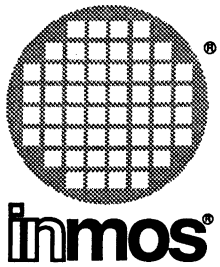


DIGITAL SIGNAL PROCESSING DATABOOK

First Edition July 1989

INMOS Databook Series

Transputer Databook
Transputer Support Databook: Development and Sub-systems
Memory Databook
Graphics Databook
Digital Signal Processing Databook

INMOS is a member of the SGS-THOMSON Microelectronics Group.

INMOS document number: 72-TRN-211-00

Printed at Bath Press Ltd, Bath

Contents overview

Contents

Preface

Digital Signal Processing is a significant area of application for INMOS devices. The INMOS Digital Signal Processing Databook has been published in response to the growing interest and requests for information concerning INMOS DSP devices.

The databook comprises an overview, engineering data and applications information for the IMS A100, A110 and A121 Digital Signal Processing devices.

The INMOS Digital Signal Processing family is a range of algorithm specific devices designed to provide high performance, cost effective solutions to signal processing problems. The summary of current devices is shown below.

In addition to DSP devices, the INMOS product range also includes transputer products, graphics devices and fast SRAMS. For further information concerning INMOS products please contact your local INMOS sales outlet.

Part	Algorithm	Order of calculation	Data rate MHz	MOPS	Military available
A100-17	1D convolution	32	2.125–8.5	68–272	yes
A100-21	1D convolution	32	2.5–10	80–320	yes
A100-30	1D convolution	32	3.75–15	120–480	yes
A110-20	1D/2D convolution	21×1, 7×3	20	420	
A121-20	2D DCT/IDCT / Filter	8×8	20	320	

Table 1

INMOS

1.1 Introduction

INMOS is a recognised leader in the development and design of high-performance integrated circuits and as a pioneer in the field of parallel processing. The company manufactures components designed to satisfy the most demanding of current processing applications and also provide an upgrade path for future applications. Current designs and development will meet the requirements of systems in the next decade. Computing requirements essentially include high-performance, flexibility and simplicity of use. These characteristics are central to the design of all INMOS products.

INMOS has a consistent record of innovation over a wide product range and supplies components to system manufacturing companies in the United States, Europe, Japan and the Far East. As developers of the Transputer, a unique microprocessor concept with a revolutionary architecture and the occam parallel processing language, INMOS has established the standards for the future exploitation of the power of parallel processing. INMOS products also include a highly successful range of high-performance graphics devices including a Colour Video Controller and a family of Colour Look Up Tables. INMOS also has a firmly established reputation as a manufacturer of high-speed static RAMs, an area in which it has achieved a greater than 10% market share.

This databook concentrates on another successful INMOS product line, an innovative range of Digital Signal Processing (DSP) devices designed for applications such as radar and sonar, communications, video-phones, robot vision and high definition television. Devices include the IMS A100 Cascadable Signal Processor, the IMS A110 Image and Signal Processor and the IMS A121 2-D Discrete Cosine Transform Image Processor.

The corporate headquarters, product design team and worldwide sales and marketing management are based at Bristol, UK.

INMOS is constantly upgrading, improving and developing its range of Digital Signal Processing products and is committed to maintaining a global position of innovation and leadership.

1.2 Manufacturing

INMOS products are currently manufactured at the INMOS Newport, Duffryn facility which began operations in 1983. This is an 8000 square metre building with a 3000 square metre cleanroom operating to Class 10 environment in the work areas.

To produce high performance products, where each microchip may consist of up to 400,000 transistors, INMOS uses advanced manufacturing equipment. Wafer steppers, plasma etchers and ion implanters form the basis of fabrication.

1.3 Assembly

Sub-contractors in Korea, Taiwan, Hong Kong and the UK are used to assemble devices.

1.4 Test

The final testing of commercial and military products is carried out at the INMOS Newport, Coed Rhedyn facility. Military final testing also takes place at Colorado Springs.

1.5 Quality and Reliability

Stringent controls of quality and reliability provide the customer with early failure rates of less than 1000 ppm and long term reliability rates of better than 100 FITs (one FIT is one failure per 1000 million hours). Requirements for military products are even more stringent.

1.6 Military

Most INMOS Digital Signal Processing devices are available, or will shortly be available, in military versions processed in full compliance with MIL-STD-883C. Further military programmes are currently in progress.

1.7 Future Developments

1.7.1 Research and Development

INMOS has achieved technical success based on a position of leadership in products and process technology in conjunction with substantial research and development investment. This investment has averaged 18% of revenues since inception and it is anticipated that the future investment levels will be increased.

1.7.2 Process Developments

One aspect of the work of the Technology Development Group at Newport is to scale the present 1.2 micron technology to 1.0 micron for products to be manufactured in 1989/90. In addition, work is in progress on the development of 0.8 micron CMOS technology.

digital signal processing overview

2.1 Introduction

Digital Signal Processing (DSP) devices permit multiplication and additions to be performed extremely quickly. This ability is particularly useful for tasks which require repetitive calculations, e.g. filtering, speech and handwriting recognition and image processing and enhancement. There are essentially two types of DSP device, low to medium performance programmable devices and high performance devices dedicated to a specific application area. Programmable devices provide the ideal solution for modest requirements (e.g. telephone echo cancelling system). These devices cannot, however, deal with the compute intensive properties of more demanding signal processing requirements.

2.2 The INMOS solution

INMOS is dedicated to the production of high performance devices. By targetting specific high performance applications the ratio of cost to performance is greatly improved. INMOS DSP devices can individually achieve between 68 and 480 MOPS. Programming is not required as the algorithms are hardwired into the device. Some of the devices can further be grouped together (cascaded) to linearly increase performance. These features combine to provide an economic, compact, low power and high performance solution.

INMOS DSP applications include mobile radio base systems, satellite communication links, studio TV equipment, image processing, handwriting recognition and radar and sonar systems. INMOS are continually working to develop new products targetted at selected signal processing applications. Current areas of interest and development are directed at such areas as video phones, where a complex image must be compressed for transmission across the telephone network, and High Definition Television (HDTV) for use in broadcasting and storage of high definition TV pictures.

INMOS DSP devices are configurable rather than programmable. While a complete system could be built solely from, for example, the A100 device (one dimensional filtering) or A110 devices (image processing) the design is such that multiple devices can be used in conjunction with a controlling microprocessor. This arrangement of controlling microprocessor and arrayable function/algorithm specific device maintains the flexibility of a programmable system but allows very high performance in a small number of devices.

INMOS DSP devices currently include the IMS A100 Cascadable Signal Processor, the IMS A110 Image and Signal Processor and The IMS A121 2-D Discrete Cosine Transform Image Processor.

2.3 The IMS A100 Cascadable Signal Processor

The IMS A100 device is suitable for processing the high-speed data streams necessary for radio, radar, sonar and satellite communications systems. The following techniques are typical:

- Convolution/correlation
- Pulse compression
- Adaptive filters
- Fourier transforms
- Beamforming
- Hilbert transforms

Radar and sonar systems

The IMS A100 device utilises CMOS technology to enable 32, 16bit multipliers to be packed into a device dissipating less than 2W. A single device is capable of between 68 and 480 MOPS. It is a simple process to connect A100 devices together to give the 1,000 to 10,000 MOPS required by radar techniques such as pulse compression. The IMS A100 device is also military qualified. It is available in both PGA and surface mount packaging to satisfy the environmental requirements of military systems.

These features are ideal for a range of radar, sonar and ultrasonic systems. Typical applications are as follows:

- Nose cone radar
- Early warning systems

- Ultrasonic weld inspection
- Phased array sonars
- Medical ultrasonics
- Towed array sonars

Communications

The IMS A100 device provides enormous filtering power. A high performance or general purpose microprocessor used to control one or more IMS A100 devices produces a high performance adaptive filter, capable of real time countering of signal attenuation, reflection and interference in systems. The following applications are typical:

- Digital satellite communications
- Mobile digital radio
- Megastream modem applications (for example, a 2Mbit/s video conferencing system)
- Telephone exchange processing for multiple ISDN data channels.

2.4 The IMS A110 Image and Signal Processor

The IMS A110 is ideal for solving a number of real time image processing problems. The following are typical:

- 1D/2D Convolution
- Statistics/histogram collection
- 1D/2D Correlation
- Non-linear data transformation
- 1D/2D Interpolation
- Image enhancement

Machine vision

The image processing requirements of machine vision systems include noise filtering, correction of distortion and enhancement of features. The IMS A110 contains a 20Mhz 7×3 multiplier array and line stores, a single device can therefore provide filters such as edge detectors in real time. More demanding requirements can readily be met by arraying IMS A110 devices. Typical applications are as follows:

- Optical alignment
- Visual inspection for defects
- Robot Vision
- Automated pathology

Image compression

The ability of the IMS A110 device to realise data compression techniques, such as sub-band/pyramid coding and linear predictive coding in real time, makes it ideal for the following applications:

- Video conferencing
- Facsimile of high-quality pictures
- Communication of images from remote observation points (e.g. satellites)
- Image archiving (e.g. medical image databases)

Contrast enhancement

Image processing systems often require contrast modification in addition to image filtering. This is supported in the IMS A110 by the inclusion of a sophisticated data transformation post processor unit providing support for the following applications:

- Histogram equalisation (enables adjustment to different lighting conditions)
- Image contouring (e.g. for simplifying medical images)
- Dynamic range compression or expansion

Other applications

- Postal sorting
- Traffic control
- Airport baggage X-ray inspection
- Handwriting and face recognition
- Bank cheque sorting and processing
- Conversion between TV standards
- Target acquisition and tracking
- Document processing
- Medical image processing
- TV special effects

2.5 The IMS A121 2-D Discrete Cosine Transform Image Processor

The IMS A121 is the latest INMOS DSP device. It has been designed to provide high speed computation of an 8×8 Discrete Cosine Transform (DCT) or Inverse Discrete Cosine Transform (IDCT) at video data rates for image processing. Typical applications are as follows:

- Image data compression and decompression (e.g. video codecs)
- Image understanding (e.g. image texture analysis)

Image compression

DCT based image coding is appropriate for a wide range of applications which require image data compression and decompression:

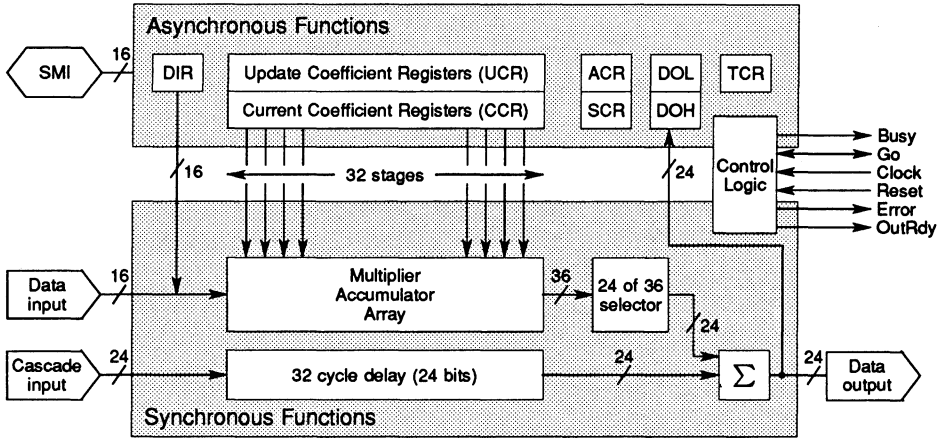
- Video conferencing and video phones
- Facsimile of colour and greyscale images
- Compact disc based interactive video systems
- Office document processing systems utilising colour or greyscale images (e.g. Document scanners, desktop publishing systems, page printers and image archiving systems)

Image understanding

The frequency domain description of an image provided by the DCT is a powerful tool for analysing textures and patterns within an image.

IMS A100
IMS A100M
Cascadable Signal Processor

Engineering Data



FEATURES

Variants for full MIL temperature range (−55°C to +125°C)

MIL-STD-883C processing

Full 16 bit, 32 stage, transversal filter

Fully cascadable with no speed degradation or reduction in dynamic range

Coefficients selectable as 4, 8, 12, or 16 bits wide

Data throughput to 15.0 MHz

High speed microprocessor compatible interface

Data input and output through dedicated ports or via the microprocessor interface

Fully static high speed CMOS implementation

Single +5V ±5% or ±10% power supply variants

TTL and CMOS compatibility

Less than 2W power dissipation

Standard 84-pin PGA or flatpack package

APPLICATIONS

Digital FIR filtering

High speed adaptive filtering

Correlation and Convolution

Discrete Fourier Transform

Speech processing using Linear Predictive Coding

Image processing

Waveform synthesis

Adaptive and fixed equalizers and echo cancellers

Spread spectrum communication

Beamforming and beamscanning in sonar and radar

Pulse compression

High speed fixed point matrix multiplication

3.1 INTRODUCTION

The IMS A100 is a high speed, high accuracy 32 stage transversal filter. Its flexible architecture allows it to be used as a 'building block' in a wide range of Digital Signal Processing (DSP) applications. The part is capable of performing high speed DFTs, convolution and correlation, as well as many filtering functions.

The input data word length is 16 bits, and coefficients are programmable to be 4, 8, 12 or 16 bits wide; two's complement numerical formats are used for both data and coefficients. The coefficients can be updated asynchronously to the system clock during normal operation, allowing the chip to be used in a variety of adaptive systems. The IMS A100 can also be cascaded to construct longer transversal filters with no additional logic or degradation in speed, whilst preserving a high degree of accuracy. The device is controlled through a standard memory interface, allowing use with any general purpose microprocessor. Data communications can be either through the memory interface, or through dedicated data ports.

3.2 DESCRIPTION

The IMS A100 is a 32 stage, cascadable, digital transversal filter. The general canonical transversal filter is shown in figure 3.1. An alternative, and functionally equivalent filter is shown in figure 3.2. It is this second realisation that is used in the IMS A100, where the input signal is supplied in parallel to all 32 multipliers, and the delay and summation operations are performed in a distributed manner.

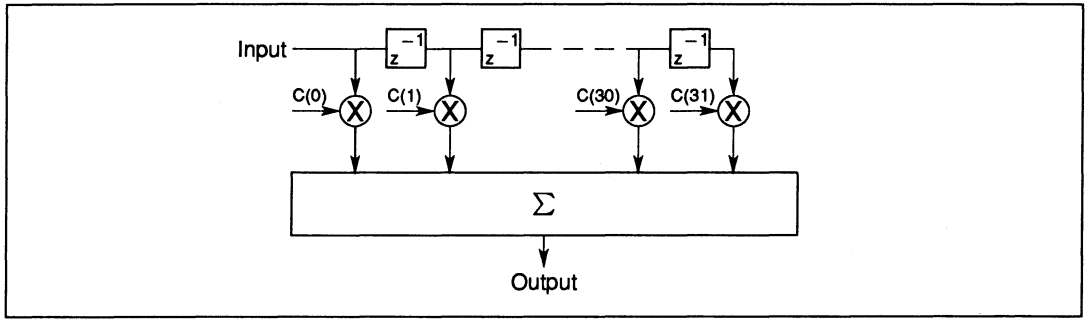


Figure 3.1 Canonical transversal filter architecture

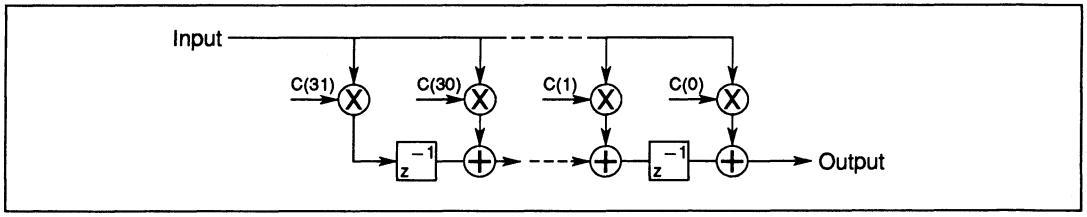


Figure 3.2 Modified transversal filter architecture

Each data sample loaded into the IMS A100 is fed in parallel to all 32 stages. At each stage the current input sample is multiplied by a coefficient stored in memory, and added to the output of the previous stage delayed by one clock cycle. The filter output at time $t = kT$ is given by:

$$y(kT) = C(0) \times x(kT) + C(1) \times x((k-1)T) + \ldots$$

$$\ldots + C(N-1) \times x((k-N+1)T)$$

where $x(kT)$ represents the kth input data sample, and $C(0)$ to $C(N-1)$ are the coefficients for the N stages.

While the IMS A100 architecture is designed as a transversal filter it contains many features which allow it to be used in a wide range of signal processing applications, e.g. adaptive filtering, matrix multiplication, discrete Fourier transforms, correlation and convolution. Figure 3.3 shows the users view of the IMS A100.

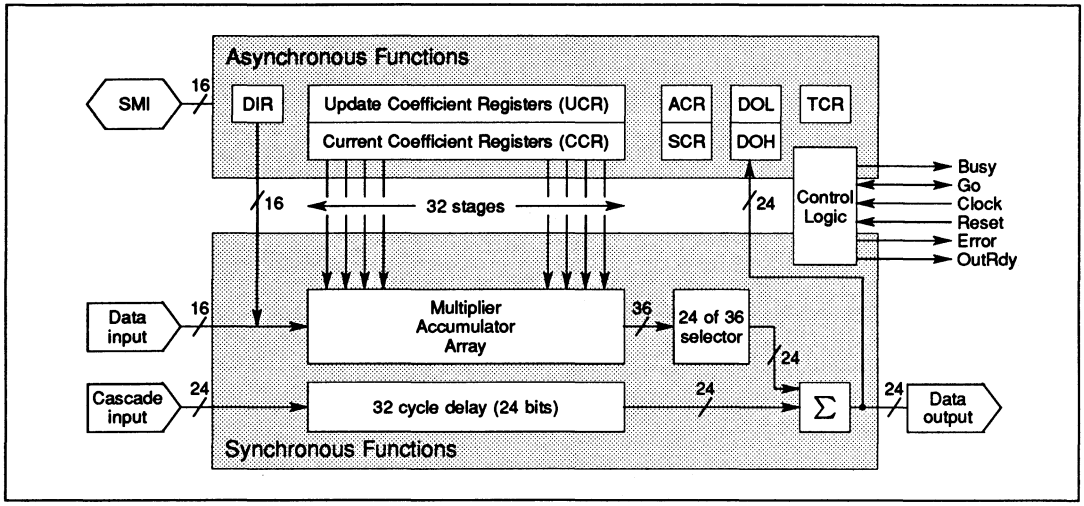


Figure 3.3 IMS A100 Users Model

The IMS A100 has four interfaces through which data can be transferred. The memory interface port allows access to the coefficent registers, the configuration and status registers and the data input and output registers for the multiplier accumulator array. Three dedicated ports are also provided, allowing high speed data input and output to the IMS A100 and the cascading of several devices.

Typically a microprocessor will configure the IMS A100 via the memory interface, then in a simple system data input and output can be performed through the data input (DIR) and data output (DOL, DOH) registers. Alternatively in a higher performance system data transfer may be performed via the dedicated input and output ports. A typical IMS A100 based system is shown in figure 3.4. Simple high-throughput fixed-configuration systems can be implemented by clocking the configuration information into the IMS A100 from a ROM.

The IMS A100 input data word width is 16 bits. The coefficient words can be programmed to be 4, 8, 12, or 16 bits wide. There is a trade off between the coefficient size and the speed of operation. If the coefficient word is L_C bits wide and the clock frequency applied to the IMS A100 is F then the maximum data throughput is $\frac{2 \times F}{L_C}$. So, for an IMS A100 operating from a 20.8 MHz clock and using 4-bit coefficients the maximum data throughput is 10.4 MHz, similarly for 16-bit coefficients the throughput is 2.6 MHz.

To preserve complete numerical accuracy, no truncation or rounding is performed on the partial products in the multiplier accumulator array. The output of this array is calculated to full precision (36 bits). A programmable barrel shifter is located at the output of this array, which allows one of five 24 bit fields to be selected from the 36 bit result. The selected 24 bits are always correctly rounded and are sign extended before being output. The selection required can be determined from analysis of the coefficients and input data used in a given application.

Two banks of coefficients are provided. At any instant one set of coefficients is in use within the multiplier accumulator array, the other set being accessible via the memory interface. Once a new set of coefficients has been loaded, the two coefficient banks can be interchanged by performing a write operation to the 'Bank Swap' bit of a control register.

So that devices can be cascaded (eg. to construct longer transversal filters), a 32 stage, 24 bit wide, shift register and 24 bit adder is included on chip. The output of one chip is connected directly to the cascade input of the next. The output of the shift register is added internally to the output of the programmable barrel

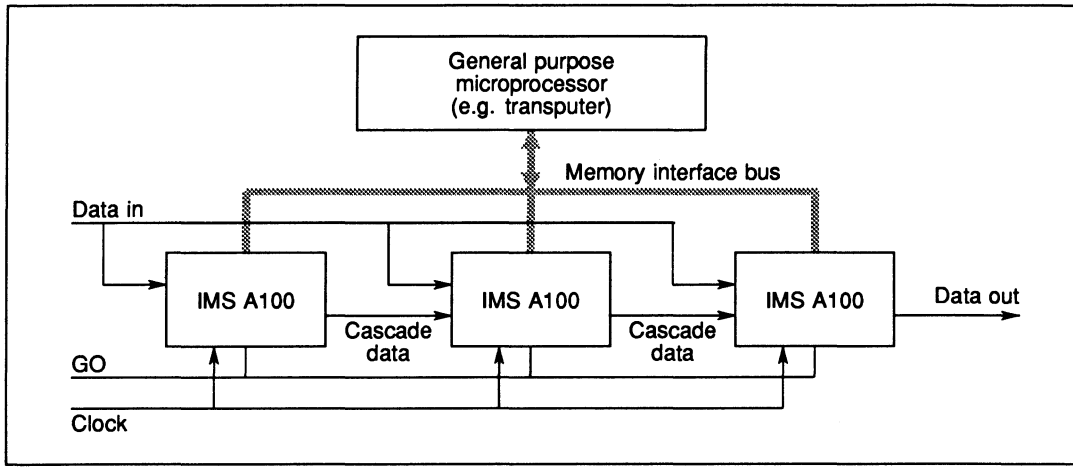


Figure 3.4 A simple IMS A100 based system

shifter to give the final 24 bit output from the chip. To minimise pin count and external buses, the data output and the cascade input ports transfer 24 bit words as a pair of 12 bit words across a 12 bit wide multiplexed interface.

As IMS A100s can be cascaded there is a price / performance trade off for most IMS A100 systems. For example, a correlation application could achieve high performance by using a cascade of IMS A100s sufficiently long to hold one of the waveforms being correlated in its coefficient registers and sending the other waveform involved in the correlation along the cascade of IMS A100s. A cheaper and slower solution would be to use a smaller number of IMS A100s and to decompose the single long correlation into a sequence of shorter correlations, the results of which are then summed.

3.3 PIN DESIGNATIONS

System services

Pin	In/out	Function
VCC, GND		Power supply and return
CLK	in	Input clock
$\overline{\text{RESET}}$	in	System reset
$\overline{\text{ERROR}}$	out	Numerical overflow error
BUSY	out	Bank swap in progress

Synchronous input/output

Pin	In/out	Function
GO	in/out	Initiate input/computation/output cycle
DIN[0-15]	in	Data input port
DOUT[0-11]	out	Data output port
CIN[0-11]	in	Cascade input port
OUTRDY	out	Output data ready

Asynchronous input/output

Pin	In/out	Function
D[0-15]	in/out	Memory interface data bus
ADR[0-6]	in	Memory interface address bus
$\overline{\text{CS}}$	in	Memory interface select
$\overline{\text{CE}}$	in	Memory interface enable
$\overline{\text{W}}$	in	Memory interface write enable

Notes

Signal names are shown with an **overbar** if they are active low, otherwise they are active high.
Pinout details are given in section 3.7

3.3.1 System services

System services include all the necessary logic to start up and maintain the IMS A100.

Power

Power is supplied to the device via the **VCC** and **GND** pins. Several of each are provided to minimise inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100nF low inductance (e.g. ceramic) capacitor between **VCC** and **GND**. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.

Input voltages must not exceed specification with respect to **VCC** and **GND**, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.

CLK

The clock input signal **CLK** controls the timing of input and output on the three dedicated ports and controls the progress of data through the multiplier accumulator array.

$\overline{\text{RESET}}$

When the IMS A100 is reset the control logic within the IMS A100 will be reset and the ACR and SCR will be initialised to their default values. Note that neither the internal data path registers nor the coefficient registers are affected by the reset. Resetting the device initialises the SCR to its default setting. So, depending on the setting of SCR before a reset, a reset may also be a device reconfiguration. The sequence of operations required to return the device to a defined state following reconfiguration is described under SCR in the register description.

A reset is initiated automatically when power is first applied to the device. This reset will be completed once four cycles of **CLK** have occured after **VCC** is valid. Alternatively reset can be initiated by taking **$\overline{\text{RESET}}$** low. This reset will be completed after at least two cycles of **CLK** have occured while **$\overline{\text{RESET}}$** is held low. **$\overline{\text{RESET}}$** should be held low for at least 200ns. Normal device operation can then continue after **$\overline{\text{RESET}}$** is taken high.

The reset should be completed before either the synchronous or asynchronous parts of the device are used.

$\overline{\text{ERROR}}$

If asserted, this pin indicates an error condition has occured, and that the condition has not been cleared. The error condition results from a numerical overflow in either the final adder or in the field selector. To allow this signal to be wire ORed between all the devices in a cascade and hence to be used as an interrupt signal to the host processor, the **$\overline{\text{ERROR}}$** outputs are open collector.

If suitably armed before the error occured the ACR error bits can be read to discriminate the two error sources. The error bits in the ACR and the error condition can be cleared and then the error bits armed to detect further errors by writing values to the ACR. The sequence of values that should be written to the ACR error bits is 0 followed by 1. An error condition can only be cleared if the error bits were suitably armed before the most recent error occured.

The ACR error bits may not observe an error occuring between clearing and arming the error bits. So, when clearing an error and arming the error bits precautions should be taken to ensure that no new error occurs. For example, first prevent the IMS A100 from initiating computation on new data; second wait for any results pending to be output; then clear and rearm. The ACR error bits will observe any error occuring after they are armed. Thus, if an error occured before the ACR error bits were armed it may be necessary to arm the error and then force an error before proceeding to clear the error (as described above).

Following power up the contents of the multiplier accumulator array and cascade path are indeterminate. As this indeterminate data flushes through a system of one or more IMS A100s errors are likely to occur. Similarly, altering the device configuration defined by the SCR is likely to result in errors. The sequence of operations required to return the device to a defined state following reconfiguration is described under SCR in the register description section of this specification.

BUSY

When high this pin indicates that an exchange of data between the Current and Update Coefficient Registers is in progress. Under certain conditions the duration of **BUSY** may be vanishingly small. **BUSY** will be active if the bank swap is caused by setting ACR[0] to request a single bank swap or when SCR[2] is set selecting Continuous Swap mode. The detailed behaviour is described in the bankswap timing diagrams.

3.3.2 Synchronous input/output

GO

The **GO** signal initiates a cycle of data input, computation and output. An IMS A100 configured as a slave will monitor the **GO** signal on the rising edge of **CLK** one cycle before it is ready to accept more data and on every rising edge thereafter until **GO** is found to be high. If **GO** is high then data input will occur on the next rising edge of **CLK**. If **GO** is low when it is sampled no new data input will occur.

In a cascade of IMS A100s one IMS A100 may be configured as a master. The master IMS A100 will drive its **GO** pin high after data has been written into its Data Input Register indicating that new data is available and that the slave IMS A100s in the casacade should start an input, computation, output cycle. When the **GO** signal goes low new data can be written to the IMS A100s. Typically a host processor will write simultaneously to the Data Input Registers of all the IMS A100s in the cascade. The host will then monitor the **GO** signal before writing new data to the cascade.

DIN[0-15]

This 16 bit wide data input port allows high speed data input to the IMS A100. The timing of this input is controlled by the **CLK** and **GO** signals. In a cascade of IMS A100s the 16 bit wide input data path and the **CLK** and **GO** signals will be bussed to all devices.

DOUT[0-11]

This 12 bit data port outputs the result from the IMS A100. The 24 bit result is multiplexed through this port as two 12 bit words, the least significant word being output first. The most significant word is output second and remains on the data pins until a new data output sequence is about to start. The **OUTRDY** signal can be used to latch these words into external circuitry. In a cascade of IMS A100s the **DOUT** pins of one device connect to the **CIN** pins of the next device in the cascade.

CIN[0-11]

The Cascade Input allows multiple IMS A100s to be cascaded. A 24 bit word is input as two 12 bit words the least significant word being input first. The 24 bit word is delayed by a shift register and summed with the output of the multiplier accumulator array. The delay from a word being input on the cascade input to that word affecting the data output is 32 data input cycles. In a typical IMS A100 based system the cascade input of each device will be connected to the data output **DOUT[0-11]** of the previous IMS A100 in the cascade. The Cascade Input of the first device in the cascade will normally be connected to ground.

OUTRDY

The output ready signal **OUTRDY** goes low just after the least significant data output word is available on the **DOUT** pins and goes high just after the most significant word is available. The rising edge of **OUTRDY** also indicates that the Data Output registers (DOL, DOH) contain the new result word. Thus the **OUTRDY** signal can either be used to latch the output of the IMS A100 into external logic or to indicate that output of the IMS A100 can be read through the memory interface from the Data Output registers.

3.3.3 Asynchronous input/output

$\overline{CS}$

This pin selects the chip; if chip select $\overline{CS}$ is low an access to the memory interface will be enabled. This signal is usually asserted by the host processors' address decoder at the begining of a memory cycle.

$\overline{CE}$

The chip enable pin. The memory interface on the IMS A100 appears to the system controlling it as 128 words of static RAM. The chip enable $\overline{CE}$ signal is similar in operation to the chip enable signal found on static RAMs. When $\overline{CE}$ is high the chip select, write enable and the address inputs are ignored and the memory interface data bus is tri-state. When chip enable is low a single read or write access is made to one of the registers within the IMS A100. Accesses to the memory interface can occur completely asynchronously to operations on the data in, cascade in and data output ports **DIN[0-15]**, **CIN[0-11]** and **DOUT[0-11]**.

$\overline{W}$

The write enable pin indicates whether the access to the IMS A100 memory interface is to be a write or a read. If $\overline{W}$ is low a write access is indicated.

ADR[0-6]

The seven bit address bus comprises pins **ADR[0-6]**. The seven bit binary value applied to the address inputs of the IMS A100 indicates which register is to be accessed.

D[0-15]

During a write to the memory interface a 16 bit word is applied to data bus pins **D[0-15]**. This word will be latched on the rising edge of chip enable $\overline{CE}$ at the end of the cycle. During a read cycle the contents of the location accessed are placed on the data pins. When $\overline{CE}$ is high the data signals are tri-state.

3.4 REGISTER DESCRIPTION

The memory map shown below indicates the primary addresses for each register. All locations between decimal addresses 64 and 75 inclusive are uniquely decoded. This group of registers is shadowed at other locations up to the 128 word boundary. The effect of reading and writing to areas in the memory map other than those shown in the table is undefined.

If the user wishes to initialise the device from a ROM addressed by a clocked counter, one of the following options applies:

1 Restrict the counter to count only from 0 to 68; this avoids writing to the data registers as well as the shadow locations.

2 Count down from 127 to zero. The initialization at the lower addresses will override spurious ones at the higher shadowed addresses.

3.4.1 Memory map †

Register	Address decimal	Address hex	Function
CCR[0-31]	32–63	20–3F	Current Coefficient Registers
UCR[0-31]	0–31	00–1F	Update Coefficient Registers
SCR	64	40	Static Control Register
	65	41	Unused location
ACR	66	42	Active Control Register
	67	43	Unused location
TCR	68	44	Test Control Register
DIR	72	48	Data Input Register
DOL	74	4A	Data Output Register (Least Significant Word)
DOH	75	4B	Data Output Register (Most Significant Word)

† All other locations accessible via the memory interface of the IMS A100 are reserved.

3.4.2 Registers

CCR[0–31]

The Current Coefficient Registers contain the coefficients currently being used by the multiplier accumlator array. CCR[0] (decimal address 32) corresponds to the coefficient register of the multiplier accumlator nearest the output of the IMS A100; i.e. this location is equivalent to C(0) in figure 3.2.

Similarly CCR[31] (decimal address 63) corresponds to C(31). The Current Coefficient Registers can be read from at any time and can be written to provided that no data processing is taking place. The effect of writing to the Current Coefficient Registers while data is being processed is undefined.

UCR[0–31]

The Update Coefficient Registers are equivalent to the Current Coefficient Registers, with the exception that the values in the Update Coefficient Registers are not currently in use within the multiplier accumlator array and can therefore be written to at any time.

A bank swap operation is equivalent to an exchange of data between the Update Coefficient Registers and the Current Coefficient Registers.

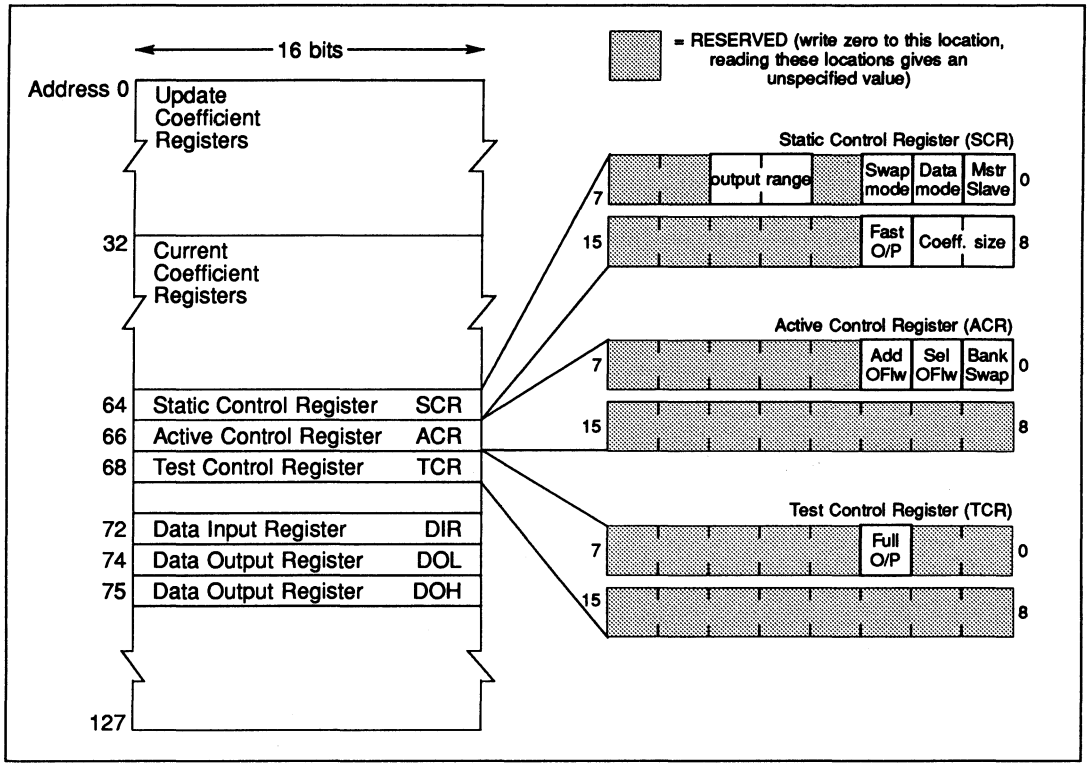


Figure 3.5 IMS A100 memory map

SCR

The Static Control Register contains the control bits which configure the IMS A100 and are unlikely to need updating after their initial configuration. The contents of the Static Control Register are not affected by the IMS A100 and can be read at any time.

Reconfiguring the SCR may result in indeterminate data values within the IMS A100 system. These values may in turn result in errors. After reconfiguring the SCR the following sequence should be followed to return the IMS A100 system to a defined, error free condition:

1 Arm error bits in ACR.

2 After SCR has been reconfigured **GO** should be held low for 20 cycles of **CLK**.

3 A series of suitable data values should then be flushed through the IMS A100 system.

4 Any errors generated should then be cleared.

5 The IMS A100 system is then ready to commence normal operation.

ACR

The Active Control Register contains status and control bits which are likely to be accessed during normal operation of the IMS A100; i.e. when handling error conditions and when requesting single coefficient bank swaps.

TCR

The Test Control Register is used for test purposes. One of the test modes provides access to the least significant part of the multiplier accumulator array output.

DIR

The Data Input Register. The IMS A100 can be configured to either take its input data from the **DIN** pins or from the Data Input Register. If the IMS A100 is configured as the master of a cascade of IMS A100s the **GO** signal will be driven in response to writing data into the Data Input Register.

In a small IMS A100 based system the Data Input Registers of all the devices in the cascade will normally be mapped into the same location within the address space of the processor controlling the cascade. Thus a single write operation can write data to all devices, the master IMS A100 generating the **GO** signal for the slaves. The Data Input Register is write only.

DOL

The least significant word of the Data Output Register. The output data from the IMS A100 is available from both the **DOUT[0–11]** pins and from the Data Output Registers. The value held in the Data Output Registers is the 24 bit output word, sign extended to 32 bits. DOL contains the least significant 16 bits of the 24 bit result; the register is read only.

DOH

The most significant word of the Data Output Register. The DOH register contains the most significant 8 bits of the 24 bit output word generated by the IMS A100. The most significant 8 bits of DOH are the sign extension of the output word. DOH is read only.

The remainder of this section describes the register details bit by bit. Each section commences with the name of the register with the bit number(s) followed by the default value, in the general format:

Name REGISTER[MSB–LSB] Default: MSB LSB

The least significant bit of a register is bit 0.

† in the tables indicates the default state of the register bit(s).

3.4.3 Static control register

Fast Output SCR[10] Default: 0

The Fast Output bit controls the way in which the 24 bit output of the IMS A100 is multiplexed across the 12 bit wide **DOUT** port. The interval between data output cycles is the same for both Normal and Fast output modes.

The difference between the modes is the time division between the least and most significant words. In fast output mode the least significant 12 bit word is available for the minimum period possible, thus allowing the most significant word to be output at the earliest possible instant. In normal output mode the least significant word is available for the same length of time as the most significant word (unless the duration of the most significant word is extended by idle cycles).

The timing constraints on data output in Normal mode are significantly simpler than those in Fast mode. Fast mode should be considered a special mode which is only used where the early availability of the output words is important, e.g. an adaptive system where the filter coefficients are being modified in response to the output data.

All devices in a cascade of IMS A100s should be configured for the same output mode. The Fast Output bit should not be altered during data processing. If it is altered the data output of the cascade will be undefined

until new input data has flushed through all stages of the cascade. If the coefficient size is 4 bits there is no difference between the fast and normal modes.

SCR[10]	Output mode
0	Normal †
1	Fast

Coefficient Size SCR[9–8] Default: 1 1

Defines size of coefficient used, in terms of word width. This also determines the minimum interval between data input cycles and thus the data throughput of the IMS A100. The Coefficient Size bits should not be altered during data processing. If they are altered the data output of the cascade will be undefined until new input data has flushed through all stages of the cascade.

In each mode the coefficient data is the least signifcant bits of the 16 bit word; e.g. in 4 bit mode, a two's complement number should be programmed into bits 0–3 of the 16 bit register. The remaining bits 4–15 are ignored.

SCR[9–8]	Coefficient size	Data input interval
0 0	4 bits	2 cycles
0 1	8 bits	4 cycles
1 0	12 bits	6 cycles
1 1	16 bits	8 cycles †

Reserved SCR[7–6] Default: 0 0

These locations are reserved. The user should write 0,0 to these locations to maintain compatability with future products. The value read from this location is undefined.

Reserved SCR[3] Default: 0

This location is reserved. The user should write 0 to this location to maintain compatability with future products. The value read from this location is undefined.

Output Word Selection SCR[5–4] Default: 1 0

These bits determine the 24 bit wide field selected from the 36 bit wide output of the multiplier accumulator array (bit positions numbered 0 to 35). The word selected will be rounded and sign extended before being output. Note that ranges '10' and '11' imply sign extension of the result.

The Output Word Selection bits should not be altered during data processing. If they are altered the data output of the cascade will be undefined until new input data has flushed through all stages of the cascade.

SCR[5–4]	Field
0 0	[7–30]
0 1	[11–34]
1 0	[15–38] †
1 1	[20–43]

Continuous Swap SCR[2] Default: 0

The Continuous Swap bit selects whether the two banks of coefficient registers are automatically exchanged after each data input and computation cycle or if individual bank swaps occur under the direction of the Bank Swap bit in the Active Control Register, ACR[0]. SCR[2] should not be set if a bankswap has been requested (by setting ACR[0]) and is still pending.

SCR[2]	Swap Mode
0	Swap on asserting ACR[0] †
1	Swap after end of each input cycle

Input Data Source SCR[1] Default: 0

The data source for the multiplier accumulator array can come from one of two sources, selected by SCR[1]. Data can either be input from the **DIN** port or it can be written into the Data Input Register via the memory interface. See also the following section.

SCR[1]	Data Source
0	From **DIN** port †
1	From DIR

Master not Slave SCR[0] Default: 0

The Master not Slave bit selects whether the IMS A100 samples the **GO** input to determine the start of a data input cycle (slave mode), or drives the **GO** pin when data is written to the DIR (master mode). If input data is supplied through the DIR one IMS A100 in the cascade should be configured as a master. If data is supplied to the **DIN** port by an external data source all the IMS A100s in the cascade should be configured as slaves and **GO** should be driven by an external system. Note that an illegal mode results if SCR[1] is 0 and SCR[0] is 1; i.e. a master cannot obtain data from the **DIN** port.

SCR[0]	Mode
0	Slave †
1	Master

3.4.4 Active control register

Cascade Adder Overflow ACR[2] Default: 0

If previously armed this status bit will be set if the addition of the 24 bit words output by the 24 from 36 bit selector (on the output of the multiply accumulator array) and the cascade shift register causes an arithmetic overflow. The $\overline{\text{ERROR}}$ pin will be driven low while this or any other error condition is active. This error bit and the error condition can be cleared by writing a zero to ACR[2], provided the data in the adder is no longer in error. After clearing this error bit the error bit should be armed (by writing a one to ACR[2]) to ensure that any future error is detected. See $\overline{\text{ERROR}}$ section.

Selector Overflow ACR[1] Default: 0

If previously armed this status bit will be set if the 24 bit output range of the selector does not include all the significant binary digits in the 36 bit result generated by the multiply accumulator array. The $\overline{\text{ERROR}}$ pin will be driven low while this or any other error condition is active. This error bit and the error condition can be cleared by writing a zero to ACR[1]. After clearing this error bit the error bit should be armed (by writing a one to ACR[1]) to ensure that any future error is detected. See $\overline{\text{ERROR}}$ section.

Initiate Bank Swap ACR[0] Default: 0

Writing a one into this control bit requests an exchange of data between the Current and Update Coefficient Registers. The bank swap will occur as soon as the current computation cycle is completed, or on the next clock cycle if the IMS A100 is idle. This control bit is cleared to zero by the IMS A100 when the bank swap is complete. No access should be made to either set of coefficient registers while a bank swap is in progress. ACR[0] should not be set if SCR[2] is already set. For a detailed description of the behaviour see the bankswap and coefficient access timing diagrams.

3.4.5 Test control register

Examine Full Output Word TCR[2] Default: 0

This bit overrides the output word selection normally made by bits SCR[5–4]. The output word selection determines the 24 bit wide field selected from the 36 bit wide output of the multiply accumulator array (bit positions numbered 0 to 35). When TCR[2] is set to '1' the output word selection is bits '–1' to 22, where bit '–1' is set to zero. The output word selection should not be altered during data processing. If altered the data output of the cascade will be undefined until new input data has flushed through all stages of the cascade.

TCR[2]	Field
0	Set by SCR[5–4] †
1	[–1–22]

Reserved TCR[1] Default: 0

This location is reserved for INMOS test purposes. For normal operation the user should write 0 to this location.

Reserved TCR[0] Default: 0

This location is reserved for INMOS test purposes. For normal operation the user should write 0 to this location.

3.5 DEVICE APPLICATIONS

The IMS A100 can be used in a variety of different applications requiring high performance computation. Some of these are described below, and are covered in detail in the IMS A100 Application Note series, available from INMOS.

3.5.1 Filtering and adaptive filtering

The IMS A100 device can be used to implement high speed FIR and IIR digital filters. The maximum sampling frequency of the input signal ranges between 2.125MHz and 15MHz, depending on the coefficient word length and speed variant that has been selected.

The continuous bank swap mode allows a single device to filter complex (I & Q) data streams. High speed random access coefficient registers enable high performance adaptive filters and equalisers to be realised with minimal complexity.

The cascadability of the device enables FIRs of greater than 32 stages to be constructed, with no degradation in data throughput.

3.5.2 Convolution and correlation

The IMS A100 is the first single-chip digital correlator capable of highly accurate computation of correlation and convolution functions (16-bit coefficients, 16-bit data and 36-bit accumulation). These functions have applications in matched filtering, noise reduction and pulse compression in communication, radar and sonar systems.

For correlations and convolutions involving a large number of data points, devices can be cascaded to several thousand stages with careful design. Alternatively, it is possible to use algorithms which allow decomposition of long correlation and convolutions into several smaller ones, which can then be carried out by a single or smaller number of devices.

3.5.3 Matrix multiplication

The architecture of the IMS A100 allows very high speed fixed point matrix multiplication. In this application the columns of the multiplier matrix are circulated as inputs to the chip while the coefficients are programmed in a suitable manner with the elements of the multiplicant matrix. Larger matrices can be handled by either cascading several chips or by decomposing the matrices into smaller ones.

3.5.4 Fourier transforms

Two algorithms, namely the Prime Number Transform (PNT) and the Chirp-Z Transform (CZT), can be used to perform high speed Fourier transforms using IMS A100s. The Fourier transform of long data sequences can be evaluated either by using cascaded IMS A100s or by using decomposition algorithms to convert a long transform into a number of short transforms (e.g. <32 points). These short transforms can then be carried out using the IMS A100s and a host processor.

The speed of transform can be traded off against the number of chips employed. Any microprocessor with a standard memory interface could be used to handle intermediate results and to control the overall system. Two IMS A100s can be used to perform a transform of about 1000 points in around 1ms to 2ms using look-up ROMs for address generation and high speed DSP controllers, or 5ms to 10ms using a microprocessor as the controller. More IMS A100s can be used if higher performance is required.

3.5.5 Waveform synthesis

The programmability of this digital transversal filter allows the IMS A100 to be used for flexible waveform generation and synthesis, by exploiting the ability to change coefficients randomly, quickly and simply. Such a configuration could be attractive for PC based synthesisers, as the chip can generate very accurate high bandwidth signals.

3.5.6 General purpose accelerator

By attaching one or more IMS A100s to any computer with DMA capability, a useful accelerator can be constructed, capable of handling all of the above applications without reconfiguration. The cascadability of the device enables users to add IMS A100s as required for extra processing performance, with minimal impact on the driving software.

3.6 ELECTRICAL SPECIFICATION

The IMS A100 is available in several speed, package and temperature variants (see section 3.9 – Ordering details) and the electrical characteristics of each are described in this section. When no variant is identified the information refers to all variants.

3.6.1 DC electrical characteristics

Absolute maximum ratings

Symbol	Parameter	Min.	Max.	Units	Notes (1)
VCC	DC supply voltage	0	7.0	V	2,3
VI, VO	Voltage on input and output pins	−1.0	VCC+0.5	V	2,3
TS	Storage temperature	−65	150	°C	2
TA	Temperature under bias	−55	125	°C	2
PDmax	Power dissipation		2.0	W	2

Notes

1 All voltages are with respect to **GND**.

2 This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedence circuit. Unused inputs should be tied to an appropriate logic level such as **VCC** or **GND**.

DC operating conditions

Symbol	Parameter	Min.	Nom.	Max.	Units	Notes (1)
VCC	DC supply voltage	4.5		5.5	V	4
VCC		4.75		5.25	V	5,6,7
VIH	Input Logic '1' Voltage **CLK**	4.0		VCC+0.5	V	2
	Input Logic '1' Voltage $\overline{\textbf{RESET}}$	2.4		VCC+0.5	V	2
	Input Logic '1' Voltage other pins	2.0		VCC+0.5	V	2
VIL	Input Logic '0' Voltage **CLK**	−0.5		0.5	V	2
	Input Logic '0' Voltage $\overline{\textbf{RESET}}$	−0.5		0.8	V	2
	Input Logic '0' Voltage other pins	−0.5		0.8	V	2
TA	Ambient Operating Temperature	0		70	°C	3,4,7
TA		−55		125	°C	3,5,6

Notes

1 All voltages are with respect to **GND**. All **GND** pins must be connected to **GND**.

2 Input signal transients up to 10 ns wide, are permitted in the voltage ranges (**GND** − 0.5 V) to (**GND** − 1.0 V) and **VCC** + 0.5 V to **VCC** + 1.0 V.

3 400 linear ft/min transverse air flow.

4 IMS A100-G21S, IMS A100-Q21S.

5 IMS A100-G21M, IMS A100-Q21M.

6 IMS A100-G17M.

7 IMS A100-G30S.

DC characteristics

Symbol	Parameter	Min.	Max.	Units	Notes (1,2)
VOH	Output Logic '1' Voltage	2.4	VCC	V	4
VOL	Output Logic '0' Voltage	0	0.4	V	5
II	Input current @ GND<VI<VCC		±10	μA	
IOZ	Tristate output current @ GND<VI<VCC		±10	μA	
ICC	Average power supply current		360	mA	3

Notes

1 All voltages are with respect to **GND**. All **GND** pins must be connected to **GND**.

2 Parameters measured over variants full voltage and temperature operating range.

3 Power dissipation is application dependent and varies with output loading. The maximum given here is for worst case data patterns and activity on all interfaces, with no DC load on outputs.

4 **OUTRDY, DOUT**: **IOut** ≤ -4.4 mA; $\overline{\textbf{ERROR}}$ is open collector; other outputs: **IOut** ≤ -5.5 mA.

5 **OUTRDY, DOUT**: **IOut** ≤ 4.4 mA; $\overline{\textbf{ERROR}}$: **IOut** ≤ 5.5 mA; other outputs: **IOut** ≤ 5.5 mA.

Capacitance

Pin	Typ.	Units	Notes
CLK	12	pF	1,2
All other pins	5	pF	1,2

1 This parameter is supplied for engineering guidance and is not guaranteed.

2 TA=25°C , F=1 MHz.

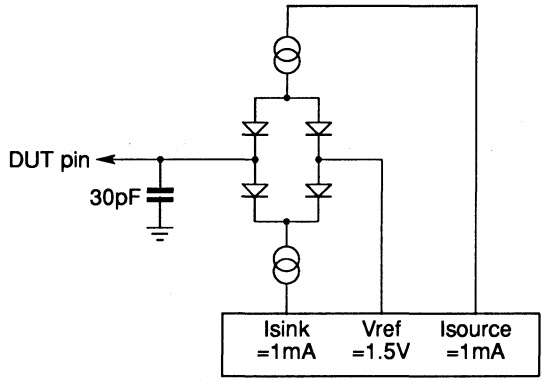
3.6.2 AC timing characteristics

AC test conditions

Output loads (except output turn-off tests)

Pin	Device mode	Load	Unit
GO	Master	20	pF
DOUT, OUTRDY	Fast output	15	pF
DOUT, OUTRDY	Normal output	30	pF
All other outputs	All modes	30	pF

Output load (output turn-off tests)



Timing reference levels

Pin	Reference levels	Notes
INPUTS	0.8V, 2.0V	1
CLK	0.5V, 4.0V	
OUTPUTS	0.4V, 2.4V	2,3
OUTPUTS	±100mV change from previous steady output voltage	4

Notes

1 Except **CLK**.

2 Output continuously driven.

3 Timings are tested using VOL=0.8V and with a suitable allowance for the time taken for the output to fall from 0.8V to 0.4V.

4 Output turn-off tests.

Clock

Symbol	Parameter	Min.	Max.	Units	Notes
t_{CHCL}	Clock pulse width high	19		ns	2
t_{CHCL}		24		ns	3
t_{CHCL}		13		ns	4
t_{CLCH}	Clock pulse width low	19		ns	2
t_{CLCH}		24		ns	3
t_{CLCH}		13		ns	4
t_{CHCH}	Clock period	48		ns	2
t_{CHCH}		58		ns	3
t_{CHCH}		33		ns	4
t_R	Clock rise time	0	50	ns	1
t_F	Clock fall time	0	50	ns	1

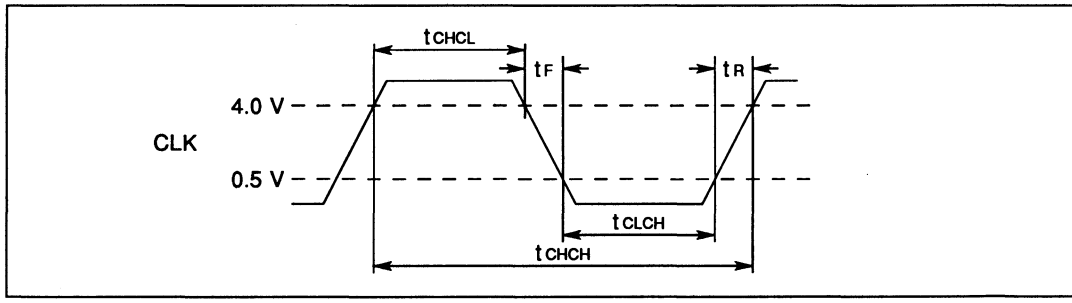
Notes

1 Clock input transitions should be monotonic between the input thresholds of 0.5 V and 4.0 V.

2 IMS A100-G21S, IMS A100-Q21S, IMS A100-G21M, IMS A100-Q21M.

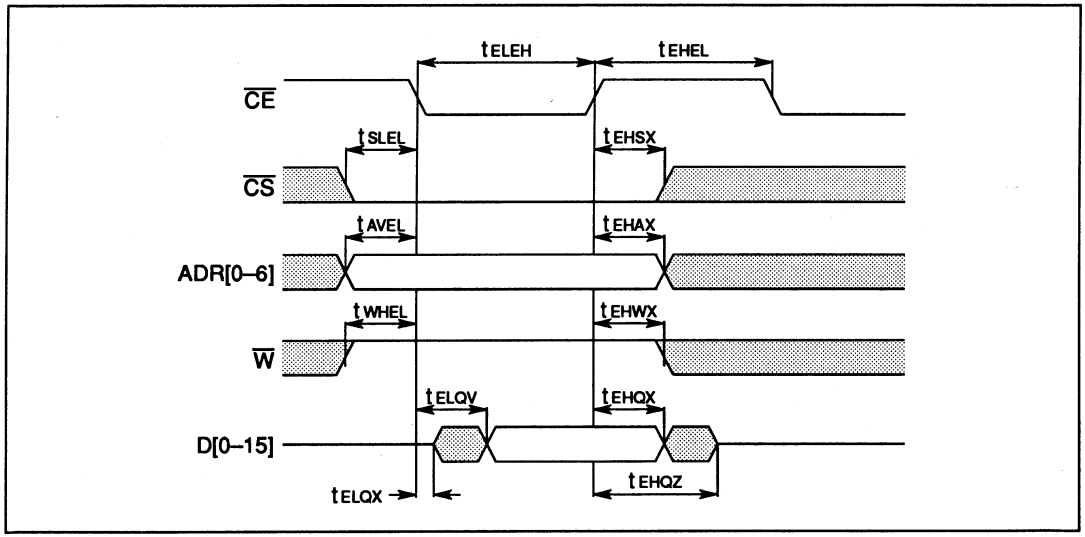
3 IMS A100-G17M

4 IMS A100-G30S



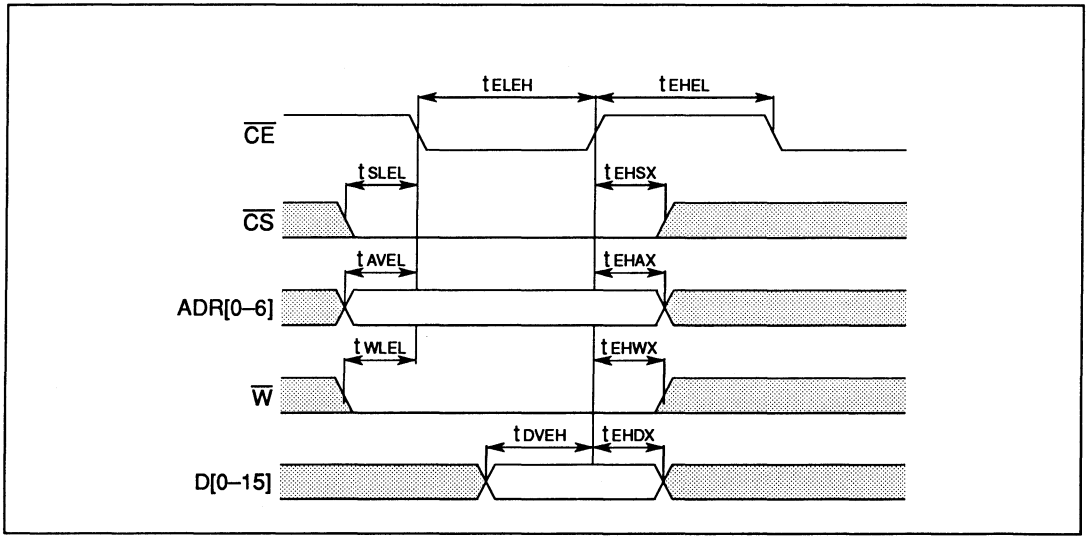
Memory interface read cycle

Symbol	Parameter	Min.	Max.	Units	Notes
t_{ELEH}	$\overline{CE}$ pulse width low	60		ns	
t_{EHEL}	$\overline{CE}$ pulse width high	50		ns	
t_{SLEL}	$\overline{CS}$ setup time	15		ns	
t_{EHSX}	$\overline{CS}$ hold time	5		ns	
t_{AVEL}	Address setup time	15		ns	
t_{EHAX}	Address hold time	5		ns	
t_{WHEL}	Read Command setup	15		ns	
t_{EHWX}	Read Command hold	5		ns	
t_{ELQX}	Output turn on delay	0		ns	
t_{ELQV}	Read data access		60	ns	
t_{EHQX}	Read data hold	0		ns	
t_{EHQZ}	Output turn off delay		25	ns	



Memory interface write cycle

Symbol	Parameter	Min.	Max.	Units	Notes
t_{ELEH}	$\overline{CE}$ pulse width low	50		ns	
t_{EHEL}	$\overline{CE}$ pulse width high	50		ns	
t_{SLEL}	$\overline{CS}$ setup time	15		ns	
t_{EHSX}	$\overline{CS}$ hold time	5		ns	
t_{AVEL}	Address setup time	15		ns	
t_{EHAX}	Address hold time	5		ns	
t_{WLEL}	Write Command setup	15		ns	
t_{EHWX}	Write Command hold	5		ns	
t_{DVEH}	Write data setup	45		ns	
t_{EHDX}	Write data hold	5		ns	



Static read accesses to DOL and DOH registers

Certain applications require to read results from the IMS A100 at high speeds. To ensure full system performance it may be necessary to read results from the DOL and DOH registers using a continuous 'static' access rather than using the normal clocked access.

During static access the $\overline{CE}$ signal is held low continuously. Under this condition it is possible to monitor either DOL or DOH continuously to observe new output words as they become available or alternatively to switch between DOL and DOH without the restriction of having to sequence $\overline{CE}$.

Symbol	Parameter	Min	Max	Units	Notes
t_{AVQV}	Address access time		75	ns	1
t_{CHQV}	Data input access time		$\mathcal{T}$+75	ns	2
t_{ELQV}	$\overline{CE}$ access time		60	ns	3
t_{AXQX}	Data hold after address change	0		ns	
t_{CHQX}	Data hold after new data input	$\mathcal{T}$+0		ns	2
t_{EHQX}	Data hold after end of read	0		ns	

Notes

1 The address access time is specified for address transitions between decimal 74 (DOL register) and decimal 75 (DOH register) only.

2 The parameter $\mathcal{T}$ describes the time taken from the input of a data word to that data word first affecting the most significant word (MSW) output. This is the time at which the DOL and DOH registers are updated.

The duration of $\mathcal{T}$ depends on the coefficient size selected and whether fast or normal output is selected.

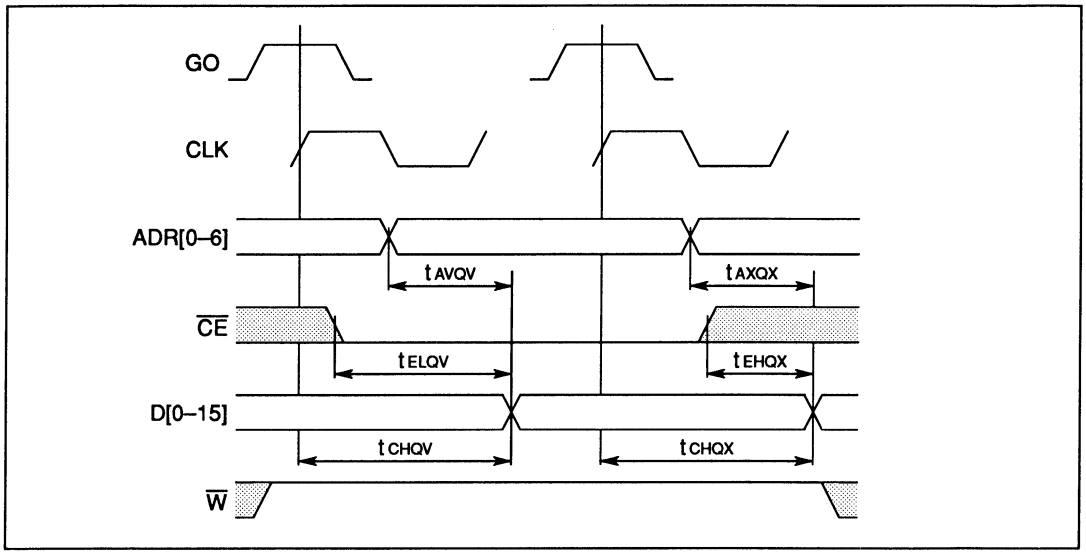
Coefficients	Output mode	$\mathcal{T}$ time
4 bit	Fast	8 CLK cycles
8 bit		10 CLK cycles
12 bit		12 CLK cycles
16 bit		14 CLK cycles
4 bit	Normal	Not defined
8 bit		11 CLK cycles
12 bit		14 CLK cycles
16 bit		17 CLK cycles

N.B. The data value read from either DOL or DOH will change as new results are computed by the device.

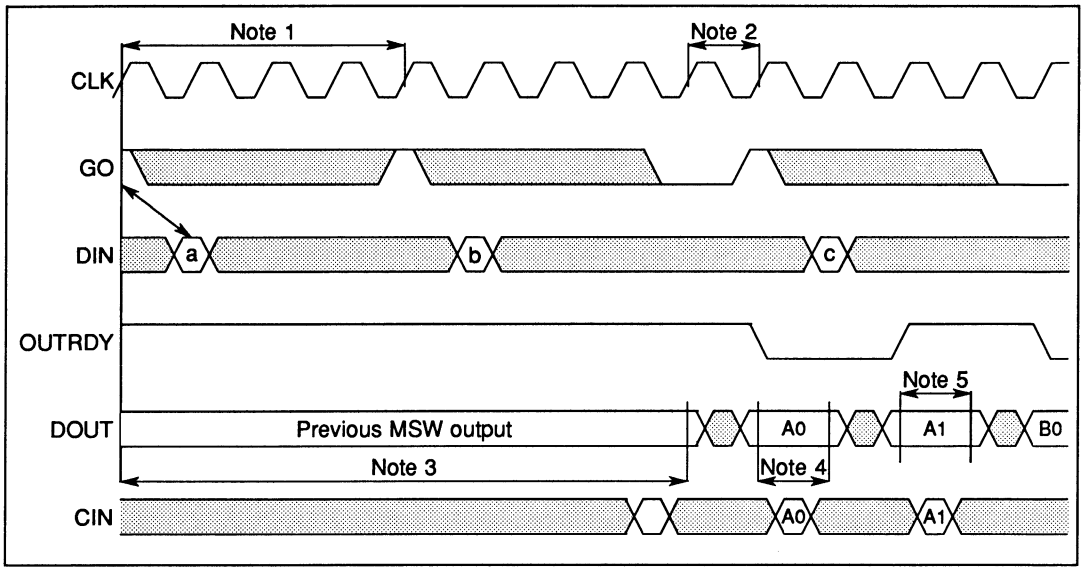
3 This parameter is the normal read access time for reading any register through the microprocessor interface. In the special case of performing reads from only DOL and DOH any number of reads from these registers can be made with $\overline{CE}$ held low continuously.

It is required that a static access (as described above) should commence like a normal clocked, random, read access to either DOL or DOH. That is **ADDRESS**, $\overline{CS}$ and $\overline{W}$ should be established with setup times to $\overline{CE}$ specified for a normal read access.

During a DOL/DOH static access sequence accesses to locations other than DOL and DOH are undefined.


GO
CLK
ADR[0–6]
tAVQV
tAXQX
CE
tELQV
tEHQX
D[0–15]
tCHQV
tCHQX
W

Typical sequence — 8 bit coefficients, normal output



Notes

1 The minimum period between sampling the **GO** input is four clock cycles for 8 bit coefficients, see the table below for the other cases.

2 After the minimum period described in note 1 has elapsed **GO** is sampled on every rising edge of **CLK** until **GO** is high.

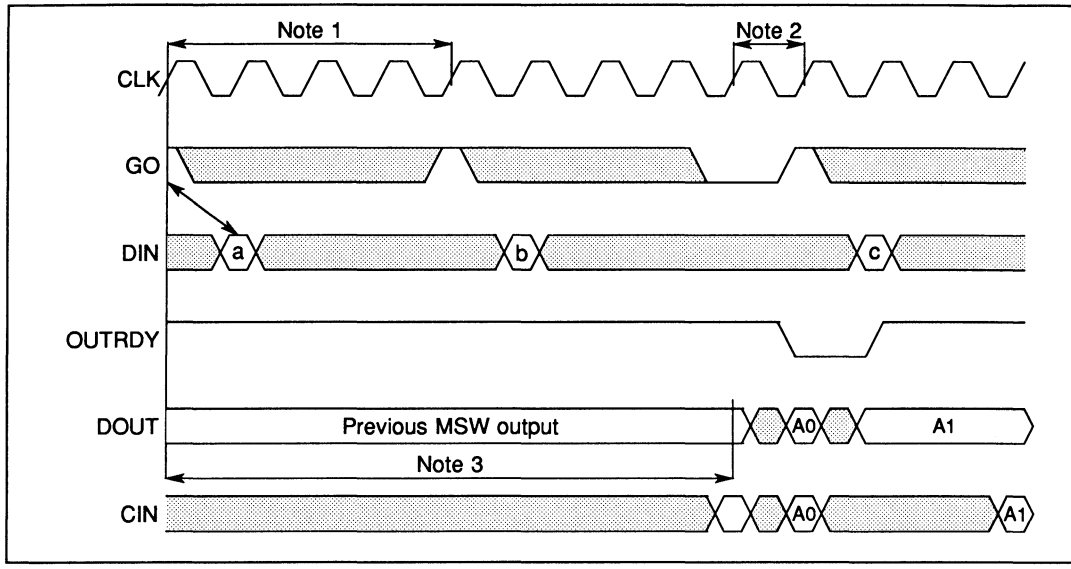
3 The delay from an output being initiated by **GO** to the output completing its previous output sequence and starting the new output sequence is 8 clock cycles for 8 bit coefficients, see the table below for the other cases.

4 The least significant word is available at the output across one complete **CLK** cycle for the 8 bit coefficient, normal output case, see the table below for the other cases.

5 The most significant word is available for the minimum period described in note 4, but will be extended by a clock cycle for each additional idle cycle inserted between data inputs.

Coefficients	Min. Output Period note 1	Delay To Output note 3	Min. LSW Output Duration notes 4 and 5
4 bit	2 CLK cycles	6 CLK cycles	Undefined, no normal output
8 bit	4 CLK cycles	8 CLK cycles	1 CLK cycle
12 bit	6 CLK cycles	10 CLK cycles	2 CLK cycles
16 bit	8 CLK cycles	12 CLK cycles	3 CLK cycles

Typical sequence — 8 bit coefficients, fast output



Notes

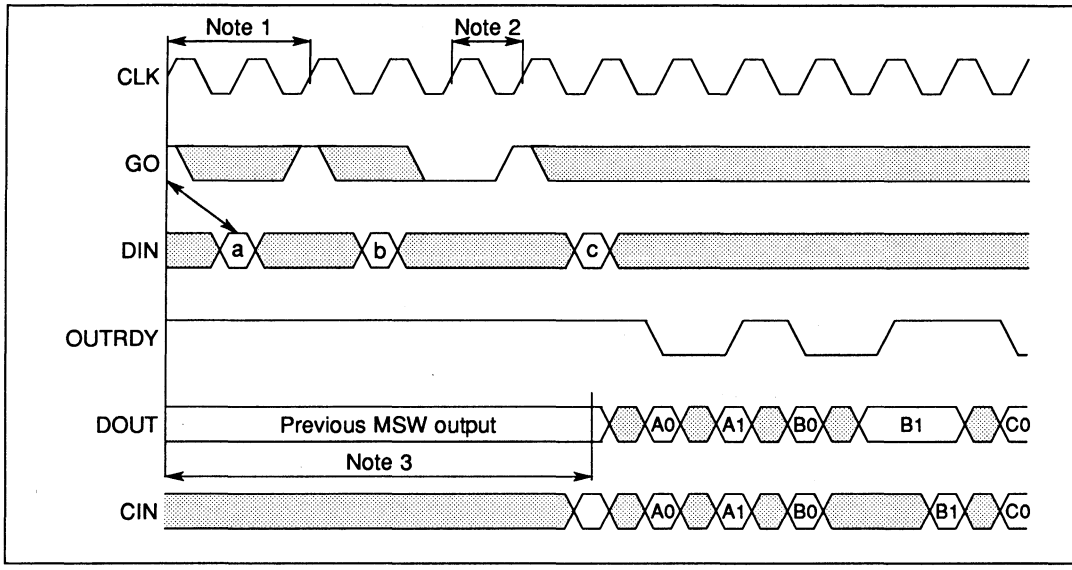
1 The minimum period between sampling the **GO** input is four clock cycles for 8 bit coefficients, see the table below for the other cases.

2 After the minimum period described in note 1 has elapsed **GO** is sampled on every rising edge of **CLK** until **GO** is high.

3 The delay from an output being initiated by **GO** to the output completing its previous output sequence and starting the new output sequence is 8 clock cycles for 8 bit coefficients, see the table below for the other cases.

Coefficients	**Min. Output Period** note 1	**Delay To Output** note 3
4 bit	2 CLK cycles	6 CLK cycles
8 bit	4 CLK cycles	8 CLK cycles
12 bit	6 CLK cycles	10 CLK cycles
16 bit	8 CLK cycles	12 CLK cycles

Typical sequence — 4 bit coefficients



Notes

1 The minimum period between sampling the **GO** input is two clock cycles for 4 bit coefficients, see the table below for the other cases.

2 After the minimum period described in note 1 has elapsed **GO** is sampled on every rising edge of **CLK** until **GO** is high.

3 The delay from an input being initiated by **GO** to the output completing its previous output sequence and starting the new output sequence is 6 clock cycles for 4 bit coefficients, see the table below for the other cases.

Coefficients	Min. Output Period note 1	Delay To Output note 3
4 bit	2 CLK cycles	6 CLK cycles
8 bit	4 CLK cycles	8 CLK cycles
12 bit	6 CLK cycles	10 CLK cycles
16 bit	8 CLK cycles	12 CLK cycles

Normal output timing — 8 bit coefficient case shown

Symbol	Parameter	Min.	Max.	Units	Notes
tCHQV	CLK high to DOUT valid delay		36	ns	3
tCHQV			25	ns	5
tCHQV			40	ns	4
tCHQX	DOUT hold time after CLK high	2		ns	
tQVRL	DOUT to OUTRDY low lead	15		ns	3,4
tQVRL		10		ns	5
tRLQX	DOUT hold time after OUTRDY low	10		ns	1
tQVRH	DOUT to OUTRDY high lead	15		ns	3,4
tQVRH		10		ns	5
tRHQX	DOUT hold time after OUTRDY high	10		ns	1,2
TIME1	LSW output duration	1	3	tCHCH	1
TIME2	MSW output duration	1	3	tCHCH	1,2
tDVCL	CASIN setup time to CLK low	10		ns	3,5
tDVCL		14		ns	4
tCLDX	CASIN hold time from CLK low	10		ns	

Notes

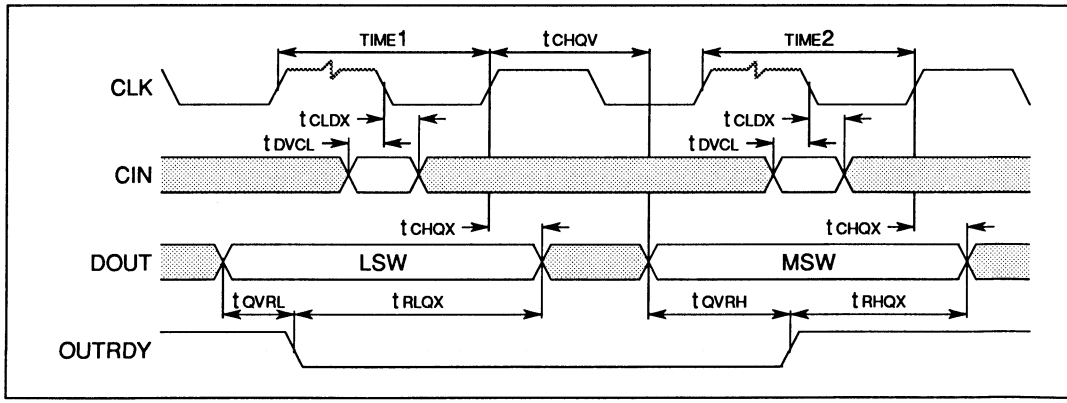
1 This parameter is determined by the coefficient size in use. The minimum value given is correct for 8 bit coefficients. This parameter is extended by 1 (or 2) periods of **CLK** if 12 (or 16) bit coefficients are used. This mode of operation is not defined if 4 bit coefficients are used.

2 These parameters are extended by one tCHCH for each idle cycle inserted between data input sequences.

3 IMS A100-G21S, IMS A100-Q21S, IMS A100-G21M, IMS A100-Q21M.

4 IMS A100-G17M.

5 IMS A100-G30S.



Fast output timing — 4 bit coefficient case shown

Symbol	Parameter	Min.	Max.	Units	Notes
t_{CHQV}	CLK high to DOUT valid delay		36	ns	1,3
t_{CHQV}			22	ns	1,5
t_{CHQV}			40	ns	1,4
t_{CHQX}	DOUT hold time after CLK	2		ns	2
t_{QVRL}	DOUT to OUTRDY low lead	5		ns	1,6
t_{RLQX}	DOUT hold time after OUTRDY low	10		ns	6
t_{QVRH}	DOUT to OUTRDY high lead	5		ns	1,6
t_{RHQX}	DOUT hold time after OUTRDY high	10		ns	2,6
t_{DVCH}	CASIN setup time to CLK high	10		ns	3,5
t_{DVCH}		14		ns	4
t_{CHDX}	CASIN hold time to CLK high	0		ns	

Notes

1 These parameters assume that each **DOUT** signal is loaded with a maximum of 15 pF.

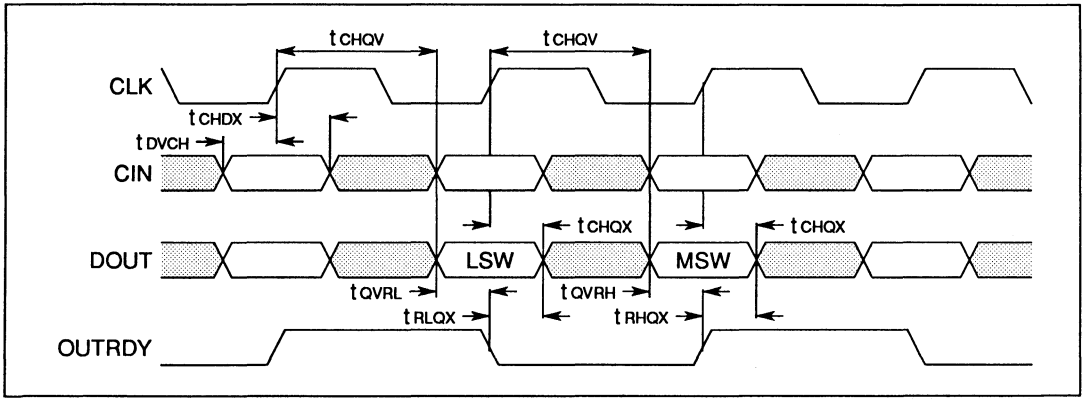
2 t_{CHQX} and t_{RHQX} for the MSW are shown here for the case where 4 bit coefficients are being used. In the other cases (8, 12 and 16 bit coefficients) the MSW is available for an additional 2, 4 or 6 **CLK** periods. In all cases the MSW will be available for an additional period of **CLK** for each idle cycle inserted between data input sequences.

3 IMS A100-G21S, IMS A100-Q21S, IMS A100-G21M, IMS A100-Q21M.

4 IMS A100-G17M.

5 IMS A100-G30S.

6 The **OUTRDY** signal should not be used in this mode using the IMS A100-G30S variant at clock frequencies above 20.8 MHz.



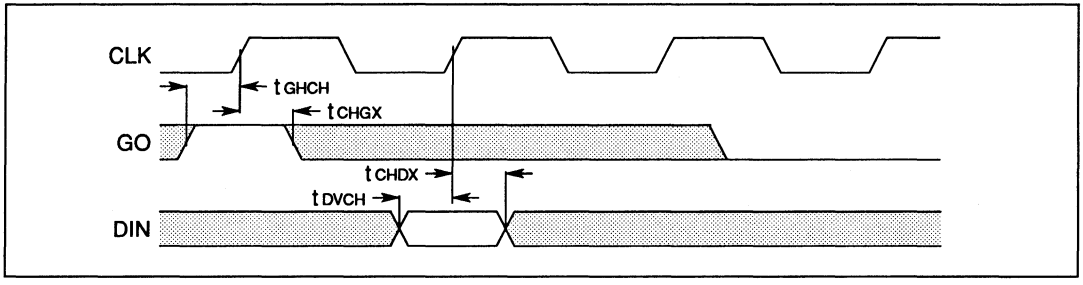
External GO and data input timing

Symbol	Parameter	Min.	Max.	Units	Notes
t_{GHCH}	GO setup time	10		ns	
t_{CHGX}	GO hold time	5		ns	
t_{DVCH}	DIN setup time	30		ns	1
t_{DVCH}		17		ns	2
t_{CHDX}	DIN hold time	5		ns	

Notes

1 IMS A100-G21S, IMS A100-Q21S, IMS A100-G21M, IMS A100-Q21M, IMS A100-G17M.

2 IMS A100-G30S.



Master generated GO

Symbol	Parameter	Min.	Max.	Units	Notes
t_{EHGH}	Write to DIR to GO high delay	25		ns	1,4
t_{GHCH}	GO high before GO sampled	10		ns	2,4
t_{GLEL}	GO low to write to DIR	0		ns	4
t_{GLCH}	GO low before GO next sampled	10		ns	2,4

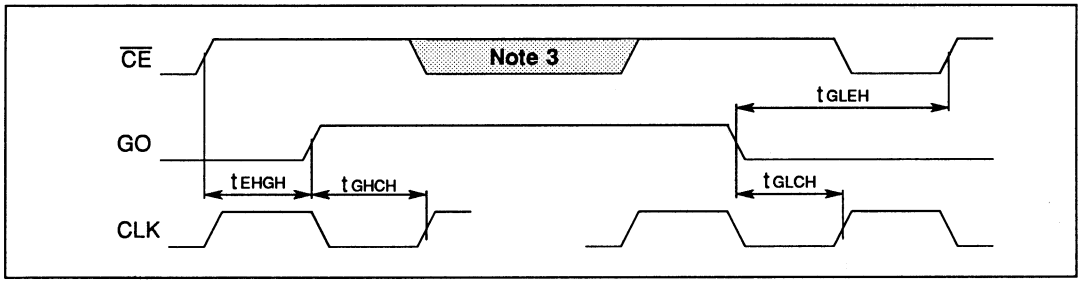
Notes

1 The maximum delay from a write to the DIR to **GO** going high is 2 * t_{CHCH} + 50 ns.

2 This parameter assumes the capacitive load on **GO** is less than 20 pF. **GO** is specified so that one master IMS A100 can drive three slave IMS A100s without buffering.

3 Accesses can be made through the external memory interface to any register other than DIR.

4 This mode should not be used with the IMS A100-G30S variant at clock frequencies above 20.8 MHz.



Bankswap timing

Symbol	Parameter	Min.	Max.	Units	Notes
t_{EHBH}	ACR[0] set to BUSY high delay		55	ns	4
t_{CHBL}	BUSY hold after bankswap		50	ns	4
t_{CHEH}	ACR[0]=0 hold after last input	20		ns	3,4
t_{EHCH}	ACR[0]=1 setup to next input	10		ns	3,4

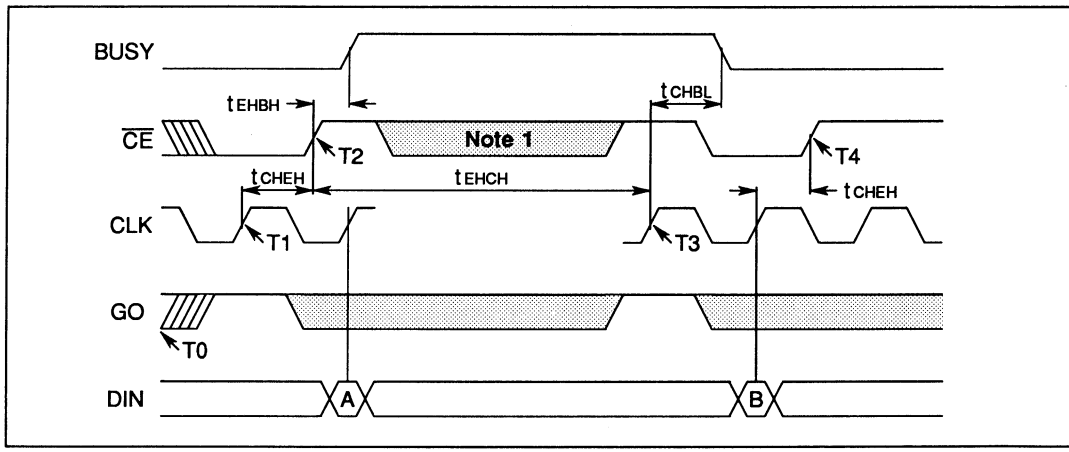
Notes

1 The activity on $\overline{CE}$ shown is for writing ACR[0]=1. During the period **Note 1** it may be possible to access other registers (subject to their own access constraints).

2 For small t_{EHCH}, **BUSY** may only occur for a short time or not occur at all.

3 If t_{CHEH} or t_{EHCH} is exceeded then bankswap may be synchronised to the previous or next input cycle.

4 This mode should not be used with the IMS A100-G30S variant at clock frequencies above 20.8 MHz.



The bankswap timing diagram shows how successive data samples (A and B) can be processed by different sets of coefficients by causing a bankswap to occur between the input of sample A and sample B.

The sequence of events is as follows:

T0 No bankswap pending.

T1 **GO** sampled and found to be high, thus initiating input of data sample A.

T2 Bankswap requested by writing ACR[0]=1. If the minimum timing requirement, t_{CHEH}, from T1 to T2 is not met it is possible (but not guaranteed) that the bankswap requested at T2 will occur immediately and thus affect the processing of data sample A.

T3 Bankswap occurs on the first rising edge of **CLK** upon which **GO** is sampled (without reference to the state of **GO**). If the minimum timing requirement, t_{EHCH}, from T2 to T3 is not met it is possible (but not guaranteed) that the bankswap requested at T2 will not occur at T3 but at the next sampling of **GO**.

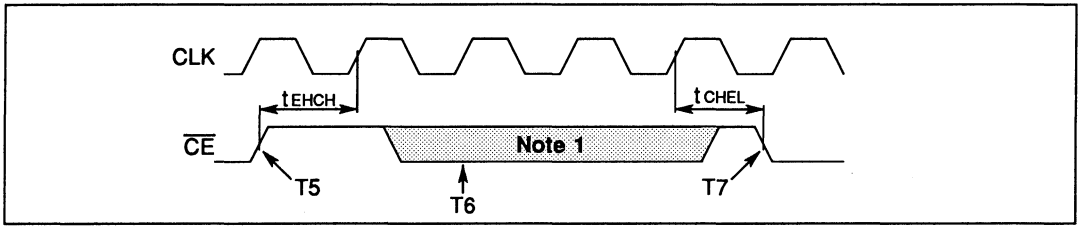
T4 This is the earliest time at which another bankswap can be requested.

Coefficient access timing

Symbol	Parameter	Min.	Max.	Units	Notes
t_{EHCH}	End coefficient access before bankswap	0		ns	
t_{CHEL}	Start coefficient access after bankswap	0		ns	

Notes

1 During this period accesses may be made to registers other than the coefficient registers (subject to their own access constraints).



If a bankswap (caused by setting either ACR[0]=1 or SCR[2]=1) occurs at the **GO** sampling point T6, then no access should be made to the coefficient registers between T5 and T7.

3.7 PACKAGE SPECIFICATIONS

3.7.1 84 pin grid array package

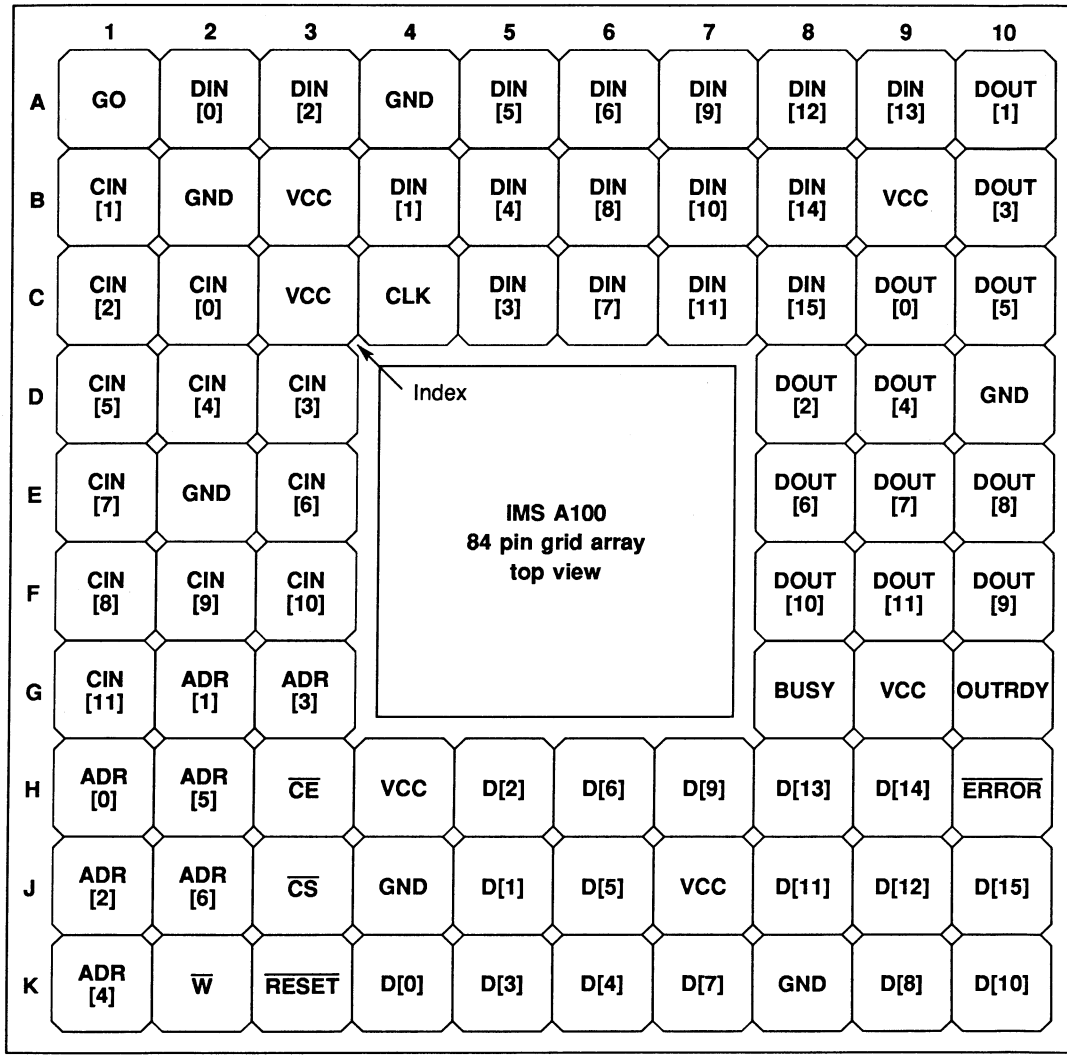


Figure 3.6 IMS A100 pin configuration

Note

All **VCC** pins **must** be connected to the 5 Volt power supply.
All **GND** pins **must** be connected to ground.

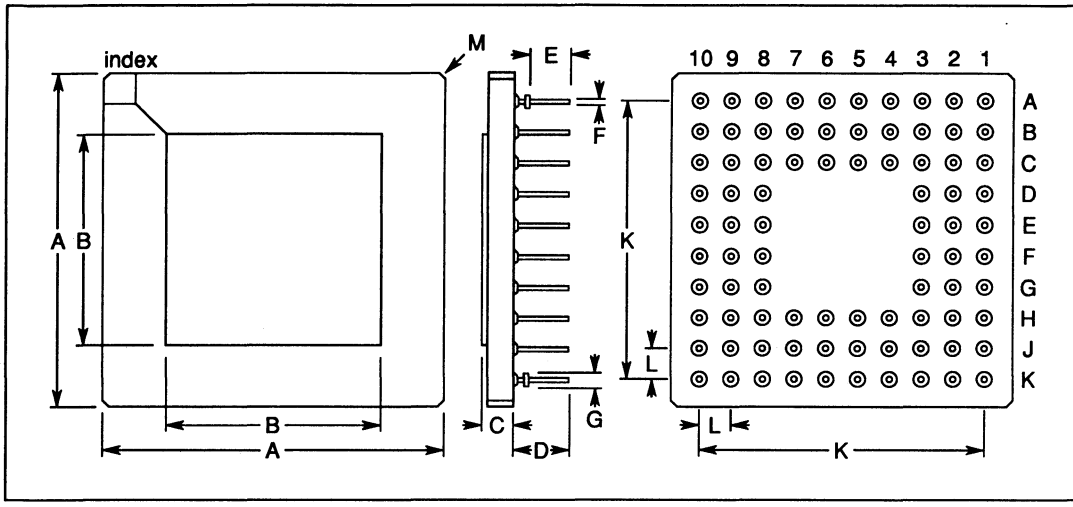


Figure 3.7 84 pin grid array package dimensions

DIM	Millimetres NOM	Millimetres TOL	Inches NOM	Inches TOL	Notes
A	26.924	±0.254	1.060	±0.010	
B	17.019	±0.127	0.670	±0.005	
C	2.456	±0.278	0.097	±0.011	
D	4.572	±0.127	0.180	±0.005	
E	3.302	±0.127	0.130	±0.005	
F	0.457	±0.025	0.018	±0.001	Pin diameter
G	1.143	±0.127	0.045	±0.005	Flange diameter
K	22.860	±0.127	0.900	±0.005	
L	2.540	±0.127	0.100	±0.005	
M	0.508		0.020		Chamfer
Package weight is approximately 7.2 grams					

Table 3.1 84 pin grid array package dimensions

Pin grid array thermal characteristics

Symbol	Parameter	Min	Nom	Max	Units	Notes
θ JA	Junction to ambient thermal resistance			35	°C /W	1,2

Notes

1 Measured at 400 linear ft/min transverse air flow.

2 This parameter is sampled and not 100% tested.

3.7.2 84 pin quad ceramic package

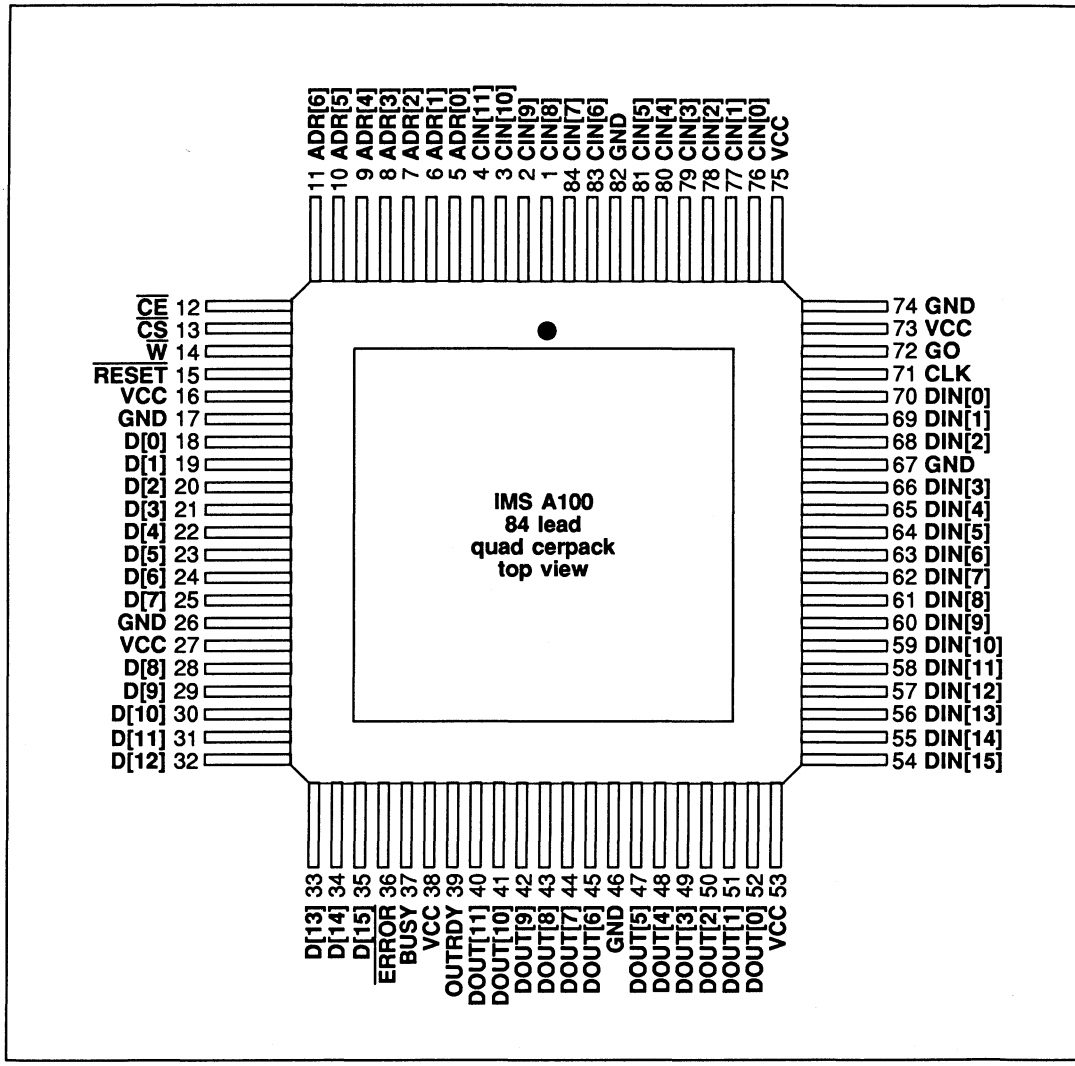


Figure 3.8 IMS A100 pin configuration

Note

All **VCC** pins **must** be connected to the 5 Volt power supply.
All **GND** pins **must** be connected to ground.

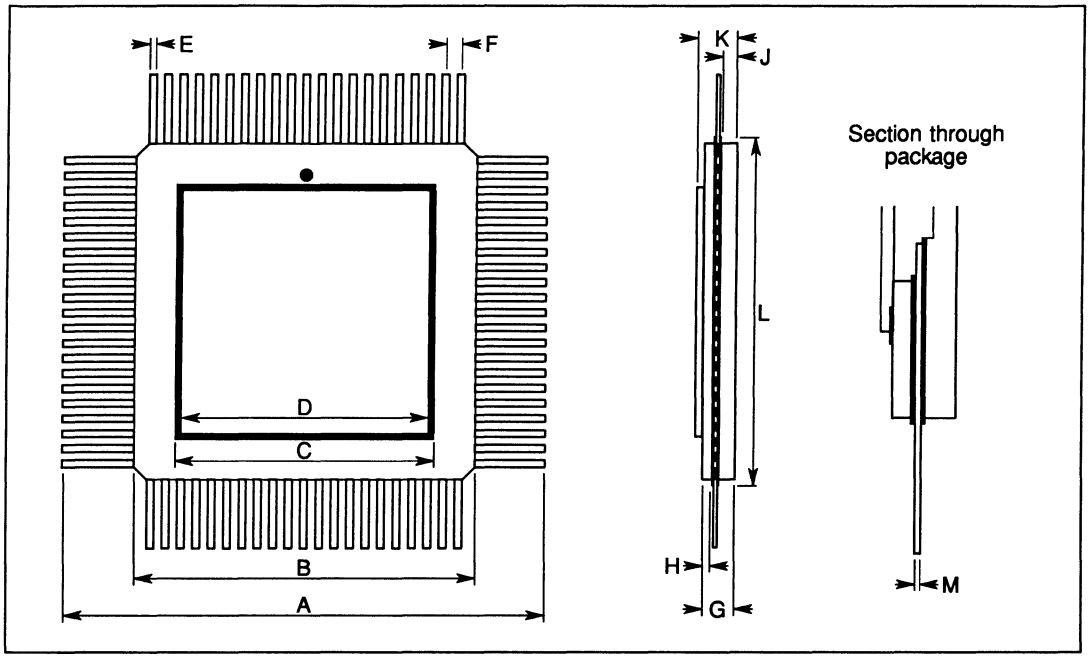


Figure 3.9 84 lead quad cerpack package dimensions

DIM	Millimetres		Inches		Notes
	NOM	TOL	NOM	TOL	
A	38.100	±0.508	1.500	±0.020	
B	26.924	±0.305	1.060	±0.012	
C	20.574	±0.203	0.810	±0.008	
D	19.558	±0.254	0.770	±0.010	
E	0.508		0.020		
F	1.270	±0.051	0.050	±0.002	
G	2.489	±0.305	0.098	±0.012	
H	0.635	±0.076	0.025	±0.003	
J	1.143	±0.102	0.045	±0.004	
K	3.099		0.122		Max.
L	27.940		1.100		Max.
M	0.178	±0.025	0.007	±0.001	

Table 3.2 84 lead quad cerpack package dimensions

Quad cerpack thermal characteristics

Symbol	Parameter	Min	Nom	Max	Units	Notes
θ JA	Junction to ambient thermal resistance			35	°C /W	1,2

Notes

1 Measured at 400 linear ft/min transverse air flow.

2 This parameter is sampled and not 100% tested.

3.8 MILITARY STANDARD PROGRAM ‡

The INMOS military program is designed to provide class B microcircuits in accordance with 1.2.1 of MIL-STD-883, 'Provisions for the use of MIL-STD-883 in conjunction with compliant non-JAN devices'. The IMS A100M is processed for general applications where component quality and reliability must conform to the guidelines and objectives of military procurement. Suitability for use in specific applications should be determined using the guidelines of MIL-STD-454.

Screening procedures are compliant with Method 5004 and the provisions of paragraph 3.3 therein. Quality conformance procedures are compliant with method 5005 using the alternate Group B provisions of paragraph 3.5.2. All electrical testing is performed to guarantee operation at −55 °C , +25 °C and +125 °C .

All INMOS military grade components are provided in hermetically sealed ceramic packages.

By specifying an INMOS military product, the user can be assured of receiving a product manufactured, tested and inspected in compliance with MIL-STD-883 and one with superior performance for those applications where quality and reliability are of the essence.

100 Percent Process Step	MIL-STD-883C Method	Test Condition	Comment
Internal visual	2010	B	
Stabilization bake	1008	C	
Temperature cycle	1010	C	
Constant acceleration	2001	D	Y-1 axis
Seal test	1014	B	
Seal test	1014	C	
Visual inspection			INMOS 89-1001
Pre burn-in electrical			+25°C data sheet
Burn-in	1015	D	
Post burn-in electrical			+25°C data sheet
PDA			5% max
Final electrical			+125°C data sheet
Final electrical			−55°C data sheet
External visual	2009		
Group A	5005	3.5.1	A1-A11
Group B	5005	3.5.2	
Group C	5005		MIL-STD-883C 1.2.1.b.17
Group D	5005		MIL-STD-883C 1.2.1.b.17

‡ See INMOS document 49-9047 'Military General Processing Specification' for full details.

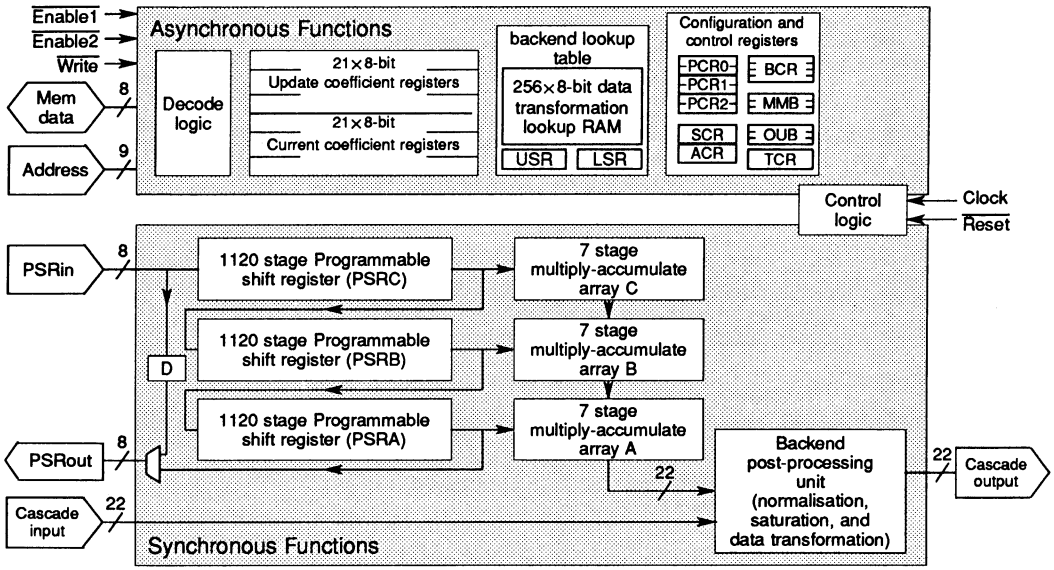
3.9 ORDERING DETAILS

The following table indicates the designation of the IMS A100 variants.

INMOS designation	Package	Clock speed	Military/commercial
IMS A100-G21M	Ceramic pin grid array	21 MHz	military
IMS A100-G21S	Ceramic pin grid array	21 MHz	commercial
IMS A100-Q21M	Flatpack	21 MHz	military
IMS A100-Q21S	Flatpack	21 MHz	commercial
IMS A100-G17M	Pin grid array	17 MHz	military
IMS A100-G30S	Pin grid array	30 MHz	commercial

IMS A110
Image and Signal Processing Sub-system

Preliminary Data



FEATURES

1-D/2-D software configurable convolver/filter
On-chip programmable line delays (0 – 1120 stages)
8-bit data and 8-bit coefficient slice
21 multiply-and-accumulate stages
1-D (21) or 2-D (3×7) convolution window
On-chip post processor for data transformation
Fully cascadable in window size and accuracy
20 MHz data throughput (420 MOPs)
Signed/unsigned data and coefficients
Microprocessor interface
High speed CMOS implementation
TTL compatible
Single +5V ±10% Supply
Power dissipation < 2.0 Watts
100 pin ceramic PGA

APPLICATIONS

1-D and 2-D digital convolution and correlation
Real time image processing and enhancement
Edge and feature detection
Data transformation and histogram equalisation
Computer vision and robotics
Template matching
Pulse compression
1-D or 2-D interpolation

4.1 INTRODUCTION

The IMS A110 is a single-chip reconfigurable and cascadable subsystem suitable for many high speed image and signal processing applications. Apart from its powerful multiply-accumulate capability (420 MOPs), the strength of the IMS A110 lies in its extensive programmable support for data conditioning and transformation.

4.2 DESCRIPTION

The IMS A110 consists of a configurable array of multiply-accumulators, three programmable length 1120 stage shift registers, a versatile post-processing unit and a microprocessor interface for configuration and control purposes. The comprehensive on-chip facilities make a single device capable of dealing with many image processing operations.

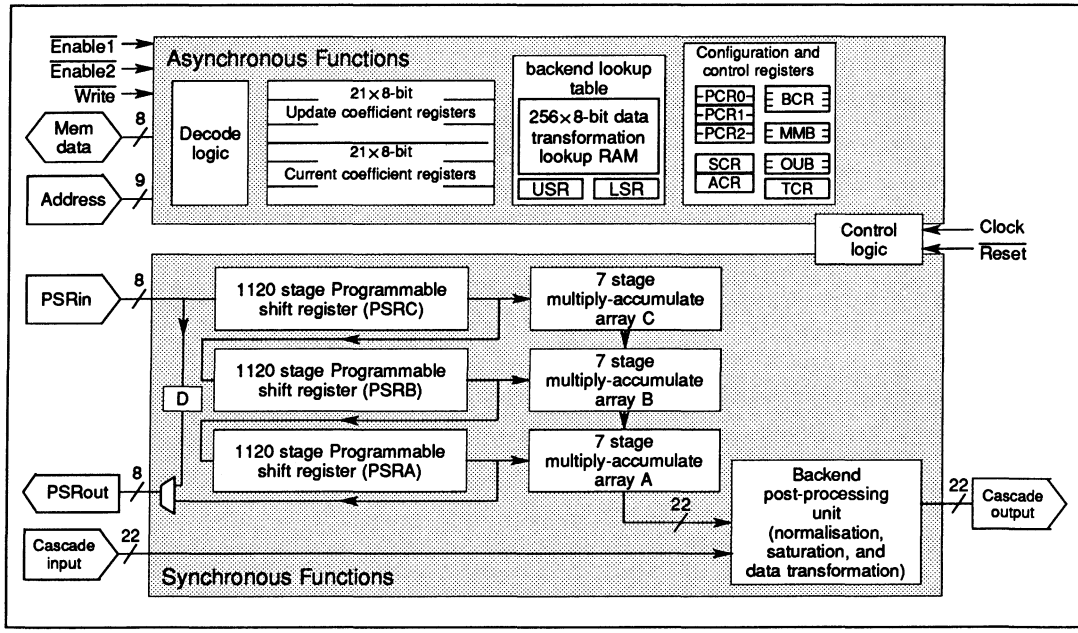


Figure 4.1 IMS A110 users model

The IMS A110 has five interfaces through which data can be transferred, figure 4.1. The microprocessor interface allows access to the coefficient registers, the configuration and status registers, and the data transformation tables. The remaining four interfaces allow high speed data input and output to the IMS A110 and the cascading of several devices. A typical IMS A110 system is shown in figure 4.3. If N devices are used in the cascade, they can be configured, entirely under software control, as a $21N$ stage 1-D transversal filter or as a $7X$ by $3Y$ 2-D window, where X and Y are any integers satisfying $N \geq XY$. For example 4 cascaded devices can be software configured as: an 84-stage 1-D filter, a 7 by 12 2-D window, a 28 by 3 2-D window, or a 14 by 6 2-D window.

The final output of the chip is 22 bits wide in twos complement format.

Figure 4.2 shows the distribution of the delays inside the part.

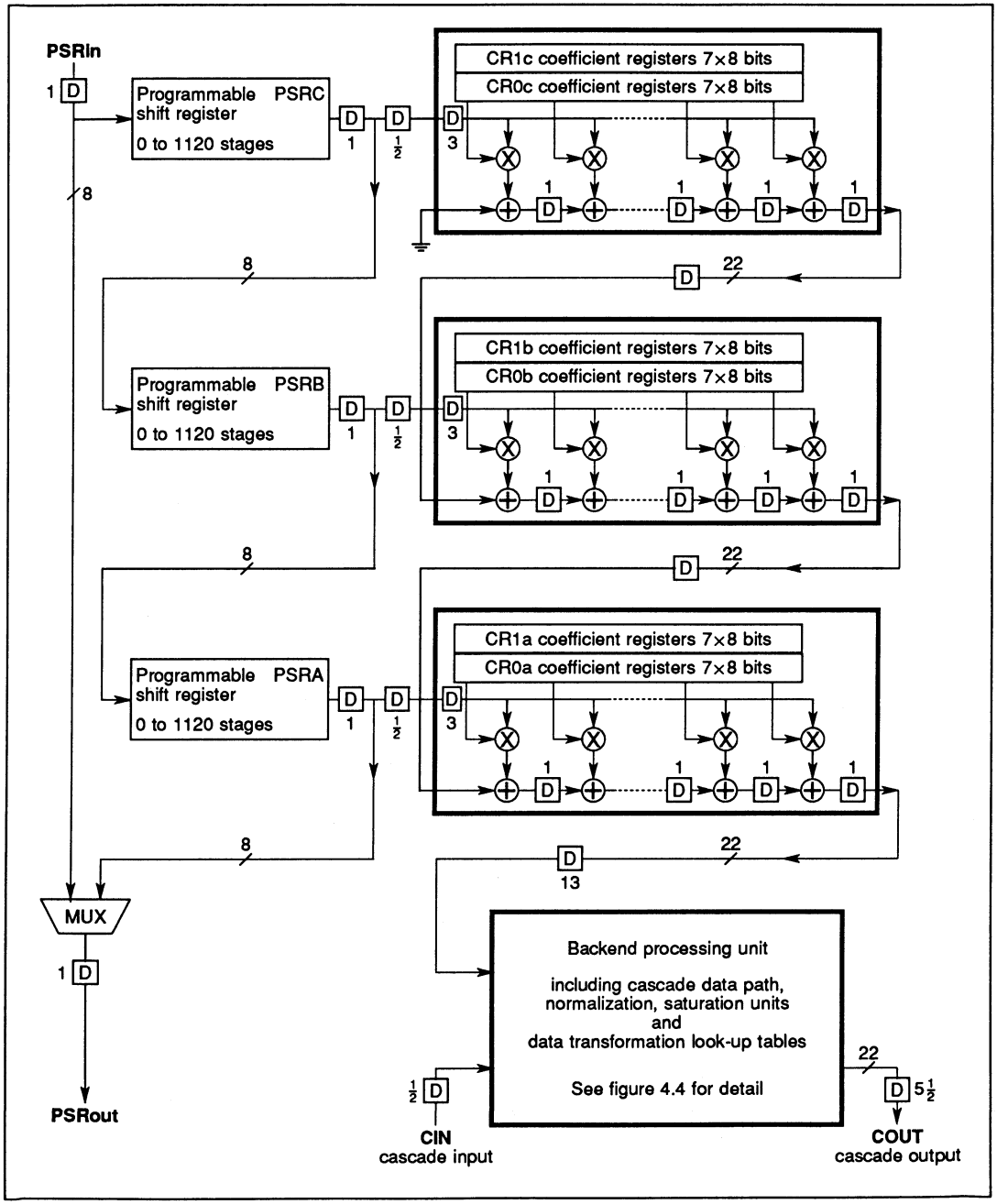


Figure 4.2 Synchronous functions of the IMS A110

The latency between PSRin and COUT is dependent upon the length of PSRc. For example, with PSRc set to 0, and all coefficients set to zero except CR0c[6] (so the data passes through all MAC stages), the COUT bus will correspond to the PSRin bus delayed by 47 clock cycles.

The latency between PSRin and PSRout is 5 cycles PLUS the lengths of PSRc, PSRb and PSRa. If the shift registers are bypassed by setting SCR[1] to 1 then PSRout will be PSRin delayed by 2 clock cycles.

The Latency between the cascade input (CIN) and cascade output (COUT) is 6 cycles. This is shown lumped at the cascade input and cascade output pads in figure 4.2. Figure 4.4 gives details of the data pipelining through the backend datapath.

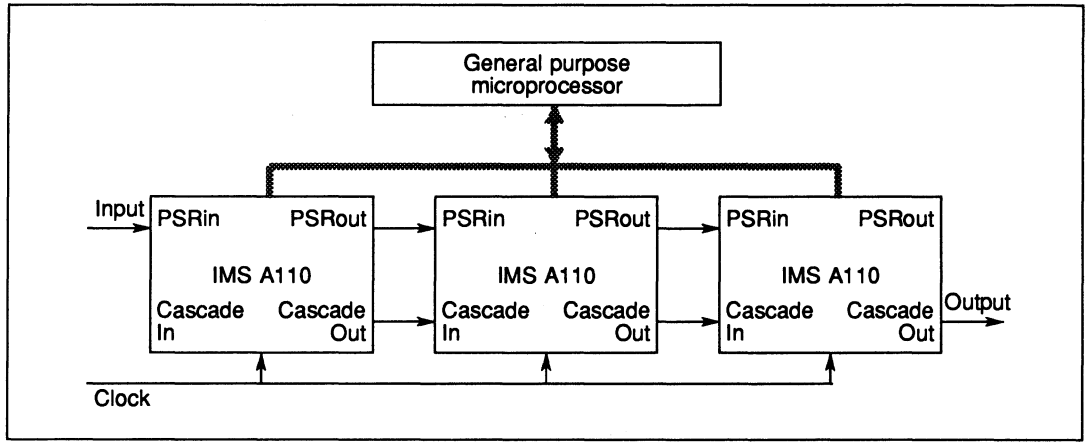


Figure 4.3 A typical IMS A110 based system

4.3 PROGRAMMABLE SHIFT REGISTERS

The three shift registers are 8 bits wide and are each programmable from 0 up to 1120 clock cycles in length. The lengths are programmed into control registers via the microprocessor interface.

Data is clocked into the device via the PSRin bus (Programmable Shift Register in) at a maximum rate of 20 MHz. On-chip, the input data is then fed through a pipeline of the three shift registers. The output of the first shift register passes to the first 7-stage mac array and also to the input of the second shift register. Having passed through all three shift registers the data is output on the PSRout bus and can be used for cascading. Alternatively, as shown in figure 4.2 the shift registers can be bypassed and the input data transferred to the PSRout bus after two delay stages. This mode can be controlled via the on-chip control registers and significantly simplifies software configuration of a cascade arrangement.

4.4 MAC ARRAY

As shown in figure 4.2, the processing core of the device consists of a configurable array of multiply-accumulators (macs). The mac array consists of three 7-stage transversal filters which can be configured either as a 21-stage linear pipeline or as a 3×7 two-dimensional window. The input data is 8 bits wide and is fed to the mac array via three programmable shift registers.

The output of each shift register is supplied as input to one of the three 7-stage transversal filters. For each of the three transversal filters the associated input data is fed simultaneously to all 7 mac stages. At each stage the input sample is multiplied by a coefficient stored in memory, and added to the output of the previous stage delayed by one clock cycle. The output of each 7-stage mac is fed, via a delay stage, to the first stage in the next transversal filter.

The coefficient word width in the mac array is 8 bits wide. Two banks of coefficients are provided. At any instant one set of coefficients is in use within the mac array. The set in use is defined by the state of the 'Current Bank' bit, ACR[0]. The other set can be altered via the microprocessor interface. Once a new set of coefficients has been loaded, the activities of the two coefficient banks can be interchanged without interrupting the flow of data. Alternatively, by setting the 'continous bank swap' bit SCR[0], the two coefficient banks are swapped automatically after each data input. In this case the 'Current Bank' bit only determines which bank is used first. Both data input and coefficients can be programmed independently to support twos complement or positive unsigned formats allowing multiple devices to be used as a 'slice' in higher accuracy systems.

Within the mac array no truncation or rounding is performed on the partial products. The mac array output is fed to the backend post-processing unit which is responsible for data transformation / normalisation and cascading function.

4.5 BACKEND POST-PROCESSOR – hardware description

The Backend Post-Processor consists of four major blocks : The input block (shifter, cascade adder and rectifier unit), a statistics monitor, the data conditioning unit which itself consists of the data transformation unit and the data normaliser, and the output block (output adder and multiplexers).

A detailed diagram of the Backend Post-Processor is given in figure 4.4

All operations performed in the backend are on twos complement signed numbers unless otherwise stated.

4.5.1 Shifter, Cascade Adder and Rectifier

Data from the mac array enters the datapath via a programmable shifter. The shifter is capable of arithmetic right shifts (divides) of up to 8 bits with rounding, and left shifts of up to 8 bits. The size of this shift is controlled by the status bits BCR0[5-1]. The output of the shifter passes into the cascade adder where it is added, along with any rounding generated by the shifter, to either the cascade input bus (BCR0[0] = 0), or a zero value (BCR[0] = 1).

If the result of this 22-bit signed addition is greater than $2^{21} - 1$, (2097151_{10}) then the adder will generate a positive overflow. Likewise, if it is less than -2^{21}, (-2097152_{10}) a negative overflow will be generated. In other words, a positive overflow is generated if the result of adding two positive numbers (both MSBs = 0) is negative (resulting MSB = 1). Conversely, a negative overflow is generated if the result of adding two negative numbers (both MSBs = 1) is positive (MSB = 0). Adding two numbers of different signs cannot cause the adder to overflow.

The output of the cascade adder can optionally be full-wave or half wave rectified under the control of BCR0[7,6]. The output of the rectifier passes onto the X bus. Overflows on the X bus are signalled to both the statistics monitor and the data conditioner.

4.5.2 Statistics Monitor

The statistics monitor allows the user to set up watch dogs on the dynamics of the data on the X bus. It cannot affect the data on the X bus. The statistics gathered provide information on the system behaviour which can be used to ensure correct data scaling and normalisation. The information is also useful in the control of the overall system's analogue frontend.

Hardware/Functions

The statistics monitor consists of a 24 bit Min/Max register (MMR), a 24 bit Min/Max Buffer (MMB), a 22 bit Over/UnderShoot Counter (OUC), a 22 bit Over/UnderShoot Buffer (OUB) and a 22 bit twos complement comparator.

It can perform one of four functions :

- **MAX REGISTER :** Capture the maximum value of data and store it in the MMR.
- **MIN REGISTER :** Capture the minimum value of data and store it in the MMR.
- **OVERSHOOT COUNTER :** Increment the OUC each time the data value exceeds the preset value in the MMR.
- **UNDERSHOOT COUNTER :** Increment the OUC each time the data value is less than the preset value in the MMR.

The mode of operation is determined by the Max/Min switch BCR1[0], and the Static Threshold switch BCR1[1].

Operation

Each sample on the X bus is compared against the threshold stored in the MMR.

If the unit is configured as an **overshoot counter** and the data on the X bus exceeds the threshold in the MMR, then the counter (OUC) is incremented. If the data is less than or equal to the threshold, then no action will occur. The OUC is unsigned and will not wrap around. Thus it behaves as a saturating counter with a maximum value of $2^{22} - 1$, ($3FFFFF_{16}$, 4194303_{10}). If there is a positive overflow on the X bus, then the counter will increment since the correct X bus value must exceed the threshold. Similarly a negative overflow on the X bus will not increment the counter since the correct X bus value cannot exceed the preset threshold.

If the unit is configured as an **undershoot counter** then the counter will be incremented whenever the sample is less than the preset threshold. In this case a negative overflow will cause the counter to increment.

If the unit is configured as a **max register** and the X bus exceeds the current threshold in the MMR, then the value on the Xbus is loaded into the MMR and becomes the new threshold and the counter is incremented. If the threshold is not exceeded then no action occurs. Thus the value in the MMR is the maximum value that has appeared on the X bus, and the value in the OUC has been incremented by the number of times that the threshold has been updated.

If the unit is configured as a **min register** then the threshold is updated and the counter incremented whenever the X bus is less than the current threshold.

When operating as a min/max register, overflows on the X bus can never cause the threshold to be updated as this would load an erroneous value into the MMR.

Overflows

Bit 22 of the MMR records the history of positive overflows on the X bus. Similarly bit 23 records the history of negative overflows. These bits are set to zero by writing to the MMR copy location and are active independently of whether the **Static Threshold** bit is set. When the MMR is read, then bits 22 and 23 are interpreted as follows :

bit 23	bit 22	condition
0	0	No overflow has occured
0	1	One or more positive overflows have occured
1	0	One or more negative overflows have occured
1	1	Both postive and negative overflows have occured

Access to registers

The MMR and OUC are accessed, through the memory interface, only via their associated buffers (MMB and OUB respectively) and are not accessible directly. In order to load the MMR with a value, the host must first write the value to the MMB and then transfer the data from the MMB to the MMR by performing a WRITE to the **copy MMR** location, $0B4_{16}$. To read the MMR the host must first perform a READ cycle from location $0B4_{16}$ (which transfers the contents of the MMR into the MMB) and then read the MMB. The OUB is accessed in the same way except that the dummy writes and reads are done to and from location $0BC_{16}$.

Copies from MMR to MMB and OUC to OUB (reads) can be performed at any time giving a snapshot of the contents of the MMR and OUC respectively. Copies from MMB to MMR and OUB to OUC (writes) can also be performed at any time allowing the threshold and counter to be updated dynamically.

4.5.3 Data transformation unit

The data transformation unit consists of a prescalar, an under/over select detector, a lookup table and a byte selector. It can be used in isolation to perform abitrary data mappings, or in conjunction with the data normaliser to implement sophisticated dynamic range compression functions.

Prescalar

This allows an 8-bit field anywhere within the 22-bit X bus to be selected as the address to the LUT. This is performed by right shifting the X bus so that the required 8 bits are at the least significant end. The amount of right shift is programmed in BCR2[4-0] and can have a value from 0 to 16.

Over/under select detector

This unit monitors whether the amount of right shift performed by the prescalar is sufficient to include all significant bits in, and maintain the sign of, the selected 8 bit field (i.e. an over or under select is generated if the most significant bit of the selected 8 bit field differs from any subsequent bit right up to and including the most significant bit of the right shifted X bus). This will be an **overselect** if the X bus is positive (Bit 21 = 0), and an **underselect** if the X bus is negative (Bit 21 = 1).

Prescalar under/over selects and X bus positive/negative overflows are passed to the LUT along with the selected 8 bit address field.

Lookup table (LUT) and byte select

The LUT consists of 64 words, 32 bits wide plus two special 32 bit locations called the **upper** and **lower saturation** registers (USR and LSR respectively). Thus the LUT is actually 66 words by 32 bits. The 32 bit output of the LUT is called the Y bus.

The most significant 6 bits of the 8 bit address field are used to address one of 64 words in the LUT. The least significant pair of bits in the 8 bit field are used to control a byte select on the output. Thus in addition to operating as a 64+2 word look up table of 32 bit words, it can be used as an 8 bit, 256+2 byte LUT providing 8bit – 8bit transformations.

Positive overflows on the X bus, and over selects in the prescalar cause the LUT to access the USR overriding the address given by the prescalar. Likewise negative overflows and under selects cause the LUT to access the LSR. Any sort of overflow on the X bus or prescalar will cause the byte select control to be overridden and the **most significant byte** (byte 3) of the appropriate Saturation Register will appear on the byte wide output of the data transformation unit.

If there are simultaneous overflows on the X bus and in the prescalar then the overflow from the X bus takes priority.

The USR and LSR can thus be used to model the saturating behaviour of analogue circuits instead of the usual 'wrap around' encountered in digital systems. Alternatively the USR and LSR could signal error conditions within the backend directly on the output pins via one of the output multiplexers.

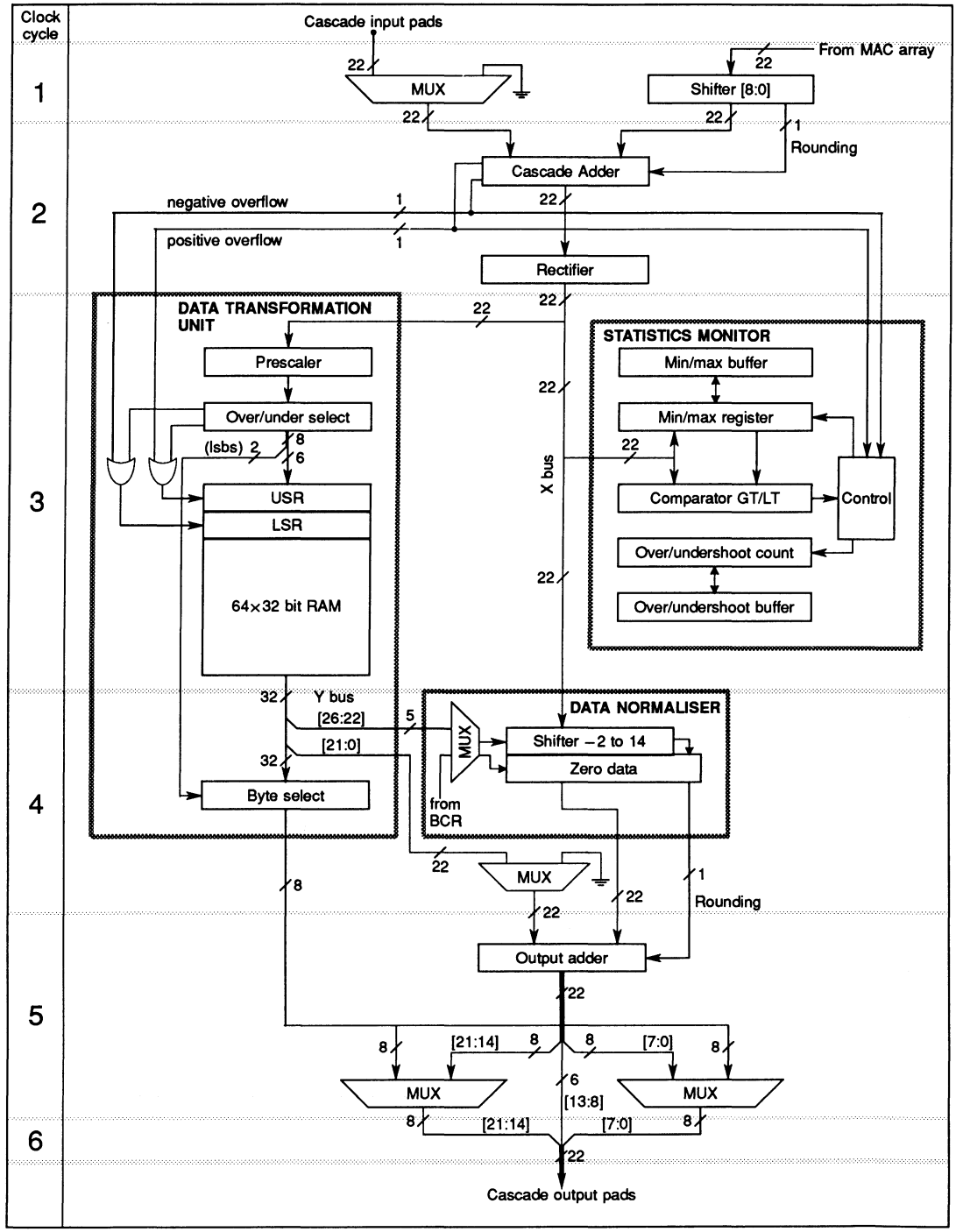


Figure 4.4 Detailed block diagram of the Backend Post-processing Unit

The LUT is loaded via the memory interface. The addressing for the LUT corresponds to the 8 bit field, assuming that the byte selector is being used. In order to access the lookup table, USR and LSR from the microprocessor interface, the **LUT Access** control bit ACR[1] must be set to zero. This will force the Y bus to zero and the normaliser to be controlled by BCR3[7-3] regardless of the setting of the dynamic normalisation bit, BCR3[2]. The LUT, USR and LSR can then be loaded with any arbitrary value via the microprocessor interface. Setting the LUT access control bit to one will then allow the LUT to be used in the data transformation unit.

4.5.4 Data normaliser

This unit consists of a shifter capable of right shifts of up to 14 bits and left shifts up to 2 bits, followed by a *zero data unit* and an adder. The shifter is controllable from one of two 5 bit sources : control bits BCR3[7-3] or bits 26 to 22 of the Y bus. The control bit **Enable Dynamic Normalisation** (BCR3[2]) determines which source is in control of the normaliser. If this bit is set to zero the normaliser is controlled by BCR3[7-3]. The five bit field is a twos complement number between 14 and −2. This indicates the amount of right shift (negative meaning left shift). Any value outside this range causes the output of the shifter to be forced to zero. The output of the shifter, with any rounding generated by the shifter, goes into the output adder.

4.5.5 Output adder

This is a 22 bit adder with one of its inputs coming from the data normaliser. The other input is either bits 21 to 0 of the Y bus from the data transformation unit, or set to zero under the control of BCR3[1]. Note that any overflow occuring due to left shifting in the normaliser or the subsequent addition in the output adder is not detected by the IMS A110.

4.5.6 Output multiplexers

These two multiplexers allow the currently selected byte from the LUT to be optionally selected to drive either the most significant byte and/or the least significant byte of the Cascade Output pins. This is controlled by the state of BCR2[5] and BCR2[6]. Enabling either of these multiplexers overrides the state of the Cascade Output pins only on the relavent 8 pins. The remaining pins will continue to represent the output of the output adder.

4.6 BACKEND POST-PROCESSOR – Modes Of Operation

The backend post-processing unit is capable of performing many functions including data scaling, transformation, dynamic range compression and histogram equalisation.

4.6.1 Default mode (after Reset)

At power up or after reset the state of the backend post-processor is such that data from the MAC array and the cascade input are added and pass straight through the datapath unaffected.

The default mode for the statistics monitor is **min register** although the values in the OUB, OUC, MMR and MMB will be undefined. Likewise the contents of the LUT, USR and LSR will be undefined, the **LUT Access** control bit will be zero forcing the Y bus to zero and allowing the microprocessor interface to access the LUT, USR and LSR.

Note that the cascade output pins and the PSR output pins are tristated.

4.6.2 Cascade adder / MAC data scalar

These units allow the cascading of IMS A110s where the output of the MAC array may be scaled before it is added to the cascade input data. The shifter can also be used for combining devices to obtain extended precision in input data, coefficient word length or both.

The ability to zero the cascade input provides a simple means of controlling the number of 'active' devices cascaded as well as a means of debugging large systems.

4.6.3 Rectification

Rectification, the removal of negative results, is needed in several image processing functions.

For example, edge detection using a Sobel operator usually requires full wave rectification due to the different signs obtained at differing edge transitions. Edge detection using a Laplacian operator produces a *change* of sign at an edge. In this case, removing negative numbers using half wave rectification can produce better results as full wave rectification can lead to some blurring of the edge transition.

4.6.4 Static scaling

This can be performed using one of two units: the MAC array output shifter (as above), and the data normaliser. In the second case the data undergoes a simple scaling operation (with rounding) within the normaliser. The normaliser can be used to scale (multiply) the data by the factors 0, 1/16384, 1/8192, 1/4096 ..., 1/2, 1, 2, 4. By controlling the normaliser from the control bits BCR3[7-3], this provides a means for simple scaling of the data before it is output. Setting BCR3[1] and BCR2[6,7] to zero ensures that the data transformation unit takes no part in the operation and the output of the normaliser is passed unchanged to the output pins.

4.6.5 Dynamic scaling

In this mode the scaling is controlled by the data itself. i.e. the scalar is controlled from the LUT (Ybus bits 26-22) by setting BCR3[2] to one, the Ybus input to the output adder being set to zero either by setting BCR3[1] to zero or programming the LUT accordingly. This mode can provide a discontinuous non-linear transformation.

4.6.6 Simple transformation

This mode allows the user to apply arbitrary transformations to the data before it is output. Here the LUT is treated as 256 by 8. The 8 bit field selected by the LUT prescalar is used to address a byte in the LUT which is passed directly to the output pins via one of the output multiplexers. Ybus control of the data normaliser is disabled, BCR3[7-3] are set out of range so as to zero the normaliser output and the Ybus input to the output adder is set to zero by BCR3[1]. One (or both) of the output multiplexers are enabled and so the addressed byte from the LUT passes straight to the cascade output pads. Only the most significant byte of the USR and LSR are applicable in this mode as overflows override the byte select control and force it to select the most significant byte.

4.6.7 Dynamic normalisation

In this mode the normaliser and transformation units in the output conditioner are used together to perform sophisticated non-linear dynamic range compression and transformations. As in the simple transformation case the prescalar selects an 8 bit field anywhere within the X bus. The most significant 6 bits, and overflows, are fed as an address to the LUT. In this case the lookup table is treated as 64+2 by 32. Bits 26 to 22 of the Y bus are used to control the normaliser block so that the input to the normaliser is dynamically scaled. The output of the normaliser is then added in the output adder to the least significant 22 bits of the Y bus (Note that only 28 bits of the 32 bit Y bus are actually used).

Thus the data is scaled, rounded, and then an offset is added to the scaled result. Each operation can be viewed as

$$output = input \times scale + offset$$

Where *scale* and *offset* are both programmable functions of *input*. One way to view this operation is to consider that the original data range is divided into 64 equal sized levels and in each level a different scale and offset is applied. The scale and offset stored in the USR and LSR would be chosen to give the desired behaviour under overflow conditions.

Note that in the case of cascade adder overflows, the data on the X bus is invalid, so the scale here would usually be set out of range so as to zero the normaliser output. The offsets in the USR and LSR would then provide the cascade output directly.

Note also that if the 5 bit scale field in the LUT is programmed so that the normaliser always zeros the data, then the output will correspond to the 22 bit offset field in the LUT. This can be viewed as a coarse transformation with wide dynamic range which is useful for applications such as image contour emphasis and equalisation.

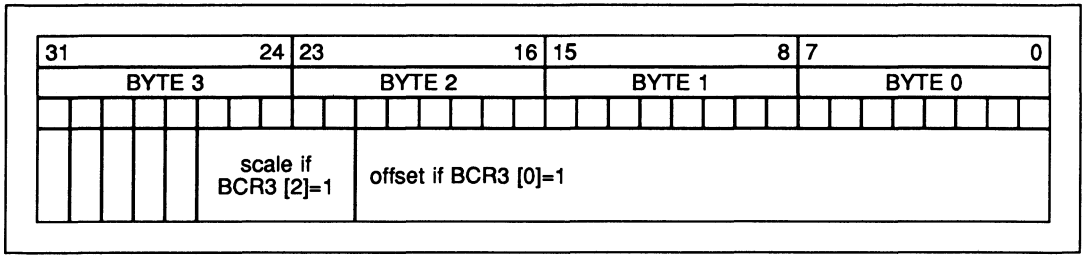


Figure 4.5 Bit format of data stored in LUT, USR and LSR

4.7 GLOSSARY

This section defines the meaning of terms used elsewhere in this data sheet.

Arithmetic Shift

For a right shift, the most significant bit is always copied into the most significant end of the result. For example shifting right by 2:

$$01000101 \rightarrow 00010001$$

$$11000101 \rightarrow 11110001$$

For a left shift, the least significant bit will become zero.

Note that left shifting can cause overflows and these are not detected in the MAC output scalar or the data normaliser.

Rounding

All rounding done within the IMS A110 is equivalent to truncating after adding 1/2 LSB. (Rounding is always applied in the positive direction). For example for 8 bit twos complement numbers undergoing a two bit right shift:

00000011 → 00000000 + 1 = 00000001	(rounded up)
00000010 → 00000000 + 1 = 00000001	(rounded up)
11111110 → 11111111 + 1 = 00000000	(rounded up)
00000001 → 00000000	(no rounding)
11111101 → 11111111	(no rounding)

Left shifts do not generate rounding.

Transversal Filter

A transversal filter is a calculation consisting of the sum of products of successive points of input data. For input data $x_i, x_{i+1}, \ldots$, and a set of coefficients, $c_6, c_5, \ldots$, the result, Y is:

$$Y = \sum_{i=0}^{6} c_i \times x_{6-i}$$

Two's Complement

Two's complement numbers allow both positive and negative numbers. For example in 8 bit numbers the most positive number is 127, the most negative is −128:

two's complement	decimal
10000000	−128
10000001	−127
11111111	−1
00000000	0
00000001	1
01111111	127

Rectification

Rectification is a method of removing negative numbers. There are two methods: Full wave and Half wave. In either case all positive numbers and zero are unaffected. In Full wave rectification, any negative numbers are negated (i.e. multiplied by −1) so that they become positive. In Half wave rectification, all negative numbers are replaced by zero.

Dynamic Range Compression

When Dynamic is used in this context, it is to indicate a change of behaviour for each data point. For example, a dynamic shift is one where the size of the shift may change on each successive clock cycle. Dynamic range compression is range compression making use of an offset and shift, which can change depending on each data point. This allows the essential non-linear transformations required in image processing to be implemented on the IMS A110.

Bit Fields

Bits, words and addresses in this data sheet are little-endian; The lowest order byte of a multiple byte word is referred to as byte 0, and is addressed in the same way. Similarly, the least significant bit of any bit field is that with the lowest bit number. For example, 'bits 26–22' refers to a 5 bit field where bit 22 is treated as the least significant, and bit 26 as the most significant.

Latency

Within the IMS A110 the latency is the number of clock cycles from an input to its corresponding output. For instance, with the programmable shift registers bypassed by setting SCR[1] to 1, the latency from PSRin to PSRout will be 2 as shown in figure 4.6.

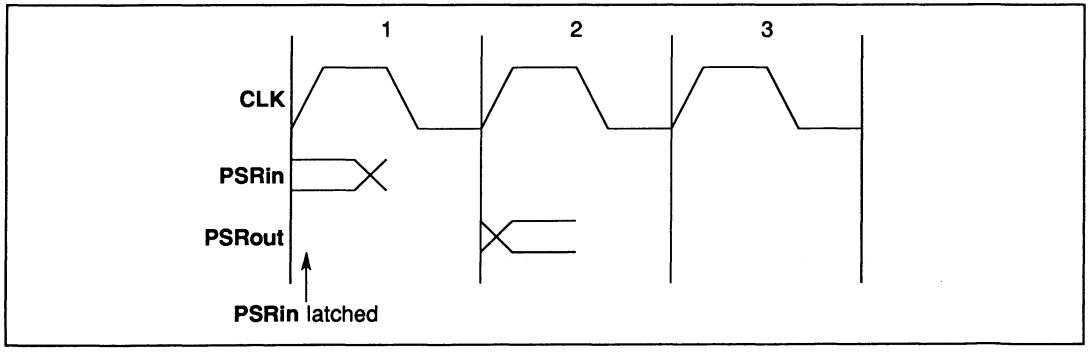


Figure 4.6

4.8 PIN DESIGNATIONS

System services

Pin	In/out	Function
VCC, GND		Power supply and return
CLK	in	Input clock
$\overline{\text{RESET}}$	in	System reset

Synchronous input/output

Pin	In/out	Function
PSRin[7-0]	in	Programmable shift register input
PSRout[7-0]	out	Programmable shift register output
Cin[21-0]	in	Cascade input port
Cout[21-0]	out	Cascade output port

Asynchronous input/output

Pin	In/out	Function
$\overline{\text{E1}}$, $\overline{\text{E2}}$	in	Memory interface enable signals
$\overline{\text{W}}$	in	Memory interface write enable
ADR[8-0]	in	Memory interface adress bus
D[7-0]	in/out	Memory interface data bus

Notes

Signal names are shown with an **overbar** if they are active low, otherwise they are active high. Pinout details are given in section 4.12.

4.8.1 System services

System services include all the necessary logic to start up and maintain the IMS A110.

Power

Power is supplied to the device via the **VCC** and **GND** pins. Several of each are provided to minimise inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100nF low inductance (e.g. ceramic) capacitor between **VCC** and **GND**.

CLK

The clock signal **CLK** controls the timing of input and the output on the four dedicated interfaces, and controls the progress of data through the shift registers, multiply-accumulate array and post-processing unit. The A110 is fully static so the clock can be slowed down or stopped in either state without corrupting data.

$\overline{\text{RESET}}$

If this pin is taken low for at least 2 clock cycles, the control logic within the IMS A110 will be reset and all of the control and configuration registers will be initialised to their default values. All other register, memory locations, datapath registers and shift registers will not be reset by this signal.

A reset is initiated automatically when power is first applied to the device. This reset will be completed once four cycles of **CLK** have occured after **VCC** is valid.

4.8.2 Synchronous services

PSRin[7-0]

This 8-bit wide bus supplies input data to the device. The input data enters the first of the three shift registers in the chain. The timing of this input is controlled by the **CLK** signal. The data on the **PSRin** port is sampled on the rising edge of the clock. In a cascade arrangement, this bus will be connected to the **PSRout** port of the previous device. In such an arrangement the **PSRin** port on the first device will be the input to the overall cascaded system.

PSRout[7-0]

This bus outputs the data from the last programmable shift register in the chain. The data on this bus is synchronously clocked by the rising edge of **CLK**. In a cascade arrangement this port will be connected to the **PSRin** port of the next device. At power up, or after a reset, the **PSRout** pins are tristated. They are enabled by SCR[5].

Cin[21-0]

The Cascade Input port allows IMS A110s to be cascaded. It also can be used for combining an external signal (e.g. a reference image or an offset) with the processed result. In a cascade arrangement, this bus will be connected to the Cascade Output of the previous device. The data on the **Cin** bus is sampled on the rising edge of **CLK**.

Cout[21-0]

This bus outputs the processed result from the IMS A110 and can also be used for cascading. The 22-bit result is synchronously clocked by the rising edge of **CLK**. In a typical cascaded system this bus will be connected to the Cascade Input port of the next device. On the last device in the cascade, this bus will be the output of the overall system. At power up, or after a reset, the **Cout** pins are tristated. They are enabled by SCR[4].

4.8.3 Asynchronous input/output

$\overline{E1}$, $\overline{E2}$

If both of these signals are low, then the microprocessor interface is enabled. The operation of these enable signals is very similar to those found on static RAMs. When either of these signals are high the Write Enable and the address inputs are ignored and the microprocessor interface Data signals are high impedance. When both Enable signals are low a read or write access is made to registers or the RAMs within the IMS A110. Access to the microprocessor interface can occur asynchronously to the synchronous pins (**PSRin**, **PSRout**, **Cin**, **Cout**) of the device.

$\overline{W}$

Write Enable indicates whether the access to the IMS A110 memory interface is to be a read or a write. If $\overline{W}$ is low a write access is indicated.

ADR[8-0]

The nine bit binary value applied to the address inputs of the IMS A110 indicates which register or RAM location within the device is to be accessed.

D[7-0]

During a write to the microprocessor interface an 8-bit word is applied to the Data pins which is written to the appropriate location. During a read cycle the contents of the location accessed are placed on the Data pins. When either of the Enables are high the Data pins are high impedance.

4.9 REGISTER DESCRIPTION

4.9.1 Memory map

Within the IMS A110 addresses are fully decoded. Reading from locations not defined in the memory map will produce zero data. Data written to such locations is ignored. This allows the part to be fully programmed using a ROM with an address incremeter. In this case, for future compatibility, zero should be written to all undefined locations.

Register	Address decimal	Address hex	Function
CR0a	0–6	000–006	Coefficient Registers Bank 0a
CR0b	16–22	010–016	Coefficient Registers Bank 0b
CR0c	32–38	020–026	Coefficient Registers Bank 0c
CR1a	64–70	040–046	Coefficient Registers Bank 1a
CR1b	80–86	050–056	Coefficient Registers Bank 1b
CR1c	96–102	060–066	Coefficient Registers Bank 1c
PCRA	128–129	080–081	PSRA Control Register
PCRB	130–131	082–083	PSRB Control Register
PCRC	132–133	084–085	PSRC Control Register
SCR	144	090	Static Control Register
ACR	146	092	Active Control Register
BCR	160–163	0A0–0A3	Backend Configuration Register
MMB	176–178	0B0–0B2	Maximum/Minimum Buffer
CMM	180	0B4	Copy MMR
OUB	184–186	0B8–0BA	Overshoot/Undershoot Buffer
COU	188	0BC	Copy OUC
TCR	208	0D0	Test Control Register
USR	248–251	0F8–0FB	Upper Saturation Register
LSR	252–255	0FC–0FF	Lower Saturation Register
LUT	256-511	100–1FF	Look-up Table

4.9.2 Registers

CR0a Coefficient registers bank 0a

These seven 8-bit locations contain coefficients which can be used by the third, of the three, 7-stage mac arrays. CR0a(0) (address #000) corresponds to the coefficient register of this mac array nearest to its output. Similarly CR0a(6) (address #006) corresponds to the coefficient register of this mac nearest to its input. These Coefficient registers can be written to provided that the other register bank is in use. Whether the coefficient written is signed or unsigned is determined by the 'Unsigned Coefficient' bit SCR[3]. Once a value is written to a coefficient register, its value can be read back from an internal duplicate register. These registers will be used by the mac array, when ACR[0], 'Current Bank' is set to zero. Writing to these Coefficient Registers while in use will result in an undefined operation of the mac array.

CR0b Coefficient registers bank 0b

These seven 8-bit locations contain coefficients which can be used by the second, of the three, 7-stage mac arrays in the chain. CR0b(0) (address #010) corresponds to the coefficient register of this mac array nearest to its output. Similarly CR0b(6) (address #016) corresponds to to the coefficient register of this mac nearest to its input. Their behaviour is otherwise identical to CR0a.

Address (Hex)	Name	bit 7	6	5	4	3	2	1	0
1FF ⋮ 100	LUT	Look Up Table							
0FC-0FF	LSR	Lower Saturation Register							
0F8-0FB	USR	Upper Saturation Register							
0D0	TCR								
0BC	COU	Copy Over/UnderShoot Buffer							
0B8-0BA	OUB	Over/UnderShoot Buffer							
0B4	CMM	Copy Min/Max Buffer							
0B0-0B2	MMB	Min/Max Buffer							
0A3	BCR3	Normaliser Control					Dynamic normali-sation	LUT to output adder	0
0A2	BCR2	0	LS output byte	MS output byte	Look Up Prescaler				
0A1	BCR1	0	0	0	0	0	0	Static threshold	Greater Than
0A0	BCR0	Full Wave	Half Wave	MAC Output Scaler					Zero Cascade
092	ACR	0	0	0	0	0	0	Backend LUT Access	Current Bank
090	SCR	0	0	PSR Out Enable	Cascade Enable	Unsigned Coef	Unsigned Data	Bypass PSRs	Cont Swap
085	PCRC	0	0	0	0	0	Shift Length (Upper Bits)		
084	PCRC	Shift Length (Lower Bits)							
083	PCRB	0	0	0	0	0	Shift Length (Upper Bits)		
082	PCRB	Shift Length (Lower Bits)							
081	PCRA	0	0	0	0	0	Shift Length (Upper Bits)		
080	PCRA	Shift Length (Lower Bits)							
066 ⋮ 060	CR1c	Bank 1 Coefficient Register							
056 ⋮ 050	CR1b	Bank 1 Coefficient Register							
046 ⋮ 040	CR1a	Bank 1 Coefficient Register							
026 ⋮ 020	CR0c	Bank 0 Coefficient Register							
016 ⋮ 010	CR0b	Bank 0 Coefficient Register							
006 ⋮ 000	CR0a	Bank 0 Coefficient Register							

Figure 4.7 IMS A110 memory map

CR0c Coefficient registers bank 0c

These seven 8-bit locations contain coefficients which can be used by the first, of the three, 7-stage mac arrays in the chain. CR0c(0) (address #020) corresponds to the coefficient register of this mac array nearest to its output. Similarly CR0c(6) (address #026) corresponds to to the coefficient register of this mac nearest to its input. Their behaviour is otherwise identical to CR0a.

CR1a Coefficient registers bank 1a

These seven 8-bit locations contain coefficients which can be used by the third, of the three, 7-stage mac arrays in the chain. CR1a(0) (address #040) corresponds to the coefficient register of this mac array nearest to its output. Similarly CR1a(6) (address #046) corresponds to to the coefficient register of this mac nearest to its input. These registers will be used provided that ACR[0], 'Current Bank' is set to one, or continuous bank swap mode is in operation (SCR[0] set to one).

CR1b Coefficient registers bank 1b

These seven 8-bit locations contain coefficients which can be used by the second, of the three, 7-stage mac arrays in the chain. CR1b(0) (address #050) corresponds to the coefficient register of this mac array nearest to its output. Similarly CR1b(6) (address #056) corresponds to to the coefficient register of this mac nearest to its input. Their behaviour is otherwise identical to CR1a.

CR1c Coefficient registers bank 1c

These seven 8-bit locations contain coefficients which can be used by the second, of the three, 7-stage mac arrays in the chain. CR1c(0) (address #060) corresponds to the coefficient register of this mac array nearest to its output. Similarly CR1c(6) (address #066) corresponds to to the coefficient register of this mac nearest to its input. Their behaviour is otherwise identical to CR1a.

PCRA PSRA Control register

This is a 16-bit register, with least significant byte at location #080, and is used to set up the length of the last shift register in the chain. Programmed lengths outside the range 0 to 1120 will cause undefined behaviour of the shift register.

PCRB PSRB Control register

This is a 16-bit register, with least significant byte at location #082, and is used to set up the length of the second shift register in the chain. Programmed lengths outside the range 0 to 1120 will cause undefined behaviour of the shift register.

PCRC PSRC Control register

This is a 16-bit register, with least significant byte at location #084, and is used to set up the length of the first shift register in the chain. Programmed lengths outside the range 0 to 1120 will cause undefined behaviour of the shift register.

SCR Static control register

The Static Control Register contains the control bits which set up parts of the IMS A110 which are likely to not need reconfiguration during processing. The contents of this register are not affected by the IMS A110 and can be read at any time. Modifying the Static Control register during processing will result in undefined behaviour. Normal operation will start to occur between 0 and 3 clock cycles after the completion of the write cycle.

ACR Active control register

The Active Control Register contains status and control bits which are likely to be accessed during normal operation of the IMS A110.

BCR Backend configuration register

The Backend Configuration Registers consist of four byte-wide registers BCR0, BCR1, BCR2, and BCR3 which are located at addresses #0A0, #0A1, #0A2, and #0A3 respectively. These four registers are used to control the backend post-processing unit. None of the control bits in these registers can be modified by the IMS A110. Modification of the values in these registers during processing may result in undefined behaviour. Normal operation will start to occur between 0 and 3 clock cycles after the completion of the write cycle.

MMB Maximum/minimum buffer

These three locations hold a 24-bit wide word, with the least significant byte at the lowest address, and act as a buffer between the MMR and the microprocessor interface. All the transactions between the MMR and the host processor must take place through this register. When the MMR is not in use, the value of this buffer is undefined.

CMM Copy MMR

This location is used to enable the data transfer between the MMB and MMR. A write to this location causes the contents of MMB to be copied into the MMR and bits 23 and 22 of the MMR (the cascade adder overflow flags) to be set to zero. A read from this location causes the reverse, i.e the contents of the MMR are copied into the MMB. The value written to this location is ignored, the value read back is undefined.

OUB Overshoot/undershoot buffer

These three memory locations hold a 22-bit word, with the least significant byte at the lowest address, and act as a buffer between the OUC and the microprocessor interface. All the transactions between the OUC and the host processor must take place through this register. When the OUC is not in use, the value of this buffer is undefined.

COU Copy OUC

This location in the memory is used to enable the data transfer between the OUB and OUC. A write to this location causes the contents of OUB to be copied into the OUC. A read from this location causes the reverse, i.e the contents of the OUC are copied into the OUB. The value written to this location is ignored, the value read back will be undefined.

TCR Test control register

This register is used for testing, and should be loaded with zero for normal operation.

USR Upper saturation register

This is a 32-bit value with the least significant byte at the lowest address. Its contents are used to replace the LUT output if positive overflow(s) occur in the lookup prescaler and / or in the cascade adder. Accesses from the microprocessor interface can only be made while ACR[1] is set to zero.

LSR Lower saturation register

This is a 32-bit value with the least significant byte at the lowest address. Its contents are used to replace the LUT output if negative overflow(s) occur in the lookup prescaler and / or in the cascade adder. Accesses from the microprocessor interface can only be made while ACR[1] is set to zero.

LUT Look-up table

These locations are for the 256-byte look-up table which is used for data mapping and transformation operations. From the microprocessor interface, these locations are addressed in the same way as that seen by the 8-bit output of lookup prescaler. When used in 32 bit mode, the locations are treated in the same way as other 32 registers: Word 0 has its most significant byte at #103, its least significant byte at #100, Word 12 has its most significant byte at #133, its least significant byte at #130. Accesses from the microprocessor interface can only be made while ACR[1] is set to zero.

4.10 REGISTERS – BIT ALLOCATION

This section describes the register details bit by bit. Each section commences with the name of the register with the bit number(s) followed by the default value, in the general format:

Name REGISTER[MSB–LSB] Default: MSB...LSB

The least significant bit of a register is bit 0.

† in the tables indicates the default state of the register bit(s).

4.10.1 PSR control registers (PCR)

PSRA control PCRA[10-0] Default: 0...0

These eleven least significant bits of the PCRA are used to specify the length of the last Programmable Shift Register (PSRA). The length of the shift register will be numerically equal to the binary value loaded in these bits. The value loaded in must be in the range of 0 to 1120 decimal. If a value outside this range is written to these bits the behaviour of the shift register will be undefined. After updating this register, the behaviour of the delay is undefined for 32 clock cycles. Hence changing the length from 1000 to 1001 delays, will result in correct output only after 1033 cycles.

Reserved PCRA[15-11] Default: 00000

These 5 most significant bits of the PCRA are reserved. The user should write zero to these locations to maintain compatibility with future products. The value read from these locations will be zero.

PSRB control PCRB[10-0] Default: 0...0

These eleven least significant bits of the PCRB are used to specify the length of the second Programmable Shift Register (PSRB). The length of the shift register will be numerically equal to the binary value loaded in these bits. The value loaded in must be in the range of 0 to 1120 decimal. If a value outside this range is written to these bits the behaviour of the shift register will be undefined.

Reserved PCRB[15-11] Default: 00000

These 5 most significant bits of the PCRB are reserved. The user should write zero to these locations to maintain compatibility with future products. The value read from these locations will be zero.

PSRC control PCRC[10-0] Default: 0...0

These eleven least significant bits of the PCRC are used to specify the length of the first Programmable Shift Register (PSRC). The length of the shift register will be numerically equal to the binary value loaded in these bits. The value loaded in must be in the range of 0 to 1120 decimal. If a value outside this range is written to these bits the behaviour of the shift register will be undefined.

Reserved PCRC[15-11] Default: 00000

These 5 most significant bits of the PCRC are reserved. The user should write zero to these locations to maintain compatibility with future products. The value read from these locations will be zero.

4.10.2 Static control register (SCR)

Reserved SCR[7-6] Default: 0 0

These locations are reserved. The user should write zero to these locations to maintain compatibility with future products. The value read from these locations will be zero.

PSR out Enable SCR[5] Default: 0

A zero at this location will force the PSR Output pins into the tristate mode.

Cascade Enable SCR[4] Default: 0

A zero at this location will force the Cascade Output pins into the tristate mode.

Unsigned coefficients SCR[3] Default: 0

If this bit is set to one, the format of subsequently loaded coefficients become unsigned, with coefficient value assuming a range between 0 and 255 decimal. An 8-bit coefficient with all its bits set to one will represent +255 decimal. When this bit is zero the format of subsequently loaded coefficients will be twos complement and the corresponding numerical value will have a range between -128 and +127. By changing this bit whilst coefficients are being loaded, coefficients between −128 and +255 can be used. The unsigned format on all coefficients is suitable when IMS A110s are combined to obtain wider coefficients for extended precision.

SCR[3]	Coefficient type
0	Signed coefficients †
1	Unsigned coefficients

Unsigned data SCR[2] Default: 0

If this bit is set to one, the IMS A110 input data format will become unsigned, with input data value assuming a range between 0 and 255 decimal. An 8-bit value with all its bits set to one will represent +255 decimal. When this bit is zero the input data format will be twos complement and the corresponding numerical value will have a range between −128 and +127. Unlike SCR[3], this bit cannot be used to dynamically alter the data format. The unsigned format is suitable when IMS A110s are combined to obtain wider input data for extended precision.

SCR[2]	Data type
0	Signed data †
1	Unsigned data

Bypass shift registers SCR[1] Default: 0

This bit is used to program the path between the PSRin and PSRout ports. A zero at this location will cause the output from the last programmable shift register to be sent to PSRout port. Writing a one to this bit will cause the three programmable shift registers to be bypassed, and the data entering the port PSRin to be fed directly, via a delay of 2 clock cycles, to the port PSRout. This bit allows full programmability of a cascade arrangement so that the same hardware can be operated in a variety of ways.

Continous bank swap SCR[0] Default: 0

The continuous bank Swap bit selects whether the the two banks of coefficient registers are used alternately after each data input or if this is controlled solely by the state of the 'Current Bank' bit in the Active Control Register ACR[0].

SCR[0]	Swap mode
0	Swap on asserting ACR[0] †
1	Swap after end of each input cycle

4.10.3 Active control register (ACR)

Reserved ACR[7-2] Default: 000000

These 6 most significant bits of the ACR are reserved. The user should write zero to these locations to maintain compatibility with future products. The value read from these locations will be zero.

Enable lookup table ACR[1] Default: 0

Writing a zero into this control bit allows the memory interface to access the Lookup table; the output to the data transformation unit will be zero. The normaliser will be controlled by BCR3[7-3], regardless of the state of BCR3[2]. Writing a one to ACR[1] allows the IMS A110 to use the Lookup Table. After changing this bit, 2 clock cycles must occur before the Lookup Table can be accessed.

ACR[1]	LUT mode
0	Memory interface access †
1	Data transformation unit

Current bank ACR[0] Default: 0

When the 'Continuous Bank Swap' bit is set to zero, writing a zero into this control bit instructs the IMS A110 to use the set of coefficient registers at addresses 0 to #X26. Setting a one to this bit instructs the IMS A110 to use the set of coefficient registers at addresses #40 to #X66. If the 'Continuous Bank Swap' bit is set to one, then this bit only indicates the bank selected for the first cycle of the continuous swap mode. Writing to this bit whilst in continuous bank swap mode (SCR[0] = 1) will result in undefined behaviour of the mac array.

ACR[0]	Coefficient bank
0	Use coefficient registers at 0 to #X26 †
1	Use coefficient registers at #40 to #X66

4.10.4 Backend control register 0 (BCR0)

Enable full-wave rectification BCR0[7] Default: 0

If this bit is set the output of the cascade adder is full-wave rectified (absolute value operation) before it is fed to the remainder of the backend. This bit will override the function of the BCR0[6].

Enable half-wave rectification BCR0[6] Default: 0

Writing a one in this bit will cause the negative values from the cascade adder to be replaced with zero. Note that writing a one into BCR0[7] will override the function of this control bit.

BCD0[7-6]	Rectifier mode
0 0	Straight through †
0 1	Half wave rectification
1 0	Full wave rectification
1 1	Full wave rectification

Mac array output scaler BCR0[5-1] Default: 00000

The contents of these five bits control the amount of right or left shift applied to the data at the output of the mac array. This field is interpreted as a two's complement number. A positive number represents a right shift (divide). Any shift in the range −8 (10000) to +8 (01000) is legal. Values outside this range will result in undefined behaviour of the mac output scaler.

Zero cascade input BCR0[0] Default: 0

This bit controls the Cascade Input Multiplexer. Writing a one to this bit will cause a zero, instead of the cascade input data, to be fed to the cascade adder.

BCR[0]	Cascade input mode
0	Cascade data
1	Zero †

4.10.5 Backend control register 1 (BCR1)

Reserved BCR1[7-2] Default: 000000

These locations are reserved. The user should write zero to these locations to maintain compatibility with future products. The values read from these locations will be zero.

Static threshold BCR1[1] Default: 0

If this bit is set to one, the signals from the comparator will be used to increment the Over / Undershoot Counter only. If this bit is zero, the signals from the comparator will be used to latch the output of the Cascade Adder into the Maximum / Minimum Register (MMR), **and** to increment the counter. In this case the counter will have been incremented by the number of times that the threshold has been updated.

Enable greater than BCR1[0] Default: 0

This control bit determines whether the comparator in the statistics monitor behaves as a 'greater than', or as a 'less than' comparator. The signal from this comparator is used to drive the Over / Undershoot Counter and the Max / Min Register. A one at this location selects 'greater than'.

BCR1[1-0]	Statistics monitor mode
0 0	Min. register †
0 1	Max. register
1 0	Undershoot counter
1 1	Overshoot counter

4.10.6 Backend control register 2 (BCR2)

Reserved BCR2[7] Default: 0

This location is reserved. The user should write zero to this location to maintain compatibility with future products. The value read from this location will be zero.

Pass LUT data to least significant output BCR2[6] Default: 0

This bit controls the output multiplexer. If this bit is set to one, the selected byte from the LUT is output on the least significant byte (bits 7 to 0) of the Cascade Output pins.

Pass LUT data to most significant output BCR2[5] Default: 0

This bit controls the output multiplexer. If this bit is set to one, the selected byte from the LUT is output on the most significant byte (bits 21 to 14) of the Cascade Output pins.

Lookup prescaler BCR2[4-0] Default: 00000

The contents of these five bits control the amount of (arithmetic) right shift applied to the data, by the Lookup Prescaler. Writing a numerical value between 0 and 16 (binary 10000) into these bits, will cause the data to be right-shifted by a corresponding number of places. For example, if the bit pattern 00101 is written to these five bit positions, a right shift of 5 places will occur. Writing any value outside the range (0 to 16) will result in undefined behaviour of the lookup Prescaler.

4.10.7 Backend control register 3 (BCR3)

Normalizer control BCR3[7-3] Default: 00000

These five bits control the number of places, that the normaliser shifts the data to the right or to the left. This field is interpreted as a twos complement number. A positive number is taken to be a right shift. Any shift in the range −2(11110) to +14(01110) is legal. Any other value will cause the number zero to be output from the normaliser.

Enable dynamic normalization BCR3[2] Default: 0

If this bit is set to one, the normaliser will be controlled by bits 26 to 22 from the output of the lookup table, instead of BCR3[7-3].

Feed LUT data to output adder BCR3[1] Default: 0

One of the inputs of the Output Adder can be either supplied by the Lookup Table or forced to zero. Setting this control bit to zero selects zero. Setting this control bit to one selects bits 21 to 0 of the Lookup Table.

Reserved BCR3[0] Default: 0

This location is reserved. The user should write zero to this location to maintain compatibility with future products. The value read from this location will be zero.

4.11 ELECTRICAL SPECIFICATION

4.11.1 DC electrical characteristics

Absolute maximum ratings

Symbol	Parameter	Min.	Max.	Units	Notes (1,2)
VCC	DC supply voltage	0	7.0	V	3
VI, VO	Voltage on any other pin	−1.0	VCC+0.5	V	3
TA	Temperature under bias	−40	85	°C	
TS	Storage temperature	−65	150	°C	
PDmax	Power dissipation		2.0	W	

Notes

1 All voltages are with respect to **GND**.

2 This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedence circuit. Unused inputs should be tied to an appropriate logic level such as **VCC** or **GND**.

DC operating conditions

Symbol	Parameter	Min.	Nom.	Max.	Units	Notes (1)
VCC	Supply Voltage	4.5	5.0	5.5	V	
VIH	Input Logic '1' Voltage **CLK**	4.0		VCC+0.5	V	2
	Input Logic '1' Voltage other pins	2.0		VCC+0.5	V	2
VIL	Input Logic '0' Voltage **CLK**	−0.5		0.5	V	2
	Input Logic '0' Voltage other pins	−0.5		0.8	V	2
TA	Ambient Operating Temperature	0		70	°C	3

Notes

1 All voltages are with respect to **GND**.

2 Input signal transients, up to 10ns wide, are permitted in the voltage ranges (**GND** − 0.5 V) to (**GND** − 1.0 V) and **VCC** + 0.5 V to **VCC** + 1.0 V.

3 400 linear ft/min transverse air flow.

DC characteristics

Symbol	Parameter	Min.	Max.	Units	Notes (1,2)
VOH	Output Logic '1' Voltage	2.4	VCC	V	4
VOL	Output Logic '0' Voltage	0	0.4	V	5
IIN	Input leakage current(any input current)		±10	μA	3
IOZ	Off state output leakage current		±10	μA	3
IDD	Average power supply current		350	mA	

Notes

1 All voltages are with respect to **GND**.

2 Parameters measured over full voltage and temperature operating range.

3 VCC = VCC(max), GND ≤ VIN ≤ VCC

4 **IOut** ≤ −4.4 mA

5 **IOut** ≤ 4.4 mA

Capacitance

Pin	Typ.	Units	Notes
CLK	12	pF	1,2
All other pins	5	pF	1,2

1 This parameter is supplied for engineering guidance and is not guaranteed.

2 TA= 25°C , F= 1 MHz.

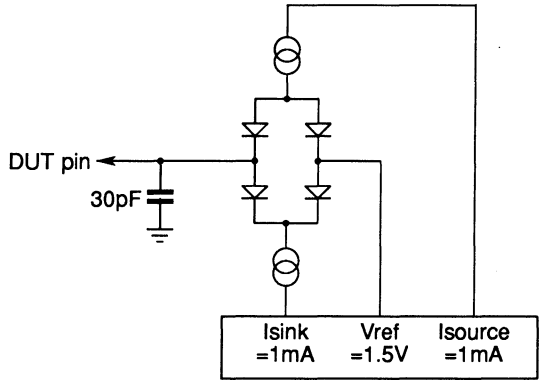
4.11.2 AC timing characteristics

AC test conditions

Output loads (except output turn-off tests)

30pF for all outputs.

Output load (output turn-off tests)



Timing reference levels

Pin	Reference levels	Notes
INPUTS	0.8V, 2.0V	1
CLK	0.5V, 4.0V	
OUTPUTS	0.4V, 2.4V	2,3
OUTPUTS	±100mV change from previous steady output voltage	4

Notes

1 Except **CLK**.

2 Output continously driven.

3 Timings are tested using VOL=0.8V and with a suitable allowance for the time taken for the output to fall from 0.8V to 0.4V.

4 Output turn-off tests.

4.11.3 Timing diagrams

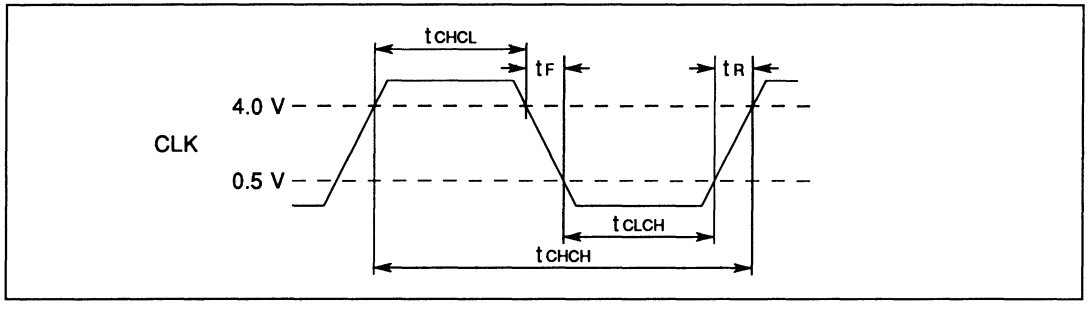
Clock Requirements

Symbol	Parameter	Min	Max	Units	Notes
t_{CHCL}	Clock Pulse High Width	20		ns	2
t_{CLCH}	Clock Pulse Low Width	20		ns	2
t_{CHCH}	Clock Period	50		ns	2
t_R	Clock rise time	0	50	ns	1
t_F	Clock fall time	0	50	ns	1

Notes

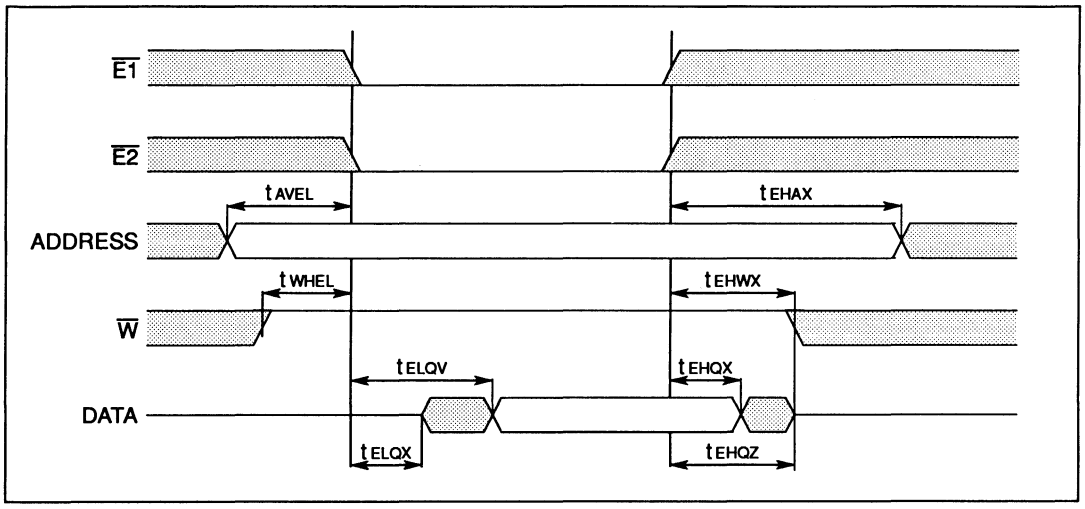
1 Clock input transitions should be monotonic between the input thresholds of 0.5 V and 4.0 V.

2 For Rev.A parts t_{CHCL}, t_{CLCH} and t_{CHCH} have **maximum** values of 50 000ns, 50 000ns and 100 000ns respectively. (A minimum clock frequency of 10kHz.)



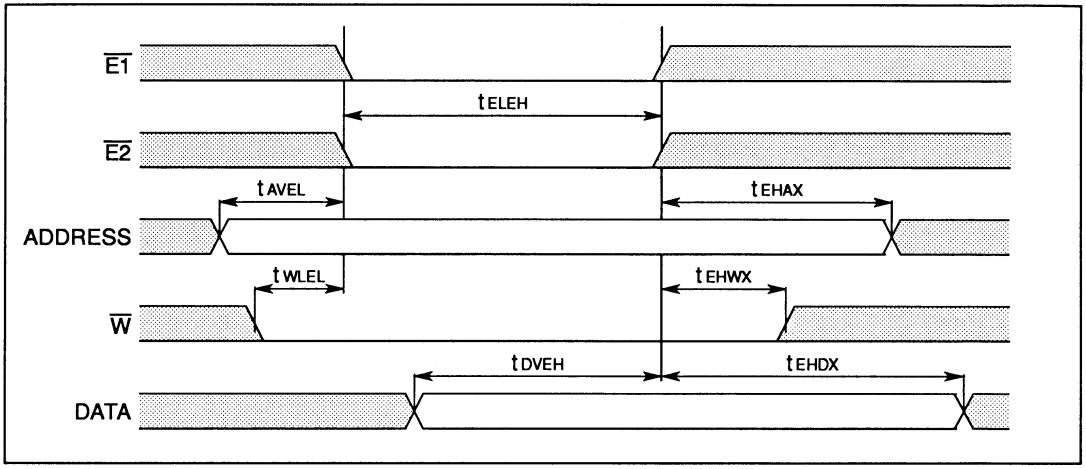
Microprocessor Interface Read Cycle

Symbol	Parameter	Min	Max	Units	Notes
t_{AVEL}	Address setup	0		ns	
t_{EHAX}	Address hold	0		ns	
t_{WHEL}	Read Command Setup	0		ns	
t_{EHWX}	Read Command Hold	0		ns	
t_{ELQX}	Output turn-on	0		ns	
t_{ELQV}	Read data access		100	ns	
t_{EHQX}	Read data hold	0		ns	
t_{EHQZ}	Output turn off		25	ns	


E1
E2
ADDRESS
W
DATA
tAVEL
tEHAX
tWHEL
tEHWX
tELQV
tEHQX
tELQX
tEHQZ

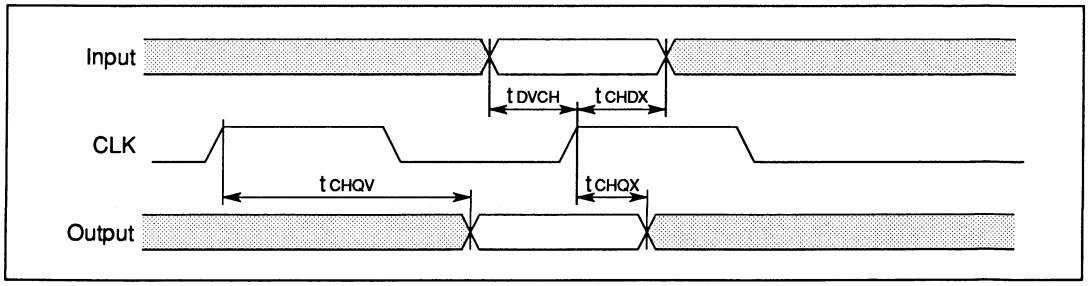
Microprocessor Interface Write Cycle

Symbol	Parameter	Min	Max	Units	Notes
t_{ELEH}	Enable Width Low	100		ns	
t_{AVEL}	Address setup	0		ns	
t_{EHAX}	Address hold	0		ns	
t_{WLEL}	Write Command Setup	0		ns	
t_{EHWX}	Write Command Hold	0		ns	
t_{DVEH}	Write data Set up	50		ns	
t_{EHDX}	Write data hold	0		ns	



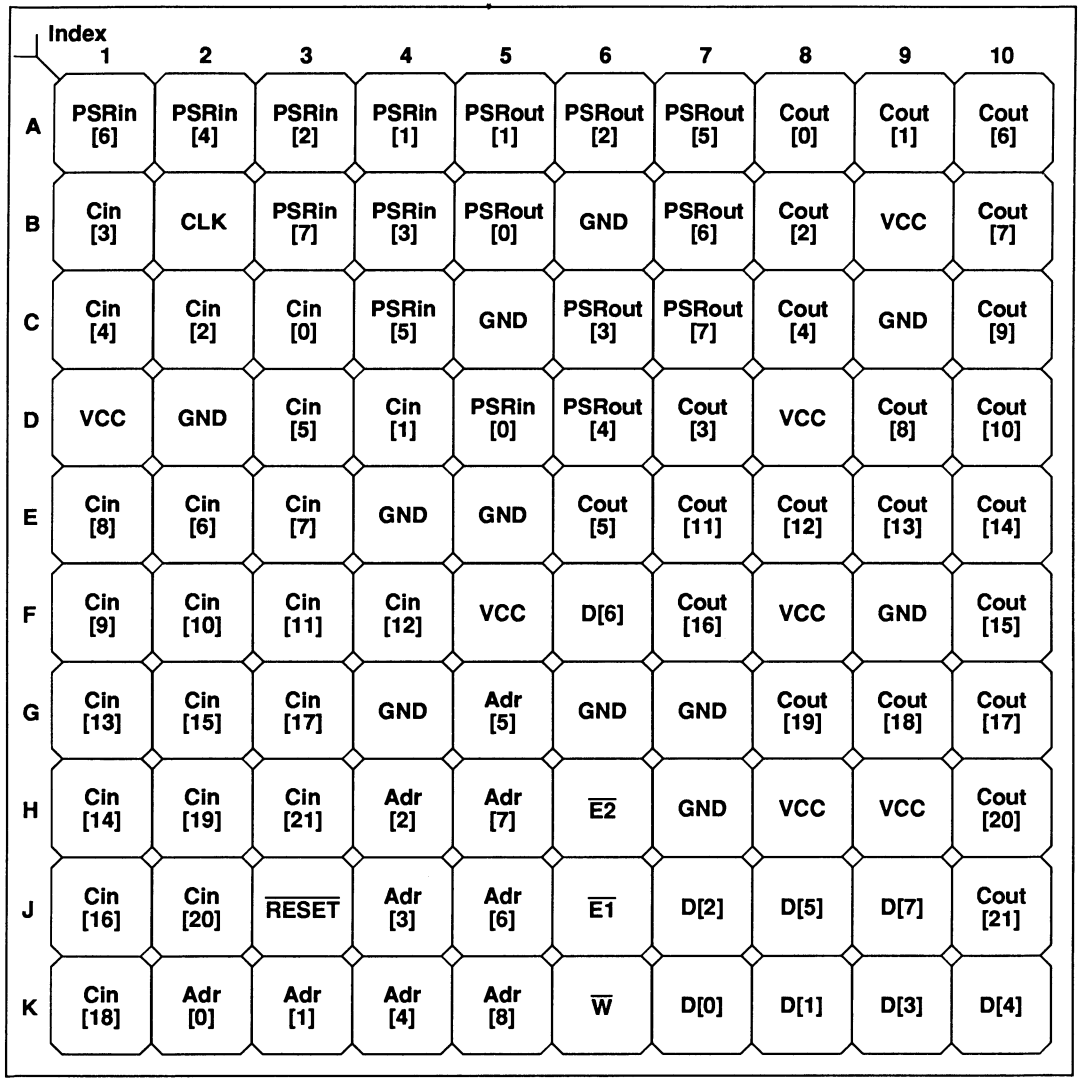
Synchronous Input and Output

Symbol	Parameter	Min	Max	Units	Notes
t_{CHQV}	CLK high to Output Valid		40	ns	
t_{CHQX}	Output hold time after CLK	2		ns	
t_{DVCH}	Input setup time to CLK high	8		ns	
t_{CHDX}	Input hold time to CLK high	0		ns	



4.12 PACKAGE SPECIFICATIONS

4.12.1 100 pin grid array package



Index	1	2	3	4	5	6	7	8	9	10
A	PSRin [6]	PSRin [4]	PSRin [2]	PSRin [1]	PSRout [1]	PSRout [2]	PSRout [5]	Cout [0]	Cout [1]	Cout [6]
B	Cin [3]	CLK	PSRin [7]	PSRin [3]	PSRout [0]	GND	PSRout [6]	Cout [2]	VCC	Cout [7]
C	Cin [4]	Cin [2]	Cin [0]	PSRin [5]	GND	PSRout [3]	PSRout [7]	Cout [4]	GND	Cout [9]
D	VCC	GND	Cin [5]	Cin [1]	PSRin [0]	PSRout [4]	Cout [3]	VCC	Cout [8]	Cout [10]
E	Cin [8]	Cin [6]	Cin [7]	GND	GND	Cout [5]	Cout [11]	Cout [12]	Cout [13]	Cout [14]
F	Cin [9]	Cin [10]	Cin [11]	Cin [12]	VCC	D[6]	Cout [16]	VCC	GND	Cout [15]
G	Cin [13]	Cin [15]	Cin [17]	GND	Adr [5]	GND	GND	Cout [19]	Cout [18]	Cout [17]
H	Cin [14]	Cin [19]	Cin [21]	Adr [2]	Adr [7]	$\overline{E2}$	GND	VCC	VCC	Cout [20]
J	Cin [16]	Cin [20]	$\overline{RESET}$	Adr [3]	Adr [6]	$\overline{E1}$	D[2]	D[5]	D[7]	Cout [21]
K	Cin [18]	Adr [0]	Adr [1]	Adr [4]	Adr [8]	$\overline{W}$	D[0]	D[1]	D[3]	D[4]

Figure 4.8 IMS A110 100 pin grid array package pinout – top view

Note

All **VCC** pins **must** be connected to the 5 Volt power supply.
All **GND** pins **must** be connected to ground.

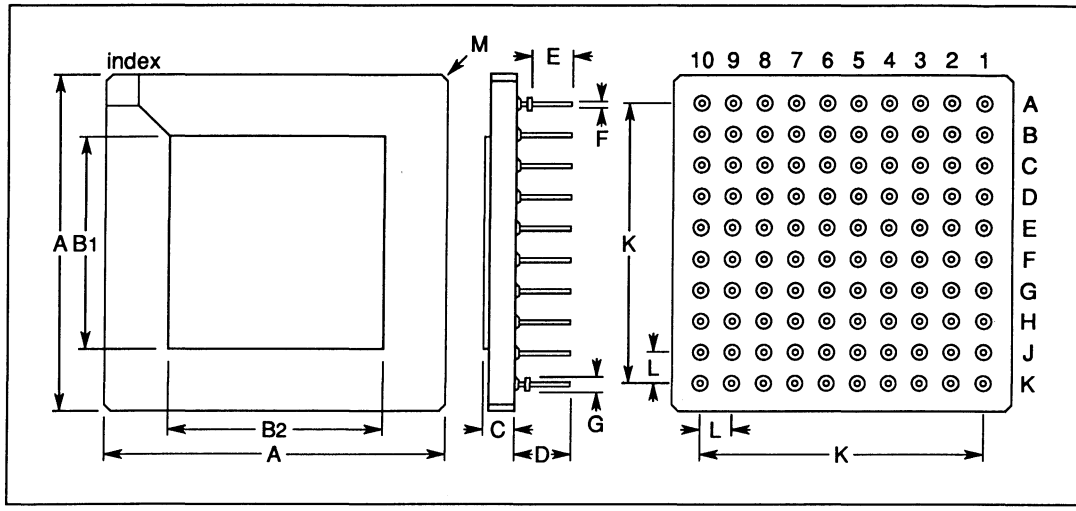


Figure 4.9 100 pin grid array package dimensions

DIM	Millimetres		Inches		Notes
	NOM	TOL	NOM	TOL	
A	26.924	±0.254	1.060	±0.010	
B1	17.019	±0.127	0.670	±0.005	
B2	18.796	±0.127	0.740	±0.005	
C	2.456	±0.278	0.097	±0.011	
D	4.572	±0.127	0.180	±0.005	
E	3.302	±0.127	0.130	±0.005	
F	0.457	±0.051	0.018	±0.002	Pin diameter
G	1.143	±0.127	0.045	±0.005	Flange diameter
K	22.860	±0.127	0.900	±0.005	
L	2.540	±0.127	0.100	±0.005	
M	0.508		0.020		Chamfer

Table 4.1 100 pin grid array package dimensions

Pin grid array thermal characteristics

Symbol	Parameter	Min	Nom	Max	Units	Notes
θ JA	Junction to ambient thermal resistance			35	°C /W	1,2

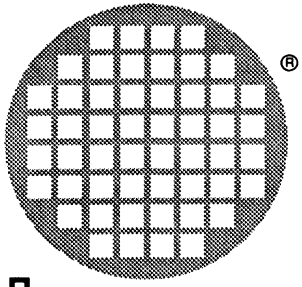
Notes

1 Measured at 400 linear ft/min transverse air flow.

2 This parameter is sampled and not 100% tested.

4.13 ORDERING DETAILS

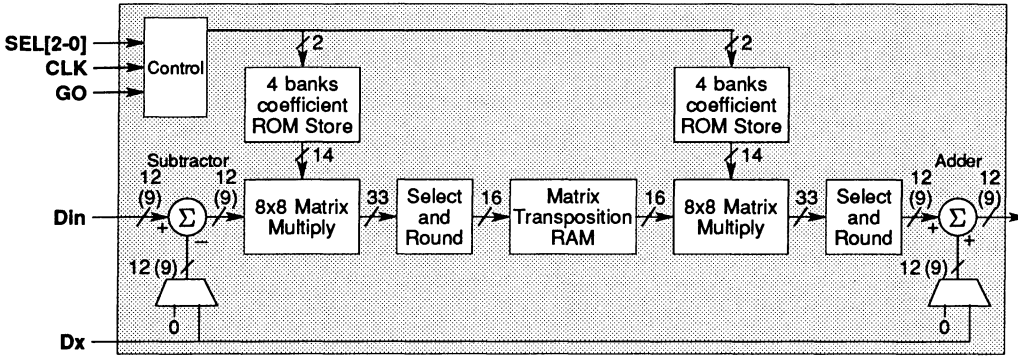
INMOS designation	Package	Clock speed	Military/commercial
IMS A110-G20S	Ceramic pin grid array	20 MHz	commercial



inmos®

IMS A121
2-D Discrete Cosine Transform Image Processor

Advance information



FEATURES

- 8 × 8 Transform size.
- 8 × 8 DCT calculation time = 3.2μs.
- DC to 20 MHz pixel rate.
- 9 bit add/subtract input.
- 12 bit input/output.
- 14 bit fixed coefficients.
- Multifunction capability (DCT, IDCT, Filter).
- Full internal precision, for each dimension.
- Fully synchronous interface.
- High speed CMOS implementation.
- TTL compatible.
- Single +5V ± 10%.
- Power dissipation < 1.5 Watt.
- 44 pin plastic package.

DESCRIPTION

The IMS A121 is a device for computing the Discrete Cosine Transform (DCT & IDCT). It will also function as a 2-D linear filter or perform matrix transposition. These 4 functions operate on blocks of data with a fixed size of 64 samples (8 × 8). The IMS A121 has other functions aimed specifically at the implementation of video codecs; on-chip subtraction and addition functions may be selected to reduce system chip count.

The main computation is performed by two identical multiplication arrays, each of which perform an 8×8 matrix multiplication in 64 cycles, with no internal rounding. The DCT/filter coefficients (14 bit) are stored in 4 banks of fixed ROM. The intermediate 8×8 matrix result is rounded to 16 bits and stored in the transposition RAM between each multiplication array. The device is fully pipelined with data sampled on the input at the clock frequency and the resultant output appearing 128 clock cycles later.

5.1 OVERALL DEVICE OPERATION

The IMS A121 is a device for computing the Discrete Cosine Transform (DCT) and the Inverse Discrete Cosine Transform (IDCT). It can also perform a simple low-pass filter operation.

The IMS A121 processes blocks of data which are 64 samples long and represent an 8×8 matrix. Data is sampled on the **Din** port every cycle and data is output every cycle on the **Dout** port.

The **GO** signal is used to indicate the start of a block. When it is sampled high the data on the **Din** port is the first sample of the block. The mode select signals **SEL[2-0]** are sampled at the same time. The remainder of the block of data is sampled on the **Din** port for the subsequent 63 cycles and during this time the **GO** signal and the **SEL** port are ignored. Each consecutive group of eight samples is treated as a column, eight such columns making a block.

The computation is in two stages, between which the block of 64 intermediate samples is stored in the transposition RAM. The transposition RAM serves a dual function of storing the intermediate results and transposing the data from column order into row order. This permits the two matrix computation elements to be identical although the first stage does the column computations and the second stage does the row computations.

Data is output on the **Dout** port in blocks of 64 samples. However, each consecutive group of eight samples now represents a row because of the internal transposition of data. The first sample of the block is output on the **Dout** port 128 cycles after the first sample of the block was sampled on the **Din** port.

An auxillary port, **Dx** is provided. The data on the **Dx** port is optionally subtracted from the data on the **Din** port (DCT mode) or added to the output (IDCT mode).

The IMS A121 views input data in column order and (because of the internal transposition) output data in row order. However, this convention is only used to define the arithmetic which the IMS A121 performs. The system in which the IMS A121 is a component may well view the data going into the IMS A121 in row order and the data coming out in column order.

5.1.1 The fixed ROM coefficients

There are four sets of fixed ROM coefficients, each corresponding to one of the four possible functions the device can perform. The two main functions which the device can perform are the DCT and the IDCT. The other two functions provide assistance for the implementation of a video codec. The filter function is provided at very little overhead because the device is essentially a 2-D filter. The transposition function which is a unity multiplication, enables a simple method of switching out the filter without any external logic.

5.1.2 Number formats

All numbers input to the IMS A121 are *signed integers*. The **Din** and **Dout** ports use 12 bit signed integers, while the **Dx** port uses 9 bit signed integers. In both cases the number format is *twos complement binary*. *Little Endian* format is assumed throughout, so that, for example, **Din[0]** is the least significant bit of the **Din** port and **Din[11]** the most significant (sign) bit. When a nine bit number is transfered over one of the 12 bit ports the most significant nine bits are used. The lowest three bits of the **Din** port are ignored and the lowest three bits of the **Dout** port will be zero.

5.1.3 Internal Bit-field Selectors and Rounding

The transforms are implemented by a matrix multiplication with no truncation or rounding. This yields a 33 bit result, with bit-field selectors provided to select the parts of the result which are of interest. 16 bits are selected from the output of the first matrix multiplication, which are stored in the matrix transposition RAM. Either 9 bits or 12 bits are selected from the output of the second matrix multiplication (depending on the selected mode).

Bits below the selected range are discarded although the result is *rounded* not truncated. This is a simple round towards $+\infty$; if the most significant bit of those bits which have been discarded is set then one is added to the bits which are retained.

5.1.4 Overflow, Saturation and Clipping

Overflow can occur in the subtraction unit, the two bit-field selectors or the addition unit. Overflow occurs whenever there are insufficient bits in the result to represent the number. When overflow occurs the result is replaced by the most positive or the most negative number which can be represented (depending on the sign of the correct result).

The device will normally be used in a feedback system. If either positive or negative overflow occurs, then inaccuracies have been introduced. However, the system will remain stable.

In some of the IDCT modes the output is clipped so that all results are positive and all negative numbers are replaced by zero. This ensures that the output is a valid (8-bit) pixel, between 0 and 255.

5.1.5 Subtraction with the DCT function

When the IMS A121 is used to perform the DCT, it is possible to enable the on-chip subtraction unit, so that before the DCT the data on the **Dx** port is subtracted from the data on the **Din** port. The data is presented to the **Dx** port at exactly the same time as to the **Din** port.

In DCT mode the data on the **Din** port is a nine bit number (the lowest 3 of 12 bits are ignored). The result of the subtraction is saturated to nine bits before being passed to the matrix multiplier.

5.1.6 Addition with the IDCT function

When the IMS A121 is used to perform the IDCT, it is possible to enable the on-chip addition unit, so that after the IDCT of the data has been done, the result may be added to the data on the **Dx** port. The timing requires careful consideration because of the latency of the device (128 cycles). The first sample of a block must be presented on the **Dx** port 124 cycles after the first sample was presented to **Din**. The data presented to the **Dx** port should be transposed and is thus in the same order as it will come out of **Dout** four cycles later.

The result of the addition is saturated to nine bits and then clipped so that all negative numbers are replaced by zero. The nine bit result is presented on **Dout[11-3]**, while **Dout[2-0]** will be zero. **Dout[11]** will be zero because all the numbers are positive.

Two modes are provided which perform the IDCT *without* addition. One of these modes disables the adder completely so that nine bit signed results appear on **Dout**. The other mode does NOT add on the value on the **Dx** port but still clips the result so that only positive values appear on **Dout**.

5.1.7 Resetting

The IMS A121 does not have a reset pin. At power-on the internal state will be undefined and as a result the first three blocks processed are not guaranteed correct. **GO** must be held low for at least 63 cycles to ensure that when it does go high it is interpreted as the start of a block.

5.2 DCT FUNCTION

The DCT function is selected when **SEL[2-0]**=000 or 100 (mode 0 or 4).

5.2.1 Internal number format

The input for the DCT is a 9 bit signed integer in the range −256 to +255. This is either an external input or the output of the on-chip subtractor depending on **SEL[2-0]**. The input is multiplied in the matrix multiplication array by 14 bit signed fixed point numbers in the range −2 to $(2-2^{-12})$. The accumulated result of 8 multiply operations is a 26 bit signed integer, the bottom 8 bits of which are rounded (see section 5.1.3) and the top 2 bits used to saturate the output (see section 5.1.4). The result of the first matrix multiply is stored as a 16 bit signed integer and the second matrix multiply performed in exactly the same manner, yielding 33 bit results. The output rounds the bottom 19 bits, saturates the top 2 bits giving a 12 bit signed integer in the range −2048 to +2047.

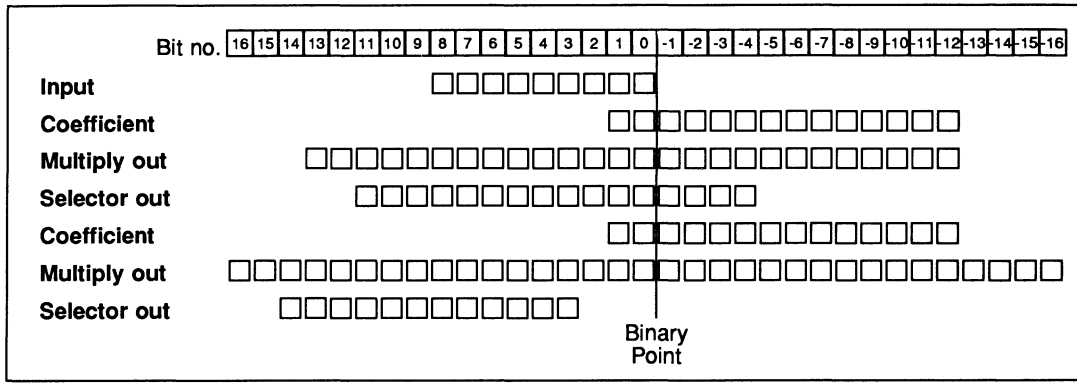
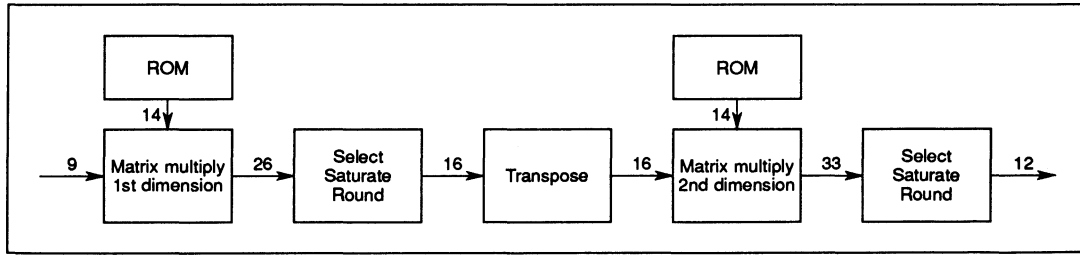


Figure 5.1 DCT internal number format

5.2.2 Internal data flow



5.2.3 The mathematical basis for the DCT

The 1 dimensional equation for the DCT is as follows:

$$\textbf{Forward transform} \quad X(k) = \sqrt{\frac{2}{N}}c(k)\sum_{m=0}^{N-1} x(m)\cos\left[\frac{(2m+1)k\pi}{2N}\right] \quad k = 0, 1, \cdots, N-1$$

$$\textbf{where} \quad c(k) = \begin{cases} \frac{1}{\sqrt{2}} & for \quad k = 0 \\ 1 & for \quad k = 1 \cdots N-1 \end{cases}$$

where $x(m)$ represent the input samples and $X(k)$ is the resulting output. The special case for the IMS A121 is with $N = 8$ and the actual filter coefficients are then calculated. The following equation is used to calculate the actual filter coefficients.

$$\textbf{DCT coefficients} \quad Coeff_{km} = \sqrt{2}c(k)\cos\left[\frac{(2m+1)k\pi}{2N}\right]$$

It should be noticed that the coefficients are $2\sqrt{2}$ times bigger than in the forward transform equation. This means that the output after the 2 dimensional DCT is 8 times too big (The 1 dimensional transform is applied twice giving $(2\sqrt{2})^2$ magnitude increase). This is in accordance with the 3 bit shift of the output data necessary to give the correct 12 bit signed output.

5.2.4 DCT coefficients

1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1.3870	1.1759	0.7857	0.2759	−0.2759	−0.7857	−1.1759	−1.3870
1.3066	0.5412	−0.5412	−1.3066	−1.3066	−0.5412	0.5412	1.3066
1.1759	−0.2759	−1.3870	−0.7857	0.7857	1.3870	0.2759	−1.1759
1.0000	−1.0000	−1.0000	1.0000	1.0000	−1.0000	−1.0000	1.0000
0.7857	−1.3870	0.2759	1.1759	−1.1759	−0.2759	1.3870	−0.7857
0.5412	−1.3066	1.3066	−0.5412	−0.5412	1.3066	−1.3066	0.5412
0.2759	−0.7857	1.1759	−1.3870	1.3870	−1.1759	0.7857	−0.2759

5.2.5 DCT coefficients (14 bit signed integers)

4096	4096	4096	4096	4096	4096	4096	4096
5681	4816	3218	1130	−1130	−3218	−4816	−5681
5352	2217	−2217	−5352	−5352	−2217	2217	5352
4816	−1130	−5681	−3218	3218	5681	1130	−4816
4096	−4096	−4096	4096	4096	−4096	−4096	4096
3218	−5681	1130	4816	−4816	−1130	5681	−3218
2217	−5352	5352	−2217	−2217	5352	−5352	2217
1130	−3218	4816	−5681	5681	−4816	3218	−1130

5.3 IDCT FUNCTION

The IDCT function is selected when **SEL[2-0]**=001, 101 or 111 (modes 1, 5 or 7).

5.3.1 Internal number format

The input for the IDCT is a 12 bit signed integer in the range −2048 to +2047. The input is multiplied in the matrix multiplication array by 14 bit signed fixed point numbers in the range −2 to $2 - 2^{-12}$. The accumulated result of 8 multiply operations is a 29 bit signed integer, the bottom 8 bits of which are rounded (see section 5.1.3) and the top 5 bits used to saturate the output (see section 5.1.4). The result of the first matrix multiply is stored as a 16 bit signed integer and the second matrix multiply performed in exactly the same manner, yielding 33 bit results. The output rounds the bottom 19 bits, saturates the top 5 bits giving a 9 bit signed integer in the range −256 to +255

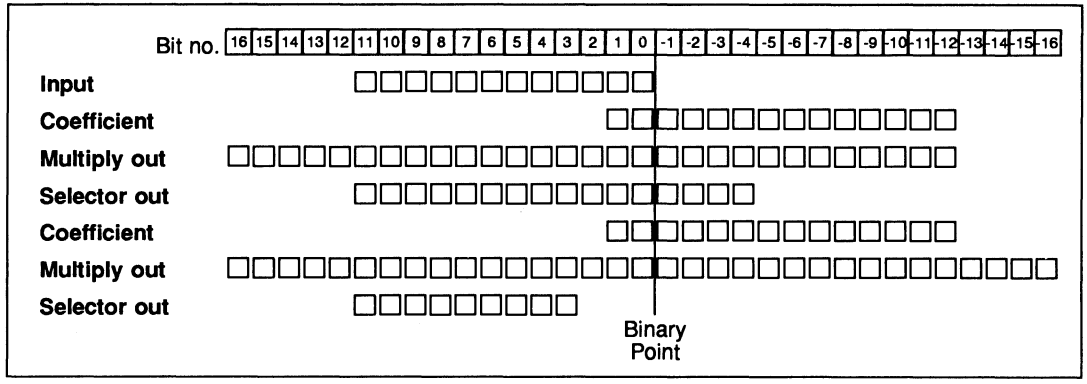
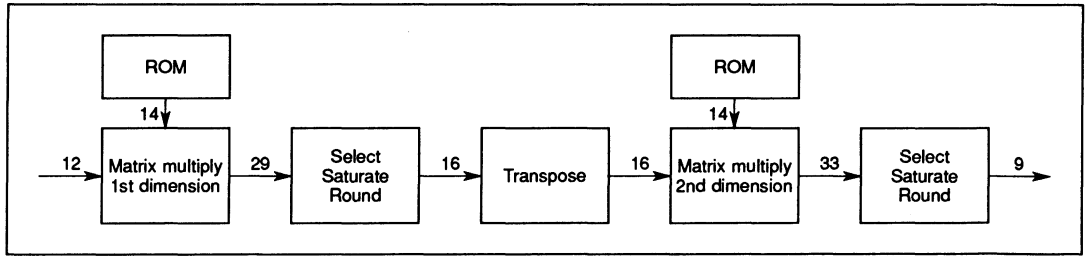


Figure 5.2 IDCT internal number format

5.3.2 Internal data flow



5.3.3 The mathematical basis for the IDCT

The 1 dimensional equation for the IDCT is as follows:

$$\textbf{Inverse transform} \quad x(m) = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} X(k)c(k)\cos\left[\frac{(2m+1)k\pi}{2N}\right] \quad m = 0, 1, \cdots, N-1$$

$$\textbf{where} \quad c(k) = \begin{cases} \frac{1}{\sqrt{2}} & for \quad k = 0 \\ 1 & for \quad k = 1 \cdots N-1 \end{cases}$$

where $x(m)$ represent the output samples and $X(k)$ is the input. The special case for the IMS A121 is for $N = 8$ and the actual filter coefficients are then calculated. The following equation is used to calculate the actual filter coefficients.

$$\textbf{IDCT coefficients} \quad Coeff_{mk} = \sqrt{2}c(k)\cos\left[\frac{(2m+1)k\pi}{2N}\right]$$

It should be noticed that the coefficients are $2\sqrt{2}$ times bigger than in the inverse transform equation. This means that the output after the 2 dimensional IDCT is 8 times too big (The 1 dimensional transform is applied twice giving $(2\sqrt{2})^2$ magnitude increase). This is in accordance with the 3 bit shift of the output data necessary to give the correct result.

5.3.4 IDCT coefficients

1.0000	1.3870	1.3066	1.1759	1.0000	0.7857	0.5412	0.2759
1.0000	1.1759	0.5412	−0.2759	−1.0000	−1.3870	−1.3066	−0.7857
1.0000	0.7857	−0.5412	−1.3870	−1.0000	0.2759	1.3066	1.1759
1.0000	0.2759	−1.3066	−0.7857	1.0000	1.1759	−0.5412	−1.3870
1.0000	−0.2759	−1.3066	0.7857	1.0000	−1.1759	−0.5412	1.3870
1.0000	−0.7857	−0.5412	1.3870	−1.0000	−0.2759	1.3066	−1.1759
1.0000	−1.1759	0.5412	0.2759	−1.0000	1.3870	−1.3066	0.7857
1.0000	−1.3870	1.3066	−1.1759	1.0000	−0.7857	0.5412	−0.2759

5.3.5 IDCT coefficients (14 bit signed integers)

4096	5681	5352	4816	4096	3218	2217	1130
4096	4816	2217	−1130	−4096	−5681	−5352	−3218
4096	3218	−2217	−5681	−4096	1130	5352	4816
4096	1130	−5352	−3218	4096	4816	−2217	−5681
4096	−1130	−5352	3218	4096	−4816	−2217	5681
4096	−3218	−2217	5681	−4096	−1130	5352	−4816
4096	−4816	2217	1130	−4096	5681	−5352	3218
4096	−5681	5352	−4816	4096	−3218	2217	−1130

5.4 FILTER FUNCTION

The filter function is selected with **SEL[2-0]**=010. (mode 2)

This filter is intended to be used for image data, taking 9 bit signed input data and giving a 9 bit signed result.

5.4.1 Internal number format

The input to the filter is a 9 bit signed integer in the range −256 to +255. The input is multiplied in the matrix multiplication array by 14 bit signed fixed-point numbers in the range −2 to $2-2^{-12}$. The accumulated result of 8 multiply operations is a 26 bit signed integer, the bottom 5 bits of which are rounded (see section 5.1.3) and the top 5 bits are used to saturate the output (see section 5.1.4). The result of the first matrix multiply is stored as a 16 bit signed integer and the second matrix multiply performed in exactly the same manner, yielding 33 bit results. The output rounds the bottom 19 bits, saturates the top 5 bits giving a 9 bit signed integer in the range −256 to +255.

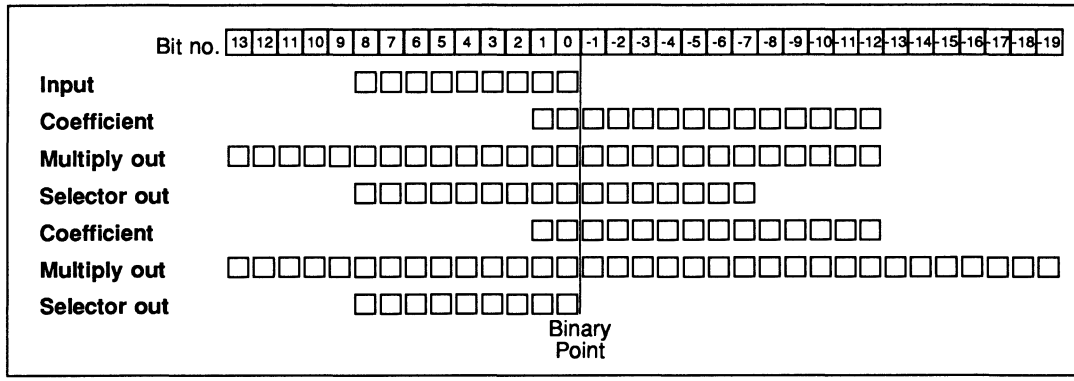
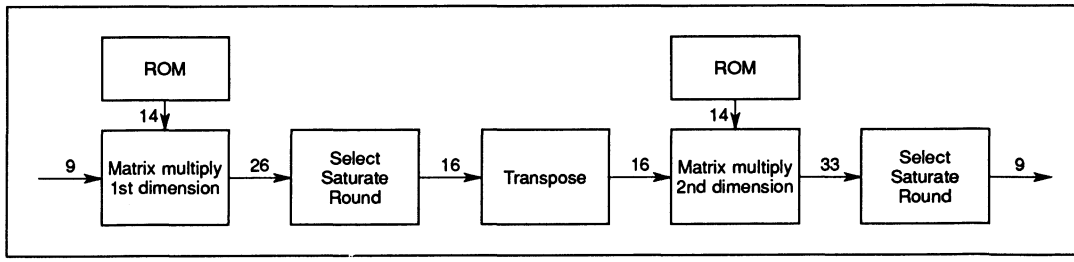


Figure 5.3 Filter and Transpose internal number format

5.4.2 Internal data flow



5.4.3 Definition of filter

The filter is a simple $\frac{1}{4}-\frac{1}{2}-\frac{1}{4}$ filter applied in both dimensions which means that the overall filter kernel is:

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

i.e. an output pixel is calculated from the corresponding pixel in the input field and its eight closest neighbours by evaluating

$$\tfrac{1}{16}(4 \times \text{pixel} + 2 \times (\textstyle\sum \text{four adjacent pixels}) + 1 \times (\textstyle\sum \text{four diagonal pixels}))$$

However, at the block edges, where some of the pixels would fall outside the block boundary, the filter is modified to 0–1–0 which means that along the edge the kernel would be:

$$\tfrac{1}{16}\begin{bmatrix} 0 & 0 & 0 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \text{ (rotated to suit)}$$

and the corner pixels are passed through unmodified.

5.4.4 Filter coefficients

$$\begin{bmatrix} 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.2500 & 0.5000 & 0.2500 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.2500 & 0.5000 & 0.2500 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.2500 & 0.5000 & 0.2500 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.2500 & 0.5000 & 0.2500 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.2500 & 0.5000 & 0.2500 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.2500 & 0.5000 & 0.2500 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 \end{bmatrix}$$

5.4.5 Filter coefficients (14 bit signed integers)

$$\begin{bmatrix} 4096 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1024 & 2048 & 1024 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1024 & 2048 & 1024 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1024 & 2048 & 1024 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1024 & 2048 & 1024 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1024 & 2048 & 1024 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1024 & 2048 & 1024 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4096 \end{bmatrix}$$

5.5 TRANSPOSER FUNCTION

The transposition function is selected with **SEL[2-0]**=011. (mode 3)

This is intended to be used for filtering image data, taking 9 bit signed input data and giving a 9 bit signed result. Data is passed through unmodified and is intended to be used in conjunction with the filter function (**SEL[2-0]**=010), so that by toggling **SEL[0]** the filter can be switched in and out.

5.5.1 Internal number format and data flow

The internal number format and data flow for the transpose function are the same as for the filter function. Refer to sections 5.4.1 and 5.4.2.

5.5.2 Transposition coefficients

$$\begin{bmatrix} 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 \end{bmatrix}$$

5.5.3 Transposition coefficients (14 bit signed integers)

$$\begin{bmatrix} 4096 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4096 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4096 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4096 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4096 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4096 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 4096 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4096 \end{bmatrix}$$

5.6 PIN DESIGNATIONS

System services

Pin	In/out	Function
VCC, GND		Power supply and return
CLK	In	Input clock

Synchronous input/output

Pin	In/out	Function
GO	In/out	Initiate input/computation/output cycle
Din[11-0]	In	Data input port
Dout[11-0]	Out	Data output port
Dx[11-3]	In	Addition/subtraction port
SEL[2-0]	In	Mode select input port

5.6.1 System services

Power

Power is supplied to the device via the **VCC** and **GND** pins. Several of each are provided to minimise inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100nF low inductance (e.g. ceramic) capacitor between **VCC** and **GND**. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.

Input voltages must not exceed specification with respect to **VCC** and **GND**.

CLK

The clock input signal **CLK** controls the timing of input and the output on the three dedicated interfaces, and controls the progress of data through the addition/subtraction units, multipliers and transposition RAM. Since the IMS A121 is fully static, the clock can be stopped in either phase without corrupting data.

5.6.2 Synchronous input/output

GO

The **GO** signal is active high and is sampled on the rising edge of the input clock. If the device is processing a previous block of data, the **GO** signal is ignored. Otherwise, the processing of a block of 64 pixels commences and the **GO** signal is ignored for a further 63 cycles. Data is always assumed to be valid for the 64 cycles from the start of a major cycle. Blocks of data may be processed at any time and with any spacing between the major blocks, by toggling the **GO** signal as necessary.

Din[11-0]

The data input port is sampled 64 times on successive clock cycles, commencing when **GO** is sampled high. Data must be valid on the rising edge of **CLK** for each of the 64 cycles. The block of data may be considered as an 8×8 matrix, where each group of 8 samples represents a column, and the 8 columns are sampled consecutively until the block is complete. The data is *twos complement, Little Endian* so that **Din[11]** gives sign information, and **Din[0]** is the least significant bit.

Dout[11-0]

The data output port will be valid for periods spanning 64 clock cycles. The data will be valid on the rising edge of the clock, exactly 128 cycles (the latency) after the data was sampled on the input. This output data, which may be considered as an 8 × 8 matrix, is transposed with respect to the input data. The data is *twos complement, Little Endian* like the input data.

Blocks of data may follow directly after one-another so that the first data of a block is presented exactly 64 cycles after the first data of the preceding block. However, if there is a gap between blocks zero will appear on the data output port between blocks of data.

Dx[11-3]

The addition/subtraction port is sampled on each clock cycle in exactly the same way as the data input port. The data on this port will either be subtracted from the signal on the data input port before matrix multiplication, or, added to the result of matrix multiplication prior to output. The addition and subtraction functions can never be used together. The function selected is determined by the **SEL[2-0]** signals. The data is *twos complement, Little Endian* like the **Din/Dout** data. Note however, that although the **Dx** port has a different width, **Dx[10]** has the same bitwise significance as **Din[10]/Dout[10]**.

The timing of data on the **Dx** port is different depending on the selected mode.

In the case of subtraction in the DCT mode, **SEL[1-0]=00**, data is presented on the **Dx** port on the same cycle as the corresponding data (from which it will be subtracted) is presented on the **Din** port.

In the case of addition in the IDCT mode, **SEL[1-0]=01**, data is presented on the **Dx** port exactly 4 cycles before the corresponding data (to which it will have been added) appears on the **Dout** port.

SEL[2-0]

The mode select input port is sampled on the rising edge of **CLK**, when **GO** is active, at the start of a block of data. This fixes the selected mode for the entire block of data.

SEL[2-0]	Mode	Function	PreSubtract	PostAdd	Clipping	**Din** width	**Dout** width
000	0	DCT	Disabled	Disabled	Disabled	9	12
001	1	IDCT	Disabled	Disabled	Disabled	12	9
010	2	Filter	Disabled	Disabled	Disabled	9	9
011	3	Transpose	Disabled	Disabled	Disabled	9	9
100	4	DCT	Enabled	Disabled	Disabled	9	12
101	5	IDCT	Disabled	Enabled	Enabled	12	9
110	6	Reserved – Do not use					
111	7	IDCT	Disabled	Disabled	Enabled	12	9

5.7 ELECTRICAL SPECIFICATION

5.7.1 DC electrical characteristics

Absolute maximum ratings

Symbol	Parameter	Min	Max	Units	Notes (1)
VCC	DC supply voltage	0	7.0	V	2
VI, VO	Voltage on input and output pins	−1.0	VCC+0.5	V	2
TA	Temperature under bias	−40	85	°C	2
TS	Storage temperature	−65	150	°C	2
PDmax	Power dissipation		1.5	W	2

1 All voltages are with respect to GND.

2 This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

DC operating conditions

Symbol	Parameter	Min.	Nom.	Max.	Units	Notes (1)
VCC	DC supply Voltage	4.5	5.0	5.5	V	
VIH	Input Logic '1' Voltage	2.0		VCC+0.5	V	2
VIL	Input Logic '0' Voltage	−0.5		0.8	V	2
TA	Ambient Operating Temperature	0		70	°C	3

Notes

1 All voltages are with respect to GND. All **GND** pins must be connected to GND.

2 Input signal transients 10 ns wide, are permitted in the voltage ranges GND – 0.5 V to GND – 1.0 V and VCC + 0.5 V to VCC + 1.0 V.

3 400 linear ft/min transverse air flow.

DC characteristics

Symbol	Parameter	Min.	Max.	Units	Notes (1,2)
VOH	Output Logic '1' Voltage	2.4	VCC	V	IO $\leq$ −4.4 mA
VOL	Output Logic '0' Voltage	0	0.4	V	IO $\leq$ 4.4 mA
IIN	Input leakage current (any input)		±10	μA	3
ICC	Average power supply current		300	mA	4

Notes

1 All voltages are with respect to GND. All **GND** pins must be connected to GND.

2 Under the conditions specified by the DC operating conditions.

3 VCC = VCC(max), GND $\leq$ VIN $\leq$ VCC

4 This applies at 20 MHz and will be less at slower clock rates

5.7.2 A.C. timing characteristics

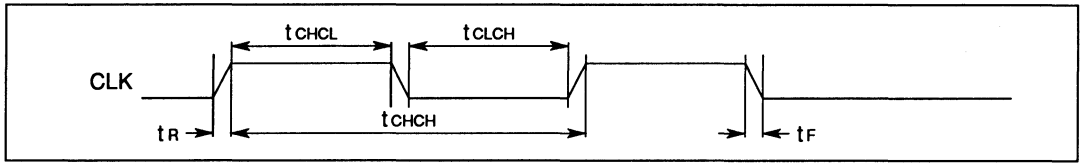
All timings are given for a load of 30pF unless otherwise stated.

Clock requirements

Symbol	Parameter	Min	Max	Units	Notes
t_{CHCL}	Clock Pulse High Width	20		ns	
t_{CLCH}	Clock Pulse Low Width	20		ns	
t_{CHCH}	Clock Period	50		ns	
t_R	Clock rise time	0	50	ns	1
t_F	Clock fall time	0	50	ns	1

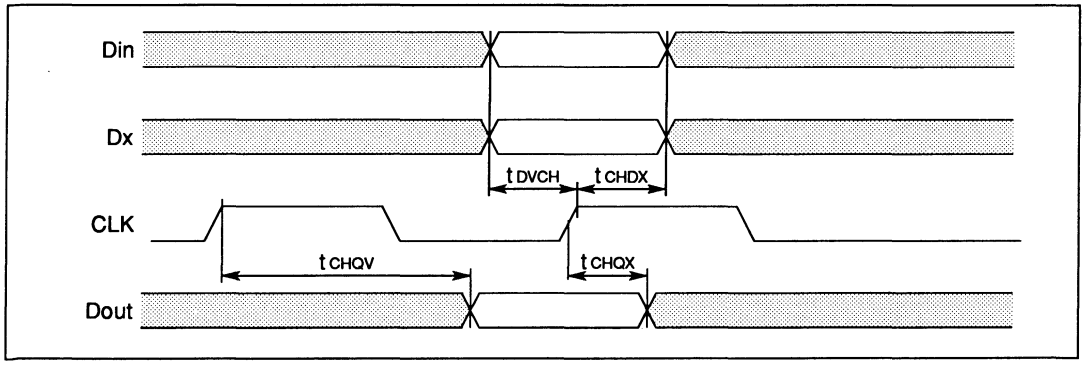
Notes

1 The clock edges should be monotonic between VIL and VIH.



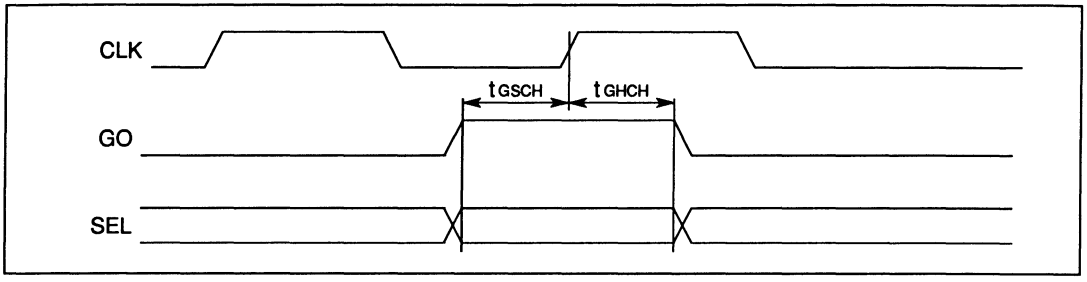
Synchronous input and output (Din, Dout, Dx)

Symbol	Parameter	Min	Max	Units	Notes
t_{CHQV}	CLK high to Dout Valid		38	ns	
t_{CHQX}	Dout hold time after CLK	2		ns	
t_{DVCH}	Din/Dx setup time to CLK high	10		ns	
t_{CHDX}	Din/Dx hold time to CLK high	0		ns	

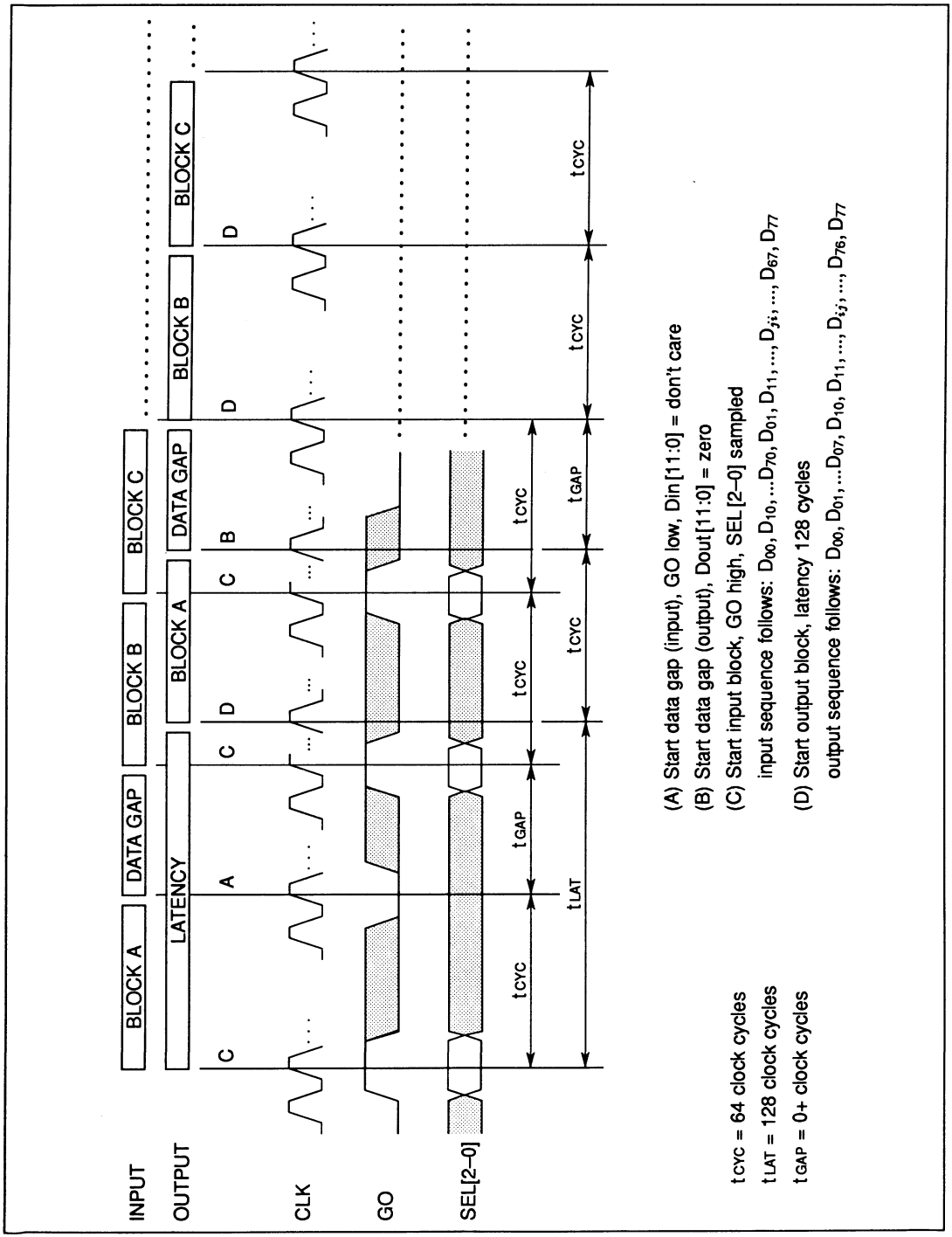


Synchronous control (GO, SEL[2-0])

Symbol	Parameter	Min	Max	Units	Notes
t_{GHCH}	GO/SEL hold to clock high	0		ns	
t_{GSCH}	GO/SEL setup to clock high	10		ns	



Overall data timing



5.8 PACKAGE SPECIFICATIONS

5.8.1 44 pin PLCC package

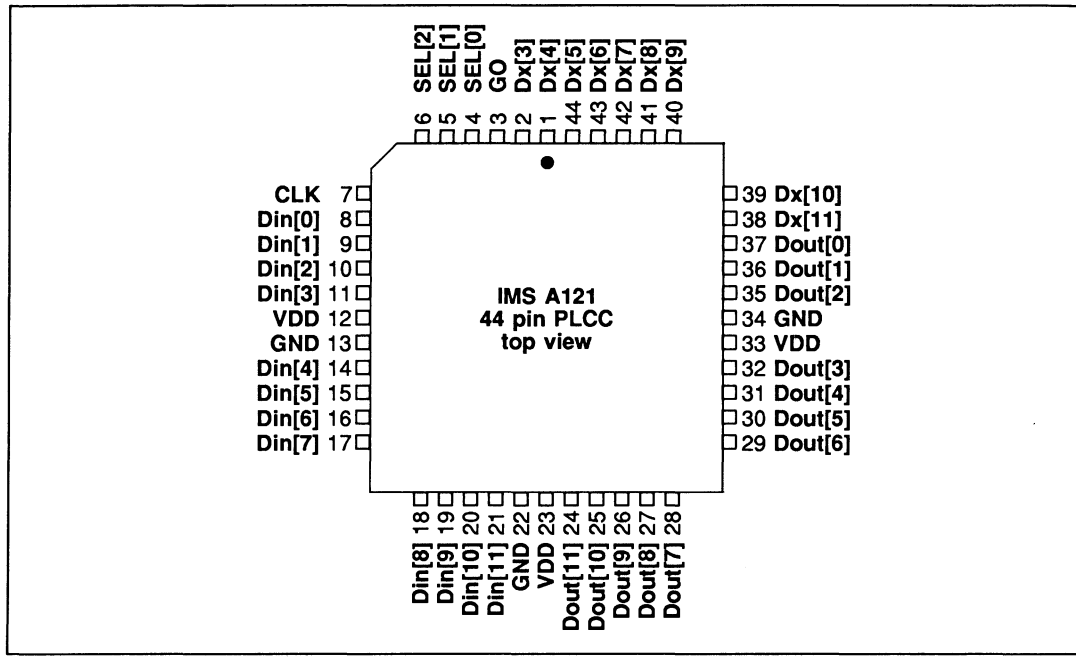


Figure 5.4 IMS A121 44 pin PLCC J-bend package pinout

Note

All **VCC** pins **must** be connected to the 5 Volt power supply.
All **GND** pins **must** be connected to ground.

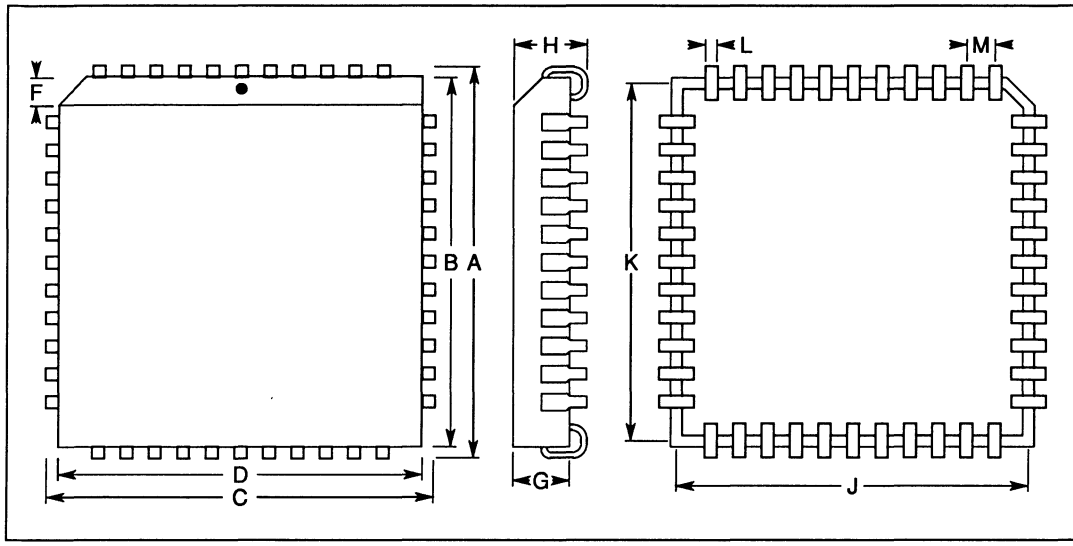


Figure 5.5 44 pin PLCC J-bend package dimensions

DIM	Millimetres		Inches		Notes
	NOM	TOL	NOM	TOL	
A	17.577	±	0.692	±	
B	16.612	±	0.654	±	
C	17.577	±	0.692	±	
D	16.612	±	0.654	±	
F	1.143		0.045		
G	3.861		0.152		
H	4.369	±	0.172	±	
J	15.748	±	0.620	±	
K	15.748	±	0.620	±	
L	0.457		0.018		
M	1.270		0.050		

Table 5.1 44 pin PLCC J-bend package dimensions

PLCC thermal characteristics

Symbol	Parameter	Min	Nom	Max	Units	Notes
θ JA	Junction to ambient thermal resistance				°C /W	1,2

Notes

1 Measured at 400 linear ft/min transverse air flow.

2 This parameter is sampled and not 100% tested.

5.9 ORDERING DETAILS

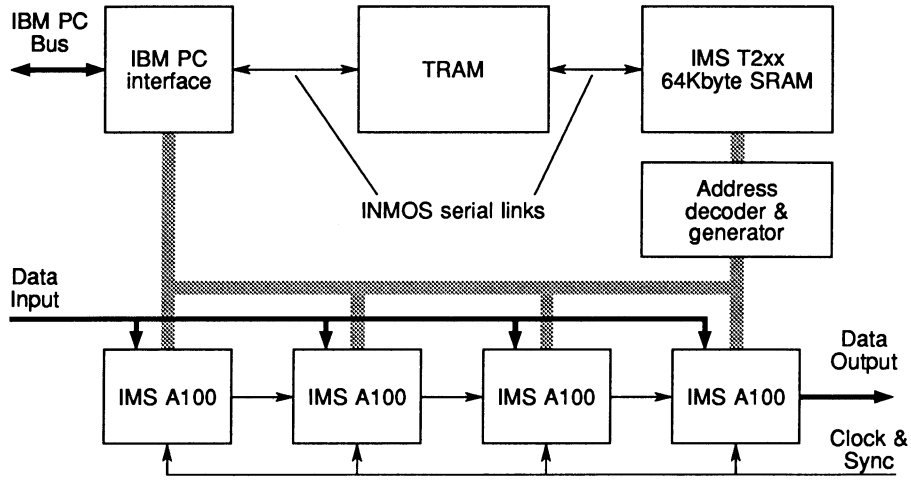
The following table indicates the designation of the IMS A121 variants.

INMOS designation	Package	Clock speed	Military/commercial
IMS A121-J20S	Plastic LCC	20 MHz	commercial

IMS B009

IMS A100 DSP System Evaluation Board

Product Overview



Features

- High performance Digital Signal Processing development board for both real-time and non-real time compute-intensive applications
- Cascade of four IMS A100s
- Up to 1280 Million Operations Per Second (MOPS) capability
- Up to 10MSamples/sec continuous data throughput
- Fully programmable using an IMS T2xx 16 bit transputer with 64Kbyte SRAM
- Option to install TRAM
- Transputers arrayable for high performance pipelined systems
- General purpose address mapper (Look Up Table) for data sequencing
- Data supplied from internal (i.e. software/file) or external sources
- Controllable from IBM PC applications under MS-DOS or other transputer systems
- IMS A100 cascade accessible directly from IBM PC bus
- Complete DSP development environment available, including IMS A100 and IMS B009 software simulators
- Compatible with full transputer board family

6.1 The IMS B009 Evaluation Board

The IMS B009 can be used to evaluate and implement a wide range of high performance DSP techniques. It can also be used by OEMs as a component for building high performance, flexible, DSP systems, where the production quantities do not justify development of a specific DSP board.

The IMS B009 is an IBM PC (XT or AT) add-in board containing 4 IMS A100 signal processors controlled by an IMS T2xx. The 4 IMS A100s can be used to implement 128 tap FIR filters, convolvers or correlators, on 16 bit data, with 16 bit coefficients at rates up to 2.5 M samples/second, or at up to 10M samples/second with 4 bit coefficients.

The IMS A100s can be controlled and configured either directly from the IBM PC or, for much greater performance, by the IMS T2xx. Data flow through the IMS A100s can be controlled by the IBM PC, the IMS T2xx or directly from an external digital signal source, via a DIN 41612 edge connector. This last option allows the IMS A100s to process data at rates up to 10MHz.

The IMS T2xx has 64Kbytes of fast SRAM. The interface between the IMS T2xx, the SRAM, and the IMS A100s is designed to allow the IMS T2xx to move data through the IMS A100s at speeds up to 1.25 M samples/second. An address mapping table allows the IMS T2xx to perform complex data sequencing tasks at high speeds. Each of the 4 transputer links on the IMS T2xx can be used to transfer data between the IMS B009 and other transputer systems at up to 0.8 Mbytes/second simultaneously. using more than one link for data I/O can provide data transfer rates of several Mbytes/second.

The IMS B009 is a Transputer Module (TRAM) motherboard. A single TRAM, up to size 4, can be installed. For example, an IMS B404 TRAM can be used to provide 2 Mbytes of data storage and additional (possibly floating point) data processing. This same TRAM could be used to run software packages such as the IMS D700 Transputer Development System and the IMS D703 DSP Development System. The IMS B009 (with a suitable TRAM) thus provides the basis for both a Transputer and a DSP development workstation.

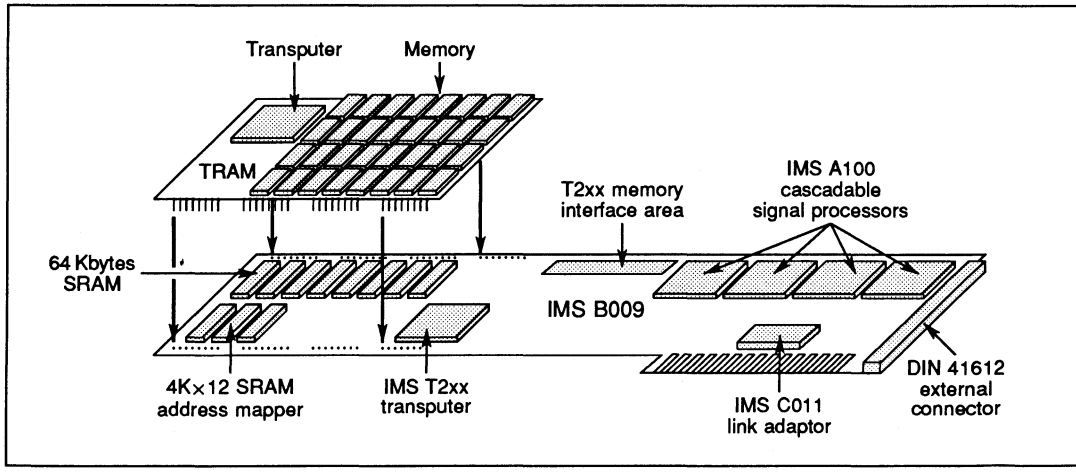


Figure 6.1 IMS B009 key components

6.2 Board Description

The IMS B009-1 contains a cascade of four IMS A100s, an IMS T2xx 16 bit transputer with 64Kbyte SRAM, and a socket for a standard TRAM.

An INMOS TRAM (e.g. IMS B404) provides a general purpose host processor, capable of supporting the full INMOS occam 2 Transputer Development System, and the IMS D703 DSP Development System. The IMS T2xx is used as a high performance controller for the IMS A100 cascade. Figure 2 shows the board with the optional TRAM, and the configuration of the IMS A100 cascade.

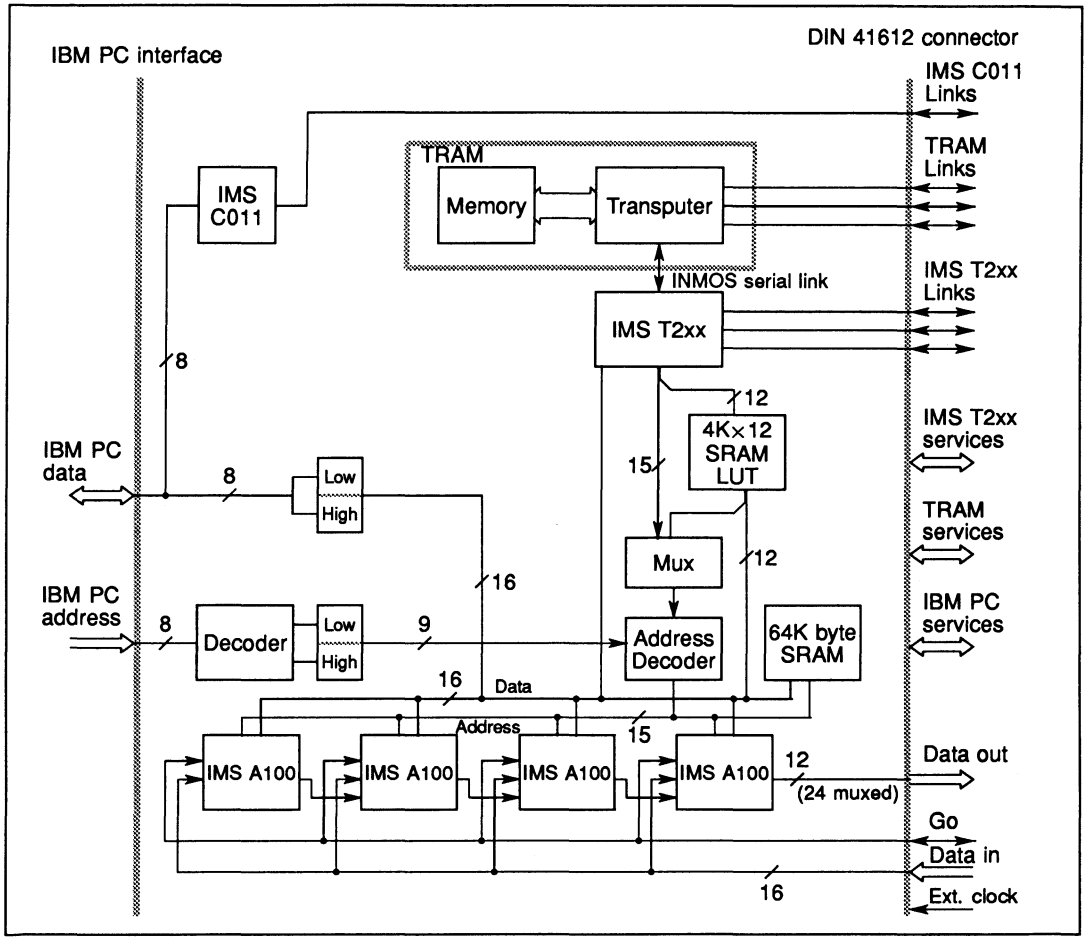


Figure 6.2 IMS B009 Detailed Block Diagram

A 4K×12 'address mapper' is provided for high speed generation of arbitrary address sequences. This mapper can be applied at any time to any addresses generated in the positive address space of the IMS T2xx, without any performance degradation. Thus, arbitrary data sequences can be preloaded, and applied at the appropriate point during data processing.

The IMS T2xx can be connected to any other transputer with one or more standard INMOS serial links, each link being capable of approx. 0.8MBytes/sec in each direction, and operation in full duplex. The transputer links can also be used to connect to other transputer evaluation boards, or for arraying IMS B009s to form a high bandwidth signal processing pipeline.

The TRAM/B009-1 combination offers a powerful concurrent processing environment, with preprocessing operations such as data pre/post ordering handled by the TRAM, whilst the IMS T2xx drives the IMS A100s. For highest performance, external ports are provided, enabling users to supply real time data to the A100 cascade, and output processed data at full speed. Thus, real time processing can be implemented with a minimum of additional hardware.

In order to maximise the range of applications of the IMS B009, most of the key control and data signals are brought to either the 96 way DIN 41612 connector, or to an internal connection area. This enables users to construct DMA interfaces to all devices on the IMS T2xx memory interface bus. Thus, a wide range of real time interfaces can be realised, making the IMS B009 ideal for general laboratory use or for prototyping final systems.

Input data can thus be supplied from one of four sources:

- External data port (10MSamples/sec continuous)
- IMS T2xx memory interface (approx. 5MBytes/sec burst)
- Transputer link (approx. 0.8MBytes/sec x 4 burst)
- IBM PC bus (approx. 0.2MBytes/sec burst)

Due to the relatively small power supplies provided with some IBM PC compatibles, special links have been provided to isolate the V_{cc} plane from the IBM PC power pins, and to provide external power directly via the DIN 41612 connector. This enables several IMS B009s to be used in a standard IBM PC chassis without danger of exceeding either power supply or backplane ratings.

6.3 Programming

The IMS B009 enables users to exploit the flexibility of the IMS A100 under a standard occam 2 environment, by running the IBM PC Transputer Development System (IMS D700) on the TRAM located on the board itself. In this way, high performance DSP systems can be realised, using high level languages throughout. The IMS D703 DSP Development System, supplied in both source code and binary form, demonstrates how to make best use of the various addressing modes and facilities of the board.

The board can also be treated as a peripheral to the IBM PC, responding to commands sent to it on the PC bus. This mode of operation disables the transputers and limits data rates to those attainable on the standard IBM PC bus, but does enable users to evaluate the potential of attaching IMS A100s directly to an IBM PC, controlled by a normal PC based program.

Alternatively, using the IMS B009 driver program supplied with the IMS D703, the IBM PC application can boot the IMS T2xx directly. It can then use the driver program in exactly the same way as any transputer application, communicating with the IMS T2xx via the link adaptor. This approach provides far higher performance for IBM PC hosted applications than controlling the IMS A100s directly via the PC bus.

6.4 Product summary

The IMS B009-1 comprises four IMS A100s, one IMS T2xx 16-bit transputer with 64Kbyte SRAM, and the 4K×12 address mapper. It also contains an unpopulated socket for a TRAM (up to size 4).

A comprehensive suite of documentation is supplied with each system, including full descriptions of the board design, software users and reference manuals, and a set of application notes. Test software is also provided, which performs extensive diagnostics of all functional components of the board.

6.5 Technical summary

Board ready for installation in a single IBM PC XT or AT system unit expansion slot

Four IMS A100-G21S cascadable signal processors

One IMS T2xx 16 bit transputer with 64 kbytes SRAM

10 or 20 MBit/sec INMOS link transmission speeds

DIN 41612 96 pin I/O connector

A100 signals:
Data in/out
Clock/Go/OutRdy

Transputer signals:
TRAM, T2xx Reset/Analyse/Error
1 INMOS link from IMS C011
3 INMOS links from IMS T2xx
3 INMOS links from TRAM

+5V, ground

Cables (suitable for connection to all INMOS evaluation boards)

INMOS links
Up/Down Reset/Analyse/Error cables
Standard Jumpers

Power supply required (from IBM PC or externally)

+5V (approx. 4 amps with TRAM)
Ground

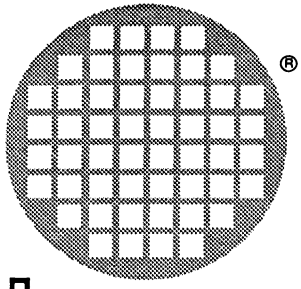
Note: *the IMS B009-1 can operate with external power supplies if required.*

6.6 Ordering details

Product	Part number
B009 Evaluation board	IMS B009-1

Available Documentation

IMS A100 Data Sheet
IMS A100 Application Note 1: Digital Filtering with the IMS A100
IMS A100 Application Note 2: Discrete Fourier Transforms with the IMS A100
IMS A100 Application Note 3: Correlation and Convolution with the IMS A100
IMS A100 Application Note 4: Complex (I & Q) Processing with the IMS A100
IMS A100 Application Note 5: Hardware considerations with the IMS A100
IMS A100 Application Note 6: Image processing with the IMS A100



inmos®

IMS D703

DSP Development System

Product Overview

Features

- Numerically accurate simulator of the IMS A100 Cascadable Signal Processor
- Software simulation of both 'raw' IMS A100 cascade and IMS B009
- Comprehensive IMS T212-based IMS A100 driver optimised for IMS B009 facilities
- Interactive mode providing direct access to all software models
- User definable application tasks, written in occam 2
- Capable of simulating any configuration of IMS A100 devices
- Wide range of examples supplied, including FIR, convolution, correlation, DFT, and algorithm partitioning techniques
- Simple and efficient software to hardware development route
- occam 2 language for data generation and control of user applications
- Multi-window graphics interface
- Range of DSP design aids, including FIR, DFT design tools
- Runs on any IBM PC (with CGA/EGA) hosted transputer/occam 2 system
- All occam software components supplied in both fully documented source and binary code forms

7.1 Introduction

The IMS D703 DSP Development System is a comprehensive software tool for developing DSP applications at a number of different levels, e.g.

- Optimal (Remez exchange) FIR filter design.
- Numerically accurate simulation of a filter on the IMS A100 software model with test data.
- Software emulation of an entire IMS A100/Transputer based DSP system.
- Evaluation of a filter on 'live' data in real time (using IMS A100s on an IMS B009 evaluation board).

The IMS D703 is based on a software harness which controls the data flow between a number of processes providing facilities such as:

- Accurate simulation of one or more IMS A100s.
- Communication with the filing system, I/O etc. of the host PC.
- Communication with other evaluation hardware (e.g. the IMS B009 or any transputer based system).
- Application software.

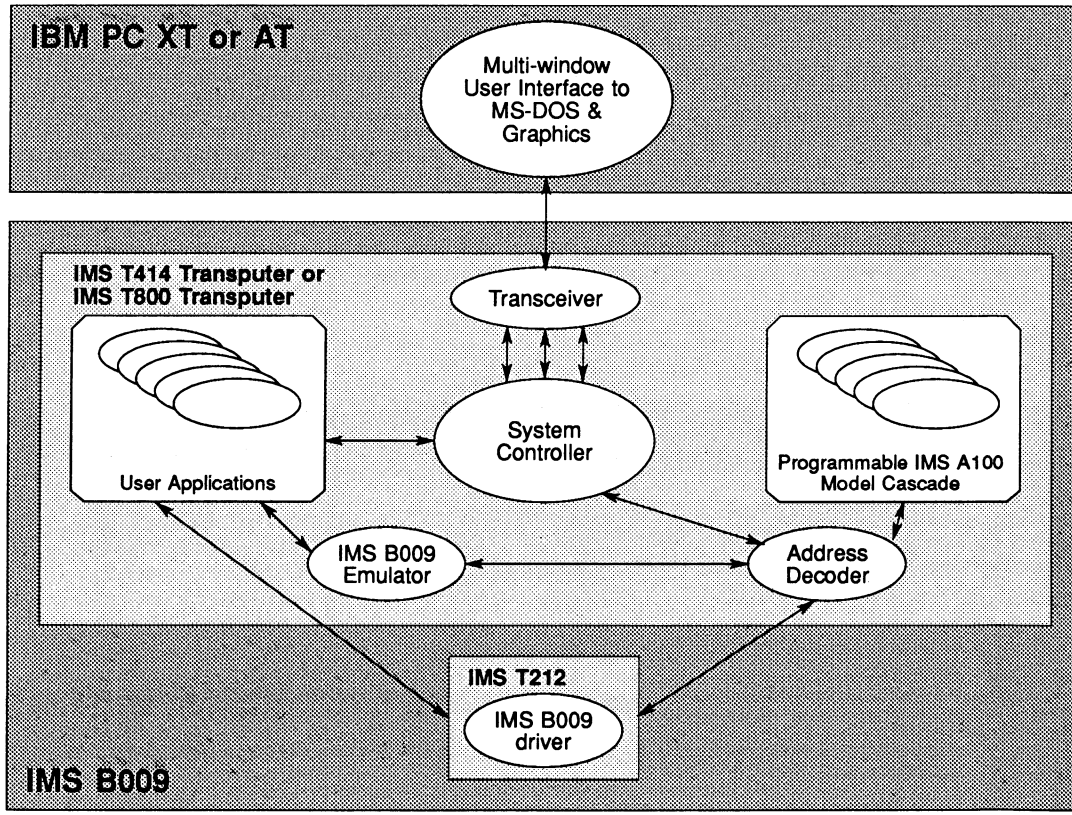


Figure 7.1 IMS D703 structure

An example of an application software package is the filter design and evaluation package provided in the IMS D703. This allows the user to interactively specify a filter which is then designed using the Remez exchange technique. Having designed a filter this application package then allows the user to apply test data patterns to either a simulation of the filter or to a hardware implementation of the filter (using the 4 IMS A100s on an IMS B009, if available). The input and output waveforms of the filter can be displayed in different graphics windows.

The application packages provided within the IMS D703 allow the user to interactively experiment in a number of DSP areas (e.g. filter design and using correlation to detect a signal in noise) without modifying the IMS D703 software.

Users can perform more elaborate DSP system simulations by writing their own application packages to work within the harness of the IMS D703. This allows the user to concentrate on developing algorithms and techniques while using the resources for file I/O, simulation etc. provided by the IMS D703.

7.2 Requirements

The IMS D703 is supplied as both (IMS T414) executable code and occam source code. If the user wants to modify the IMS D703 or to recompile it to execute on a different 32 bit transputer (e.g. IMS T800) the IMS D700 Transputer Development System is required. The IMS D703 and IMS D700 packages are designed to run on a transputer (T4 or T8 family) with at least 1 Mbyte of memory, hosted by an IBM PC XT, AT or compatible running MS-DOS. The IMS D703 also requires an EGA/CGA graphics display.

A suitable combination of products is:

IMS D703, IMS D700, IMS B009-1, IMS B404

Alternatively an IMS B008 can be substituted for the IMS B009-1; this only allows software simulation of IMS A100s.

7.3 Software Description

7.3.1 User Applications

A 'user application' is an occam process which can control any of the IMS A100 systems accessible to the IMS D703. By combining the services of the system controller with those of the IMS B009 driver, application programs can be written to perform any required task. In order to make these programs as simple a possible to write, a standard occam 'harness' is provided which includes a comprehensive set of standard procedures, constants, and interfaces to the rest of the IMS D703 environment. All the example applications supplied with the IMS D703 use this standard structure. They include code for convolutions, Discrete Fourier Transforms and FIR design using both hardware and software versions of the IMS A100.

Since the IMS D703 is implemented using the occam 2 language, user applications themselves are written in occam. Thus an advanced high level language is used for programming the system, rather than awkward text files or specialised macro languages. The range of standard procedures and examples supplied means that users can quickly modify an application to meet their specific requirements, whilst achieving a high level of performance. As users become more confident, they can develop their own optimised systems using parts of the IMS D703 as required. Indeed, because the IMS D703 executes entirely on the transputer host, dedicated applications can be developed which enable the IMS B009 to act as a high performance 'turbocharger' to an existing MS-DOS applications.

```
PROC FIR.example(CHAN OF ANY to.controller, from.controller)
  ... STANDARD     services
  ... USER_DEFINED data and PROCs

  VAL descriptor IS "128 tap FIR demo" :

  SEQ
    ... STANDARD initialisation
    WHILE TRUE -- loop forever
      SEQ
        ... evaluate coefficients
        -- Load coefficients into the A100s
        SEQ address=ccr.base FOR number.of.stages
          A100.write(address, coefft[i])

        -- Pass test data through A100s
        SEQ i = 0 FOR test.data.size
          SEQ
            A100.write(DIR, test.data[i])
            A100.read(DOL, result.data[i])
    stop()
:
```

Figure 7.2 Example user application for an FIR filter

A range of applications are provided to demonstrate the system, and provide a valuable means of evaluating the IMS A100. These include:

- 1D and 2D Convolution
- Correlation for Pulse Compression
- Discrete Fourier Transforms using the Prime Number Transform
- FIR Coefficient Design (using Remez Exchange method)

7.3.2 IMS A100 Model

At the heart of the IMS D703 is the system level occam 2 software model of the IMS A100. This provides a complete numerically accurate model of the IMS A100 in all modes of operation. Any configuration of cascaded IMS A100s can also be modelled, by joining the models together with occam channels. Also, since the model is written in occam, a model cascade can be distributed across an array of transputers for improved performance.

System simulations can be performed by connecting the Standard Microprocessor Interface (SMI) channels of one or more IMS A100 models to an address decoder model. This enables memory mapped systems to be modelled, and the address decoding scheme of a prototype system tested. Several examples of these are provided with the IMS D703, including a simple linear address space model, and an emulation of the IMS B009 address decoder. Alternatively, test data can be supplied to the data input channels, which emulate the external Data INput (DIN) port of the IMS A100. Likewise, the output can be received on the data output channel of the model (DOUT of the IMS A100), or read from the DOL/DOH registers via the SMI channels.

7.3.3 Address Decoder

To model the behaviour of a microprocessor-controlled IMS A100 based system, an address decoder process is provided. This receives all memory read and write requests for the IMS A100s, and decodes the addresses according to the selected decoding scheme.

Three modes of operation are provided:

- Raw Cascade Mode: Each device of the IMS A100 software model cascade is allocated 128 contiguous address locations. A special address is also provided so that a single write operation loads data into the Data Input Registers (DIRs) of all the devices in the cascade at the same time.
- IMS B009 Emulator Mode: A fixed cascade size of four IMS A100 devices is used, and the address decoder emulates the decoding used on the IMS B009 hardware.
- IMS B009 Hardware: All memory requests are passed directly to the IMS B009 driver. Additional commands are provided to reset and boot the IMS T2xx on the IMS B009 with a new driver if required.

7.3.4 IMS B009 driver

Another feature of the IMS D703 is the IMS B009 driver. The IMS B009 is a high performance DSP board, which contains four IMS A100s with a hardware optimised interface to the IMS T2xx controller. In order to control this board, and make full use of features such as address mapping and special block move modes, a software driver is provided which executes on the IMS T2xx of the IMS B009.

During startup, the IMS D703 will look for the IMS T2xx, and if present it will automatically bootstrap the IMS T2xx with the IMS B009 driver. This enables the user to access the resources of the IMS B009, and provides a means whereby user applications can perform complex operations at high speed with minimal knowledge of the IMS B009 hardware.

For certain applications demanding maximum performance, more optimised drivers may be required. The standard IMS T2xx-based driver provided with the IMS D703 gives users an excellent framework from which highly optimised drivers can be rapidly and reliably produced. Also, user applications can bootstrap the IMS T2xx directly, so that experienced users can create optimised drivers for each application if necessary, within a common environment.

7.3.5 IMS B009 Emulator

A software emulation of the IMS B009 loaded with the driver described above is also provided. IMS D703 users can thus develop code for the IMS B009 without having the hardware physically present. Using the standard procedures provided, users can produce applications that can execute with either the IMS B009 software emulator or the IMS B009 hardware without modification.

7.3.6 System Controller

The user environment is designed to enable DSP engineers to develop new algorithms quickly, and explore their performance without learning large command repertoires or understanding the software complexities of the IMS D703. The system controller controls this environment by providing a range of interactive services for debugging and monitoring of the system. 'Interactive' commands are entered using the keyboard, and are processed by a simple command line interpreter. Included are commands for reading and writing locations, plotting data both graphically and textually, monitoring messages from any part of the system, executing user applications, and controlling the various modes of operation. A full hierarchical online help facility is also provided, which can be customised to users' needs. A summary of commands is given in figure 7.3.

User application tasks can access additional functions via a special group of services, known as 'Binary' commands. These services are designed to execute far faster than the interactive commands, by processing commands in internal binary format directly rather than converting each command from text to binary prior to execution. Facilities that involve large amounts of data, such as plotting, are only provided with binary commands. The IMS B009 driver can also be accessed using binary commands. Both interactive and binary commands can be executed by user applications.

A keystroke file facility is provided, so that users can save all keyboard actions to file, and replay them. Keystroke files can also be created externally using any text editor or MS-DOS application, so that users can feed data gathered from external sources into the system, and capture the results for subsequent processing.

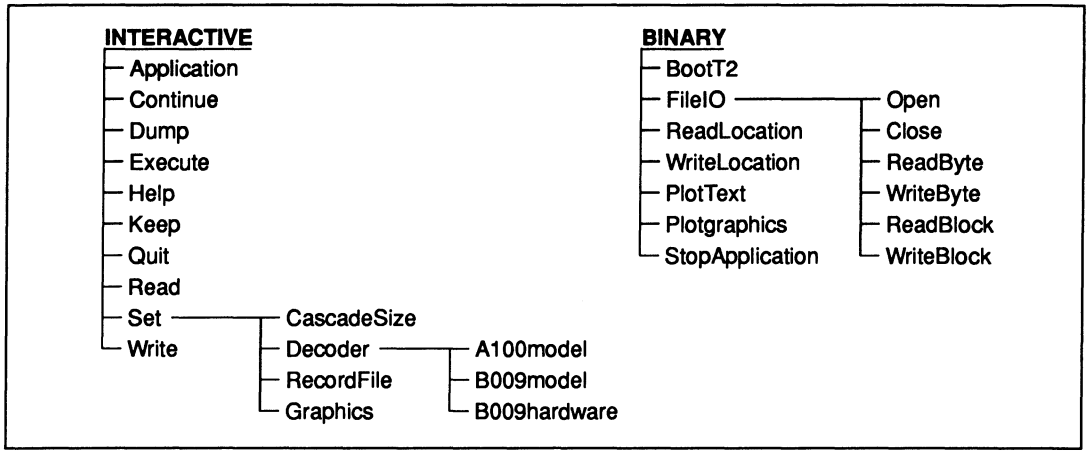


Figure 7.3 IMS D703 Command Summary

7.3.7 Multi-Window MS-DOS Interface

A multi-window user interface MS-DOS program is provided, which acts as a bridge between the resources of the IBM PC and the transputers. This program enables users to boot the host transputers, examine system status, plot data, access files, and manipulate the windows. In order to display the windows, and for plotting data graphically, an IBM PC fitted with CGA or EGA graphics hardware is required.

A comprehensive graphics suite is available to user applications, so that they can also make use of facilities such as automatic scaling, curve fitting, windows, and screen dumping. The graphics is based on the Turbo Pascal Graphics Toolbox, and access to most of the facilities of this package are provided.

7.4 Host Environment

The IMS D703 executes on any 32-bit transputer with at least 1Mbyte of RAM. If an IMS B009-1 and TRAM are being used, or the IMS D703 is being executed on an IMS B004 connected to an IMS B009-1, the standard IMS B009 driver will be automatically booted into the IMS T2xx. To modify the IMS D703 source code, users will require an IMS D700 Transputer Development System (TDS) for occam 2. The TDS runs on the same hardware as the IMS D703.

The IMS D703 graphics environment and run time support requires an IBM PC XT or AT with at least 512kbytes of RAM, a hard disk, MS-DOS version 3.1 or later, and an IBM compatible CGA or EGA graphics adaptor with monochrome or colour monitor.

7.5 Ordering Details

Product	Part number
B009 Evaluation board	IMS B009-1
Transputer Development System	IMS D700
DSP Development Software	IMS D703

digital filtering with the IMS A100

8.1 Introduction

When an analogue signal is sampled in time, the sampled signal is referred to as a discrete-time signal. If each sample in this discrete-time signal is also quantised in amplitude, (e.g. represented by an arbitrary n-bit number), then it is usually referred to as a digital signal. In the subject of digital filtering it is these types of signals which are processed and operated on. The fact that the digital signals are quantised both in time and amplitude gives one greater control over the processing as compared to analogue signal processing.

In these application notes the concept of the digital filtering is first introduced. This is done by starting from a simple RC analogue filter and deriving a corresponding digital filter. The classification of digital filters is then summarized, followed by giving a summary of techniques applicable to filter design using the IMS A100 device.

8.2 From analogue to digital

Figure 8.1a shows a simple first-order RC filter. The simple differential equation describing this circuit in terms of its input and output voltages is:

$$v_0(t) + RC\frac{dv_0(t)}{dt} = v_i(t) \tag{1}$$

where $v_0(t)$ and $v_i(t)$ are analogue output and input voltage waveforms. In the analogue world both input and output voltages are continuous-time waveforms and the complexity of the solution would depend on the input voltage function $v_i(t)$. Given an input waveform $v_i(t)$, the solution can be obtained using:

(i) Standard mathematical techniques which solve the differential equation and obtain the output waveform in closed form.

(ii) Numerical techniques which calculate the approximate output waveform in a digital computer. This would necessitate the sampling of the input and output waveforms.

The second method above provides the basis for digital filtering techniques. Consider that the input and the output voltages are sampled with a sampling interval T such that $v_i(nT)$ and $v_o(nT)$ represent the values of $v_i(t)$ and $v_o(t)$ at time $t = nT$.

If T is sufficiently small then the derivative $\frac{dv_o(t)}{dt}$ at time $t = NT$ can be approximated by:

$$\frac{dv_o(nT)}{dt} \simeq \frac{v_o(nT) - v_o((n-1)T)}{T} \tag{2}$$

substituting this in equation (1) we obtain:

$$v_o(nT) + \frac{RC}{T}v_o(nT) - \frac{RC}{T}v_o((n-1)T) = v_i(nT) \tag{3a}$$

Equation (3a) is a linear difference equation that approximates the differential equation (1). Equation (3a) can be rewritten as:

$$v_o(nT) = \frac{1}{1+(RC/T)}v_i(nT) + \frac{(RC/T)}{1+(RC/T)}v_o((n-1)T) \tag{3b}$$

This is now a recursion formula in which the present input sample and the previous output sample are used to calculate the present output sample. The notation can be simplified to:

$$v_o(n) = b_0 v_i(n) + a_1 v_o(n) \tag{4a}$$

where $b_0 = \frac{1}{1+(RC/T)}$ and $a_1 = \frac{(RC/T)}{1+(RC/T)}$.

The signal-flow diagram for this filter is shown in figure 8.1b. The block labelled 'D' represents a delay equal to one sampling period T. In digital filter notations a delay of n sampling periods is usually denoted by z^{-n}. Therefore a delay of one sampling period can be represented by z^{-1}.

It is important to note that a common element in all filter structures is the concept of storage. In the analogue RC filter (figure 8.1a) the storage is present in the form of a capacitor and in its digital equivalent (figure 8.1b) the storage takes the form of a delay stage. In fact the storage element is the essential ingredient for any filter whether analogue or a digital. This is because filters are used to operate on the signal 'changes' and as such they need to have some knowledge of the history of the signal to allow them to perform their function.

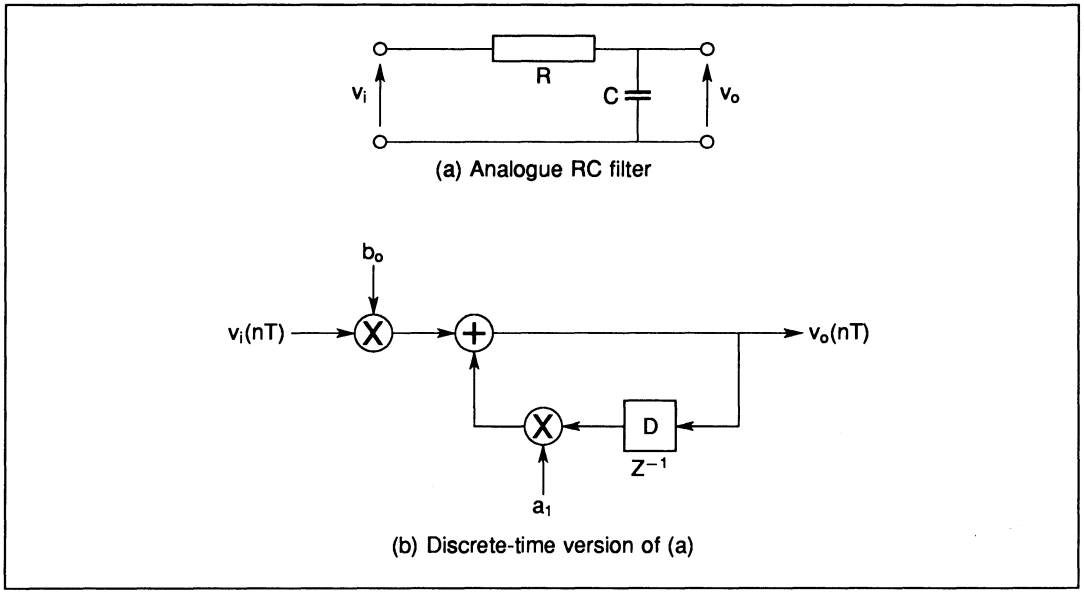


Figure 8.1 Analogue RC filter and its discrete-time equivalent

An important characteristic feature of any filters is its so called 'impulse response'. This is defined as the output waveform of the filter when a unity impulse is applied to the input. Using equation (4a) and assuming a unity impulse as the input waveform i.e.

$$\begin{aligned} v_i(0) &= 1 \\ v_i(n) &= 0 \quad for \quad n > 0 \end{aligned}$$

then the output sequence would be:

$$b_0,\ a_1 b_0,\ a_1^2 b_0, \ldots\ldots,\ a_1^n b_0, \ldots\ldots\ldots$$

or in short

$$v_0(n) = a_1^n b_0$$

It should be noted that the above impulse response has, in theory, infinite length. This is due to the recursive nature of this particular filter structure. This types of filters are often referred to as infinite-impulse-response (IIR) filters.

An alternative way of looking at the filter in this example is to use equation (4a) in successive substitutions i.e.

$$\begin{aligned} v_o(n) &= b_0 v_i(n) + a_1 v_o(n-1) \\ &= b_0 v_i(n) + a_1[b_0 v_i(n-1) + a_1 v_o(n-2)] \\ &= b_0 v_i(n) + a_1 b_0 v_i(n-1) + a_1^2[b_0 v_i(n-2) + a_1 v_o(n-3)] \\ &= \ldots\ldots \\ &= \ldots\ldots \\ &= b_0 v_i(n) + a_1 b_0 v_i(n-1) + a_1^2 b_0 v_i(n-2) + a_1^3 b_0 v_i(n-3) + \ldots\ldots \end{aligned} \tag{4b}$$

Equation (4b) expresses the output waveform as a linear combination of input samples only, but this involves infinite number of input samples. Notice also that the coefficients b_0 and a_1 have positive values less than unity (R and C are assumed to be finite and non-zero). This means that in equation (4b) the coefficients decrease for older input samples. It may therefore be reasonable to assume that these coefficients approximate to zero beyond a certain point. In this way only a finite number of terms would be involved in equation (4b), or in other words, the infinite impulse response is approximated by a finite impulse response since it decays rapidly to zero. This modified filter with its finite duration impulse response falls in the category of FIR (Finite-Impulse Response) filters. In the next section these concepts are generalized.

8.3 Digital filter classifications

Linear difference equations, similar to equation (4a & 4b) are the basis for the theory of digital filters. The general difference equation can be expressed as:

$$y(n) + \sum_{m=1}^{M} a_m y(n-m) = \sum_{k=0}^{N} b_k x(n-k) \tag{5}$$

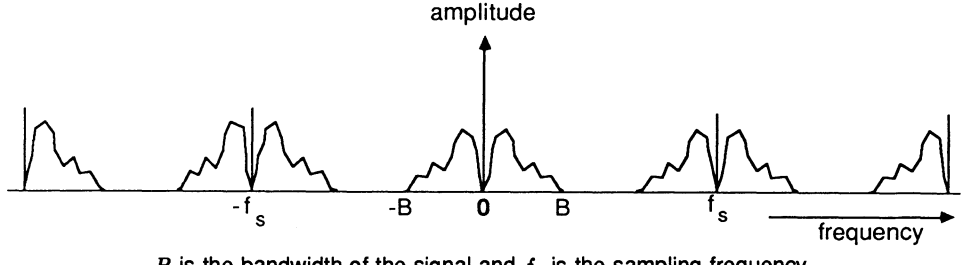
Where the x and y sequences are the input and the output of the filter and a_m's and b_k's are the coefficients of the filter.

As mentioned earlier the notation z^{-1} is often used to denote a delay equal to one sampling period. In the theory of the dicrete-time signals, the concept of z has been developed further and is referred to as the z-transform. This is a discrete-time version of the well known Laplace transform (sometimes referred to as the s-transform) which is mainly used for dealing with continuous signals. In the s-domain a delay of T seconds corresponds to e^{-sT}. Therefore the two variables s and z are related by:

$$z^{-1} = e^{-sT} \tag{6}$$

where T is the sampling period.

In the s-domain the spectrum of a signal with a bandwidth B and sampled at a frequency f_s, is periodic with a period equal to f_s. This is depicted in figure 8.2. This periodicity in the spectrum of a sampled signal is the basic reason behind the Nyquist criterion which requires a minimum sampling frequency of twice the signal bandwidth (i.e. $f_{s_{min}} = 2 \times B$), in order to avoid aliasing effects.



B is the bandwidth of the signal and f_s is the sampling frequency.

Figure 8.2 Spectrum of a sampled signal

Equation (6) allows a mapping between the two domains. Part of the imaginary axis between $-\frac{f_s}{2}$ to $+\frac{f_s}{2}$, in the s-plane, is mapped into a unit circle in the z-domain as shown in figure 8.3. The fact that the imaginary axis in the s-plane is mapped onto a circle is a consequence of the periodic nature of the spectrum. As shown in figure 8.3, the left-hand half of the s-plane (between $-\frac{f_s}{2}$ and $+\frac{f_s}{2}$) is mapped onto the inside of the unit circle, while the right-hand half is mapped onto the outside of the circle.

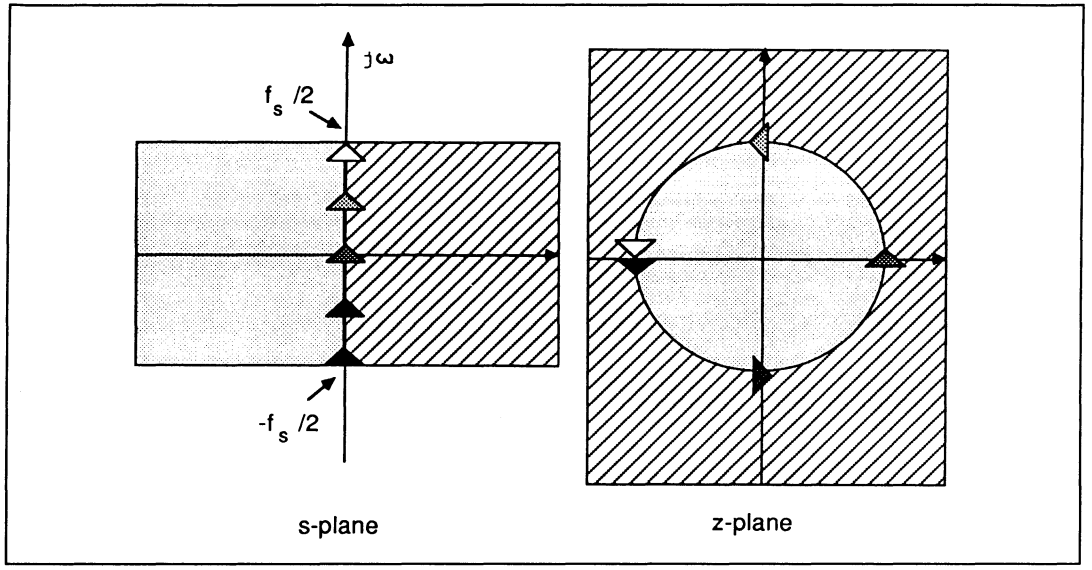


Figure 8.3 Relationship between the s-domain and the z-domain

As in the analogue design (s-domain) where a pole in the wrong place, i.e. in the right-half plane, indicates instability, in the case of discrete-time signals (z-domain) a pole outside the unit circle causes instabilities. In both cases zeroes can be anywhere.

Using the z-transform notation, the general linear equation (5) can be expressed as:

$$Y(z)(1 + \sum_{m=1}^{M} a_m z^{-m}) = X(z) \sum_{k=o}^{N} b_k z^{-k} \tag{7}$$

Where $X(z)$ and $Y(z)$ are the z-transforms of the input and output waveforms. The discrete-time (or digital) transfer function of the general filter is thus given by:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{N} b_k z^{-k}}{1 + \sum_{m=1}^{M} a_m z^{-m}} \tag{8}$$

In terms of realization, digital filters are classified into nonrecursive and recursive types. The nonrecursive structure contains only feed-forward paths and as such all the a_m terms (equation (8)) are zero. This means that for the nonrecursive filters the output is a sum of linearly weighted present and a number of past samples of the input signal as shown in figure 8.4. Referring to equation (8), for the nonrecursive filters the transfer function has only zeroes and as such is always stable.

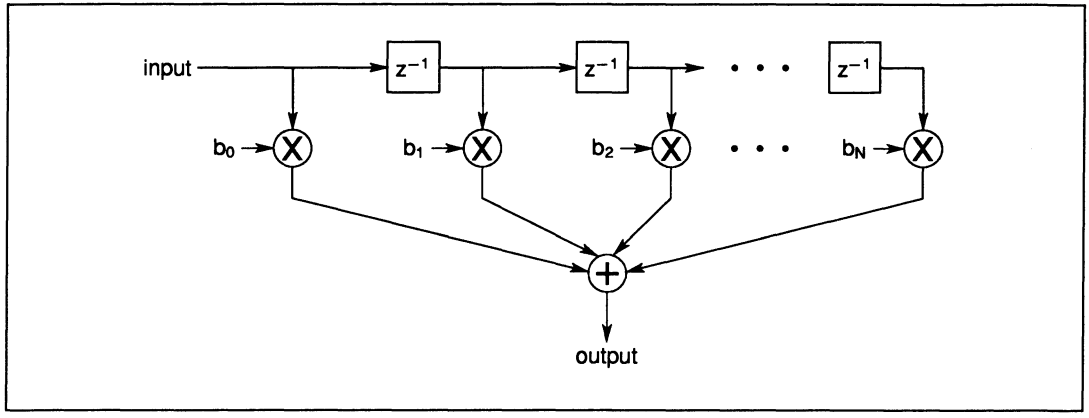


Figure 8.4 Nonrecursive digital filter structure

In the recursive filters on the other hand some or all of the a_m terms are non-zero resulting in the presence of both poles and zeroes in the transfer function. Figure 8.5 shows the general recursive filter structure. Figure 8.6 shows an alternative structure for the same transfer function with a reduced number of delay stages.

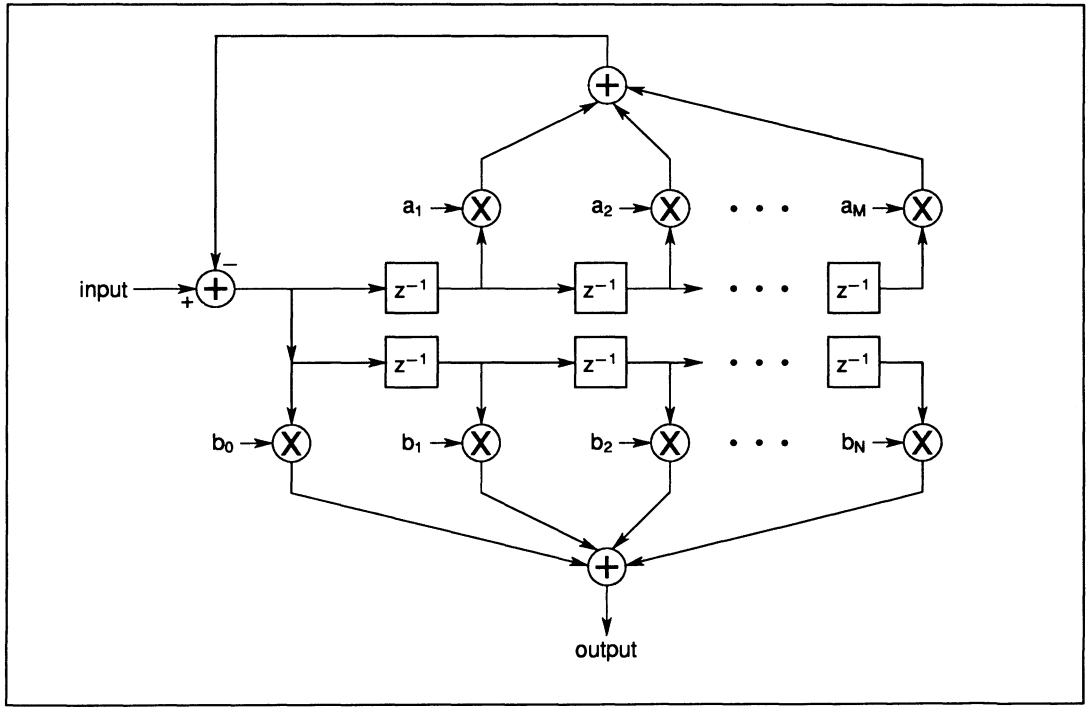


Figure 8.5 Recursive (IIR) digital filter structure

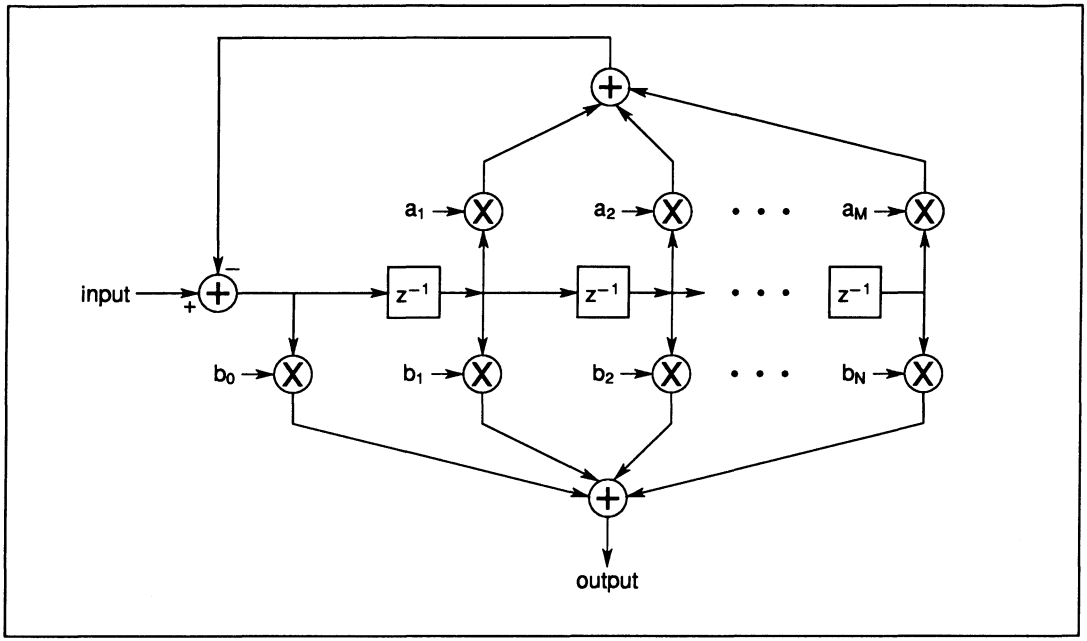


Figure 8.6 Alternative recursive (IIR) digital filter structure with reduced number of delay stages

Digital filters are also classified in terms of their impulse responses. In this classification those filters with a finite duration impulse response are referred to as FIR filters and those with an infinite duration impulse response are called IIR filters. The simplest FIR filter realization is in the nonrecursive form. For example in figure 8.4, if a unit impulse is clocked through the filter, the sequence,

$$b_0, b_1, b_2, ...b_N, 0, 0, 0, 0, 0, ...0, 0, 0 \tag{9}$$

will be output. Notice that the response consists of a sequence of samples corresponding to the filter coefficients followed by zeroes, i.e. the nonrecursive structure is an FIR filter. On the other hand the impulse response of the recursive structure (figures 8.5 & 8.6), because of the feedback paths, is infinite in duration, making the configuration an IIR filter.

8.4 Digital filter design

Digital filter design methods can be divided into two categories:

(a) Design techniques suitable for FIR filters.

(b) Design techniques suitable for IIR filters.

In both cases the requirement is simply the choice of filter coefficients in such a way that the specification for the required transfer function is met. The IMS A100 can be used to implement high performance FIR filters directly. It can also be used to implement IIR filters, although the general problems associated with IIR filter design are then introduced. In this section a brief comparison between FIR and IIR filters is given and some of their associated design techniques are summarized. Where necessary the IMS A100 implementation issues are also discussed.

8.4.1 Comparison between FIR and IIR filters

FIR filters, because of their finite-impulse response have no counterparts among analogue filters and as such can implement transfer functions which cannot be realized in the analogue world. One such property

is the excellent linear-phase characteristic which can easily be realized with FIR filters. Since a linear-phase response corresponds to only a fixed delay, attention can be focussed on approximating the desired magnitude response without concern for the phase. The design techniques for FIR filters are generally simpler than those for IIR filters, and as there are no feedback paths in an FIR filter, the stability of the filter is guaranteed. Also FIR filters have been employed, and algorithms have been developed, for adaptive processing while the use of IIR filters in these types of systems is not common.

IIR filters on the other hand have infinite impulse responses and thus their design can be closely related to analogue filter design. IIR filters in general require fewer stages compared to FIR filters but their stability is not unconditional and great care should be taken to insure stability. Furthermore IIR filters do not generally result in linear-phase characteristics which is important in many applications.

8.4.2 Basic design parameters

In digital filter design, for the reason of convenience, the frequency axis is usually normalised with respect to the sampling frequency f_s. For example for a filter with an actual pass-band cut-off frequency of $20kHz$, a stop-band cut-off frequency of $30kHz$ and a sampling frequency of $100kHz$ we have:

The normalised pass-band cut-off frequency $f_{pb} = \frac{20}{100} = 0.2$

The normalised stop-band cut-off frequency $f_{sb} = \frac{30}{100} = 0.3$

As shown in figure 8.7 the useful frequency axis (normalised) extends from 0.0 to 0.5, because the Nyquist sampling theorem requires a signal to be sampled at more than twice its highest frequency. This means that the ratio of the frequency of any component in the signal to the sampling frequency must always be less than 0.5.

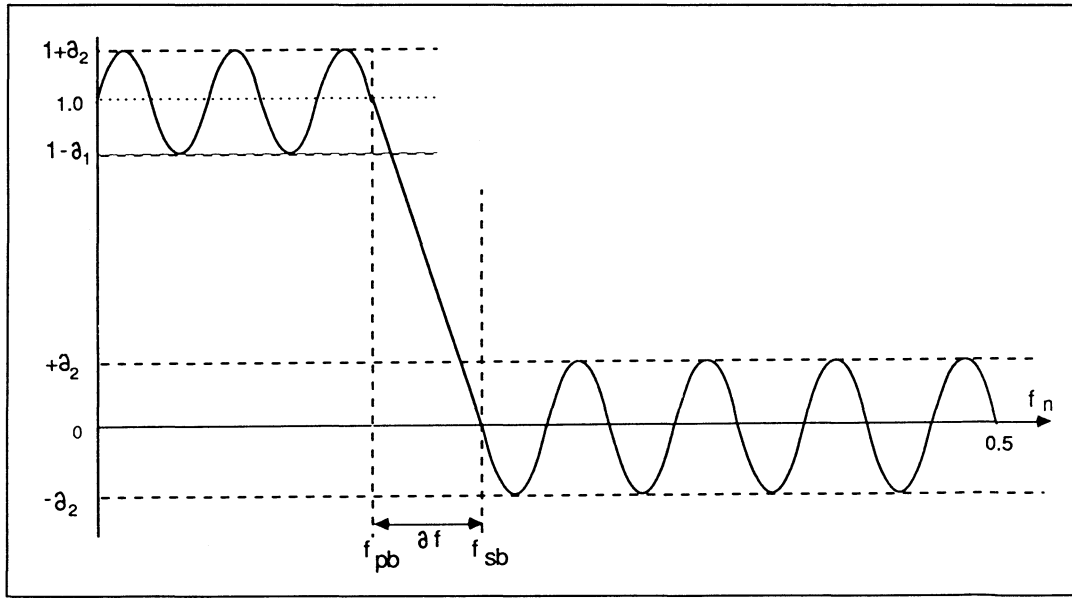


Figure 8.7 Specification parameters for a low pass filter.
Similar parameters exist for high pass and band pass filters.

Referring to figure 8.7, the pass-band and the stop-band ripples are usually expressed in dBs i.e:

pass-band ripple $(dB) = 20\log_{10}(1 + \delta_1)$

stop-band ripple $(dB) = -20\log_{10}(\delta_2)$.

The parameters f_{pb}, f_{sb}, δ_1, δ_2 and the sampling frequency define the basic specification of a filter prior to its design.

8.4.3 Design techniques suitable for FIR filters

As mentioned earlier one of the major advantages of FIR filters is the ease with which linear-phase behaviour can be obtained from these types of filters. Before summarizing the design techniques for FIR filters let us briefly consider the necessary conditions for linear-phase behaviour. It can readily be shown that in order to obtain an FIR filter with a linear-phase characteristic, the following condition has to be met (references 1 & 2):

$$\begin{aligned} h(i) &= \pm h(N-i) \quad for \quad 0 \leq i \leq N \\ &= 0 \quad otherwise \end{aligned} \tag{10}$$

This condition requires that the the impulse response of the FIR filter, $h(i)$, to have either positive or negative symmetry.

In the case of positive symmetry the frequency response will be of the form

$$H(e^{j\omega T}) = A(\omega T)e^{-j\omega TN/2} \tag{11a}$$

where $A(\omega T)$ is a real function of ω. Notice that the phase is a linear function of frequency. These types of filters are appropriate for frequency selective filters.

In the case of negative symmetry the filter transfer function will have the following form:

$$H(e^{j\omega T}) = jB(\omega T)e^{-j\omega TN/2} \tag{11b}$$

Again $B(\omega T)$ is a real function of ω. Note that the phase is again linear with frequency, but we also have a j term which indicates an extra phase shift of $\frac{\pi}{2}$. These types of frequency responses are required to realise approximate differentiators and Hilbert transforms which implement a $\frac{\pi}{2}$ phase shift over a specified frequency range.

There are essentially three well-established classes of design methods for (linear phase) FIR filters which are:

(i) window method

(ii) frequency sampling

(iii) optimal design (Remez Exchange Algorithm)

Each one of these techniques has its own merits and the choice of which would depend on the application requirements and the design time involved.

Window method

This is the most straight-forward approach to the design of FIR filters. In this method having defined an ideal frequency-response function, the corresponding ideal impulse response is determined by evaluating the inverse Fourier transform of the ideal frequency response. In the selection of the ideal frequency response, the linear phase condition may or may not be applied depending on the application.

As mentioned earlier because digital filters deal with signals sampled at a frequency f_s,it therefore follows that this frequency response is periodic in frequency with a period equal to f_s (Nyquist theorm). It is therefore possible to relate the impulse response and the frequency response of a digital filter via the following Fourier pairs:

$$H(\omega) = \sum_{n=-\infty}^{+\infty} h(n)e^{-jn\omega T} \tag{12}$$

$$h(n) = \frac{1}{\omega_s}\int_{\frac{-\omega_s}{2}}^{\frac{+\omega_s}{2}} H(\omega)e^{jn\omega T}d\omega \tag{13}$$

where ω_s, is the sampling frequency in radians/s and T is the sampling period. Having defined an ideal frequency response, $H(\omega)$, equation (13) can be used to obtain the impulse response, $h(n)$, of the filter. As an example consider the ideal low-pass frequency response characteristics with a cut-off frequency ω_c as shown in figure 8.8a. Using equation (13), and equating $H(\omega)$ to 1.0 for $-\omega_c \leq \omega \leq +\omega_c$ and to zero elsewhere, we can calculate the impulse response $h(n)$ which is given by:

$$h(n) = \frac{\omega_c T}{\pi}\frac{\sin(n\omega_c T)}{(n\omega_c T)} \tag{14}$$

where $-\infty < n < +\infty$. This impulse response is shown in figure 8.8b. There are two problems associated with this impulse response obtained in this way:

(i) The filter impulse response is infinite in duration and as such an FIR filter of infinite length is required (remember as discussed earlier for FIR filter the impulse response sample values are effectively the filter coefficients).

(ii) The filter is unrealizable since the impulse response begins at $-\infty$, indicating that no finite amount of delay can make the impulse response realizable.

One way to obtain an FIR filter which approximates the required frequency response is to truncate the infinite impulse response at $n = \pm\frac{N}{2}$, (see figure 8.8c), and shift the impulse response to the right to avoid negative time (figure 8.8d). This would result in a realizable FIR filter with $N + 1$ coefficients which are equal to the impulse response samples.

The problem with this direct truncation of the impulse response is that it results in a fixed amount of overshoot (approximately 9%) before and after the discontinuity in the frequency response. In the literature this problem is referred to as the Gibbs phenomenon. For this reason, direct truncation is not often a reasonable way of designing FIR filters.

The frequency response of a truncated time series can be improved considerably by using a window function, $w(n)$, which modifies the impulse response to $w(n) \times h(n)$. In the previous example the window was simply a rectangular window. Figure 8.9 shows the application of a different window function to the example of the ideal low-pass filter. Figure 8.9a shows the ideal infinite duration impulse response. Figure 8.9b shows the window function and figure 8.9c shows the impulse response after the application of the window function. Figure 8.9d shows the shifted impulse response which avoids unrealizable negative delays. The filter coefficients (b_k's) correspond to the sample values of this modified impulse response which is now finite and realizable. Several window functions have been suggested in the literature some of which are:

(i) Hamming window

(ii) Hanning window

(iii) Kaiser window

(iv) Dolph-Chebyshev window

(v) Blackman window

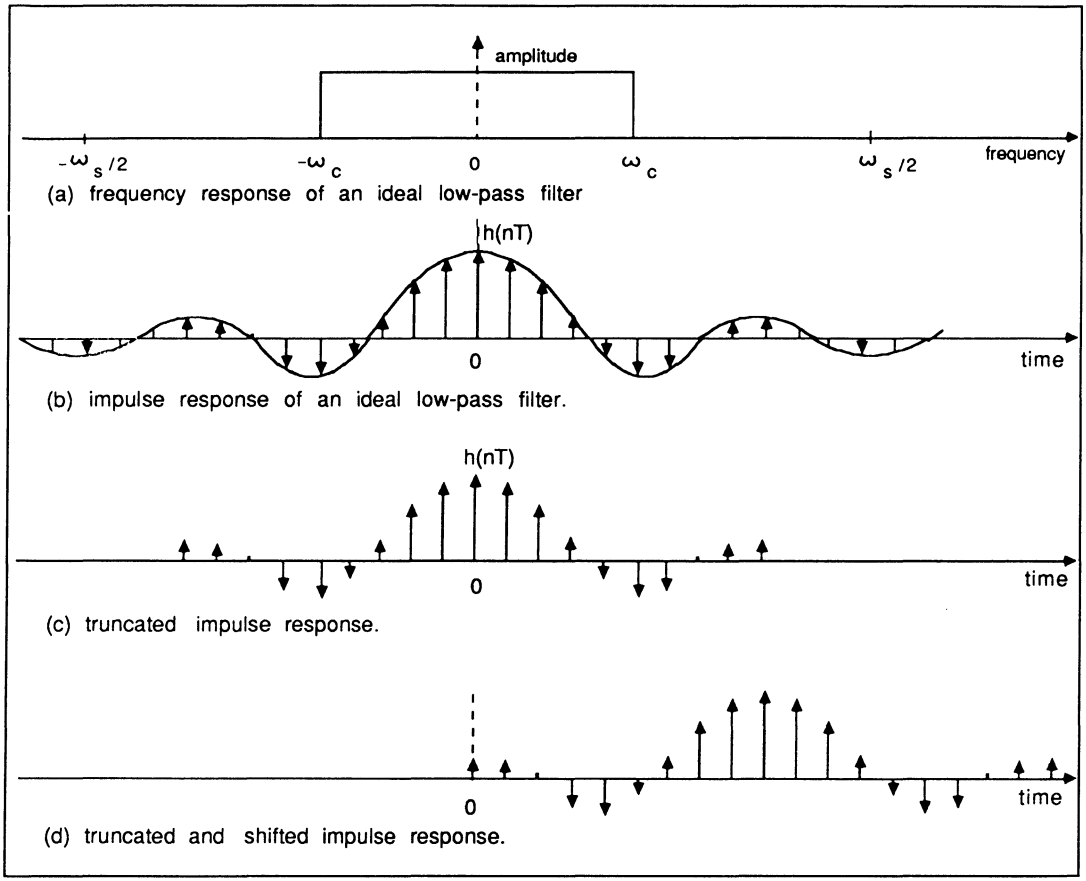


Figure 8.8

The generalized Hamming window function is given by:

$$\begin{aligned} w_H(n) &= \alpha + (1-\alpha)\cos(\tfrac{2\pi n}{N}) \quad for \quad -(\tfrac{N-1}{2}) \le n \le -(\tfrac{N-1}{2}) \\ &= 0 \qquad\qquad otherwise \end{aligned} \tag{15}$$

where $0 \le \alpha \le 1$. If $\alpha = 0.54$ the window is called a Hamming window, and if $\alpha = 0.50$ it is called a Hanning window.

For the Hamming window the main lobe of the frequency response is twice the width of that of the simple rectangular window. The amplitudes of the ripples of the Hamming window frequency response are considerably smaller than those of the rectangular window. For the rectangular window the peak side lobe (in the stop band) is only 14dB below the main-lobe (pass-band) peak. For the Hamming window the peak side lobe ripple is about 40dB below the pass band peak. Furthermore for the Hamming window 99.96% of the spectral energy is in the main-lobe peak.

Another family of windows are those proposed by Kaiser:

$$\begin{aligned} W_K(n) &= \frac{I_0(\beta\sqrt{1-[2n(N-1)]^2})}{I_0(\beta)} \quad for \quad -(\tfrac{N-1}{2}) \le n \le -(\tfrac{N-1}{2}) \\ &= 0 \qquad\qquad otherwise \end{aligned} \tag{16}$$

Where I_0 is the modified Bessel function of the first kind. The parameters β is used to specify the main-lobe width and the side-lobe level of the frequency response. β is usually specified to have a value between 4

and 9. This range of β corresponds to a range of side-lobe peaks of 3.1% to 0.047% of the main-lobe peak. The Kaiser window is essentially an optimum window in the sense that it is a finite duration sequence that has the minimum spectral energy beyond some specified frequency. For the Kaiser window the width of main lobe is almost three times that of the rectangular window, while the peak side lobe in the stop band is 57dB below the pass-band peak. The side-lobe ripple envelope decays to 94dB below the pass-band peak at half the sampling frequency.

The Dolph-Chebyshev window function has the minimum width of the main lobe in its frequency response for a given peak value of side-lobe ripple. For this window the stop-band ripples all have the same amptitude. Recursive equations exist which allow this window function to be evaluated.

References 1 and 2 contain further information on this design method and the associated window functions.

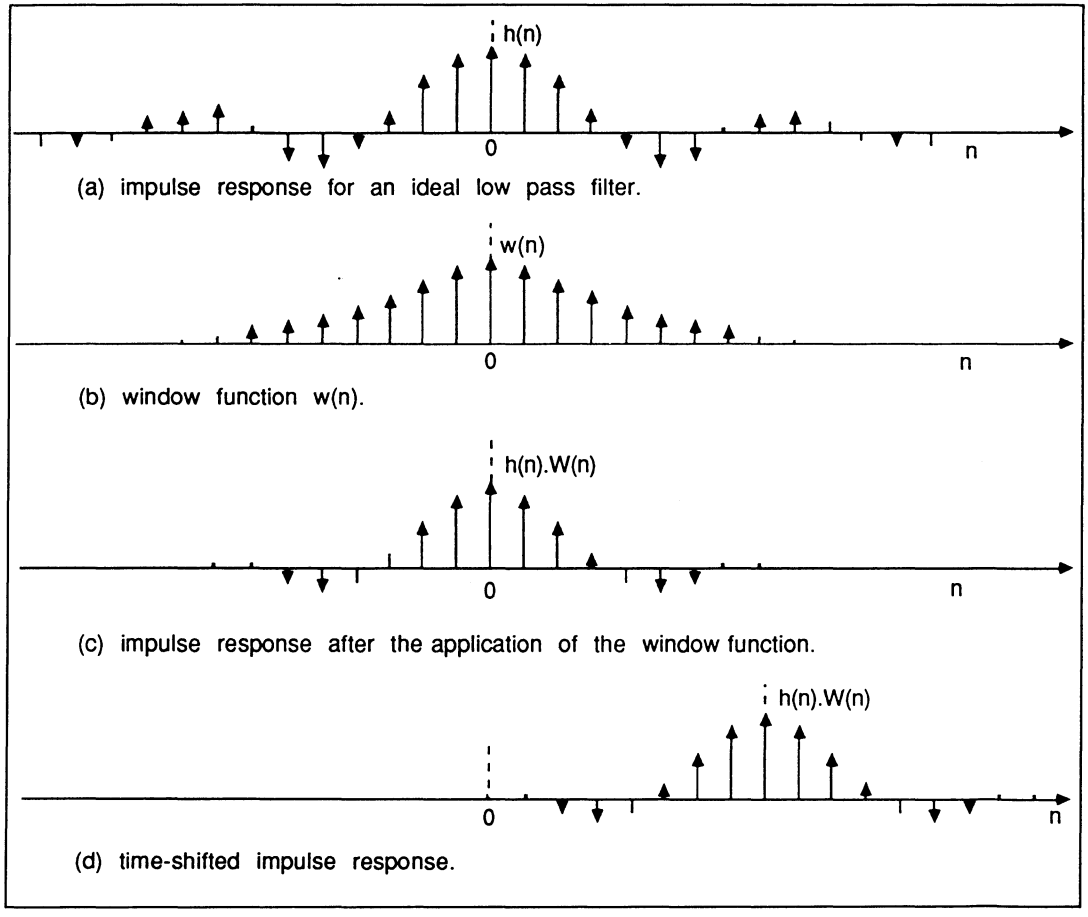


Figure 8.9

Frequency sampling technique

This technique is less common than the other two design methods, however for the sake of completeness it is briefly mentioned here.

The basic idea behind this technique is that the given (desired) frequency response is approximated by sampling it at N equally-spaced points along the frequency axis between 0 and f_s (corresponding to N samples on the unit circle in the z-plane). An N-point inverse DFT is then performed on these N frequency

samples to give N samples of the impulse response $h(n)$ which corresponds to the filter coefficients. The z-transform of the filter impulse response is then given by

$$H(z) = \sum_{k=0}^{N-1} h(n)z^{-n}$$

Substituting $e^{j\omega T}$ for z, the resulting frequency response of the filter may be evaluated which would be an approximation of the desired frequency response. The approximation error would be exactly zero at points where the desired frequency response was sampled and would be finite between them. This process is depicted in figure 8.10.

To reduce these approximation errors a number of frequency samples (particularly those in the transition band between band-pass and band-stop regions, i.e. points T_1, T_2, T_3 and T_4 in figure 8.10) can be made unconstrained variables. The values of these unconstrained variables are then optimised using computer optimisation techniques involving linear-programming methods. This involves the solution of a set of linear unequalities in the unconstrained frequency samples. In this way, by adjusting the frequency sample values at T_1, T_2, T_3 and T_4, considerable ripple cancellation, both in the pass-band and stop-band, can be achieved resulting in very good filter characteristics. The detail of these techniques are beyond the objectives of this application note, however interested readers can refer to reference 1 for further information.

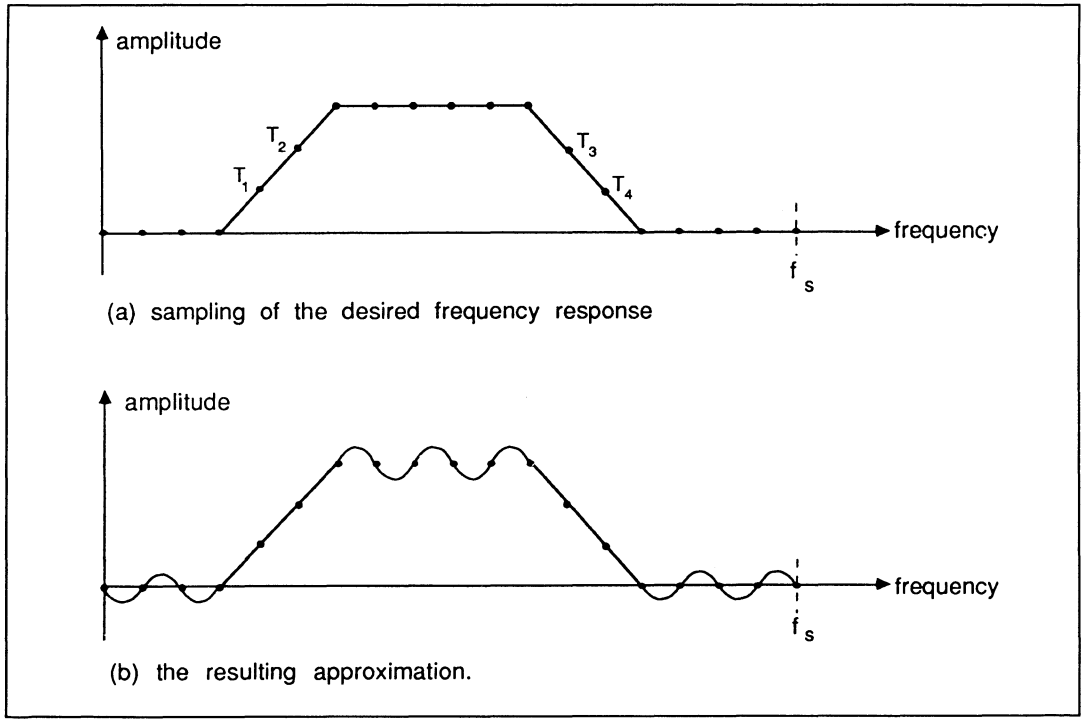


Figure 8.10

Optimal filter design – (Remez exchange algorithm)

In the frequency sampling technique, discussed in the previous section, some degree of improvement in the filter characteristics is obtained by allowing only a few of the frequency samples to be adjusted via a linear-programming technique.

An even more powerful technique which results in truly optimal filters, in the sense of having the sharpest transition between pass bands and stop bands (for a given filter length and a given approximation error) has been formulated based on the so-called Chebyshev approximations. Computer optimisation techniques based on linear programming have been developed (references 3, 4, 5 & 6) which allowed engineers to design optimal FIR filters with a minimum amount of knowledge about the actual optimisation algorithm. These iterative algorithms are based upon the principles of the Remez exchange algorithm. This algorithm yields optimal filters that satisfy the so-called minimax error criterion (reference 1), where for a given number of coefficients, the filter minimizes the maximum ripple amplitude in the pass band. The implications of this optimal design are:

(a) The Remez exchange algorithm results in an FIR filter with the smallest number of coefficients satisfying the required specification.

(b) The pass-band ripple components all have the same magnitude and need not be equal to the stop-band ripples, but their ratio must be specified.

The input to the Remez exchange program usually includes the type of filter (frequency selective filters, differentiators and Hilbert transform filters), normalised stop-band and pass-band edges, the desired minimum stop-band attenuations, the maximum pass-band ripple and the ratio of the pass-band to stop-band ripples.

The output of the program include estimated filter length, and impulse response (filter coefficients). It also includes first pass computed values for design parameters, such as pass-band ripple, stop-band attenuation. If the computed values do not satisfy the design requirements, the filter length may be increased slightly and the program is run again. Interested readers can find copies of this program in references 1, 2 & 4.

Implementing FIR filters with the IMS A100

The coefficient word size in the IMS A100 can be programmed to be 4, 8, 12 or 16 bits. Having calculated the filter coefficients using one of the techniques described earlier, these coefficients are then expressed in a 4, 8, 12 or 16-bit format, depending on the required accuracy. The filter can then be implemented by simply loading these coefficients into the IMS A100 coefficient memories. If the number of coefficients (filter stages) required is less than or equal to 32, a single IMS A100 would be sufficient, any unused coefficient locations being set to zero. If however, more than 32 coefficients are involved a number of IMS A100 devices can be cascaded to obtain the required filter order. Alternatively it is possible to partition a long FIR transfer function into product terms where each term has an order equal or less than 32. Then, using a single IMS A100, the data can be recirculated through the same device with different coefficients (associated with each term in the transfer function) for each circulation. In this way a very long FIR filter can be implemented with a single device at the expense of a reduction in the data rate.

The IMS A100 can be cascaded very easily, without the need for any external components, to obtain high order filters with a high degree of accuracy. The device has a versatile architecture which allows it to be used in various system configurations. The coefficients can be programmed via a standard memory interface, while the input and output data can be communicated either via the memory interface or dedicated I/O ports. Figure 8.11 shows some of the possible system configurations for the IMS A100. In this diagram the interface between the host and the IMS A100 consists of data and address buses of the processor plus standard memory-type control signals such as R/W, CE and CS. In figure 8.11a the host processor controls the filter coefficients, while the actual data to be processed is supplied directly from an A/D to IMS A100. In this example the filtered output is fed directly to a D/A. Using the IMS A100 and a host processor it is possible to supply the input data to the device and also to collect the filtered samples via the memory interface. This allows system configuration such as those shown in figures 8.11b&c. In figure 8.11b the host processor receives the input data from a peripheral such as an A/D and writes it (may be after some preprocessing) into the data-input register (DIR) of the IMS A100. The filtered output sample is also collected by the host via the memory interface and output (possibly after post processing) to a peripheral such as a D/A. Figure 8.11c shows a configuration where the IMS A100 is used purely as a signal processing accelerator to the host. Numerous other configurations are possible including integrating an IMS A100 into existing microprogrammed systems in order to improve the overall system performance.

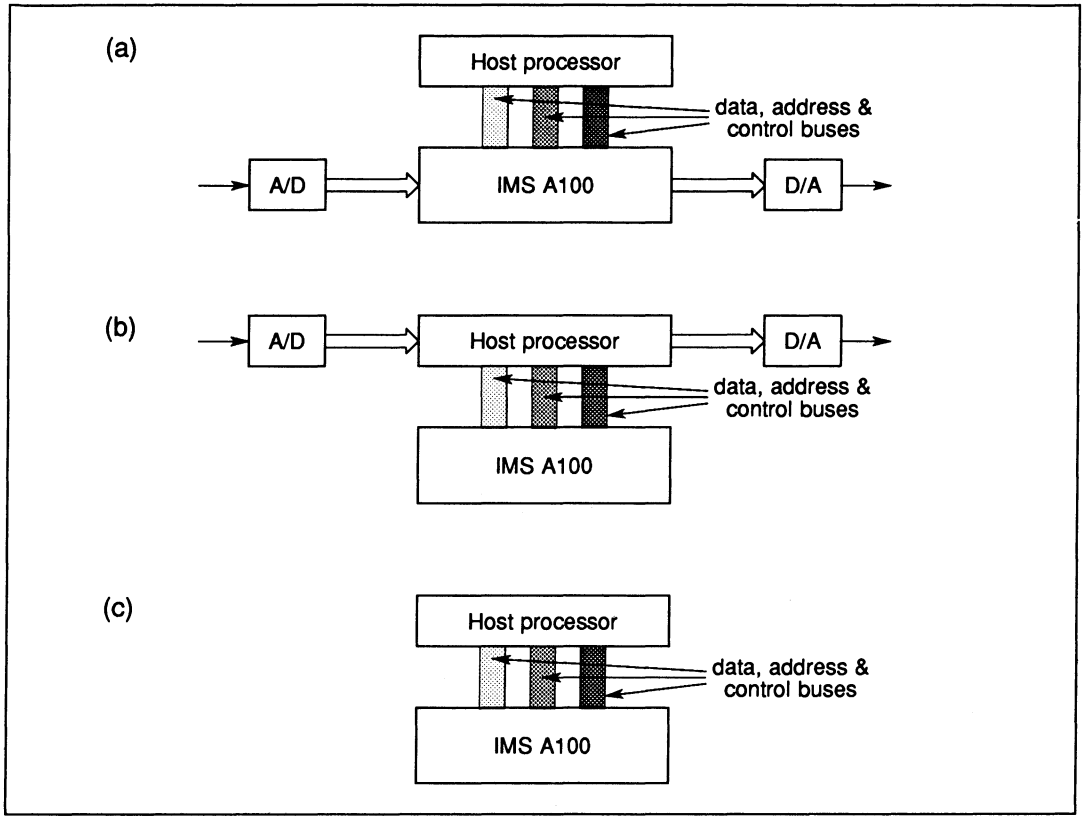


Figure 8.11 Possible system configurations using the IMS A100 in digital filtering applications

As mentioned earlier large numbers of the IMS A100 devices can be cascaded to construct FIR filters of a high order. The cascading does not involve any external components and is simply a matter of connecting the output of the previous device to the cascade input of the next chip and joining the data input ports together (if they are being used rather than the memory interface). In normal operation the cascade input of the first device should be grounded. Figure 8.12a shows this cascading arrangement for two IMS A100 devices and figure 8.12b depicts the block diagram of a system consisting of a host processor and two cascaded devices. In the latter case the data-input register (DIR) of both devices should be associated with the same address in the host's address space; and one of the devices should be selected as a master to generate the GO signal (see product data sheet for further detail).

Another important feature of the IMS A100 is a selector that is incorporated after the multiply-accumulator array. As discussed in the data sheet, the 32 multiply and accumulation in the array are performed to a precision of 36 bits which ensures that no intermediate overflows occur. The output selector can then be used to select and round a 24-bit word from this 36 bit result. This selection and rounding can be programmed to start from bits 7, 11, 15 or 20 and the selected word is sign extended if needed. One particularly useful selection is available when the input data and coefficients are in the form of 16 bit two's complement numbers normalised to between +1 and −1. In this case, if the selection is taken to start from bit 15, the output will have the same format as the input data (i.e. normalised to between +1 and −1).

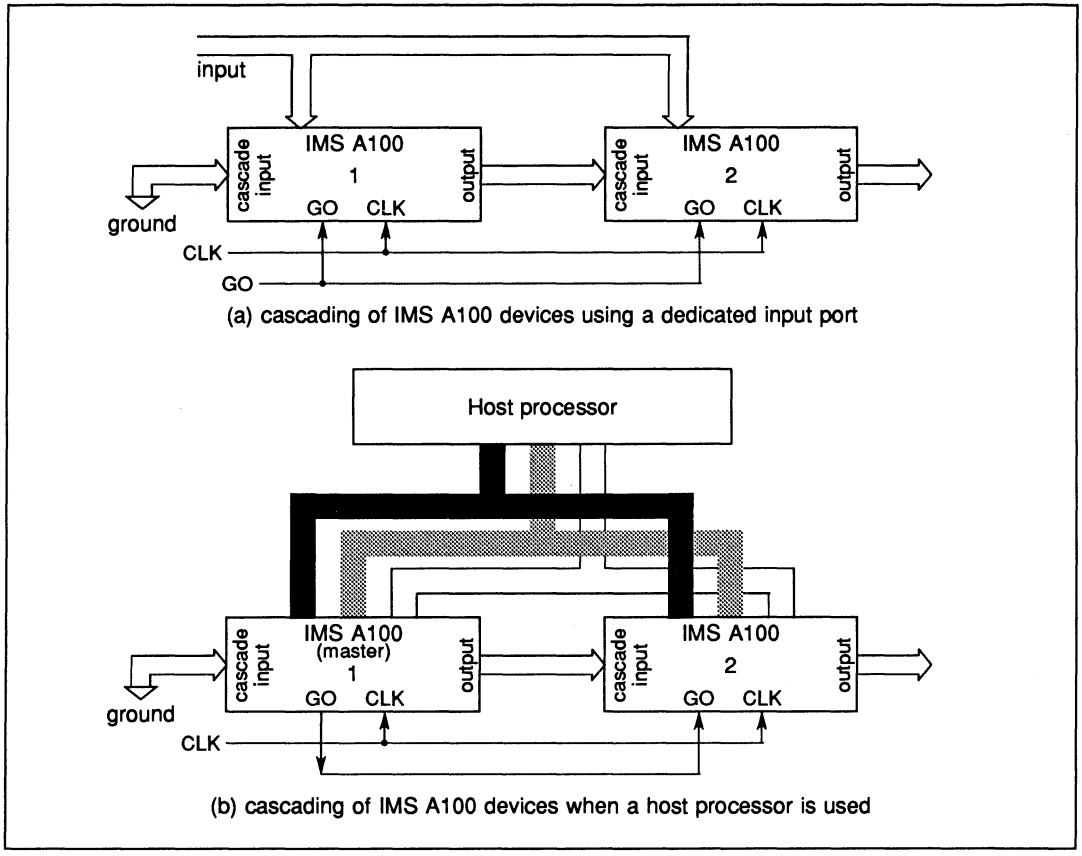


Figure 8.12 Cascading IMS A100 devices

8.4.4 The IMS A100 and IIR filters

Although the IMS A100 is designed primarily for FIR type filter implementations, it can also be used in realizing IIR filters. Referring to figure 8.5 it can be seen that two IMS A100 devices can be used to implement an IIR filter of order 32 or less in the direct form. One chip performing the calculation in the feed-forward path while the other does the feed-back path. Note that in figure 8.5 the output of the feed-back filter has to be combined with the input sequence in a subtractor and fed into the input of the second chip. This subtraction can be performed either by the host processor controlling the two IMS A100s or by an external adder.

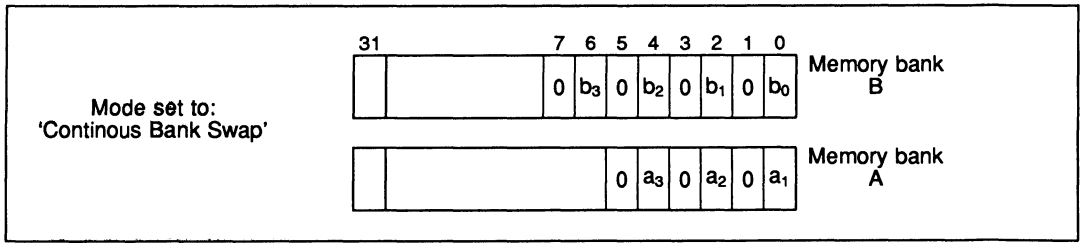


Figure 8.13 Coefficient memory allocation for IIR filter implementation

A simpler and more elegant technique to implement IIR filters using IMS A100 is to make use of the continuous bank swap feature on the IMS A100 coefficient memories. This allows a single IMS A100 to be sufficient for the implementation of IIR filters whose order is less than or equal to 16. (Before describing how this can be achieved it is worth noting that IIR filters generally require considerably fewer stages than their FIR counterparts, and as such a 16th order IIR filter implementable on a single IMS A100 can be considered as having quite a high order). Figure 8.13 shows the coefficient memory allocations in this approach, where a's and b's are the feedback and feedforward coefficients of the IIR filter respectively (see figures 8.5 & 8.6) and are loaded by the host processor. Note that in figure 8.13 alternate coefficients are set to zero in the two memory banks. The chip is also set to the continuous bank swap mode so that in one cycle the feedback coefficients (a's) and in the next cycle the feedforward coefficients (b's) are used in the calculation. It will be shown in the following paragraphs that if the difference between data samples and alternate output samples are written to the data input register of the IMS A100, then the remaining output samples would correspond to the correct filter output. The sequence of operations is as follows:

The host starts the filter operation by writing the first data value, x_0, to the data input register of the IMS A100. Remembering that the coefficient allocation is as shown in figure 8.13, the first output of the device would be a_1x_0. Referring to figure 8.6, it can readily be seen that this is indeed the feed back contribution needed to be subtracted from the next data sample x_1. The host reads this value (a_1x_0) from the data output registers (DOH and/or DOL) and stores it and then writes x_0, for a second time, to the IMS A100 input. This time the coefficient memory banks would have been swapped and the output would correspond to b_0x_0 which can readily be confirmed to be the first correct filter output (see figure 8.6). The host then reads this result as the first valid sample of the filtered output.

Next the host subtracts the feedback factor, read in earlier (a_0x_0), from the second data sample x_1, and writes the difference to the input register of the IMS A100. Remembering that the memory banks are automatically swapped every cycle, the corresponding ouput of the IMS A100 will be:

$$a_2x_0 + a_1(x_1 - a_1x_0)$$

Referring to figure 8.6 you should be able to confirm that this value corresponds to the feedback contribution needed for the third input sample. The host reads this value and stores it and as before writes the input value ($x_1 - a_1x_0$) to the IMS A100 input register for a second time. This will yield the second valid filtered sample i.e:

$$b_1x_0 + b_0(x_1 - a_1x_0) \tag{17}$$

The process is then continued in the same manner. The output of the IMS A100 will alternate between the feedback contribution and the filtered output samples. It should be emphasized that although the host is performing a single subtraction for every output value, it is the IMS A100 device which is performing the bulk of the processing. Having established how the IMS A100 can be configured to implement IIR filters, the next section deals with some of the design techniques that are used for determining the IIR filter coefficients.

8.4.5 Summary of the IIR filter design techniques

The problem of designing recursive filters is one of determining the feedforward and feedback coefficients (i.e. b_n's and a_m's in equation (8). The design techniques for IIR filters can be categorised into two basic groups:

(i) Indirect approaches.

(ii) Direct approaches.

Indirect approaches for the design of IIR filters

As mentioned earlier digital recursive filters are closely related to conventional analogue filters. In the indirect method this similarity is exploited and the digital filter coefficients are determined from a suitable analogue filter, using some form of transformation technique. In other words the indirect approach uses the wealth of knowledge already available on analogue filters (such as Butterworth, Chebyshev and Elliptic filters) and develops a corresponding recursive digital filter. This method involves the following two steps:

(1) the determination of a suitable analogue filter transfer function $H(s)$

(2) transformation and digitization of this analogue filter

Some of the most popular design techniques falling into the indirect category are:

(a) Impulse-invariant transformation.

(b) Bilinear z-transform.

(c) Matched z-transform.

These three techniques can be employed to derive recursive digital filters from conventional analogue filter structures. Before discussing these three techniques the basic characteristics of the common analogue filters, from which IIR filters are derived, will be briefly reviewed. The starting point in the indirect IIR design techniques is often one of the following analogue filter types.

1 **Butterworth filters:** These filters are characterised by the property that their magnitude characteristic is maximally flat at the origin of the s-plane. Butterworth filters are specified by their magnitude-square functions i.e:

$$|H(s)|^2 = \frac{1}{1+(\frac{s}{s_c})^{2n}} \tag{18}$$

The pole locations in the s-plane are equally spaced around a circle of radius ω_c ($s_c = j\omega_c$). These filters have a monotonically decreasing amplitude function with a roll-off of approximately $6n$ dB/decade. Figure 8.14 shows the overall amplitude response of this type of filter.

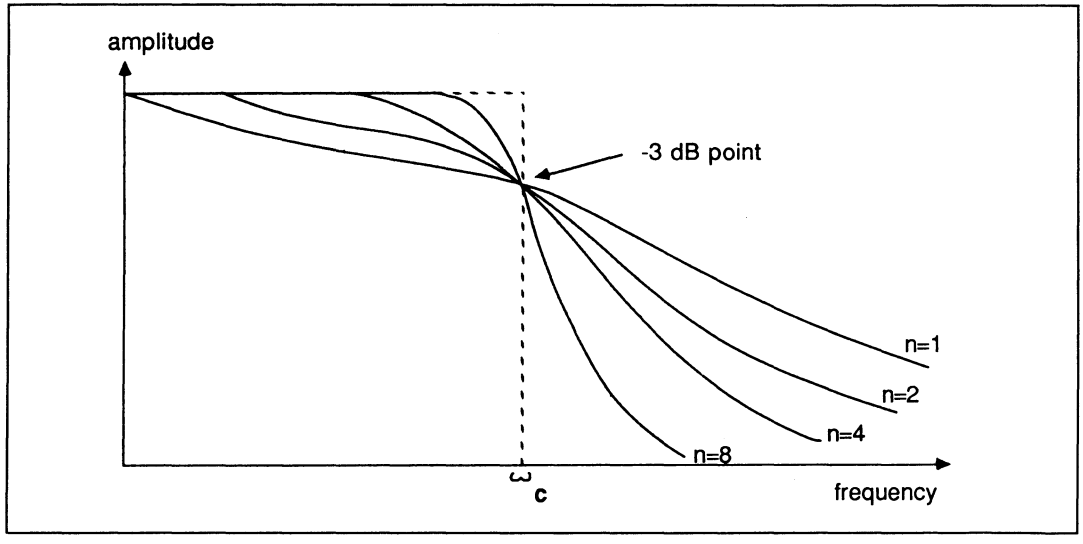


Figure 8.14 Frequency response of the Butterworth filter

2 **Chebyshev filters:** In these types of filters the peak magnitude of the approximation error is minimized over a prescribed band of frequencies and is also equiripple over the band. Chebyshev filters are specified by the magnitude-square function:

$$|H(s)|^2 = \frac{1}{1+\epsilon^2 C_N^2(\frac{s}{s_c})} \tag{19}$$

where $C_N(s)$ is a Chebyshev polynomial of order N. The parameter ϵ is used to specify a magnitude function with equal ripple in the pass band and monotonic decay in the stop band. Figure 8.15 shows the magnitude-square transfer function for the Chebyshev filter (type I) where the amplitude of the ripple is given by:

$$\delta = 1 - \frac{1}{\sqrt{1+\epsilon^2}} \tag{20}$$

The poles of the Chebyshev filter lie on an ellipse determined from the parameters ϵ, N and s_c. Chebyshev filters of type II on the other hand have monotonic behaviour in the pass band (maximally flat around ω_0) and exhibit equiripple behaviour in the stop band. For further details refer to references 1 & 2.

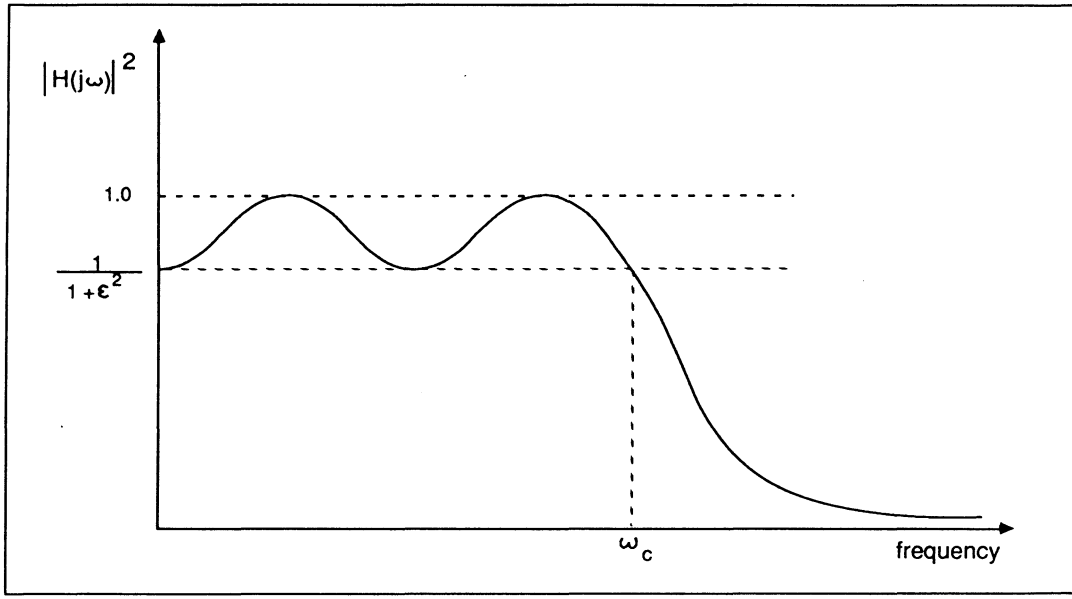


Figure 8.15 Frequency response of the Chebyshev filter (type I)

3 **Elliptic filters:** These filters exhibit a magnitude response that is equiripple in both the pass band and the stop band. These filters are optimum in the sense that for a given order and for a given ripple specification the transition band is the shortest possible. Elliptic filters are specified by the magnitude-square transfer function:

$$|H(j\omega)|^2 = \frac{1}{1+\epsilon^2 C_N^2(\omega)} \tag{21}$$

Where $C_N(\omega)$ is a rational Chebyshev function involving elliptical functions. Figure 8.16 illustrates the magnitude-square response for an elliptic filter.

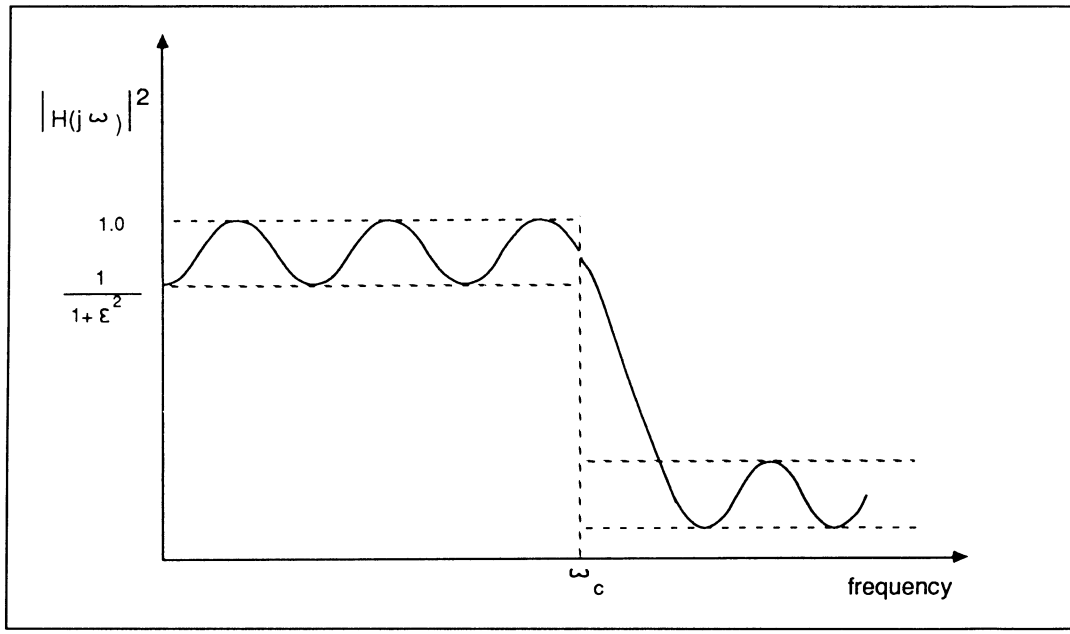


Figure 8.16 Frequency response for an elliptic filter

It is not possible to discuss all analogue filter types in this applications note as the main objective here is to summarize the basic design technique which allow transformation of analogue filters to digital realizations. Interested readers can refer to numerous books available on analogue filters.

Having decided the type and the specification of the analogue filter that satisfies the requirement, the next step in the indirect design method is to use one of the three following techniques to obtain the corresponding digital filter.

Impulse Invariant Transformation

One of the most common techniques for deriving a digital filter from a given analogue filter is the impulse-invariant transformation. As the name suggests this technique consists of using a sampled version of the impulse response of the analogue filter as the impulse response of the digital filter, i.e. the transformation does not change the impulse response of the analogue filter. Figure 8.17 illustrates the relationship between the analogue and the resulting digital responses of a typical low-pass filter obtained via the impulse-invariant method. The important point to note here is that sampling the analogue impulse response results in the frequency response of the resulting digital filter being periodic with a period equal to the sampling frequency f_s. This means that the digital filter will have a frequency response similar to a repetitive version of that of the analogue filter. If the frequency response of the analogue filter does not decay to near zero beyond $\frac{f_s}{2}$ then serious aliasing would occur and the digital filter response would be corrupted. This aliasing problem means that this design technique is not suitable for high pass filters. However for low-pass and band-pass filters the

problem can be avoided by choosing the sampling frequency high enough to ensure that the magnitude of the analogue filter response is negligible beyond $\frac{f_s}{2}$. (Note that the IMS A100 is capable of a sampling rate of 2.5MHz for 16-bit data and coefficients).

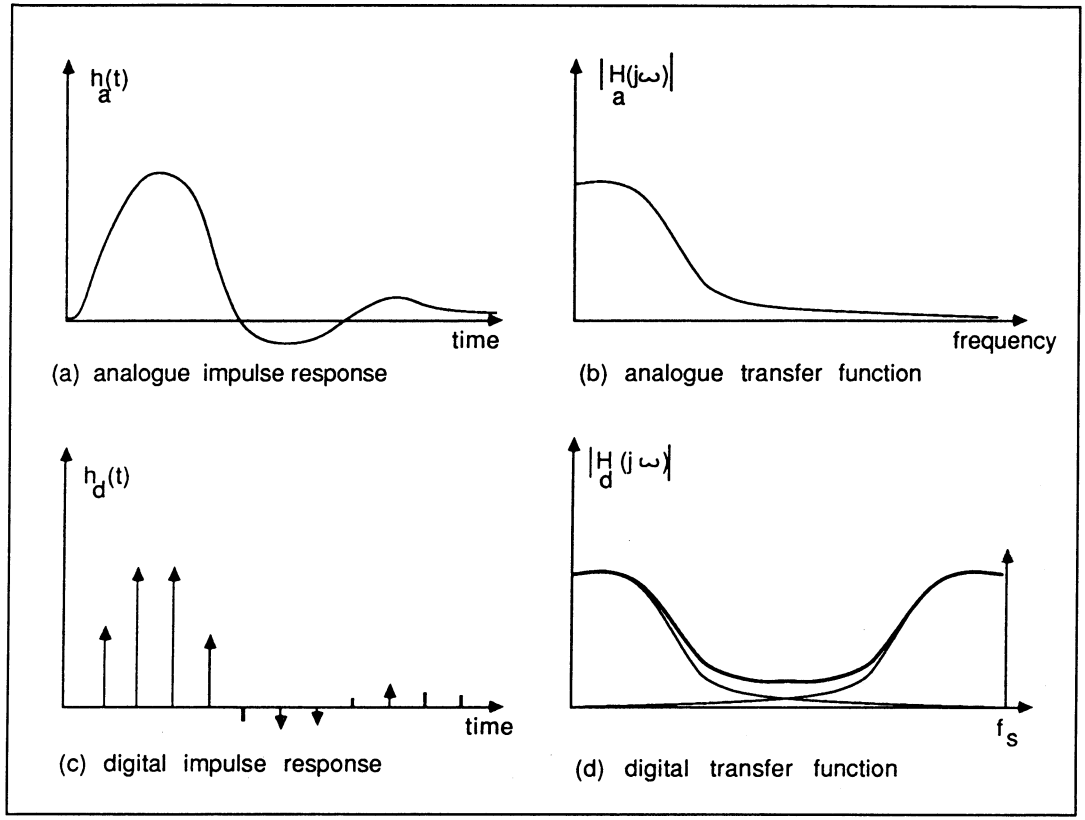


Figure 8.17 The impulsive invariance transformation relationship between analogue and digital impulse and frequency responses

To demonstrate how the impulse-invariant transformation is used to digitize an analogue filter, consider the simple case of an analogue filter with an impulse response $h_a(t) = Ae^{-\alpha t}$ i.e. a simple RC filter (the s-domain transfer function of this filter is $\frac{A}{s+\alpha}$). We start by sampling the impulse response of this analogue filter with a sampling interval T to obtain the corresponding impulse response for the digital filter, i.e.

$$h_a(kT) = Ae^{-\alpha kT} \tag{22}$$

The z-transform of equation (22) is

$$H_d(z) = \sum_{k=0}^{\infty} Ae^{-\alpha kT}\, z^{-k} \tag{23}$$

Noting that as equation (23) is a geometric series the result of the summation would be

$$H_d(z) = \frac{A}{1 - z^{-1}e^{-\alpha T}} \tag{24}$$

Equation (24) provides the z-domain transfer function of the resulting digital filter. To determine the filter coefficient (b_k's and a_m's), equation (24) can be compared with equation (8). For this simple example it can be seen that we have

$$a_1 = -e^{-\alpha T} \qquad \textit{and} \qquad b_0 = A.$$

In this example, for the sake of clarity, the impulse responses were used to arrive at the z-domain transfer function. As analogue filters are often specified in the s-domain, it is more convenient to perform the impulse-invariant transformation directly from the s-domain to the z-domain. It should be obvious to the reader from the previous example that the required mapping is of the form

$$\frac{1}{s+\alpha} \Rightarrow \frac{1}{1-e^{-kT}\,z^{-1}} \tag{25}$$

It can be shown that this is indeed a general mapping (reference 1), applicable to the impulse-invariant method for both real and complex s-plane poles.

As a second example consider the two-pole analogue filter specified by:

$$H_a(s) = \frac{2}{(s+3)(s+1)}$$

expanding using partial fraction yields

$$H_a(s) = \frac{1}{s+1} - \frac{1}{s+3} \tag{26}$$

Using equation (25) the digital transfer function would be:

$$\begin{aligned} H_d(z) &= \frac{1}{1-e^{-T}\,z^{-1}} - \frac{1}{1-e^{-3T}\,z^{-1}} \\ &= \frac{(e^{-T}-e^{-3T})z^{-1}}{1-(e^{-T}+e^{-3T})z^{-1}+e^{-4T}z^{-2}} \end{aligned} \tag{27}$$

Again by comparing equation (27) with (8) we obtain the filter coefficients,

$$b_0 = 0 \qquad b_1 = e^{-T} - e^{-3T}$$

and

$$a_1 = -(e^{-T} + e^{-3T}) \qquad a_2 = e^{-4T}.$$

As described earlier the sampling period T is chosen to ensure negligible aliasing in the filter transfer function.

The bilinear z-transformation

Another indirect design method commonly used for recursive filters is the bilinear z-transformation. The major characteristic of this transformation is that it avoids the aliasing problem which was inherent in the impulse-invariant transformation. Given an analogue transfer function $H(s)$, let us rename the variable s to s_a to indicate the reference to the analogue world i.e. $H(s) = H(s_a)$. Now let us define a new variable s_d related to s_a by the following mapping:

$$s_a = j\frac{2}{T}\tan(\frac{s_d T}{2j}) \tag{28}$$

where T is the sampling period.

Since the analogue frequency variable ω_a is related to the s-plane variable by $s_a = j\omega_a$, we can also express the above mapping as:

$$\omega_a = \frac{2}{T}\tan(\frac{\omega_d T}{2}) \tag{29}$$

where ω_d is defined as $s_d = j\omega_d$.

Starting from an analogue transfer function $H(j\omega_a)$, figure 8.18 illustrates the effect of this mapping on this transfer function. It can be seen from this diagram that the bilinear transformation compresses the entire analogue frequency range ($\omega_a = 0 \rightarrow \infty$) into a finite range equal to half the sampling frequency. This means that the spectral folding problem is completely eliminated and aliasing is therefore avoided. This compression of analogue frequency axis is usually referred to as frequency warping.

The price that is paid for this advantage is a distorted digital frequency scale resulting from this frequency warping. It can be seen from figure 8.18 that due to the non-linear mapping the specification of the resulting filter, such as the cut-off frequency, would be somewhat different from the starting analogue filter. This distortion can be taken into account in the course of digital filter design. For example the cut-off frequency of the original analogue filters are modified slightly so as after the mapping the resulting filter has the desired cut-off frequencies.

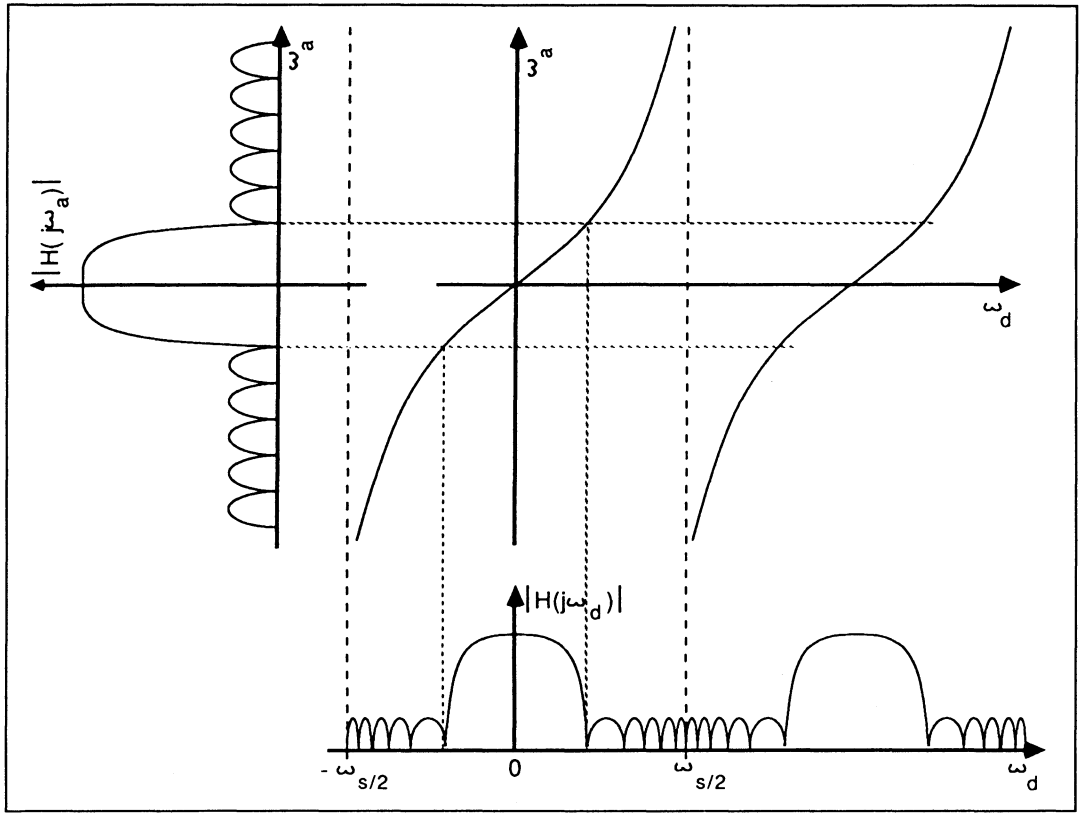


Figure 8.18 Graphical illustration of the bilinear z-transform

Returning to the transformation equation (28), we can rewrite it as:

$$s_a = \frac{2}{T}\left(\frac{1 - e^{-s_a T}}{1 + e^{-s_a T}}\right) \tag{30}$$

and remembering that $z^{-1} = e^{-s_a T}$ we can write

$$s_a = \frac{2}{T}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right) \tag{31}$$

Equation (31) provides the means for bilinear transformation directly from the s-domain to the z-domain suitable for digital filter implementation. To illustrate how the bilinear transformation technique is used consider the following example:

Filter specification

Low pass: 0 → 10kHz pass band
Sampling rate: 100kHz
Transition band: 10kHz to 20kHz
Stop-band attenuation: $-10dB$ (starting at 20kHz)
Filter must be monotonic in pass and stop band.

Design

The monotonicity requirement indicates a Butterworth filter (see previous sections).We have:

digital filter cut-off frequency=$\omega_{cd} = 2\pi \times 10000$
start of digital filter stop band=$\omega_{sd} = 2\pi \times 20000$.

Since the sampling rate is 100kHz, the sampling period would be

$$T = 10^{-5}$$

therefore

$$\omega_{cd}T = 0.2\pi \qquad and \qquad \omega_{sd}T = 0.4\pi$$

Using equation (29) we can calculate the corresponding analogue filter frequencies i.e.

analogue filter cut-off frequency=$\omega_{ca} = \frac{2}{T}\tan(0.1\pi) = 0.6498 \times 10^5$

Start of analogue filter stop band=$\omega_{sa} = \frac{2}{T}tan(0.2\pi) = 1.4531 \times 10^5$.

The required order of the Butterworth filter can be determined by using equation(18) and ensuring at least $10dBs$ attenuation at $\omega = \omega_{sa} = 1.4531 \times 10^5$ i.e.

$$10\log[1 + (\frac{1.4531 \times 10^5}{0.6498 \times 10^5})^{2n}] = 10$$

or

$$1 + (\frac{1.4531 \times 10^5}{0.6498 \times 10^5})^{2n} = 10$$

This gives $n = 1.367$, therefore we choose $n = 2$.

A second order butterworth filter with a cut-off at $\omega_{ca} = 0.650 \times 10^5$ has two equally-spaced poles on a circle of radius ω_{ca} (reference 1) given by

$$s_1,\ s_2 = -0.6498 \times 10^5(0.7071 \pm 0.7071j) = -0.4595(1.0 \pm j) \times 10^5$$

and the transfer function is given by:

$$H(s) = \frac{s_1 s_2}{(s - s_1)(s - s_2)} = \frac{4.223 \times 10^9}{s^2 + 0.919 \times 10^5 + 4.223 \times 10^9}$$

Now we apply the bilinear-z transformation by substituting for s in the above transfer function from equation (31). This gives the following digital filter transfer function:

$$H(z) = \frac{0.0675 + 0.1349z^{-1} + 0.0675z^{-2}}{1 - 1.1430z^{-1} + 0.4128z^{-2}} \tag{32}$$

The digital filter coefficients can be obtained by comparing equation (32) with (8) giving:

$$b_0 = 0.0675 \quad - - -$$
$$b_1 = 0.1349 \quad a_1 = -1.1430$$
$$b_2 = 0.0675 \quad a_2 = 0.4128$$

These coefficient values are then expressed in binary with the number of bits governed by the required accuracy. The factors affecting the necessary accuracy are discussed in section 5 of this application note.

Matched z-transform

This transformation is a direct mapping from the poles and zeroes in the s-plane to the poles and zeroes in the z-plane.

In general the two previous method i.e. the impulse invariant and the bilinear transformations are preferred to the matched z-transform as there are many cases where the matched z-transformation is not applicable. For this reason this technique is not detailed here. It would be sufficient to point out that the mapping is defined by the replacement relationship:

$$s + k = 1 - z^{-1}e^{-kT}. \tag{33}$$

The direct design techniques for IIR filters

The IIR design techniques described so far were based on transforming a known analogue transfer function into the required digital filter transfer function. It is however possible to design digital IIR filters directly without reference to an analogue filter. Direct design methods fall into two categories namely direct closed form designs and optimisation techniques.

The direct closed form design techniques begin with the desired response of the filter from which one can often decide where to place poles and zeroes to approximate this response. These techniques are not very common and as such will not be discussed here.

The second classes of direct IIR filter design techniques are based on computer optimisation. In these approaches the set of design equations cannot be solved explicitly, instead mathematical optimization techniques are employed to determine the filter coefficients that minimize some error criterion, subject to a set of design equations. The algorithms involved in these optimisation techniques are of an iterative nature and are terminated when the error reaches a minimum or the number of iterations exceeds a specified limit.

Among the most commonly used optimisation technique is one which minimizes the pass-band ripples in filters exhibiting a given stop-band attenuation. This technique is sometimes referred to as the minimax method and the optimization algorithm involved has been developed by Fletcher and Powell (reference 7). The Fletcher-Powell optimization algorithm generates the filter coefficients by using a convergent descent method.

The spectral flatness approach is another optimisation technique and is based on the fact that multiplying the desired frequency response by its inverse should result in unity throughout the frequency spectrum (i.e. a flat spectral line). Any deviations from the ideal response would result in ripples in this flat spectral line. Optimisation techniques have been developed which attempt to minimize these ripples (reference 8). The difficulty with this technique is the modeling of the desired frequency response.

Mean-square-error optimization techniques have also been developed for IIR filter design. One such technique has been described by Steiglitz (reference 9) which involves minimizing the square of the difference between actual filter behaviour and the desired performance. This algorithm searches an error *.vs.* design-parameter curve for a local minimum.

The details of the above optimisation techniques are beyond the objectives of this application note. However the references given should prove adequate for interested readers.

8.5 Finite word-length considerations and problems

In implementing digital filters both the input samples and the filter coefficients have to be quantised and expressed in a limited number of bits. In the IMS A100 chip both the coefficients and data samples can be quantized up to 16-bits of accuracy, although smaller word-lengths can be used if desired.

The problems of finite word length in digital filters apply to both FIR and IIR filters but their implications are much more severe for the IIR filters, due to their inherent feedback nature. In the fixed-point implementations of digital filters it is usual to normalize the numbers so as to make their absolute values less than one i.e. in the form of

$$d_{N-1}.d_{N-2}d_{N-3} - - - - - - - - - d_5d_4d_3d_2d_1d_0$$

where d_n represents the nth bit in the word and (.) indicates the binary point. Using this format (and two's complement notation) the number

$$0.1111\ldots1111$$

would represent a vlaue very nearly equal to +1, while the number

$$1.0000\ldots0000$$

would represent a value equal to −1.0.

If purely integer numbers were to be used the process of truncation or rounding after multiplications would become meaningless. However using the above fractional-number representation, where the numbers are

normalized to be less than one, the problems would not arise as the product of two numbers which are less than one would also be less than one.

In general there are three sources of error arising in the implementation of digital filters these are:

(i) Finite precision of the filter coefficients

(ii) Limited word length of the input data

(iii) Round-off and truncation errors in the multiplication and addition operations.

The finite precision in the representation of the filter coefficients will obviously cause the frequency response of the filter to depart to some extent from that desired for both FIR and IIR filters.

Furthermore in the case of the recursive IIR implementation, because of the existence of feedback paths, this finite precision may cause instabilities in the filter behaviour. This happens because the inaccuracies may move the z-plane poles outside the unit circle hence causing instabilities. The chances of this happening depends on how close the poles are to the unit circle in the first place. If multiplication and addition operations are followed by truncation and rounding (in order to contain word growth) further difficulties may arise. These problems may manifest themselves in undesirable oscillations in the form of 'limit cycle' or 'overflow' oscillations (discussed later). It is therefore absolutely essential for the filter behaviour to be simulated using the precision and roundings involved in the intended implementation. This is particularly relevant to recursive IIR filter where a risk of instability exists.

One of the consequences of rounding and quantisation in the digital recursive(IIR) filters is the limit-cycle phenomenon, which takes the form of a stable periodic non-zero output for zero or constant input. The limit cycle behaviour of a digital filter in general is complex and difficult to analyse. However for simple first order filters, it is possible to illustrate the effect by way of an example. Consider the first order recursive filter with the following equation:

$$y(n) = 0.09x(n) + 0.91y(n-1)$$

Assume that each output y(n) gets rounded to the nearest integer, also assume that the input is constant at 100 and the previous output is 90.

The following table shows the resulting rounded output sequence for each iteration.The last column shows the perfect output (without rounding) for comparison.

n	x(n)	y(n)	rounded y(n)	perfect y(n)
0	100	–	90	–
1	100	90.9	91	90.9
2	100	91.81	92	91.72
3	100	92.72	93	92.46
4	100	93.63	94	93.14
5	100	94.54	95	93.76
6	100	95.45	95	94.32
7	100	95.45	95	94.83
8	100	95.45	95	95.30
9	100	95.45	95	95.72
10	100	95.45	95	96.11
..	...	.....	..	.....
..	...	.....	..	.....
..	...	.....	..	.....
..	100	95.45	95	100.0

It is observed that the output sticks at a value of 95. However if the same filter is implemented with very high precision and no rounding the filter output would closely approach 100 (last column in the table).

If we approach the limit from the opposite side by starting with a value of $y(n)$ of say 110, the output would arrive at a limit of 105. You can see from this example that the system has a dead zone of ±5 units around the ideal output of 100.

In fact it can mathematically be shown that for a first-order recursive filter of the form

$$y(n) = bx(n) + ay(n-1)$$

The dead zone is given by

$$|dead\ zone| \leq \frac{\frac{q}{2}}{1-|a|} \tag{34}$$

where q is the quantisation step. In the above example a quantised step of 1 was used and equation (34) gives a dead zone of ±5 too.

For second-order systems similar results to (34) have been derived in the literature (reference 1).

Overflow oscillation is another problem associated with digital recursive filters. In the IMS A100 chip the full internal precision ensures that no overflow occurs in the multiply-accumulator array. The only source of possible overflow is the external addition which is performed in combining the feedback terms with the input samples (see section 4.4). A simple but effective way to eliminate these oscillations is to perform this addition in a saturating manner (similar to analogue adders). This operation can easily be taken care of by the controlling host processor.

In the IMS A100 device the data and coefficients can be expressed to a precision as high as 16 bits. The 32 multiplications and additions are carried out to 36-bit precision. This ensures that no overflow occurs in the multiply-accumulation array (unless all the coefficients and 32 consecutive data items have values equal to the most negative 16-bit number i.e. 1000000000000000 in binary, which is of course highly unlikely). The selector at the output of the multiply and accumulate array allows the rounding and selection of 24 bits out of this 36 bits. The combination of full internal accuracy, the selector functionality and the fact that the IMS A100 devices can easily be cascaded allows high quality FIR filters to be readily implemented. As described earlier the device can also be used to implement efficient IIR filters only in direct forms. It is well known that for high order filters direct implementations of IIR filters are more prone to instabilities compared to cascade or parallel arrangements. However the full internal precision of the IMS A100 combined with comprehensive filter simulations should minimize these instabilities. It should however be emphasized that it is possible to implement a high order high precision IIR filter in the cascade form on the IMS A100 at the expense of processing speed. In this case the IMS A100 should be used to implement low order (2nd or 4th order) sections of a cascade arrangement in turn by reloading suitable coefficients. The functionality of the whole filter is obtained by recirculating the first output batch through the chip with its coefficients modified to implement the 2nd section in the cascade array and so on.

For the IIR filter implementations figure 8.11c & 8.11b can be considered as possible system configurations.

8.6 Adaptive filters

So far we have discussed digital filters with fixed characteristics. Fixed filters are used in many practical situations to combat noise or interfering signals (e.g. a matched filter) or to select a desired frequency band (e.g. a band-pass filter). In digital signal processing the parameters of such fixed filters are determined once and remain unchanged during processing. Adaptive filters on the other hand automatically adjust their own parameters and seek to optimize their performance according to a specific criterion. The adaptive nature of such filters makes them particularly suitable for situations where signal properties are unknown or variable with time.

Figure 8.19 illustrates the basic structure of an adaptive filter. The input signal $x(t)$ is filtered or weighted in a programmable filter to yield an output $y(t)$. The filter output $y(t)$ is then compared with a reference (sometimes called a training signal) waveform to yield an error signal $e(t)$. This error is then used to update the filter coefficients in such a way that the error is progressively minimized. Several algorithms for updating the filter coefficients have been developed and can be found in references 10, 11 & 12.

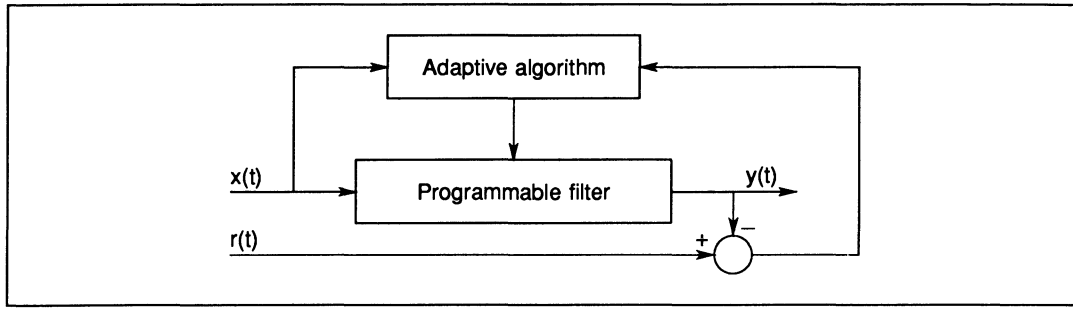


Figure 8.19 Basic structure of an adaptive filter

One example of adaptive filtering is echo cancellation in telephony. Echoes are the result of impedance mismatches in the communication circuits. The hybrid couplers which are used at the interface between two-wire and four-wire circuits are a major source of echoes. Figure 8.20 shows how an adaptive filter arrangement can be used to cancel these echoes at the hybrid interface. Notice that in this case the training signal contains the echo, while the input to the adaptive filter is the signal arriving at the hybrid. Effectively the filter adaptively models the echo path and produces a synthetic antiphase echo return which cancels the echo in the 4-wire path returning from the hybrid.

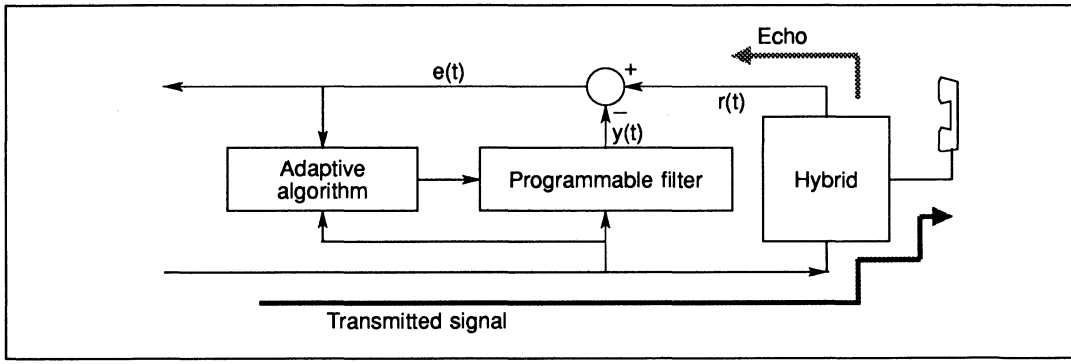


Figure 8.20 Application of adaptive filtering techniques to echo cancellation

Adaptive filters have application in low-bit rate speech coding based on linear prediction where the filter coefficients, after adaption, are transmitted instead of the speech signal itself.

The programmability of the IMS A100 can be exploited in the implementation of adaptive filters as well as fixed filters discussed earlier.

8.7 References

1 *Theory and application of digital signal processing*, Rabiner L.R., and Gold B., Prentice-Hall Inc, Englewood Cliffs, NJ, 1975.

2 *Digital signal processing*, Oppenheim A.V., and Schafer R.W., Prentice-Hall Inc, Englewood Cliffs, NJ, 1975.

3 *A computer program for designing optimum FIR linear-phase digital filters*, McClellan J.H., Parks T.W., and Rabiner L.R., IEEE Transactions on Audio and Electroacoustics, Vol. AU-21, No.8 , December 1973.

4 *Digital signal processing*, Peled A., and Liu B., John Wiley and Sons Inc., New York, 1976.

5 *Practical design rules for optimum finite impulse response low-pass digital filters*, Rabiner L.R., Bell Systems Technical Journal, Vol. 52, No.6, July-August 1973.

6 *Approximate design relationships for low-pass FIR digital filters*, Rabiner L.R., IEEE Transactions on Audio and Electroacoustics, Vol. AU-21, No.5, October 1973.

7 *A rapidly convergent descent method for minimization*, Fletcher R., and Powell M., Computer Journal 6 (No.2) 1963, pp 163-168.

8 *Time series analysis: Forcasting and control*, Box G., and Jenkins G., Holden Day, San Francisco, CA, 1975

9 *Computer aided design of recursive digital filters*, Steiglitz K., IEEE Transactions on Audio and Electroacoustics 18 (No.2), June 1970, pp123-129.

10 *Adaptive noise cancelling: principle and applications*, Widrow B. *et al*, Proc. IEEE, 1975, Vol.63, No.12, pp 1692-1716.

11 *Design and application of adaptive filters*, Grant P.M., Cowan C.F.N., Electronics & Power, February 1985.

discrete Fourier transform with the IMS A100

9.1 Introduction

In the time-domain representation, signals are expressed as a function of time. For example $x = Ae^{-at}$ is a time-domain description of a signal whose amplitude decays exponentially with time.

In the 18th century J.B. Fourier showed any signal that can be generated in a laboratory can be expressed as a sum of sinusoids of various frequencies. In other words each signal can be said to have a frequency spectrum represented by the amplitude and phases of various sinosoidal components. The frequency spectrum of a signal completely specifies it and is referred to as frequency-domain description of the signal.

The Fourier integrals provide the means for obtaining the frequency-domain representation (the spectrum) of a signal from its time-domain representation or vise-versa, i.e.

Fourier Transform:

$$X(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t}dt \tag{1}$$

Inverse Fourier Transform:

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} X(\omega)e^{j\omega t}d\omega \tag{2}$$

where $x(t)$ is a time-domain signal, $X(\omega)$ is the frequency spectrum of $x(t)$ and ω is the frequency variable.

These transforms are fundamental to the description of many real world phenomena in the fields of science and engineering. In the area of signal processing the Fourier transform is an important mathematical (and nowadays practical) tool in understanding, analysing and solving system-level problems. The Fourier transform allows us to translate time serial information into the frequency domain in a reversible way. The components of a signal, although dispersed in the time domain, may have restricted occupancy or a characteristic relationship in the frequency domain. In fact many physical processes can be categorised by the frequencies they generate and their relative strength. This is one reason why the Fourier transform, with its ability to segregate frequency components of a signal, has gained such an importance in many signal processing environments.

Apart from the ability of the Fourier transform to provide spectral information, many signal processing functions such as correlation, filtering and beamforming can be expressed in terms of the Fourier transform and its inverse. For these reasons considerable effort has gone into the development of efficient algorithms for evaluating the Fourier transform. The continuous-time Fourier transform, given by equation (1), can be made suitable for digital computation by sampling the time and frequency variables and limiting the computation to a finite set of data points. This modified version of the Fourier Transform is often referred to as the Discrete Fourier Transform (DFT) and will be discussed in more detail in the next section.

Many algorithms have been developed for efficient digital computation of the DFT. Until recently the digital multipliers needed to implement DFT's were costly, large and relatively slow, and the general purpose microprocessors were extremely slow at performing multiplications. Consequently it was necessary to calculate DFT's using a minimum number of multiplications and to use data and coefficient storage economically and this led to development of several Fast Fourier Transform (FFT) (references 1 & 2) algorithms. These FFT algorithms make use of the redundancies, that occur in the DFT, to reduce the arithmetic operations involved. Most FFT algorithms were designed simply to minimise the total number of multiplications required to calculate the DFT, often at the expense of an increase in the number of additions, memory accesses and control complexity. One such algorithm is the Cooley-Tuckey radix-2 FFT algorithm which necessitates a data size equal to a power of two ($N = 2^n$). Winogard FFT algorithms on the other hand requires that the number of data points to be prime. Both algorithms simply minimize the number of multiplications in the DFT by the use of redundancies resulting from the particular choice of the data size. These algorithms are particularly suitable for general-purpose computers and microprocessors where the major limit on processing speed is the time taken to perform the multiply instruction.

Other algorithms have been developed which map the DFT process into particular hardware structures. Two such techniques are the Rader's Prime Number Transform (PNT) (reference 3) and the Chirp-Z Transform (CZT) (reference 4) which convert the DFT into circular correlation/convolutions. These algorithms are particu-

larly suitable for implementation using transversal filter type structures. In the past, CCD and SAW transversal filters have been used to implement high-throughput wide-bandwidth DFT processors using these algorithms. The analogue nature of the CCD and SAW technologies has restricted the precision of these processors.

The availability of the IMS A100, the first high performance cascadable digital transversal filter, means that the same algorithms can now be implemented digitally offering both high speed and high accuracy. This application note deals with the concepts behind these algorithms and their implementations using the IMS A100 signal processor. Generalised mapping techniques which facilitate the DFT evaluation of a long data sequence via a number of short transforms are also discussed (The radix-2 FFT is a special case of these more general partitioning techniques). The approach described here is particularly suitable if a long DFT is to be evaluated with a transversal filter of limited size. Also, these decomposition methods are applicable to concurrent architectures and as such provide the basis for the trade-off between speed and cost involved in a particular implementation. These issues are of major importance when combining the IMS A100(s) with the INMOS transputer family of parallel processors.

Figure 9.1a shows the structure for an N-stage canonical transversal filter where the output is the weighted sum of the N most recent input samples. The IMS A100 implementation of the transversal filter is depicted in figure 9.1b where the multi-input summation of the canonical form has been replaced by a delay and add chain. You should be able to convince yourself that the two structures in figure 9.1 have the same functional behaviour. The main difference is that in figure 9.1b the partial product terms are passed down the delay-and-add chain whilst in figure 9.1a, the input samples are delayed and the sum of products is calculated simultaneously.

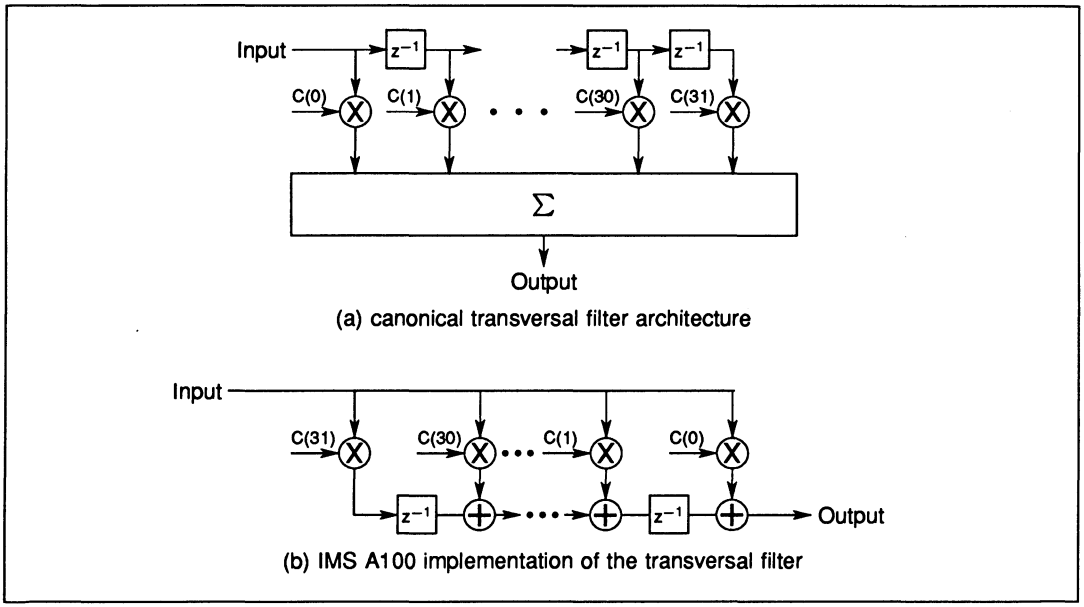


Figure 9.1 Transversal filter architecture

A simplified functional diagram for the IMS A100 is shown in figure 9.2. The major processing part of the chip incorporates 32 multipliers and a 32-stage delay-and-add chain. For the IMS A100 the input data word length in 16 bits. The coefficient word length can be programmed to be 4, 8, 12 or 16 bits. The data throughput ranges from 2.5 million samples/s to 10 million samples/s depending on the coefficient word size. Two complete sets of coefficient memories are provided. At any instant one set of coefficients is applied to the transversal filter, whilst the other set can be accessed via a standard memory interface (capable of 100ns cycle time). The function of the two coefficient memories can be exchanged by writing to control registers. Further this exchange can be made continuous, i.e alternate sets of coefficients can automatically be selected for successive computation cycles. This is particularly useful for complex number processing.

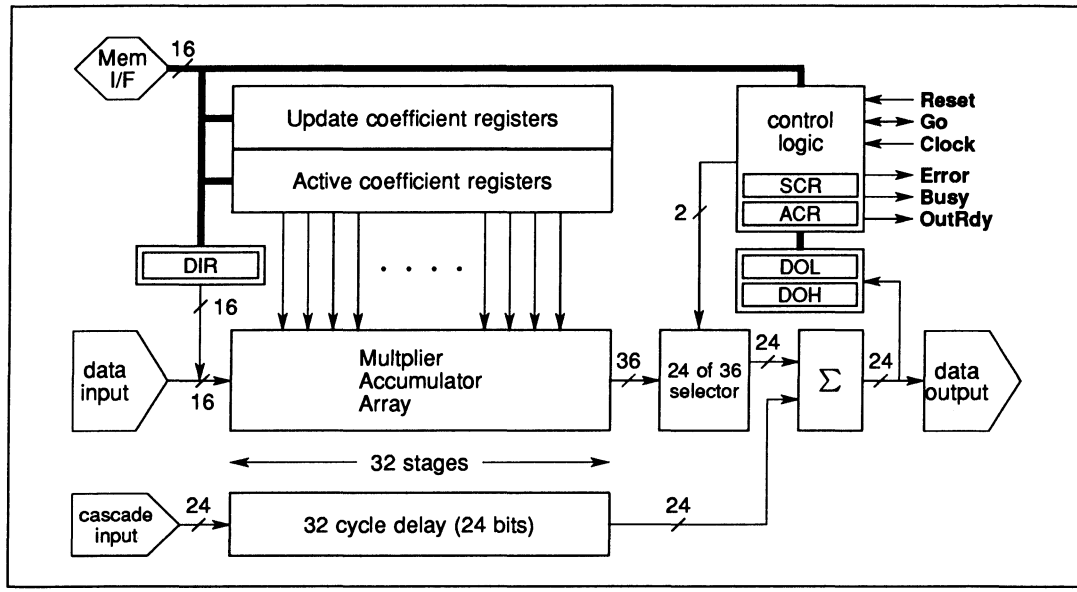


Figure 9.2 User's model of the IMS A100

Data input and output are available both through dedicated ports or via the memory interface. This selection can be programmed via the control and status registers in the IMS A100.

To preserve complete numerical accuracy, no truncation or rounding is performed on the partial products in the multiplications and delay-and-add chain. The output of the chain is calculated with a precision of 36 bits which is sufficient to ensure no overflow occurs (the only time that the output of the delay-and-add chains exceeds 36 bits is when all 32 coefficients and 32 successive input samples have the maximum possible negative value i.e. 1000000000000000 in two's complement binary notation, this is of course highly unlikely). A programmable barrel shifter is located at the output of this chain, which allows 24 bits (starting at bits 7, 11, 15 or 20 of the full 36 bit result) to be selected and rounded for output. To allow devices to be cascaded without any external components, a 32-stage 24-bit wide shift register and a 24-bit adder are included on the chip. For cascading purposes the output of one chip is connected directly to the cascade input of the next.

The control registers accessible via the memory interface allows various operational parameters to be programmed. For the full detail of the specification you are advised to refer to the IMS A100 data sheet.

In the following parts of this application note the basic concepts of DFT will be reviewed and some algorithms for its evaluation will be summarized. This is followed by a detailed description of those DFT algorithms suitable for implementation using the IMS A100 transversal filter. A multi-dimensional mapping technique is also described which allows efficient computations of long-length DFTs via the IMS A100 implementations.

9.2 The basic concepts of DFT

The equation for the Fourier transform and its inverse (equations 1 & 2) can be made suitable for digital processing by discretizing both the time variable t and the frequency variable ω; and by constraining the integration to finite limits. Referring to equation 1, for the forward transform, this can be done by making $t = nT$ and $\omega = k\omega_0$. Where T is the sampling period of the time function and ω_0 is the frequency resolution of the discrete spectrum. If the integration limits are confined over N time samples for which N independent frequency samples can be calculated we have

$$n = 0, 1, 2, 3, ..., N-1$$

$$k = 0, 1, 2, 3, ..., N-1$$

The Fourier-transform integral then becomes a Fourier-transform sum given by:

$$X(k\omega_0) = \sum_{n=0}^{N-1} x(nT)e^{-jk\omega_0 nT} \qquad k = 0, 1, ...N-1 \tag{3}$$

It can be shown that the frequency resolution in the f-domain is given by:

$$\omega_0 = \frac{2\pi}{NT} \tag{4}$$

Substituting this in (3) gives:

$$X(k\omega_0) = \sum_{n=0}^{N-1} x(nT)e^{-2\pi jkn/N} \qquad k = 0, 1, ...N-1 \tag{5}$$

For convenience the terms T and ω_0 are usually dropped from the indicies giving the DFT equation as:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-2\pi jnk/N} = \sum_{n=0}^{N-1} x(n)W^{nk}{}_N \qquad k = 0, 1, ...N-1 \tag{6}$$

where

$$W_N = e^{-2\pi j/N} = \cos(\frac{2\pi}{N}) - j\sin(\frac{2\pi}{N}) \tag{7}$$

The inverse discrete Fourier transform (IDFT) can be derived in a similar manner from its corresponding continuous form and is given by:

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{2\pi jnk/N} = \frac{1}{N}\sum_{k=0}^{N-1} X(k)W_N^{-nk} \qquad n = 0, 1, ...N-1 \tag{8}$$

Note that the DFT and its inverse, IDFT, are very similar, the only difference is the factor $\frac{1}{N}$ and the negative exponent in the IDFT. This similarity has important practical significance as it allows an algorithm or a hardware developed for DFT to be used for IDFT with minor modifications. For example an inverse DFT on the data sequence $x(0), x(1)....x(N-1)$ can be carried out by first reversing this sequence to generate a new data set $x'(\,)$ such that $x'(0) = x(N-1)$, $x'(1) = x(N-2)$.....$x'(N-1) = x(0)$ and then performing a DFT and dividing the result by N. This technique works simply because $x'(k) = x(-k)$, (In the DFT both $x(n)$ and $X(k)$ are assumed to be periodic) which converts the positive exponential in (8) into a negative one, representing a DFT. For this reason any algorithm or implementation in the following subsections will only be described for DFT as the extension to an IDFT is trivial.

It is worth noting that some authorities write the DFT and its inverse as:

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)e^{-2\pi jnk/N} \qquad DFT$$

$$x(n) = \sum_{k=0}^{N-1} X(k)e^{2\pi jnk/N} \qquad IDFT$$

i.e. the factor $\frac{1}{N}$ is applied to the DFT rather than its inverse. This version can be seen to have a physical meaning since $X(0)$, as defined, represents the average of the sampled time waveform i.e. the 'd.c.' value. Other authorities express the DFT and its inverse as:

$$X(k) = \frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} x(n)e^{-2\pi jnk/N} \qquad DFT$$

$$x(n) = \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} X(k)e^{2\pi jnk/N} \qquad IDFT.$$

This last formulation is necessary if the power contents of the time-domain and frequency-domain signals are to be identical.

Throughout this application note, the definitions given by equations (6) and (8) are used for DFT and IDFT. However, the techniques described here are applicable to all three formulations of the DFT and its inverse.

9.3 Algorithms for efficient evaluation of DFT

From equation (6), it is apparent that the direct evaluation of the DFT is very much computation intensive. Assuming complex data, $x(n) = xr(n) + j\ xi(n)$, we have

$$X(k) = XR(k) + jXI(k) = \sum_{n=0}^{N-1} [xr(n) + jxi(n)](\cos(\frac{2\pi kn}{N}) - j\sin(\frac{2\pi kn}{N})$$

or

$$XR(k) = \sum_{n=0}^{N-1} xr(n)\cos(\frac{2\pi nk}{N}) + xi(n)\sin(\frac{2\pi nk}{N}) \tag{9a}$$

and

$$XI(k) = \sum_{n=0}^{N-1} xi(n)\cos(\frac{2\pi nk}{N}) - xi(n)\sin(\frac{2\pi nk}{N}) \tag{9b}$$

where $k = 0, 1, 2.....N-1$, $XR(k)$ and $XI(k)$ are the real and imaginary parts of the spectrum respectively.

From equation (9) it can be deduced that the direct evaluation of DFT involves $4N^2$ multiplications and approximately $4N^2$ additions. In these estimates the computations involved in the evaluation of the trigonometric functions, (sin and cos) have been ignored as it is possible to precalculate the trigonometric values in a look-up table and use them appropriately. Historically, multiplications were very slow compared to other operations, therefore algorithms were developed to minimize the number of required multiplications at the expense of other operations. These algorithms made use of the cyclic nature of the exponential $\exp(\frac{-2\pi jnk}{N})$ to reduce the number of multiplications involved.

To demonstrate some of the resulting redundancies, consider the case where N is even. It is fairly straightforward to show that for this case

$$W_{\frac{N}{2}}^{k} = W_N^{2k} \tag{10}$$

and

$$W_N^k = -W_N^{k+\frac{N}{2}} \tag{11}$$

One algorithm which uses these types of redundancies is the Cooley-Tuckey radix-2 FFT (reference 1). This algorithm requires the number of data points to be equal to a power of two, ie $N = 2^m$ where m is an integer. Using the identities given by (10) and (11) the algorithm expresses the N-point DFT in terms of two $\frac{N}{2}$-point DFT's. Then the $\frac{N}{2}$-point DFT's are expressed as two $\frac{N}{4}$-point transforms using identities similar to (10) and (11). This decomposition is carried out until all the DFT's involved are only two-point transforms. The net result of this decomposition is a considerable reduction in the number of multiplications. In fact it can be shown that for the Cooley-Tuckey radix-2 FFT algorithm the number of multiplications involved is approximately $2N\log_2(N)$ and the number of additions is $3N\log_2(N)$. Compared to the $4N^2$ operations involved in the direct DFT calculation, for a large N, the radix-2 FFT algorithm reduces the number of multiplications and additions considerably.

Another algorithm which also minimises the number of multiplications is Winogard's prime-length transform (reference 2). This algorithm is applicable to cases where the data size is a prime number. In practice this algorithm is used only for short-length transforms and mapping techniques are used to extend it to large data sizes (references 5 & 6).

The argument behind the efficiency of these algorithms is only valid if the multiplication time is longer than other operations such as indexing and memory accesses. This is indeed the case for most general-purpose processors.

Todays digital technology is capable of providing extremely powerful processing engines which mean that the minimization of the number of multiplications is not always the best approach. For high performance systems, other issues such as the memory bandwidth, architecture efficiency and parallelism potential have to be seriously considered. The advances in digital technology allow other algorithms particularly those which map the DFT onto special VLSI hardware structures to be exploited. The following sections deal with algorithms that map the DFT into correlation/convolutions, ideal for implementation using the IMS A100 transversal filter. This algorithms make use of the higher level functional nature of the device and its on-chip

memory to minimise the required host's memory bandwidth. For this reason the combination of a medium-speed microprocessor and the IMS A100 device(s) results in a very high performance system capable of competing with bit-slice DSP processors.

9.4 DFT algorithms suitable for the IMS A100 implementation

There are basically two algorithms which map the DFT into a correlation (convolution) process. These are

(i) the Prime Number Transform (PNT)

(ii) the Chirp-Z-Transform (CZT)

The PNT was developed by Radar and is applicable when the number of data points is prime. The CZT on the other hand is applicable to any data size; it can, however, be simplified if the data size is an even number. The following two sections deal with each one of these algorithms and their implementations using the IMS A100 transversal filter. The final part of this application note describes mapping techniques which allow the DFT of a large number of data points to be evaluated via a number of short transforms. This mapping technique is of vital practical significance when implemanting PNT & CZT processors.

9.4.1 Rader's Prime Number Transform

The PNT algorithm has its origin in number theory (reference 7) and consists of three seperate operations. The first is a permutation (re-ordering) of the input data. The second operation is correlation of the permuted input data with permuted discrete cosine and sine samples. The third operation is a repermutation, which yields the DFT components in the conventional order of linear frequency. This final stage may be ignored in applications which also involve an inverse DFT.

In this section the mathematical background for the PNT will be summarized. Where necessary examples are provided to assist in the understanding of the concepts.

If the standard DFT equation, i.e.

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-2\pi jnk/N} = \sum_{n=0}^{N-1} x(n)W_N^{nk} \qquad k = 0, 1, ...N-1, \tag{12}$$

is to be converted to correlation between $x(n)$ and the twiddle factors W_N's, the nk product needs to be converted to a sum $n+k$. For cases where N is prime number theory allows us to achieve this.

According to number theory (reference 7), for each prime number N, there exist integers r, known as primitive roots, whose successive integer powers modulo-N will generate a permuted version of the sequence $1, 2, 3, ...N-1$.

What this means is that for a prime number N, it is possible to map the sequence $\{p\} = 0, 1, 2, ...N-2$ via the equation

$$q = (r^q) mod\ N \quad where\ \{p\} = \{0, 1, 2, 3, ...N-2\} \tag{13}$$

to a sequence $\{q\}$ where q is a one-to-one map of the original sequence $\{p\}$ and consists of a permuted version of the sequence $\{1, 2, 3, ...N-1\}$. For such a unique map to be possible r must be a primitive root of N. Let us consider $N = 7$, for which one of the primitive roots of 7 is 3. From (13) the mapping equation is

$$q = (3^p) mod\ N \quad where\ p = 0, 1, 2, ..., 5$$

For $p = 3$, $q = (27) mod\ 7 = 6$. Table 9.1 gives the corresponding values of p and q which confirm the one-to-one nature of the mapping.

It should be emphasized that for any prime N, the primitive root r, is not unique. For example table 9.2 illustrates the mapping given by (13) for $N = 7$. From this table you can see that the mapping is unique and cyclic for $r = 3$ and $r = 5$ which are the primitive roots of 7. In most practical cases the smallest primitive root is often selected.

p	0	1	2	3	4	5	6	7
q	1	3	2	6	4	5	1	3

Table 9.1 The mapping corresponding to equation (13) for $r = 3$

$r \setminus p$	0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	1	2	4	1	2	4	1	2	4	1
3*	1	3	2	6	4	5	1	3	2	6	4	5	1
4	1	4	2	1	4	2	1	4	2	1	4	2	1
5*	1	5	4	6	2	3	1	5	4	6	2	3	1
6	1	6	1	6	1	6	1	6	1	6	1	6	1
7	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	1	1	1	1	1	1	1	1	1	1	1	1
9	1	2	4	1	2	4	1	2	4	1	2	4	1
10	1	3	2	6	4	5	1	3	2	6	4	5	1
11	1	4	2	1	4	2	1	4	2	1	4	2	1
12	1	5	4	6	2	3	1	5	4	6	2	3	1
13	1	6	1	6	1	6	1	6	1	6	1	6	1
* Note that $r = 3$ and $r = 5$ give unique mapping and are therefore the primitive roots of 7.													

Table 9.2 Values for q in the mapping $q = (r^p) mod 7$

If N is prime and r is primitive root of N then we would like to apply the mapping given by (13) to an N-point DFT. Referring to the DFT equation (12), it can be seen that the subscripts n and k vary from 0 to $N-1$ whilst in the mapping given by (13) the variable q assumes values from 1 to $N-1$ i.e it excludes zero. To overcome this problem we write the DFT equation (12) in the following form

$$X(0) = \sum_{n=0}^{N-1} x(n) \tag{14a}$$

$$X(k) - x(0) = \sum_{n=1}^{N-1} x(n) W_N^{nk} \qquad k = 1, \ldots, N-1 \tag{14b}$$

i.e. we separate the expressions for the zero-frequency DFT component $X(0)$ (the d.c. term) which is very simple. This expression consists of N additions only and if required can be calculated directly.

We are left with DFT components $X(k)$ corresponding, to $k = 1, 2, ..N-1$, which are given by (14b). Note that in this equation we have taken $x(0)$ to the left-hand side so as the summation over n is from $n = 1$ to $n = N-1$. We can now apply the mapping given by (13) to equation (14) via the following transformations,

$$n = (r^m) \ mod \ N \qquad m = 0, 1, ..., N-2 \tag{15}$$

$$k = (r^l) \ mod \ N \qquad l = 0, 1, ..., N-2 \tag{16}$$

which result in a permutation of the terms in the summation and a change in the order of the equations. Equation (14b) then becomes:

$$X[(r^l) mod N] - x(0) = \sum_{m=0}^{N-2} x[(r^m) mod N] W_N^{[(r^m) mod N][(r^l) mod N]} \tag{17}$$

where $l = 0, 1, 2, ...N-2$

Remember that the twiddle factor, W_N, is cyclic in N, therefore we have

$$X[(r^l) mod N] - x(0) = \sum_{m=0}^{N-2} x[(r^m) mod N] W_N^{r^{(m+l)}} \tag{18}$$

where $l = 0, 1, 2, ...N-2$ This equation indicates that the sequence $X[(r^i) mod N] - x(0)$ can be calculated via a circular correlation of the permuted input sequence $x[(r^i) mod\ N]$ and the sequence $e^{-2\pi j(r^i)/N}$. To see this clearly let us consider in detail an example for a DFT of length 7.

The expression for a 7-point DFT is given by

$$X(k) = \sum_{n=0}^{6} x(n) W_N^{nk} = \sum_{n=0}^{6} x(n) e^{-2\pi jnk/7} \tag{19}$$

where $n, k = 0, 1, 2, ...N-1$

This DFT equation can be expressed in matrix form as: (The subscript 7 has been dropped from W_7 for convenience)

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 & W^4 & W^5 & W^6 \\ W^0 & W^2 & W^4 & W^6 & W^1 & W^3 & W^5 \\ W^0 & W^3 & W^6 & W^2 & W^5 & W^1 & W^4 \\ W^0 & W^4 & W^1 & W^5 & W^2 & W^6 & W^3 \\ W^0 & W^5 & W^3 & W^1 & W^6 & W^4 & W^2 \\ W^0 & W^6 & W^5 & W^4 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \end{bmatrix} \tag{20}$$

The superscript in each W^{nk} is evaluated mod 6. Noting that $W^0 = 1$ and separating the equation for $X(0)$ we can rewrite equation (20) as:

$$\begin{bmatrix} X(1) - x(0) \\ X(2) - x(0) \\ X(3) - x(0) \\ X(4) - x(0) \\ X(5) - x(0) \\ X(6) - x(0) \end{bmatrix} = \begin{bmatrix} W^1 & W^2 & W^3 & W^4 & W^5 & W^6 \\ W^2 & W^4 & W^6 & W^1 & W^3 & W^5 \\ W^3 & W^6 & W^2 & W^5 & W^1 & W^4 \\ W^4 & W^1 & W^5 & W^2 & W^6 & W^3 \\ W^5 & W^3 & W^1 & W^6 & W^4 & W^2 \\ W^6 & W^5 & W^4 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \end{bmatrix} \tag{21}$$

and

$$X(0) = \sum_{n=0}^{6} x(n) \tag{22}$$

The expression for $X(0)$ i.e equation (22) is a simple summation and is assumed to be evaluated separately.

Dealing with the computationally intensive part of the transform i.e. equation (21), we can apply the mapping given by (13) to this equation which would convert equation 21 into a cyclic correlation suitable for implementation using IMS A100 transversal filter. We choose $r = 3$ which is the smallest primitive root of 7. The mapping would thus correspond to that given by table 9.1. We first apply the permutation given by this mapping to the input sequence of $x(n)'s$. This would correspond to a column permutation of the twiddle matrix as shown in (23).

$$\begin{bmatrix} X(1) - x(0) \\ X(2) - x(0) \\ X(3) - x(0) \\ X(4) - x(0) \\ X(5) - x(0) \\ X(6) - x(0) \end{bmatrix} = \begin{bmatrix} W^1 & W^3 & W^2 & W^6 & W^4 & W^5 \\ W^2 & W^6 & W^4 & W^5 & W^1 & W^3 \\ W^3 & W^2 & W^6 & W^4 & W^5 & W^1 \\ W^4 & W^5 & W^1 & W^3 & W^2 & W^6 \\ W^5 & W^1 & W^3 & W^2 & W^6 & W^4 \\ W^6 & W^4 & W^5 & W^1 & W^3 & W^2 \end{bmatrix} \begin{bmatrix} x(1) \\ x(3) \\ x(2) \\ x(6) \\ x(4) \\ x(5) \end{bmatrix} \tag{23}$$

Note that the matrix equations (23) and (21) are essentially the same and their difference is only in the order of the terms. Next we apply the same mapping to the column matrix containing $X(k) - x(0)$ terms in equation

(23), this would of course correspond to a similar permutation of the rows of twiddle matrix in (23); and the result is given as equation (24).

$$\begin{bmatrix} X(1)-x(0) \\ X(3)-x(0) \\ X(2)-x(0) \\ X(6)-x(0) \\ X(4)-x(0) \\ X(5)-x(0) \end{bmatrix} = \begin{bmatrix} W^1 & W^3 & W^2 & W^6 & W^4 & W^5 \\ W^3 & W^2 & W^6 & W^4 & W^5 & W^1 \\ W^2 & W^6 & W^4 & W^5 & W^1 & W^3 \\ W^6 & W^4 & W^5 & W^1 & W^3 & W^2 \\ W^4 & W^5 & W^1 & W^3 & W^2 & W^6 \\ W^5 & W^1 & W^3 & W^2 & W^6 & W^4 \end{bmatrix} \begin{bmatrix} x(1) \\ x(3) \\ x(2) \\ x(6) \\ x(4) \\ x(5) \end{bmatrix} \tag{24}$$

Referring to equation (24) it can be seen that the twiddle-factor matrix has the property that each row can be obtained by a left-shift (rotate) of the previous row. This means that the sequence $\{X(1)-x(0),\ X(3)-x(0),\ X(2)-x(0),\ X(6)-x(0),\ X(4)-x(0),\ X(5)-x(0)\}$ can be obtained by performing a circular convolution between the sequence $\{x(1), x(3), x(2)x(6), x(4), x(5)\}$ and a permuted twiddle factor set given by $\{W^1, W^3, W^2, W^6, W^4, W^5\}$.

Figure 9.3 shows how this circular convolution can be implemented using a transversal filter structure. For the moment let us confine our attention to the canonical transversal filter structure and assume that the transversal filter is capable of complex processing i.e. both input data $x(n)$ and the twiddle factors are complex. It will be shown later how a single IMS A100 device can be used to implement this complex processing. Two implementations are shown in figure 9.3.

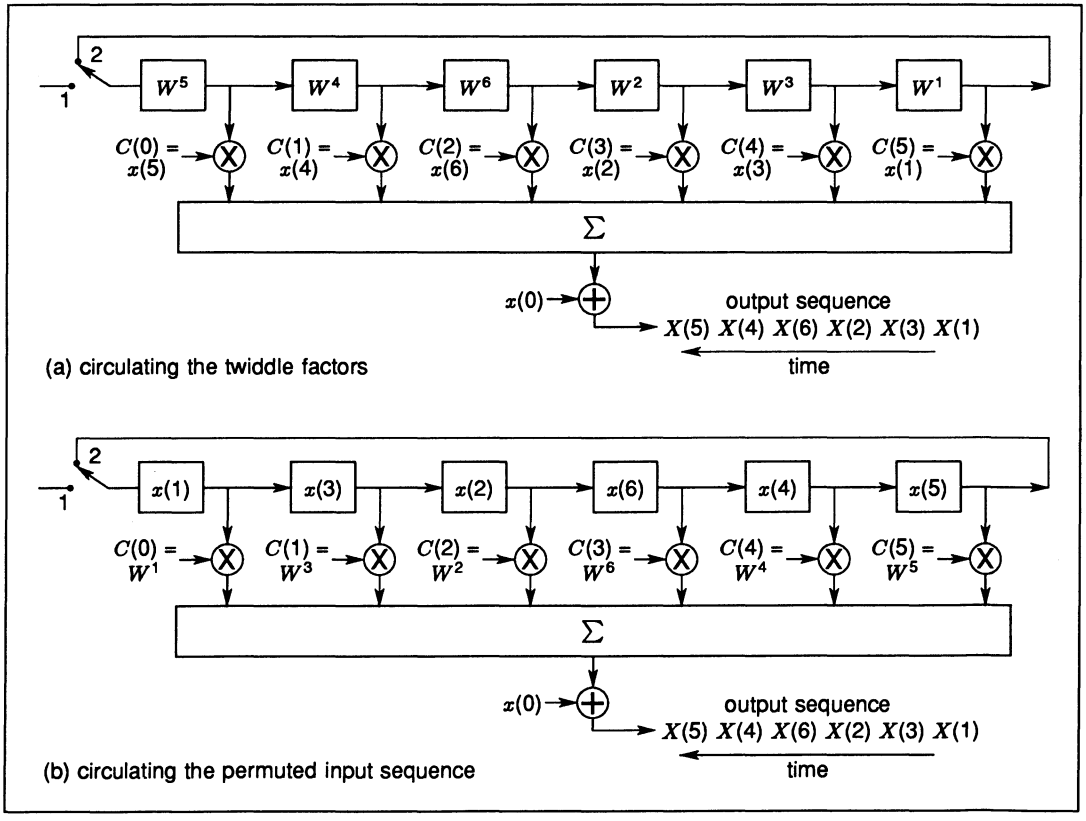


Figure 9.3 Prime number transform implementation based on the transversal filter structures

In the first implementation, figure 9.3a, the permuted twiddle factors are used as the inputs to the transversal filter. These twiddle factors are first loaded into the filter with the input switch at position 1 and then circulated

with the input switch at position 2. The output samples, as shown in figure 9.3a, would correspond to the DFT of the input sequence $x(n)$. You should be able to confirm this by referring to the matrix equation given by (24). For the arrangement in figure 9.3a the allocation of the data sequence $x(n)$ to the coefficient memory can be formulated as:

$$C(i) = x[(r^{N-2-i}) mod\ N] \quad i = 0, 1, ..., N-2 \tag{25}$$

where N is 7 in this case. Similarly the twiddle factor sequencing, at the input of figure 9.3a, can be mathematically expressed as:

$$Input(i) = W^{[(r^i) mod\ N]} \quad i = 0, 1, ..., N-2 \tag{26}$$

The output sequence for figure 9.3a is given by:

$$Output(i) = X[(r^i) mod\ N] \quad i = 0, 1, ..., N-2 \tag{27}$$

Figure 9.3b shows the second possible implementation in which the coefficient memory contains the permuted twiddle factors and the permuted data sequence is loaded at the input of the filter and is circulated to generate the DFT of the input samples. For this implementation the generalized equations for the input and output sequences and the allocation of coefficient memory are:

$$C(i) = W^{[(r^i) mod\ N]} \quad i = 0, 1, ..., N-2 \tag{28}$$

$$Input(i) = x[(r^{N-2-i} mod\ N] \quad i = 0, 1, ..., N-2 \tag{29}$$

$$Output(i) = X[(r^i) mod\ N] \quad i = 0, 1, ..., N-2 \tag{30}$$

Equation (25) to (27) and (28) to (30) define the required permutation and sequencing for a generalised prime number transform based on the canonical transversal filter structure.

It was argued earlier that the IMS A100 implementation of the transversal filter structure (figure 9.1b) is identical in behaviour to that of the canonical form (figure 9.1a). The only difference is that in the canonical form the first coefficient, $C(0)$, is associated with the left most memory location (see figure 9.1a) while in the IMS A100 implementation the right most coefficient register is allocated to $C(0)$. We will now show how our 7-Point DFT can be implemented in complex form using the IMS A100 transversal filters. The input data samples $x(n)$, the twiddle factors W^n, and the DFT output samples $X(n)$ can be expressed in terms of their real and imaginary parts as:

$$x(n) = xr(n) + jxi(n) \tag{31}$$

$$W_N^n = e^{-2\pi/N} = \cos(\frac{2\pi n}{N}) + j\sin(\frac{-2\pi n}{N}) = WR(n) + jWI(n) \tag{32}$$

$$X(n) = XR(n) + jXI(n) \tag{33}$$

As mentioned earlier the IMS A100 device contains two sets of coefficient memories; at any instant one set of the coefficients is used in the computation whilst the other set can be accessed via a standard memory interface. One very important feature of the device is that the two memory banks can be exchanged automatically at the beginning of every computation cycle. i.e. alternate set of coefficients are applied to the filter successively. This feature allows complex convolution and correlation to be performed in a single device. (This is unlike most conventional realizations of complex convolution/correlation where, as shown in figure 9.4, four transversal filters are often used to implement these complex functions). This is achieved by appropriately loading the two coefficient memories with combinations of real and imaginary samples of the reference signal and using the continuous memory-swap mode to implement complex processing. The real and imaginary parts of the signal to be correlated (or convolved) with the reference signal are then applied alternatively to the input of the IMS A100 device. An application note entitled 'Complex Processing Using the IMS A100 Transversal Filters' covers this topic in detail and is available from INMOS. The remainder of this section gives an overview of the topic in relation to complex DFTs.

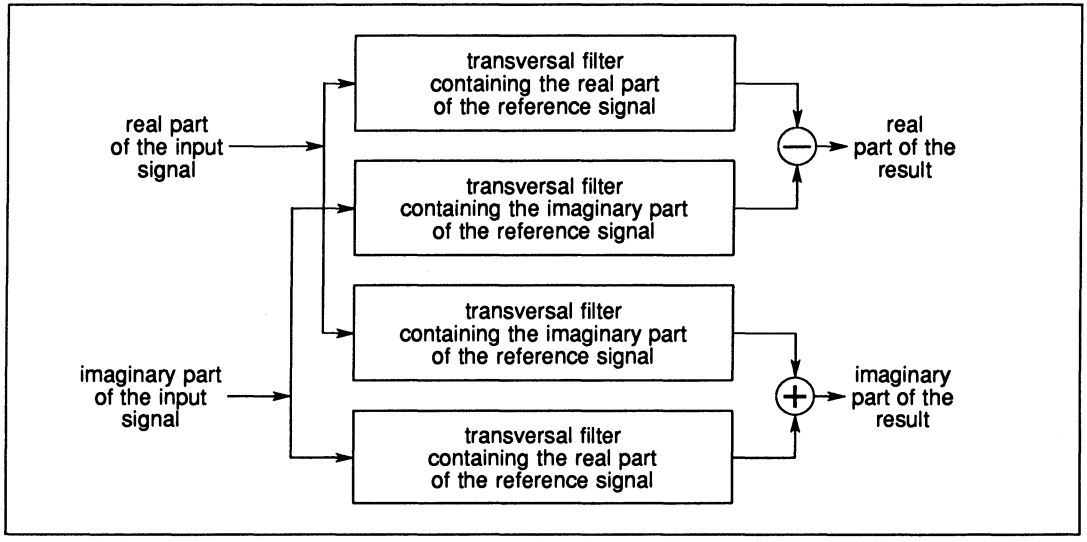


Figure 9.4 Conventional complex correlator/convolver involving four transversal filters

Figure 9.5 shows the IMS A100 implementation of the 7-point DFT example, corresponding to figure 9.3a, where the twiddle factors are circulated. The notation used for real and imaginary parts of the signals is that given by equations (31) to (33). The twiddle factors are applied to the input in a sequence identical to that used in figure 9.3a, with the real part of each sample followed by its imaginary part. The output sequence is also shown. It is assumed that the delay-and-add chain in the IMS A100 is cleared first by writing several zero's to the input. (Note that this is only needed once and any further transforms do not need this flushing). The memory banks are set in their continuous-swap mode. In the first computation cycle, with $WR(1)$ on the input, the memory bank 'A' is used in the computation; in the second cycle, when $WI(1)$ is applied to the input, the memory bank 'B' is used in the computation; in the third cycle $WR(3)$ is the input sample and the memory bank A is used in the computation and so on. Note also that for each output sample an external addition (with either $xr(0)$ or $xi(0)$ depending on whether the output corresponds to real or imaginary part of the result) has to be carried out as dictated by equation (24). This is a negligible overhead compared to the computation performed by the transversal filter and can easily be carried out by the host processor.

In figure 9.5 the first eleven output samples (denoted with $*$) are partial results and as such are not fully valid. This is due to the inherent delay associated with any transversal filter implementation. In continuous processing, however, it is possible to avoid these undefined output samples and to achieve a duty cycle of 100% by updating two coefficient memory locations with new data samples. For example in figure 9.5, assumming that we have applied the first cycle of the twiddle factor values i.e. $WR(1), WI(1), WR(3), WI(3)\ldots\ldots WR(5), WI(5)$ to the input, it is possible to update the coefficient locations corresponding to $xi(1)$ and $xr(1)$ in memory bank A with the imaginary and real parts of the first sample of the new input data block. This can be done during the latest computation cycle (with $WI(5)$ as the input twiddle factor) when the memory bank A is free. In the next cycle when $WR(1)$ is applied to the input, $xr(1)$ and $-xi(1)$ in the memory bank B can be updated with new data values while this memory bank is free. In the following computation cycle when $WI(1)$ is the input twiddle factor, the coefficient memory locations corresponding to $xi(3)$ and $xr(3)$ in the memory bank A are updated and so on. This technique removes the undefined output samples and achieves a duty cycle of 100%.

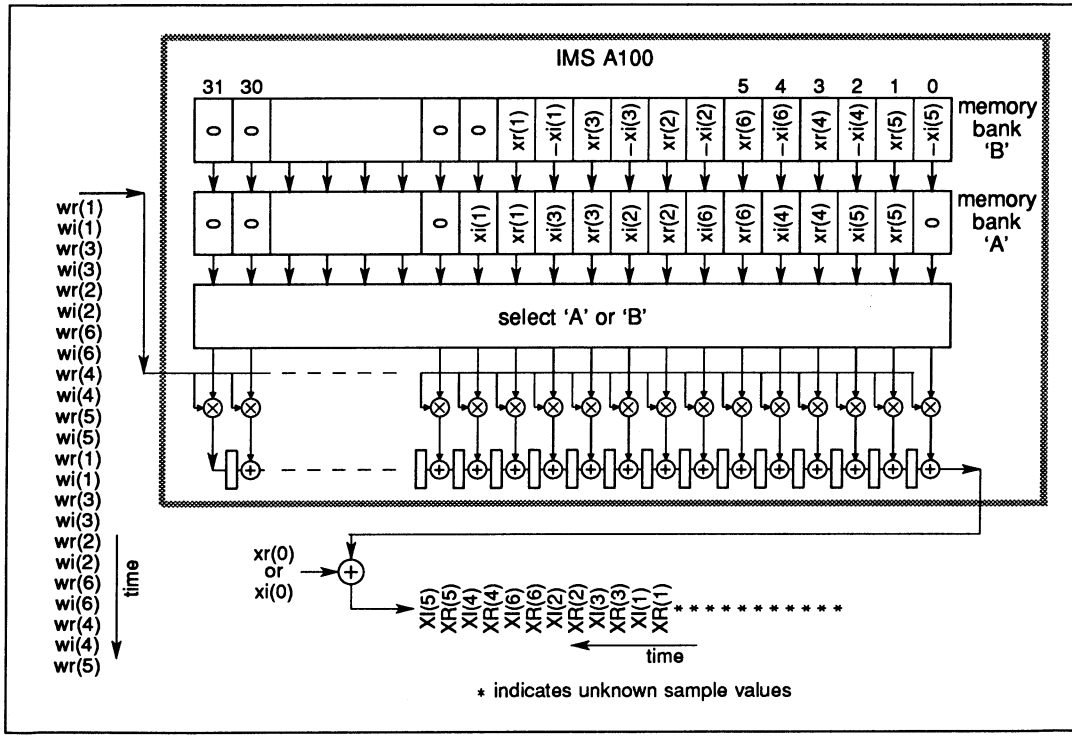


Figure 9.5 IMS A100 implementation of a 7 point complex DFT, corresponding to the canonical transversal filter realization of figure 9.3a

Figure 9.6 depicts the IMS A100 implementation of the 7-point DFT, corresponding to figure 9.3b, where the input data samples are recirculated and the twiddle factors are stored in the coefficient memories. The input data samples are applied to the input in a sequence identical to that used in figure 9.3b, with real part of each sample followed by its imaginary part. The output sequence is also shown. Other characteristics of this implementation are identical to the previous case with the exception that in this implementation it is impossible to avoid initial undefined output samples even when several continuous transforms are to be performed.

The IMS A100 devices can be cascaded without any external components by simply connecting the output of the first device to the cascade-input of the second device. This simple cascading allows transversal filters/correlators with many stages to be easily implemented.

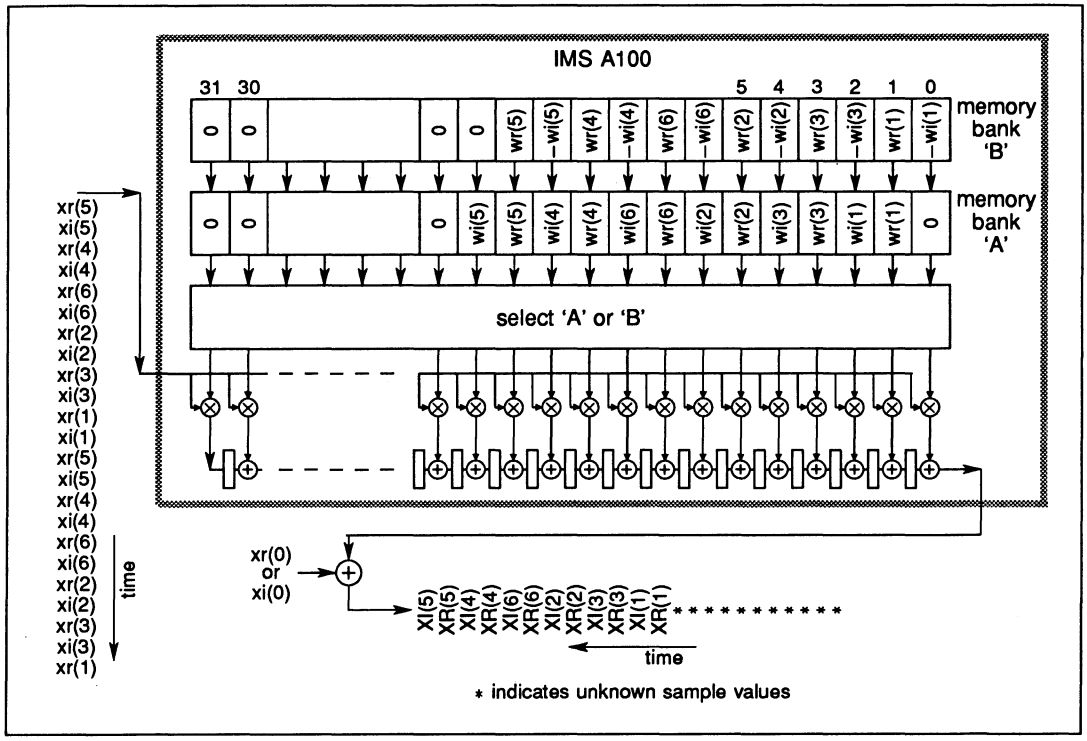


Figure 9.6 IMS A100 implementation of a 7 point complex DFT corresponding to the canonical transversal filter realization of figure 9.3b

Using prime-number algorithm there are basically two ways to implement A100 based DFT processors capable of handling long data blocks. The obvious approach is to cascade several devices resulting in a sufficiently large correlator/convolver capable of dealing with the whole data block size. This approach is only acceptable for moderate block sizes and becomes impractical if the data size is very large. The second approach is based on mapping techniques which convert a large DFT into several independent short transforms. These short transforms can then be evaluated either concurrently or sequentially, depending on the required performance. This means that the decomposition techniques described here are particularly useful as they provide the basis for trade-offs between cost and speed. This subject will be discussed in detail in section 5 of this application note.

As mentioned earlier the IMS A100 transversal filter has an on-chip industry standard memory interface which allows the part to be fully memory-mapable. Figure 9.7 shows a schematic diagram of a simple system making use of this memory interface. When implementing the prime transform algorithm on this system, the IMS A100 (or arrays of them) will perform the bulk of the computation and the host processor will be responsible for data permutation (using look-up tables), evaluating the X(0) term (equation 22), and performing the auxiliary addition involving either $xr(0)$ or $xi(0)$ (see figures 9.5–9.6).

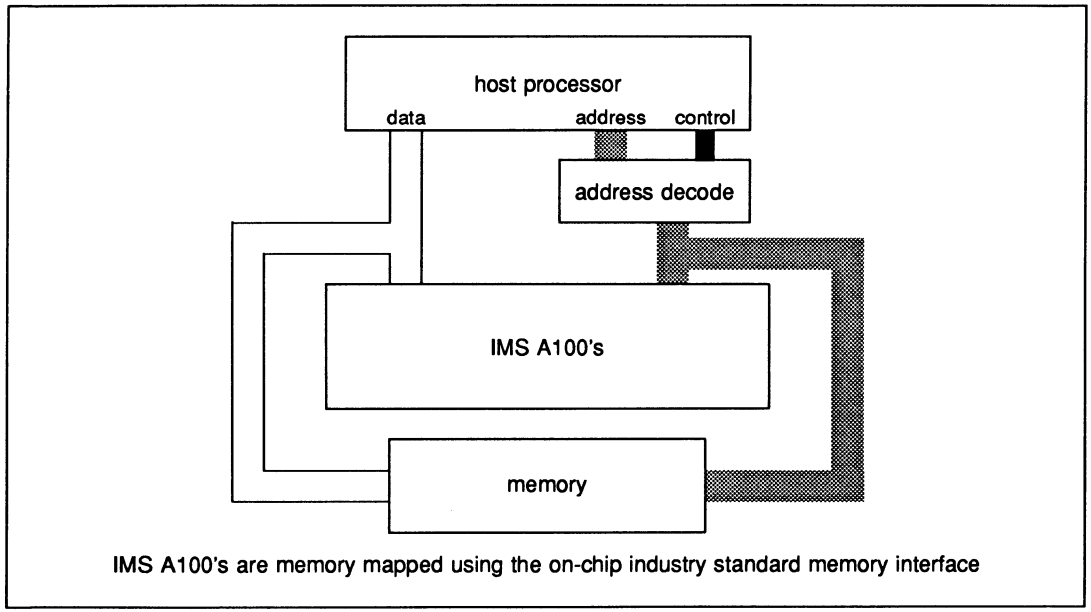


Figure 9.7 Schematic diagram of a simple IMS A100 based system

Another possible system configuration is shown in figure 9.8. This is particularly suitable for the arrangement of figure 9.5. The real and imaginary parts of the twiddle factors are pre-loaded in the memory MEM1 and are supplied to the A100 via the dedicted input port. The sequencer shown in figure 9.8 could be a simple counter. The processor accesses the coefficient memories and the output result via the IMS A100's memory interface. Other system configurations are possible. For example figure 9.9 shows the schematic of a high performance signal processing system using a dedicated controller.

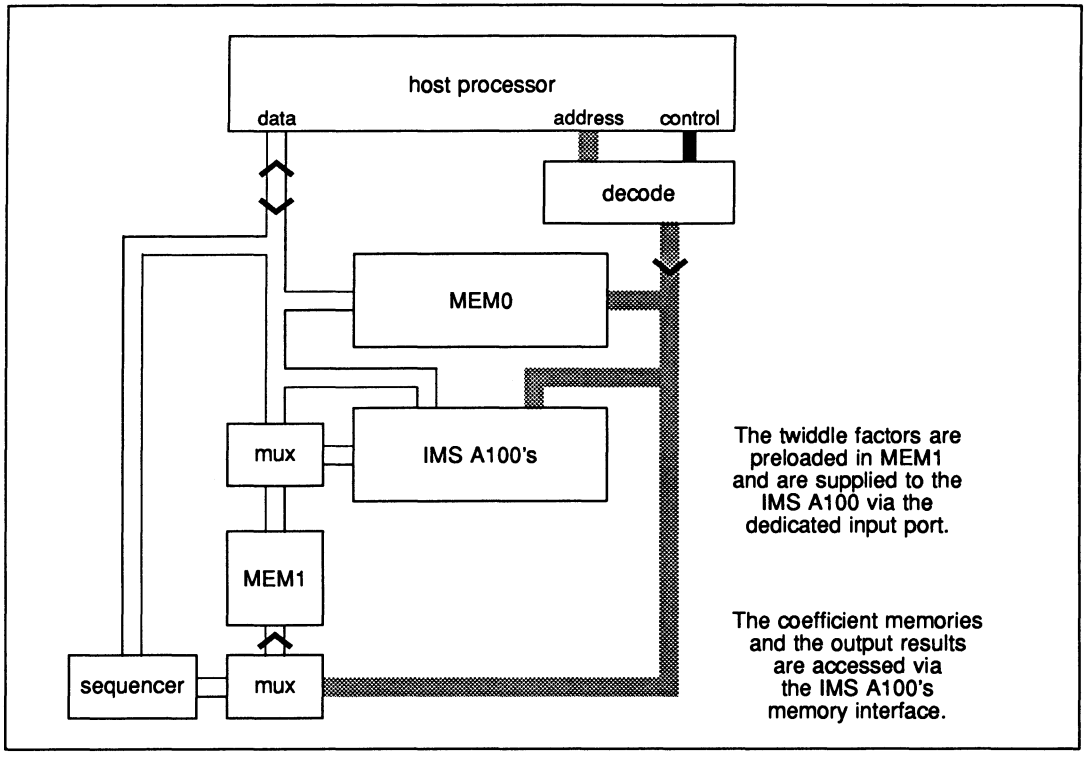


Figure 9.8 An implementation particularly suitable for the arrangement in figure 9.5

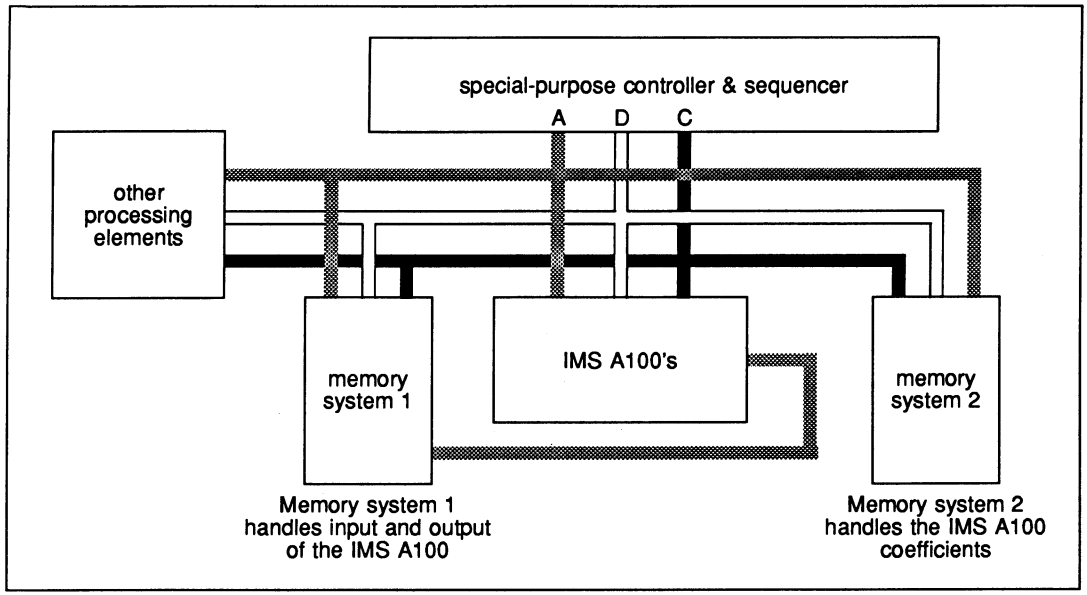


Figure 9.9 Schematic block diagram of a high performance involving a special purpose controller

9.4.2 The chirp-z transform

Another algorithm which converts the DFT equation into a convolution (or correlation) is the chirp-z transform. The reason for the name chirp is that the transform uses a sampled linear frequency-modulated carrier which in signal processing is often termed a 'chirp signal'. In the previous section we saw that the prime number transform consisted of three operations, namely

(i) Data input permutation.

(ii) Convolution (or correlation) of the permuted data with a permuted sequence of twiddle factors.

(iii) A final permutation to obtain the correct output sequence.

Figure 9.10 summarizes the principles of the prime number transform algorithm. The auxillary computation for zeroth input sample and evaluation of $X(0)$ are also shown in this schematic diagram. The structure of the chirp-z DFT algorithm is similar to that of the prime number transform technique and consists of the following sequence of three operations:

(i) Premultiplication of the input sequence $x(n)$ by the chirp $e^{(-\pi jn^2/N)}$.

(ii) Convolution (or correlation) of the resulting sequence with a second chirp signal.

(iii) Post-multiplication of the resulting sequence by the chirp signal $e^{(-\pi jk^2/N)}$.

These operations are summarized in figure 9.11. Comparing figure 9.10 and figure 9.11, it can be seen that the major difference between the two algorithms is that in the chirp-z transform the permutation operations are replaced by multiplications.

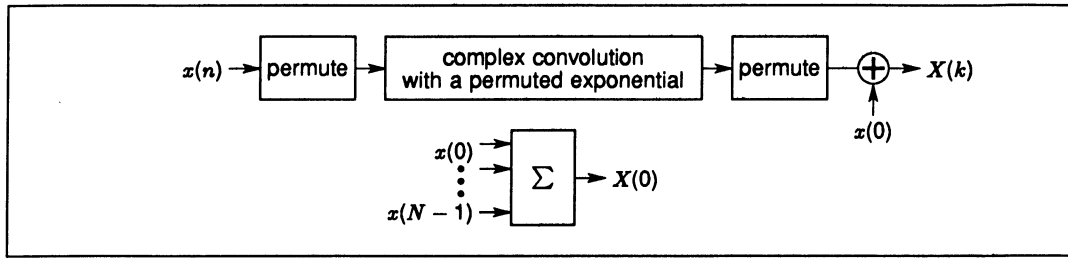


Figure 9.10 The principle of the prime number transform

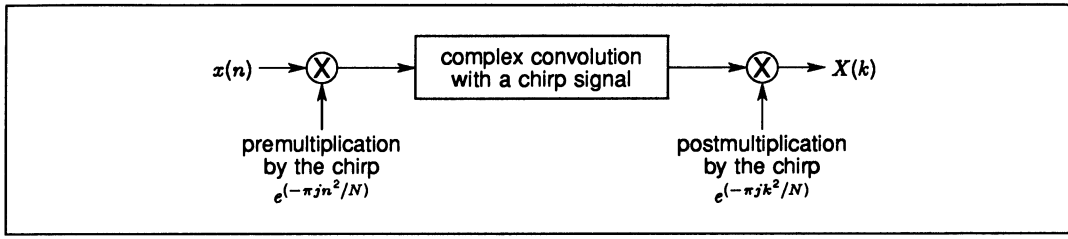


Figure 9.11 The basic principle of the CZT

In the CZT the convolution (correlation) operations can be implemented using the IMS A100 transversal filter, but the pre- and post-multiplications have to be done externally. In many applications data permutation may be preferred to multiplication in which case the prime numbers approach may be considered more advantageous. However there are applications (e.g beamforming) where the CZT, in particular a simplified version of it referred to as sliding CZT, is preferred.

To understand the CZT algorithm we start from the DFT equation

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-2\pi jnk/N} = \sum_{n=0}^{N-1} x(n)W_N^{nk} \qquad k = 0, 1, \ldots N-1 \tag{34}$$

and replace the term $-2nk$ in the complex exponential with the seemingly more complicated expression

$$-2nk = (k-n)^2 - k^2 - n^2 \tag{35}$$

hence

$$X(k) = e^{-j\pi k^2/N} \sum_{n=0}^{N-1} [x(n)e^{-j\pi n^2/N}][e^{j\pi(k-n)^2/N}] = e^{-j\pi k^2/N} \sum_{n=0}^{N-1} y(n)[e^{j\pi(k-n)^2/N}] \tag{36}$$

where $k = 0, 1, \ldots\ldots, N-1$.

In equation 36, the term $X(k)$ consists of three operations:

(i) Multiplications of the samples $X(n)$ with a complex linear frequency-modulated signal $e^{-j\pi n^2/N}$ to form a new set of samples $y(n)$; This operation is often referred to as premultiplication by a chirp,

(ii) the convolution of $y(n)$ with a second-linear frequency-modulated signal (the term $e^{j\pi(k-n)^2/N}$ and

(iii) post multiplication by $e^{-j\pi k^2/N}$.

Note that if only the power spectrum of the signal is required the final operation can be omitted, since $e^{-j\pi k^2/N}$ represent only a phase-shift and $|e^{-j\pi k^2/N}| = 1$. Also in operation (ii) the term $(k-n)^2$ in the complex exponential is equal to $(n-k)^2$ so that a convolution operation, in this case, is identical to that of a correlation. Figures 9.12 and 9.13 show examples for a 6-point CZT implemented using the canonical transversal filter structure. In these diagrams it is assumed that the transversal filter is capable of complex processing. As described in the previous section the complex conolution/correlation can easily be implemented using the IMS A100 transversal filter chip.

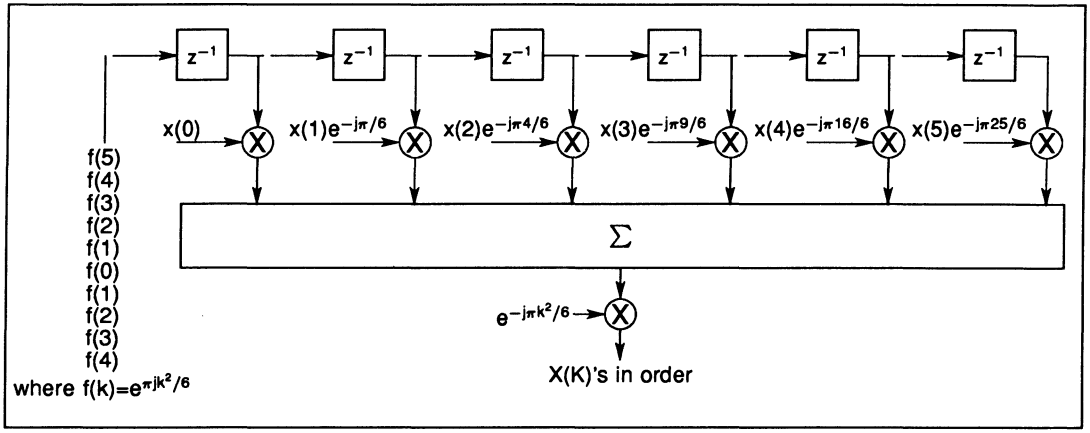


Figure 9.12 Schematic diagram for 6 point CZT using canonical transversal filter structure

In figure 9.12, samples corresponding to the product of the input data and the premultiplying chirp are stored as the filter coefficients and a second sampled chirp is fed to the input of the convolver. Note that the convolver has N-complex points. Figure 9.13 shows an alternative implementation where the product of the input data samples and the premultiplying chirp samples are the inputs to the convolver and a chirp signal is used as the reference signal in the coefficient store. Note that in this arrangement the convolver has to have $2N-1$ complex points. However when the number of points in the transform are even (N =even), it can easily be shown that the sampled chirp signal $f(n) = e^{j\pi n^2/N}$ has the following properties:

(i) it is periodic with a periodicity equal to N i.e.

$$f(n) = f(n+N) \tag{37}$$

(ii) It is symmetrical about $n = \frac{N}{2}$ i.e.

$$f(n) = f(N-n) \tag{38}$$

These properties convert the convolution in figure 9.13 into a circular one which can be implemented with an N-point complex transversal filter.

In many applications the PNT may be preferred to the CZT because of the requirement of pre- and post-multiplications in the latter. (Remember that in the PNT, permutation operations are involved rather than multiplications).

As there are considerable amount of literature available on CZT, it will not be considered here in any more detail.

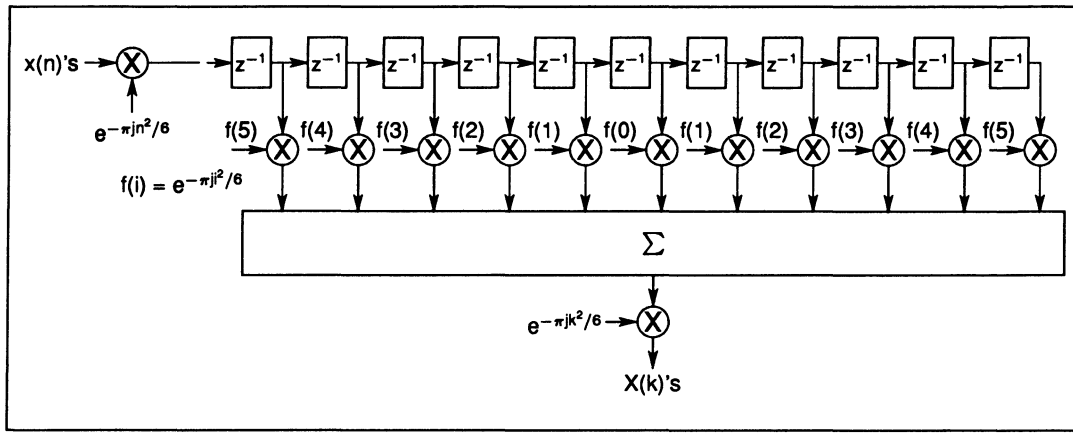


Figure 9.13 Alternative implementation of the 6 point CZT

9.5 Multidimensional index mapping for DFT decomposition

In the previous sections, algorithms which convert the DFT into convolution/correlation operations were presented. These algorithms, particularly the PNT, are suitable for implementation using the IMS A100 transversal filter.

In order to compute the DFT of long data sequences, one approach is to cascade several the IMS A100 devices so that sufficient convolution/correlation points are made available. This approach is only acceptable for moderate data sizes, and does not provide the optimum performance for a given number of devices.

In this section, index mapping techniques are described which allow long DFT's to be decomposed into several shorter transforms. These shorter transforms can then be efficiently implemented by using IMS A100 transversal filters. The decomposition techniques described here can be viewed as generalised algorithms, with radix-2 FFT being a special case of these more general partitioning techniques. These mapping techniques provide the basis for designing highly concurrent systems and optimization in terms of performance and cost.

9.5.1 Basic concepts of index mapping

The essence of these mapping techniques is that by a simple change of variable, the original complex problem is converted into several easy ones. Before applying these techniques to the DFT, let us consider a few examples which should help to familiarize the reader with the terminologies used in general index mapping.

Consider a one-dimensional array of data $x(n)$, $n = 0$ *to* $N-1$, where N is the total number of elements in the array. For $N = 6$, the array elements will be given by

$$\{x(0)\ x(1)\ x(2)\ x(3)\ x(4)\ x(5)\}$$

Let us rearrange this one-dimensional array into a two-dimensional array as shown below:

$$\begin{bmatrix} x(0) & x(1) & x(2) & x(3) & x(4) & x(5) \end{bmatrix} \overset{map}{\Longleftrightarrow} \begin{bmatrix} x(0) & x(1) & x(2) \\ x(3) & x(4) & x(5) \end{bmatrix} = \begin{bmatrix} y(0,0) & y(0,1) & y(0,2) \\ y(1,0) & y(1,1) & y(1,2) \end{bmatrix} \quad (39)$$

We have 'mapped' the original one-dimensional array, $x(n)$, into a two-dimentional array $y(n_1, n_2)$. It can be seen that in this example the mapping is given by the following linear equation.

$$n = 3n_1 + n_2 \quad , \quad x(n) = y(n_1, n_2) \tag{40}$$

where $n_1 = 0, 1$ and $n_2 = 0, 1, 2$.

The mapping is said to be one-to-one (unique) as all the elements in the original array, $x(n)$, appear in the two-dimensional array $y(n_1, n_2)$.

As a second example consider the following mapping:

$$\begin{bmatrix} x(0) & x(1) & x(2) & x(3) & x(4) & x(5) \end{bmatrix} \overset{map}{\Longleftrightarrow} \begin{bmatrix} x(0) & x(2) & x(4) \\ x(3) & x(5) & x(1) \end{bmatrix} = \begin{bmatrix} y(0,0) & y(0,1) & y(0,2) \\ y(1,0) & y(1,1) & y(1,2) \end{bmatrix} \tag{41}$$

This mapping has been obtained from

$$n = (3n_1 + 2n_2) \ mod \ 6 \tag{42}$$

Note that in equation (42) the index n is evaluated modulo N (in this case N=6) and it is therefore cyclic in N.

As a final example let us apply the following mapping

$$n = (2n_1 + 2n_2) \ mod \ 6 \tag{43}$$

to our 6-element array, which gives;

$$\begin{bmatrix} x(0) & x(1) & x(2) & x(3) & x(4) & x(5) \end{bmatrix} \overset{map}{\Longleftrightarrow} \begin{bmatrix} x(0) & x(2) & x(4) \\ x(2) & x(4) & x(0) \end{bmatrix} = \begin{bmatrix} y(0,0) & y(0,1) & y(0,2) \\ y(1,0) & y(1,1) & y(1,2) \end{bmatrix} \tag{44}$$

Obviously this map is not unique as $x(1), x(3)$ and $x(5)$ are not represented in the matrix $y(n_1, n_2)$.

9.5.2 Generalisation and conditions for uniqueness

Let us now generalize the ideas developed in the previous examples. We are interested in mapping a one-dimensional array of length $N = N_1 \times N_2$ into a two-demensional array that is N_1, by N_2 in size. In other words, the one-dimensional array

$$x(n) \ \ for \quad n = 0, 1, ...N - 1$$

is to be mapped into a two-dimensional array

$$y(n_1, n_2) \ \ for \ \ n_1 = 0, 1, ...N_1 - 1, \ and \ n_2 = 0, 1, ...N_2 - 1.$$

The major requirement is that the mapping must be unique. This mapping can be represented by

$$\begin{bmatrix} x(0) & x(1) & x(2) & \dots & x(N-1) \end{bmatrix} \overset{map}{\Longleftrightarrow} \begin{bmatrix} y(0,0) & y(0,1) & \dots & y(0,N_2-1) \\ y(1,0) & y(1,1) & \dots & y(1,N_2-1) \\ \vdots & \vdots & \ddots & \vdots \\ y(N_1-1,0) & y(N_1,1) & \dots & y(N_1-1,N_2-1) \end{bmatrix} \tag{45}$$

where

$$x(n) = y(n_1, n_2) \tag{46}.$$

This map, in general, can assume many different forms, but the one particularly useful to the DFT is the linear form,

$$n = (M_1 n_1 + M_2 n_2) \ mod \ N. \tag{47}$$

Note that n is evaluated modulo N, making the map cyclic in n. In order for this map to be unique and one-to-one, the mapping constants M_1 and M_2 must satisfy certain conditions. In the literature (references 5 & 6) these conditions have been derived from number theory for two cases which are described in the following two subsections.

Relatively prime case

In this case N_1 and N_2 are relatively prime and have no common factor. In the literature this case is often denoted by:

$$(N_1, N_2) = 1 \tag{48}$$

which means that the greatest common divisor of N_1 and N_2 is unity. For example $(5,7) = 1$, $(8,9) = 1$ and $(6,25) = 1$. For this case the conditions on M_1 and M_2 which make the mapping, given by (47), unique and one-to-one are: (references 5 & 6)

$$[(M_1 = \alpha N_2) \; and/or \; (M_2 = \beta N_1)] \; and \; (M_1, N_1) = (M_2, N_2) = 1 \tag{49}$$

where α and β are integers.In other words, to ensure a unique maping for this case:

(M_1 must be a multiple of N_2)
or (M_2 must be a multiple of N_1)
or (M_1 and M_2 must be multiples of of N_2 and N_1 respectively)

and M_1 and N_1 must be relatively prime.

and M_2 and N_2 must be relatively prime.

As an example consider $N_1 = 5$, $N_2 = 7, N = 35$. From (49) we have to choose M_1 a multiple of N_2 or M_2 a multiple of N_1 or both. Let us make M_1 the simplest multiple of N_2 i.e. $M_1 = \alpha N_2 = N_2 = 7$, this also satisfies $(M_1, N_1) = (7,5) = 1$. Then noting that we must have $(M_2, N_2) = (M_2, 7) = 1$, possible values for M_2 are: $1,2,3,4,5,6,8,9,10,11,12,13,15$. While $M_2 = 7,14,21...$ are not allowed as they are not relatively prime with N_2. (Note also that for $M_1 = 5,10,15$, we also have $M_2 = \alpha N_1$, which is allowed.)

If M_1 is chosen to be $M_1 = 2 \times N_2 = 14$, then again the same values of M_2 as above are valid.

This example shows that a large class of unique mappings exist for this case.

Common factor case

In this case N_1 and N_2 have a common factor r. i.e. their greatest common divisor is r and we have:

$$(N_1, N_2) = r \tag{50}$$

For example $(10,5) = 5$, $(9,12) = 3$, and $(7,21) = 7$. For this case the conditions on M_1 and M_2, making the mapping given by (47) unique, have been shown to be: (references 5 & 6)

$$\textbf{1)} \quad (M_1 = \alpha N_2) \; and \; (M_2 \neq \beta N_1) \; and \; (\alpha, N_1) = (M_2, N_2) = 1 \tag{51a}$$

or

$$\textbf{2)} \quad (M_1 \neq \alpha N_2) \; and \; (M_2 = \beta N_1) \; and \; (\beta, N_2) = (M_1, N_1) = 1 \tag{51b}$$

where α and β are integers.

As an example consider $N_1 = 9, N_2 = 15, N = 135$. From (51a) we can choose, $M_1 = \alpha N_2 = N_2 = 15$. The condition $(\alpha, N_1) = (1,9) = 1$ is already satisfied. From (51a), values of M_2 which are allowed are those which satisfy $(M_2 \neq \beta \times 9)$ and $(M_2, 15) = 1$. Therefore following values of M_2 are allowed

$$M2 = 1,2,4,7,8,11,13,14,16,17,19,22,.....$$

$M_2 = 3,5,6,9,10,...$ are not allowed as they do not satisfy $(M_2, 15) = 1$.

Alternatively we could have chosen $M_1 = \alpha N_2 = 2 \times 15 = 30$, then possible values of M_2 would again be

$$M_2 = 1,2,4,7,8,11,13,14,.....$$

However we could not have chosen $M_1 = \alpha N_2 = 3 \times N_2$ since this violates the requirement $(\alpha, N_1) = 1$.

9.5.3 Application of index mapping to DFT decomposition

Having covered the basic principle of index mapping and the required conditions for uniqueness, let us apply the mapping given by (47) to the DFT and investigate its consequences.

Remember that the DFT equation is given by

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-2\pi jnk/N} = \sum_{n=0}^{N-1} x(n)W_N^{nk} \qquad k = 0, 1, \ldots N-1 \tag{52}$$

and that the exponent of W is evaluated modulo N since

$$W_N^{nk} = W_N^{nk+N}. \tag{53}$$

Let us apply the following mappings to the indices n and k; i.e. to both the input data array $x(n)$, and the result array $X(k)$;

$$n = (M_1 n_1 + M_2 n_2) \ mod \ N \tag{54}$$

$$k = (L_1 k_1 + L_2 k_2) \ mod \ N \tag{55}$$

Where n_1, and k_1, are indexed to $(N_1 - 1)$ and n_2 and k_2 to $(N_2 - 1)$. As shown below these mappings convert the one dimensional arrays $x(n)$ and $X(k)$ into two-dimensional matrices $y(n_1, n_2)$ and $Y(n_1, n_2)$ respectively.

$$[\,x(0)\,x(1)\ldots x(n)\ldots x(N-1)\,] \overset{map}{\Longleftrightarrow} \begin{bmatrix} y(0,0) & y(0,1) & \ldots\ldots & \ldots y(0,N_2-1) \\ y(1,0) & y(1,1) & \ldots\ldots & \ldots y(1,N_2-1) \\ \vdots & \vdots & \ddots\ddots & \ddots\vdots \\ \vdots & \vdots & \ddots\, y(n_1,n_2) & \ddots\vdots \\ \vdots & \vdots & \ddots\ddots & \ddots\vdots \\ y(N_1-1,0) & y(N_1,1) & \ldots\ldots & \ldots y(N_1-1,N_2-1) \end{bmatrix}$$

where $y(n_1, n_2) = x(n)$.

And

$$[\,X(0)\,X(1)\,..\,X(k)\,..\,X(N-1)\,] \overset{map}{\Longleftrightarrow} \begin{bmatrix} Y(0,0) & Y(0,1) & \ldots\ldots & \ldots Y(0,N_2-1) \\ Y(1,0) & Y(1,1) & \ldots\ldots & \ldots Y(1,N_2-1) \\ \vdots & \vdots & \ddots\ddots & \ddots\vdots \\ \vdots & \vdots & \ddots\, Y(k_1,k_2) & \ddots\vdots \\ \vdots & \vdots & \ddots\ddots & \ddots\vdots \\ Y(N_1-1,0) & Y(N_1,1) & \ldots\ldots & \ldots Y(N_1-1,N_2-1) \end{bmatrix}$$

where $Y(k_1, k_2) = X(k)$.

Applying these mappings to the DFT equation gives:

$$X(L_1 k_1 + L_2 k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(M_1 n_1 + M_2 n_2) W_N^{nk} \tag{56}$$

or

$$Y(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} y(n_1, n_2) W_N^{nk} \tag{57}$$

with

$$W_N^{nk} = W_N^{M_2 L_2 n_2 k_2} \, W_N^{M_1 L_2 n_1 k_2} \, W_N^{M_1 L_1 n_1 k_1} \, W_N^{M_2 L_1 n_2 k_1} \tag{58}$$

Let us now partially define the maps in (54) and (55) by setting;

$$M_2 = \beta N_1 \quad and \quad L_1 = \gamma N_2 \tag{59}$$

Where β and γ are integers.
These assignments make the last term in (58) i.e $W_N^{M_2 L_1 n_2 k_1}$ equals to unity.Let us now separately consider each one of the two cases studied earlier and investigate the effect on the three remaining terms in (58).

Case 1 – prime factor decomposition

For this case N_1 and N_2 are assummed to be relatively prime. From (49) it can be seen that it is possible to also set:

$$M_1 = \alpha N_2 \quad and \quad L_2 = \delta N_1. \tag{60}$$

This makes the second term in (58) i.e. $W_N^{M_1 L_2 n_1 k_2}$ also equal to unity. The remaining two terms in (58) can be written as:

$$W_N^{M_2 L_2 N_2 k_2} = W_N^{\beta N_1 \delta N_1 n_2 k_2} = W_{N_2}^{\beta\delta N_1 n_2 k_2} \tag{61}$$

$$W_N^{M_1 L_1 N_1 k_1} = W_N^{\alpha N_2 \gamma N_2 n_1 k_1} = W_{N_1}^{\alpha\gamma N_2 n_1 k_1} \tag{62}$$

The DFT equation will therefore become:

$$Y(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \left[\sum_{n_2=0}^{N_2-1} y(n_1, n_2) W_{N_2}^{\beta\delta N_1 n_2 k_2} \right] W_{N_1}^{\alpha\gamma N_2 n_1 k_1} \tag{63}$$

where $k_1 = 0, 1, 2, ..., N_1 - 1$ and $k_2 = 0, 1, 2, ..., N_2 - 1$
The advantage of this equation is that it uncouples the DFT calculations in the sense that the N-point DFT can be mapped into two completely separate sets of short DFT's. In evaluating the $Y(k_1, k_2)$'s, in equation (63), the inner summations over n_2 are operations involving separate rows of the matrix $y(n_1, n_2)$. The outer summations over n_1, on the other hand, are column operations and can be carried out after the row operations are completed. By suitable choice of α, β, γ, and δ, each one of these summations can be expressed as a DFT of the corresponding row or column. Goods (reference 8) suggested:

$$\begin{aligned} \alpha &= \beta = 1 \\ \gamma &= (N_2^{-1}) \ mod \ N_1 \\ \delta &= (N_1^{-1}) \ mod \ N_2 \end{aligned} \tag{64}$$

[Note that in modulo arithmetic the reciprocal of a number $(g) \ mod \ N$ is denoted by $(g^{-1}) \ mod \ N$ and is defined as:

$$[(g) \ mod \ N][(g^{-1} \ mod \ N] = 1.$$

For example

$$(3^{-1}) \ mod \ 7 = (5) \ mod \ 7$$

since $5 \times 3 = 15$ which is 1 modulo 7.]

Applying (64) to (63) gives:

$$Y(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \left[\sum_{n_2=0}^{N_2-1} y(n_1, n_2) W_{N_2}^{n_2 k_2} \right] W_{N_1}^{n_1 k_1} = \sum_{n_1=0}^{N_1-1} \left[u(n_1, k_2) \right] W_{N_1}^{n_1 k_1} \tag{65}$$

This is now a true two-dimensional DFT with the mapping of n and k given by:

$$n = (N_2 n_1 + N_1 n_2) \ mod \ N \tag{66a}$$

$$k = \{[(N_2^{-1}) \ mod \ N_1] N_2 k_1 + [(N_1^{-1}) \ mod \ N_2] N_1 . k_2\} \ mod \ N \tag{66b}$$

In this example the mapping of n is of the simplest form and that of k is the so called Chinese Remainder Theorem (CRT).

Having mapped the original sequence $x(n)$ into the two dimensional array $y(n_1, n_2)$, equation (65) indicates that the desired DFT can be evaluated by the following two steps:

(i) Performing an N_2-point DFT on each row of matrix $y(n_1, n_2)$. This corresponds to a total of N_1 N_2-point row DFT's and would convert the matrix $y(n_1, n_2)$ into the matrix $u(n_1, k_2)$ as shown in figure 9.14.

(ii) Performing an N_1-point DFT on each column of the resultant matrix, $u(n_1, k_2)$, to yield $Y(k_1, k_2)$. The desired output sequence $X(k)$ is related to $Y(k_1, k_2)$ via the mapping given by (66b).

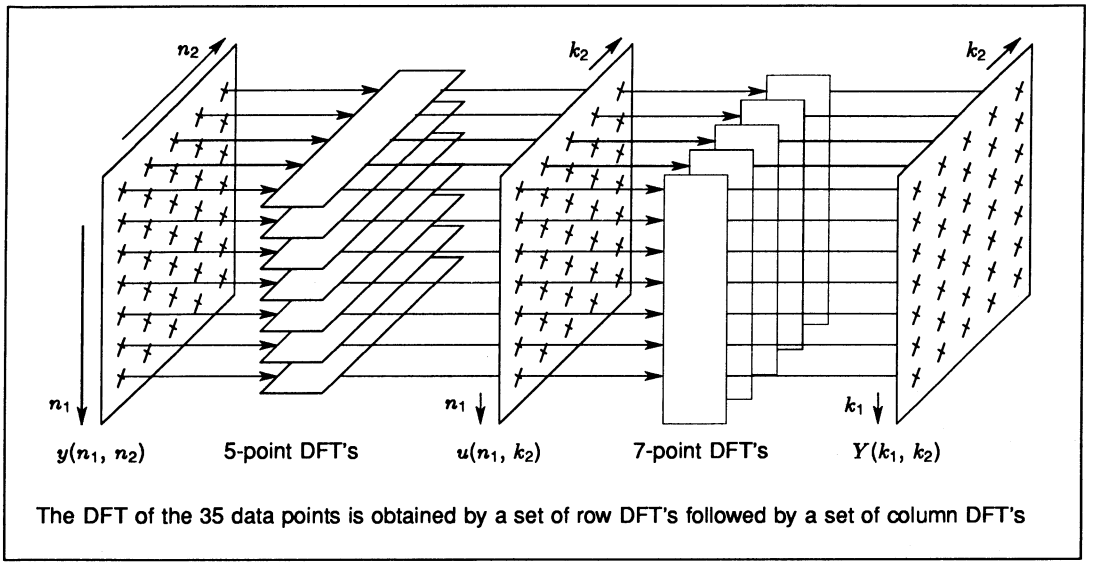


Figure 9.14 Schematic representation of equation 65 for $N = N_1 \times N_2 = 7 \times 5$

Example:

In this example we consider the evaluation of the DFT of a 35-element data array $x(n)$ ($n = 0$ *to* 34) via the mapping techniques discussed so far. Let us take $N = N_1 \times N_2 = 7 \times 5 = 35$ i.e. $N_1 = 7$ and $N_2 = 5$. The data array $x(n)$ is first mapped into a two dimensional array $y(n_1, n_2)$ via the mapping given by equation (66a) i.e.

$$n = (5n_1 + 7n_2) \; mod \; 35 \tag{67}$$

The array $y(n_1, n_2)$ would thus be as follows:

$$y(n_1, n_2) = \begin{bmatrix} x(0) & x(7) & x(14) & x(21) & x(28) \\ x(5) & x(12) & x(19) & x(26) & x(33) \\ x(10) & x(17) & x(24) & x(31) & x(3) \\ x(15) & x(22) & x(29) & x(1) & x(8) \\ x(20) & x(27) & x(34) & x(6) & x(13) \\ x(25) & x(32) & x(4) & x(11) & x(18) \\ x(30) & x(2) & x(9) & x(16) & x(23) \end{bmatrix}$$

The next step is to perform the DFT of each row of this matrix. Obviously in this example this involves seven 5-point DFT's as shown in figure 9.14. The result of these row DFT's is a new matrix denoted by $u(n_1, k_2)$.

Next the DFT of each column of the matrix u is evaluated. As shown in figure 9.14 this involves five 7-point DFT's and yields the matrix $Y(k_1, k_2)$. The two dimensional array $Y(k_1, k_2)$ contains desired DFT results $X(k)$, with the allocations governed by the mapping given by equation (66b) i.e. $X(k) = Y(k_1, k_2)$ with

$$\begin{aligned} k &= \{[(N_2^{-1}) \ mod \ N_1]N_2k_1 + [(N_1^{-1}) \ mod \ N_2]N_1.k_2\} \ mod \ N \\ k &= \{[3]\ 5\ k_1 + [3]\ 7\ k_2\} \ mod \ 35 \\ &= \{15k_1 + 21k_2\} \ mod \ 35 \end{aligned}$$

The array $Y(k_1, k_2)$ would therefore have the following arrangement:

$$Y(k_1, k_2) = \begin{bmatrix} X(0) & X(21) & X(7) & X(28) & X(14) \\ X(15) & X(1) & X(22) & X(8) & X(29) \\ X(30) & X(16) & X(2) & X(23) & X(9) \\ X(10) & X(31) & X(17) & X(3) & X(24) \\ X(25) & X(11) & X(32) & X(18) & X(4) \\ X(5) & X(26) & X(12) & X(33) & X(19) \\ X(20) & X(6) & X(27) & X(13) & X(34) \end{bmatrix}$$

In a practical implementation, the IMS A100 transversal filter can be used to perform these short row and column DFT's via the prime number transform algorithm described earlier. The important fact to note here is that each set of row (or column) DFT's consists of a number of totally independent short transforms (see figure 9.14). This allows various degrees of parallelism to be exploited very easily in acheiving the required specification.

For example a single A100 based DFT processor can be used to sequentially perform all the row DFT's followed by the column DFT's, or when extremeley high processing speed is essential, several such DFT processors can be employed in parallel to complete the independent row (or column) DFT's. In the extreme case, it is possible to compute all row and column DFT's concurrently in a pipelined system arrangement. The INMOS concurrent processor family (transputers) when combined with the IMS A100(s) provide an ideal environment for exploiting these algorithms.

In arriving at equation (65) we applied the conditions given by (64) to equation (63). This resulted in the mapping given by (66) on which the last example was based . It is possible to use other values for α, β, γ, and δ than those given by (64).

For example we could have used:

$$\begin{aligned} \gamma &= \delta = 1 \\ \alpha &= (N_2^{-1}) \ mod \ N_1 \\ \beta &= (N_1^{-1}) \ mod \ N_2 \end{aligned} \tag{68}$$

This would have resulted in the mappings for n and k to be interchanged i.e.

$$n = \{[(N_2^{-1}) \ mod \ N_1]N_2n_1 + [(N_1^{-1}) \ mod \ N_2]N_1.n_2\} \ mod \ N \tag{69a}$$

$$k = (N_2k_1 + N_1k_2) \ mod \ N \tag{69b}$$

Another interesting possibility is

$$\alpha = \beta = \gamma = \delta = 1 \tag{70}$$

This would result in the simple mapping for both n and k i.e.

$$n = N_2n_1 + N_1n_2 \tag{71a}$$

$$k = N_2k_1 + N_1k_2 \tag{71b}$$

but requires a modification in the W. We can see this by substituting (70) into (63) which gives:

$$Y(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \left[\sum_{n_2=0}^{N_2-1} y(n_1, n_2) W_{N_2}^{N_1n_2k_2} \right] W_{N_1}^{N_2n_1k_1} \tag{72}$$

By defining

$$W'_{N_2} = W_{N_2}^{N_1} = e^{-2\pi j N_1/N_2}$$

and

$$W'_{N_1} = W_{N_1}^{N_2} = e^{-2\pi j N_2/N_1}$$

Equation (72) can be rewritten as:

$$Y(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \left[\sum_{n_2=0}^{N_2-1} y(n_1, n_2) W'_{N_2}{}^{n_2 k_2} \right] W'_{N_1}{}^{n_1 k_1} \tag{73}$$

This equation is very similar to (65), with the exception that a modified W is used. Equation (73) also maps into an arrangement similar to figure 9.14. The DFT's of course, would have to be calculated with the modified W. This still can be done via the prime number transforms by simply replacing W with W' in the transform.

Case 2 – common factor decomposition

Having covered the case where N_1 and N_2 were relatively prime, we now go back to equation (58) and consider the case where N_1 and N_2 have a common factor r, i.e.

$$(N_1, N_2) = r$$

Remember that we applied (59) to (58) which made the last term in (58) equal to unity i.e. we have :

$$Y(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \left[\sum_{n_2=0}^{N_2-1} y(n_1, n_2) W_{N_2}^{\beta L_2 n_2 k_2} \right] W_N^{M_1 L_2 n_1 k_2} W_{N_1}^{M_1 \gamma n_1 k_1} \tag{74}$$

Unlike the previous case we cannot use equation (60) to make the second terms in (74) equal to unity. This is because the equations in (60) would violate the necessary requirement, specified by (51), for one-to-one mapping.

The term $W_N^{M_1 L_2 n_1 k_2}$ is referred to as a 'twiddle factor'. Referring to (51) and remembering that we have already used the conditions given by (59), we can set

$$M_1 = L_2 = \beta = \gamma = 1 \tag{75}$$

which gives

$$Y(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \left[\sum_{n_2=0}^{N_2-1} y(n_1, n_2) W_{N_2}^{n_2 k_2} \right] W_N^{n_1 k_2} W_{N_1}^{n_1 k_1} \tag{76}$$

with the mapping given by:

$$n = n_1 + N_1 n_2 \tag{77a}$$

$$k = N_2 k_1 + k_2 \tag{77b}$$

Note that equation (76) is very similar to equation (65) with the exception of the existence of the twiddle term. Equation (76) can be interpreted as shown in figure 9.15. It can be seen that when N_1 and N_2 have a common divisor, N-complex multiplications have to be performed between the row and column DFT operations. In the previous case where N_1 and N_2 were assumed to be relatively prime no such multiplications were needed making the former mapping more efficient and easier to implement.

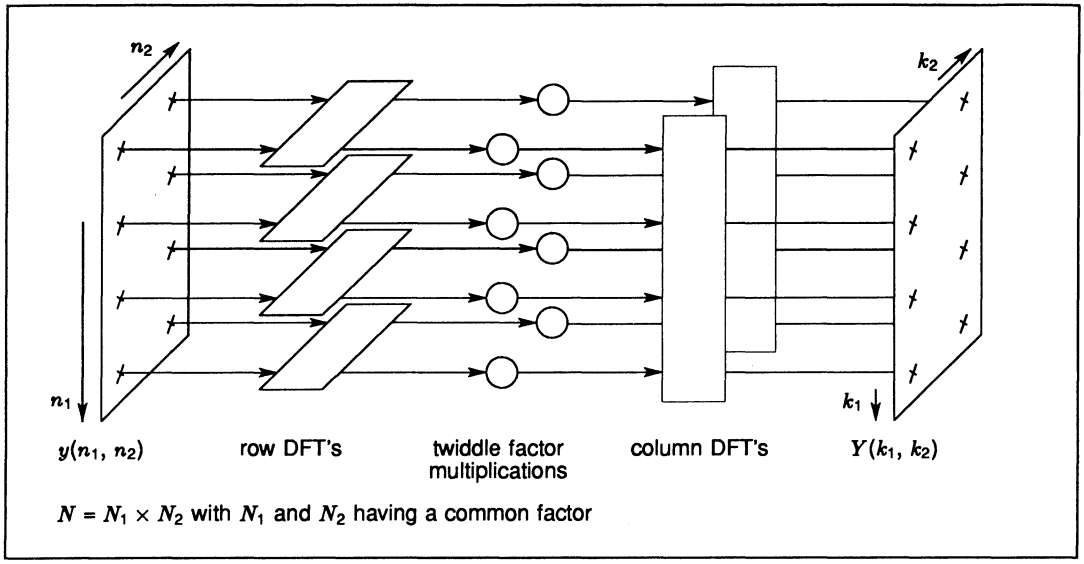


Figure 9.15 Mapping of an N point DFT into dimensions

9.5.4 Extension to multiple dimensions

The concepts presented in this application note were concentrated around a two-dimensional mapping. There is no reason why the same concepts cannot be extended to more dimensions. For example if

$$N = N_1 \times N_2 \times N_3$$

where N_1, N_2 and N_3 are relatively prime. The original N-point transform can be carried out via $N_2 \times N_3$, N_1-point, transforms followed by $N_1 \times N_3$, N_2-point, transforms followed by $N_1 \times N_2$, N_3-point, transforms. The easiest way to see this is to first map the N-point transform into a two-dimensional one with dimensions

$$N_1' = N_3 \qquad and \qquad N_2' = N_1 N_2$$

This consists of N_3, $N_1 \times N_2$-point, transforms (row DFT's) followed by $N_1 \times N_2$, N_3-point, transforms (column DFT's). Each one of the $N_1 \times N_2$-point transforms can then be decomposed into N_1, N_2-point, transforms followed by N_2, N_1-point, transforms.

Note that these multidimensional index mappings apply to both prime factor and common factor decompositions. In fact radix-2 FFT is nothing more than a common factor decomposition where all the factors N_1, N_2, N_3, N_4..... are made equal to 2. The advantage of the prime factor over the common factor decomposition is in that no twiddle matrix multiplications are needed for the prime factor case.

9.6 References

1 *An algorithm for the machine calculation of complex Fourier series*, Cooley J.W., Tukey J.W., Math. Comput., Vol.19, pp 297-301, April 1965.

2 *On computing the discrete Fourier transform*, Winograd S., Proc. Nat. Acad. Sci. USA, Vol.73, No. 4, pp. 1005-1006, April 1976.

3 *Discrete Fourier transforms when the number of data samples is prime*, Rader C.M., Proc. IEEE, Vol.56, pp 1107-1109, June 1968.

4 *The chirp z-transform algorithm and its application*, Rabiner L.R., et al, The Bell System Technical Journal, May-June 1969, pp 1249-1292.

5 *Index mappings for multidimensional formulation of the DFT an convolution*, Burrus C.S., IEEE Trans. on ASSP, Vol.25, June 1977, pp 239-242.

6 *DFT/FFT and convolutional algorithms-theory and implementation*, Burrus C.S., Parks T.W., Wiley-interscience Publication, New York 1985.

7 *Number theory in digital signal processing*, McClellan J.H., Rader C.M., Englewood Cliffs, NJ, Prentice-Hall Inc, 1979.

8 *The relationship between two fast Fourier transforms*, Good I.J., IEEE Trans. on Comput., Vol. C-20, pp 310-317, March 1971.

correlation and convolution with the IMS A100

10.1 Introduction

The correlation process is widely used in many electronic systems including instrumentation, communication, medical ultrasonics, radar, sonar, control systems and other signal processing environments. The basic reasons for this widespread use can be attributed to the many useful characteristics exhibited by the correlation process. These properties include

- The ability to recover a desired signal masked by noise or other interferences. This is particularly useful in noisy environments that arise in communication, radar, sonar and ultrasonic applications.
- Delay estimation capability which is essential in many applications including range measurement in navigation systems, radar, sonar and also system identification.
- The ability to recognize a given pattern within a signal.
- The auto-correlation of a signal is closely related to the power spectrum which has resulted in the application of the correlation process to spectral analysis.
- The correlation process provides a good characterization of many signals and has therefore been used in many prediction and estimation algorithms.

Convolution is closely related to the correlation process. Mathematically convolution is what happens in the process of filtering. It will be shown in the next section that both these functions involve a large number of multiplications and additions. Up to now, for the time domain implementation of these processes, many systems have used multiply-accumulator devices. Because of their inherent concurrency, the numerical evaluations involved in the convolution and correlation functions can be performed in parallel. But due to the high cost, power consumption requirement, and size restriction many digital systems use only a single (or possibly two) multiply-accumulator(s). This has resulted in a processing bottle-neck in the time domain evaluation of these functions. For example using a 16-bit multiply and accumulator chip available today it is possible, for a 32-point digital correlator, to achieve at best a sampling frequency of around 100 to 300KHz. This is further reduced as the number of correlation points increases. Additional complexities occur as some form of address generator has to be used to sequence the data and the reference coefficients through the multiply-accumulator chip.

The IMS A100 VLSI chip overcomes these problems by incorporating 32 multiply-accumulators on a single chip. The sampling speed of the IMS A100 ranges from 2.5 MHz to 10 MHz depending on the reference-waveform word-size. (4, 8, 12 or 16 bits). It is the true parallelism incorporated in the systolic structure of the IMS A100 that allows such speed increases. The architecture of the IMS A100 has been designed in such a way that large numbers of these chips can be cascaded to perform high precision correlations involving more than 32 points at full speed. Alternatively it is possible to use multidimensional index mapping to decompose a long correlation/convolution into a number of short ones which can then be carried out by usoing a single or a small number of devices.

By suitable allocation of the coefficients, the IMS A100 can be used to perform 3×3, 5×5, or larger two-dimensional image convolutions.

In this application note the concepts of correlation and convolution are first introduced followed by their IMS A100 implementation issues. Partitioning techniques for decomposing a long correlation/convolution into a number of short ones are then described. Next an example of a two-dimensional image convolution is given. Finally some application areas of correlation and convolution are summarised.

10.2 Correlation concepts

The correlation between two functions is a measure of their similarity. This is illustrated in figure 10.1 where three extreme cases are depicted. Figure 10.1a shows two waveforms which are absolutely identical and they have maximum positive correlation. The two waveforms in figure 10.1b are similar, except for their polarities and as such they have maximum negative correlation. Finally figure 10.1c shows one of the waveforms of figure 10.1a and a noise like signal. As these two waveforms are completely dissimilar the correlation between them is expected to be very small or even zero.

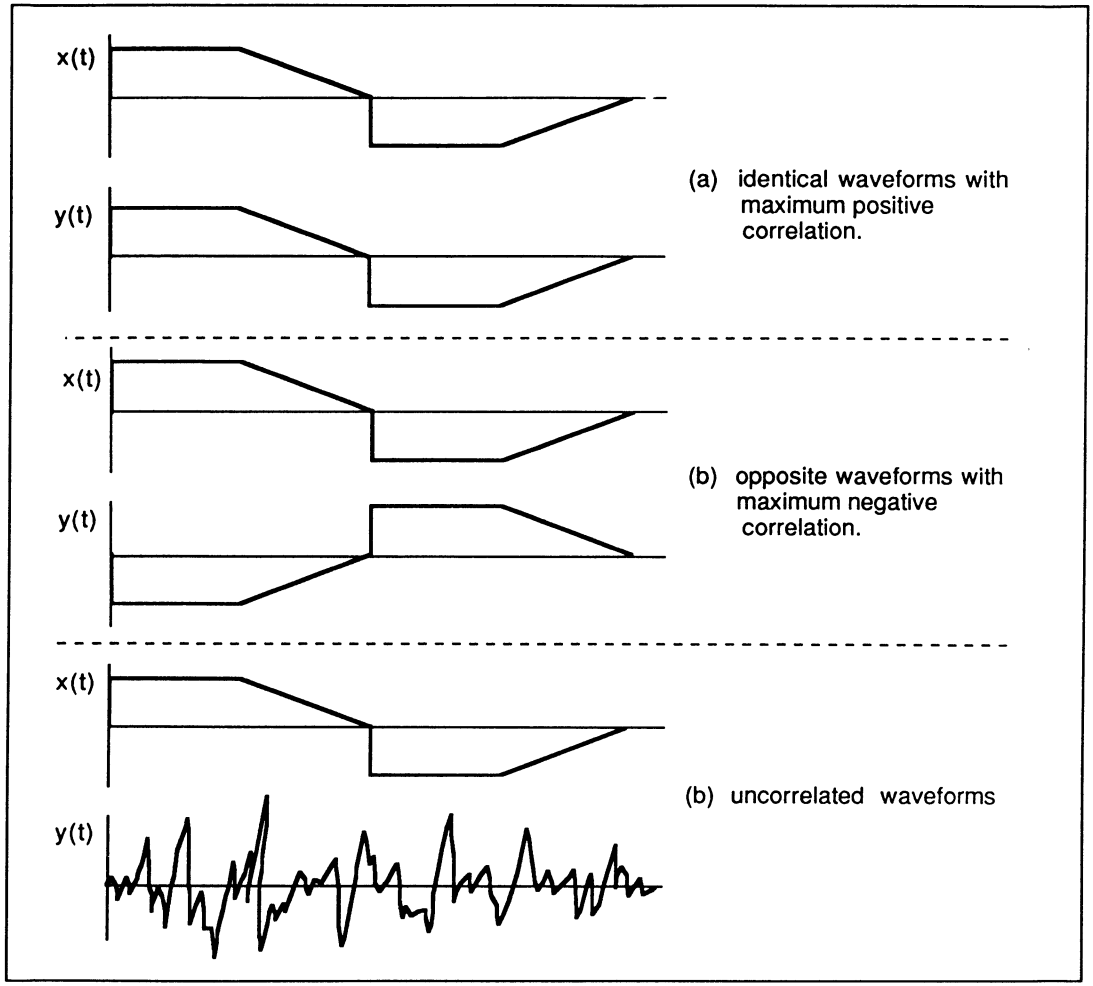


Figure 10.1 Illustration of correlation process

Mathematically the correlation function between two waveforms $x(t)$ and $y(t)$ is expressed as

$$R_{xy}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{+\frac{T}{2}} x(t)y(t+\tau)dt \qquad (1)$$

$R_{xy}(\tau)$ is also referred to as cross-correlation between the two waveforms. For identical waveforms (ie correlating a waveform $x(t)$ with itself) the correlation function is denoted by $R_{xx}(\tau)$ and is called the auto-correlation function.

Equation (1) can be interpreted as follows:

The cross-correlation function, $R_{xy}(\tau)$ between the two waveforms $x(t)$ and $y(t)$ is obtained by shifting one of the two signals in time by an amount equal to τ (i.e. modifying $y(t)$ to $y(t+\tau)$), multiplying the shifted waveforms by the other signal and integrating the product.

If the waveforms are periodic with a period T_0, equation (1) can be modified to:

$$R_{xy}(\tau) = \frac{1}{T_0}\int_{\frac{-T_0}{2}}^{\frac{+T_0}{2}} x(t)y(t+\tau)dt \qquad (2a)$$

i.e. the integration is evaluated only over one period of the signal.

Figure 10.2 provides a graphical illustration of the process of cross correlation between two waveforms $x(t)$ (figure 10.2a) and $y(t)$ figure 10.2b. We start by multiplying the two waveforms and integrating over the interval $\frac{-T_0}{2} \leq t \leq \frac{+T_0}{2}$ yielding $R_{xy}(0)$. With $\tau = 0$, (ie no shift) it can be seen from figures 10.3a & 10.3b that there is no overlap between non-zero parts of the two waveforms resulting in $R_{xy}(0) = 0$ as shown in figure 10.2f. To evaluate $R_{xy}(\tau)$, the waveform $y(t)$ is left shifted by an amount equal to τ, giving $y(t+\tau)$, and the multiplication and integration is repeated. As the waveform $y(t)$ is shifted to the left, there will be no overlap between non-zero parts of $y(t+\tau)$ and $x(t)$ untill $\tau > \tau_1$, see figure 10.2c. At $\tau = \tau_1$, the non-zero parts of the two waveforms just begin to overlap. Figure 10.2d shows the position of $y(t)$ when it is shifted by $\tau = \tau_2$. Here the non-zero parts of $x(t)$ and $y(t+\tau_2)$ have have overlapped and the integration of the product of the two waveforms therefore yields a non-zero value for $R_{xy}(\tau_2)$ as shown in Figure 10.2f. As $y(t)$ is shifted further the non-zero overlapping section of the two waveforms and hence the value of $R_{xy}(\tau)$ increase. When $y(t)$ is shifted to the position shown in figure 10.2e, full overlap occurs and $R_{xy}(\tau)$ will attain its maximum value of $R_{xy}(\tau_3)$ as shown in figure 10.2f. Shifting $y(t)$ further causes the value of $R_{xy}(\tau)$ to decrease as the two waveforms pass each other. Figure 10.2f shows the complete cross-correlation function between the two waveforms. You can confirm the shape of this cross-correlation function by evaluating equation (2a).

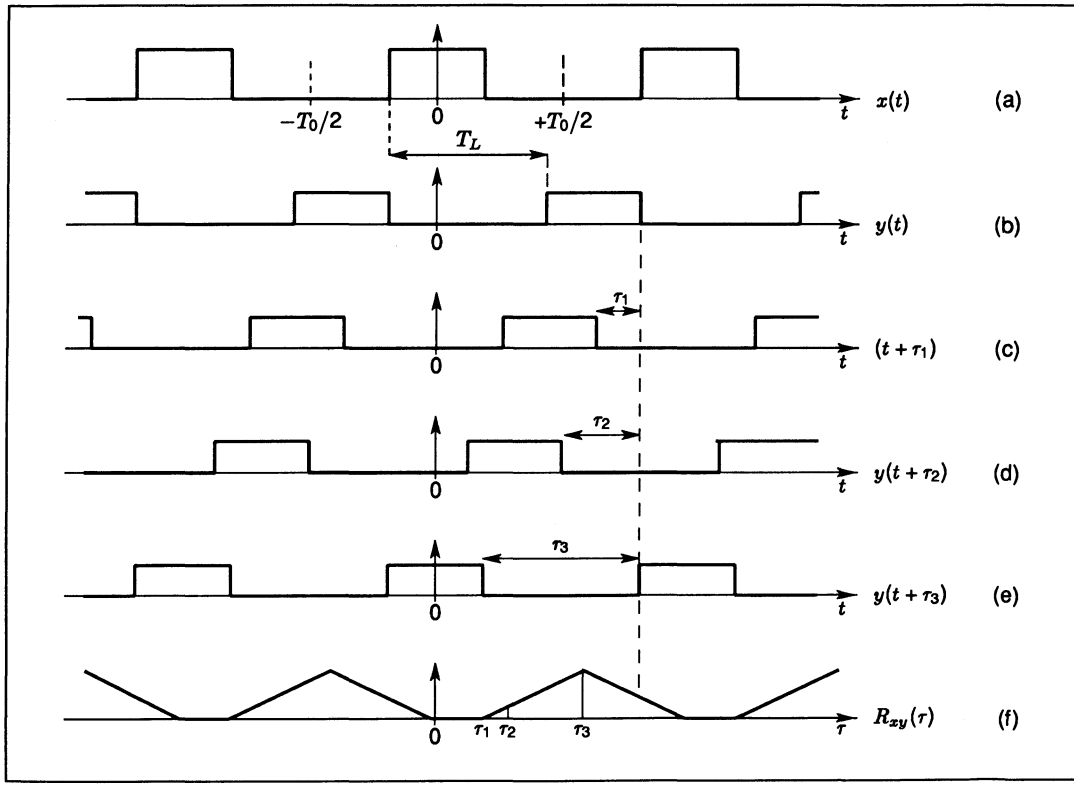


Figure 10.2 Graphical illustration of the correlation process

One interesting point to note here is that the maximum value of $R_{xy}(\tau)$ occurs at $t = \tau_3$ which is equal to the time-lag, T_L, between the two waveforms $x(t)$ and $y(t)$. This is how the correlation process is used to measure delays.

From figure 10.2 it can be seen that the cross correlation function could have been evaluated by shifting $x(t)$ to the right instead of left shifting $y(t)$. Mathematically this can be confirmed by defining a new variable $t' = t + \tau$ and substituting in (2a) which gives:

$$R_{xy}(\tau) = \frac{1}{T_0} \int_{\frac{-T_0}{2}}^{\frac{+T_0}{2}} x(t' - \tau) y(t) dt \tag{2b}$$

So far we have dealt with the correlation of analogue signals. For digital processing both waveforms $x(t)$ and $y(t)$ have to be sampled and digitalized. For discrete-time signals the process of correlation can be expressed as:

$$R_{xy}(mT) = \frac{1}{N} \sum_{k=0}^{N-1} x(kT) y((k+m)T) \tag{3a}$$

At time $t = kT$ equation (3a) requires future samples of $y(t)$. Similar to the analogue case, (equation 2b), the above equation can be modified so that it only uses past samples of $y(t)$, i.e.

$$R_{xy}(mT) = \frac{1}{N} \sum_{k=0}^{N-1} x((k-m)T) y(kT) \tag{3b}$$

In equation (3b), T, donates the sampling period and should be chosen to ensure that the sampling rate is greater than twice the signals bandwidth (Nyquist rate). For the sake of simplicity the factor, T, is usually dropped from the indices of equations (3a) & (3b), i.e.

$$R_{xy}(m) = \frac{1}{N} \sum_{k=0}^{N-1} x(k) y(k+m) \tag{4a}$$

and

$$R_{xy}(m) = \frac{1}{N} \sum_{k=0}^{N-1} x(k-m) y(k) \tag{4b}$$

Where k and m are used to index the samples and N is the number of correlation points involved. In practice the correlation size N will depend on the duration of the two functions, and on their periodicity if they are periodic.

From equations (4a) or (4b) it can be observed that direct evaluation of M samples of the cross-correlation function, R_{xy}, will involve $M \times N$ multiply-and-accumulate operations.

10.3 Convolution concepts

The convolution function is closely related to that of correlation. The convolution of two signals $x(t)$ and $y(t)$ is mathematically defined by:

$$C_{xy}(\tau) = \lim_{T \to \infty} \frac{1}{T} \int_{\frac{-T}{2}}^{\frac{+T}{2}} x(t) y(\tau - t) dt \tag{5}$$

This equation is very much similar to equation (1) defining the correlation process. Their difference is that in convolution the signal $y(t)$ is first time-reversed (i.e. is mirrored around $t = 0$) and then shifted by τ. This time-reversed and shifted signal is then multiplied by $x(t)$ and the product is integrated over all t's. Figure 10.3 graphically illustrates the process of convolution.

The process of convolution occurs in filters where the output of a filter is in fact the convolution of the input function, $d(t)$, and the impulse response, $h(t)$, of the filter (see the application note entitled 'Digital Filtering with the IMS A100'):

$$f(\tau) = \int_{-\infty}^{+\infty} d(t) h(\tau - t) dt \tag{6}$$

where $f(\tau)$ is the filter output.

For discrete-time signals equation (5) becomes

$$C_{xy}(m) = \frac{1}{N} \sum_{k=0}^{N-1} x(k) y(m-k) \tag{7}$$

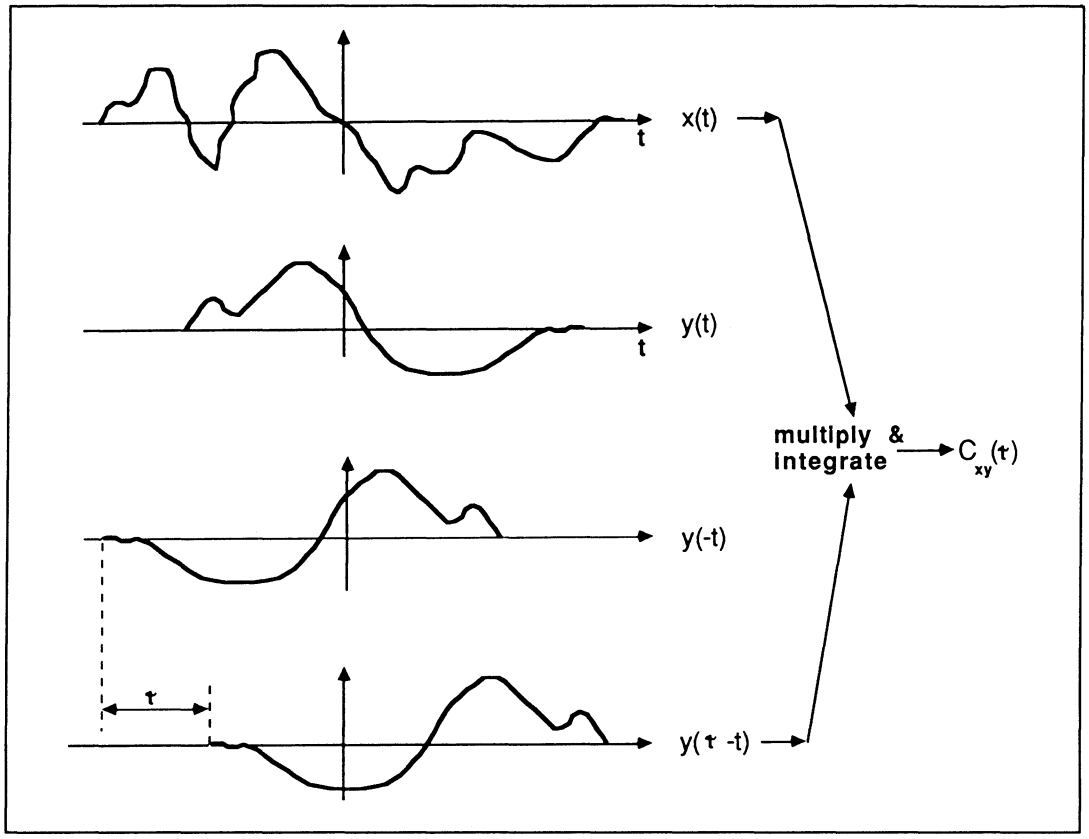


Figure 10.3 Illustrating the convolution process

which defines the convolution function for digital signals. Notice that like correlation, convolution involves carrying out N multiply-and-accumulations for each sample of $C_{xy}(m)$. Due to the high degree of similarity between correlation and convolution functions, the same hardware can be used to perform both functions. All that needs to be done is to time-reverse one of the waveforms for the convolution process.

The following two sections deal with hardware implementations for the correlation and convolution functions. The first section deals with the conventional approach involving multiply-and-accumulator chips and points out the processing bottle-necks associated with these solutions. The second section shows how the IMS A100 signal processor can be configured to perform these functions efficiently and simply, at speeds not economically feasible with the conventional approach.

10.4 Conventional hardware for time-domain evaluation of correlation

As discussed earlier, the processes of correlation and convolution are based on multiplying a delayed version of a sequence of samples by another sequence and summing the products. Conceptually this could be mechanised, as shown in figure 10.4, by providing two shift registers to hold all the values of x's and y's required for the computation, a further shift register to provide the delay (mT), an array of multipliers for forming the products, and a multi-input adder for the final summation. In the example of figure 10.4 the output would correspond to $R_{xy}(2)$, as a delay of two stages is incorporated in the path of signal $x(kT)$ giving $x((k-2)T)$ (see equation 3b).

Up to now, due to the large number of multipliers and adders involved, it has not been possible to eco-

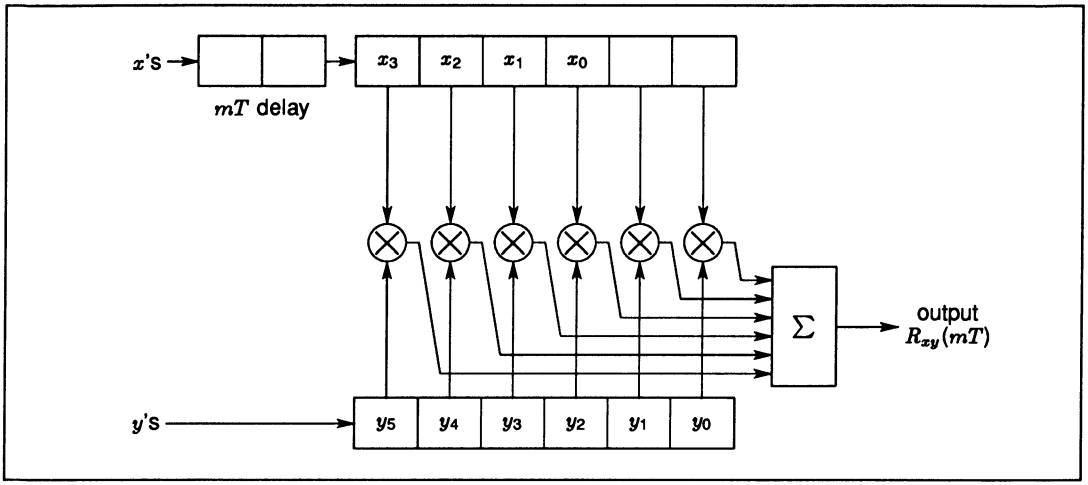


Figure 10.4 Schematic diagram for an ideal correlator hardware

nomically implement high precision correlators directly in the form given by figure 10.4. Instead to minimize the size, cost and power consumption, a single multiply-and-accumulator is usually used and time-shared between all the multiplications. Figure 10.5 shows a schematic block diagram of a conventional correlator implementation. The system consists of memories to hold samples of the two signals to be correlated, a multiply-accumulator and an address generator hardware which is responsible for sequencing the correct order of signal samples through the multiply-and-accumulator. The obvious disadvantage of this arrangement is the processing bottle-neck caused by using a single multiply-and-accumulator to sequentially evaluate what is inherently a concurrent problem. Assuming a multiply-accumulate time of T_{mac}, for an N-point correlator implemented using a single multiply-accumulator, the maximum sampling rate would be

$$f_{max} = \frac{1}{NT_{mac}} \tag{8}$$

For example if T_{mac} = 100ns and N = 100, then a signal sampling frequency of at most 100kHz would be possible.

Many applications such as radar and communication require faster processing rates than can be achieved using a single multiply-and-accumulator. (Some improvements can be achieved by carrying out the processing in the frequency domain at the cost of introducing some complexity. However here we are only concerned with the time-domain approach. A separate application note entitled 'Discrete Fourier Transform with the IMS A100' deals with the time-domain to frequency-domain transformations)

In applications where a fast processing rate is essential, a trade-off is often made between the correlator precision and its speed. For example if one or both of the signals x and y are assumed binary, the multiplications become simple binary AND operations, and it would be possible to implement a high speed low precision correlator. In fact many correlator chips available today are of this type and have very low precision compared to those implemented from multiply-accumulators.

The IMS A100 chip on the other hand is the first high-precision high-speed VLSI implementation of a single-chip correlator. It provides a numerical accuracy in excess of that of the 16-bit multiply and accumulators while allowing sampling rates in the MHz region. The next section illustrates how this chip can be used to perform fast and highly accurate correlation and convolution functions.

10.5 The IMS A100 implementation of correlation/convolution

The IMS A100 is a 32-stage correlator (convolver) in which the samples of the two signals to be correlated can be represented as up to 16-bit words. This corresponds to a signal dynamic range of 96 dB's. A number

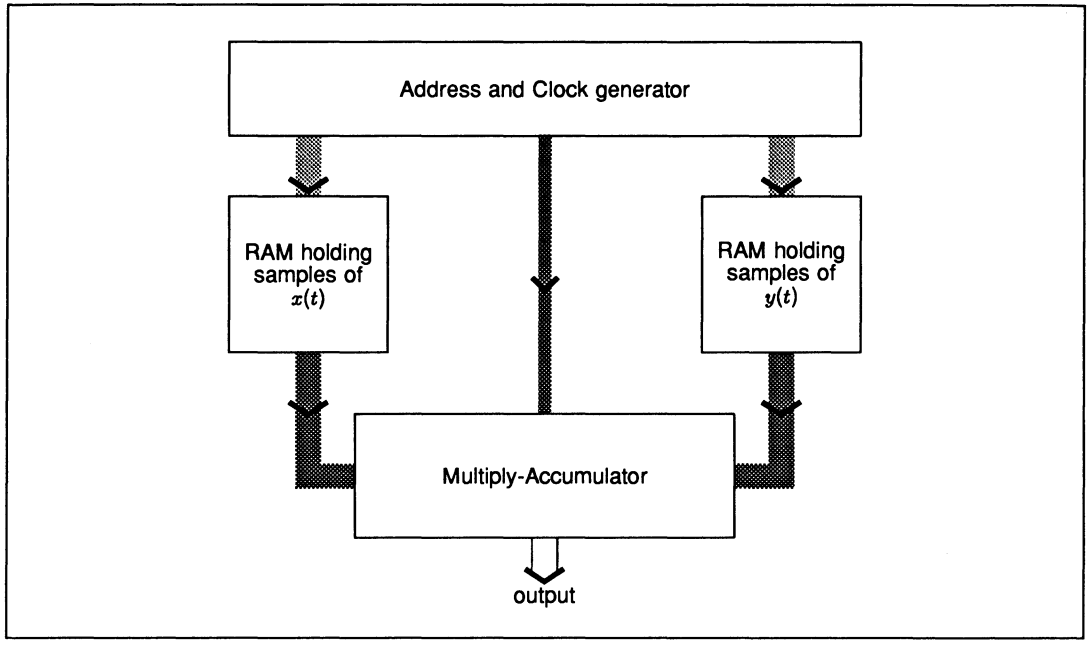


Figure 10.5 Block diagram of a conventional correlator/convolver

of these devices can also be cascaded, without the need for any external components, to provide much longer correlators while preserving a high degree of accuracy. The IMS A100 chip (or cascaded chips) can be fully memory mapped and used as a peripheral accelerator to a host processor.

To understand the architecture of the IMS A100 let us first consider the basic function of a simple 3-point correlator shown in figure 10.6a. The three samples of the first signal (reference signal) i.e. x_0, x_1 and x_2 are loaded in three registers feeding an array of three multipliers. The samples of the second signal i.e. y_0, y_1, y_2,.... are fed into a 3-stage shift register whose outputs are also connected to the multipliers. A three input adder combines the products to give the correlation function. As the samples of the signal y are shifted through the shift register, the output of this hypothetical correlator (assuming the shift register is reset at the start) will be:

$$y_0x_2,\ y_0x_1 + y_1x_2,\ y_0x_0 + y_1x_1 + y_2x_2,\ y_1x_0 + y_2x_1 + y_3x_2, \ldots\ldots$$

The correlator structure in figure 10.6a can be modified to that given in figure 10.6b without affecting the functionality. In figure 10.6b the multi-input summation process is avoided and replaced by a chain of delay-and-add units. The input, supplying the signal y, is also made common to all of the multipliers. Note also that the signal samples x_0, x_1, x_2 are stored in the opposite direction to that of figure 10.6a. Supplying the input sequence of samples $y_0, y_1, y_2, y_3, \ldots\ldots$ to the structure of figure 10.6b and simultaneously shifting the partial products along the delay-and-add chain, it is straightforward to confirm that the output sequence would be

$$y_0x_2,\ y_0x_1 + y_1x_2,\ y_0x_0 + y_1x_1 + y_2x_2,\ y_1x_0 + y_2x_1 + y_3x_2, \ldots\ldots$$

This sequence is absolutely identical to that obtained from figure 10.6a. In other words the structure in figure 10.6a & b have identical functionality and both can be used to perform correlation between two sequences. The IMS A100 architecture is based around this modified structure. The major processing part of the chip incorporates 32 multipliers and a 32-stage delay-and-add chain as shown in figure 10.7.

At this point the interested reader is advised to consult the data sheet of the IMS A100 for full details.

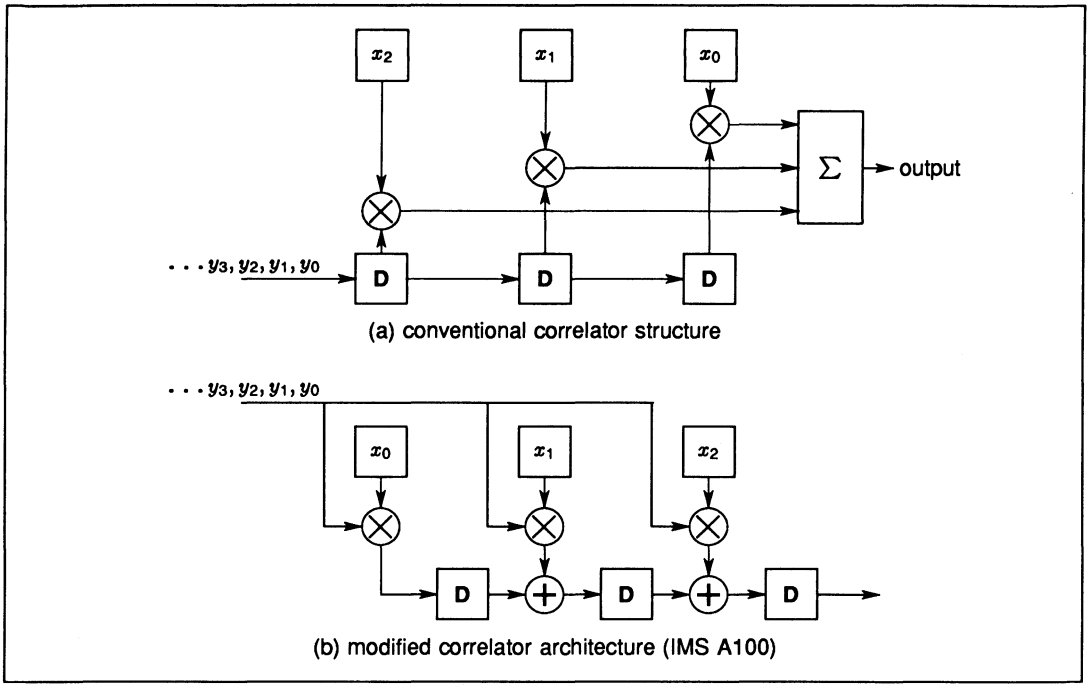


Figure 10.6 Relating the IMS A100 architecture to that of a correlator

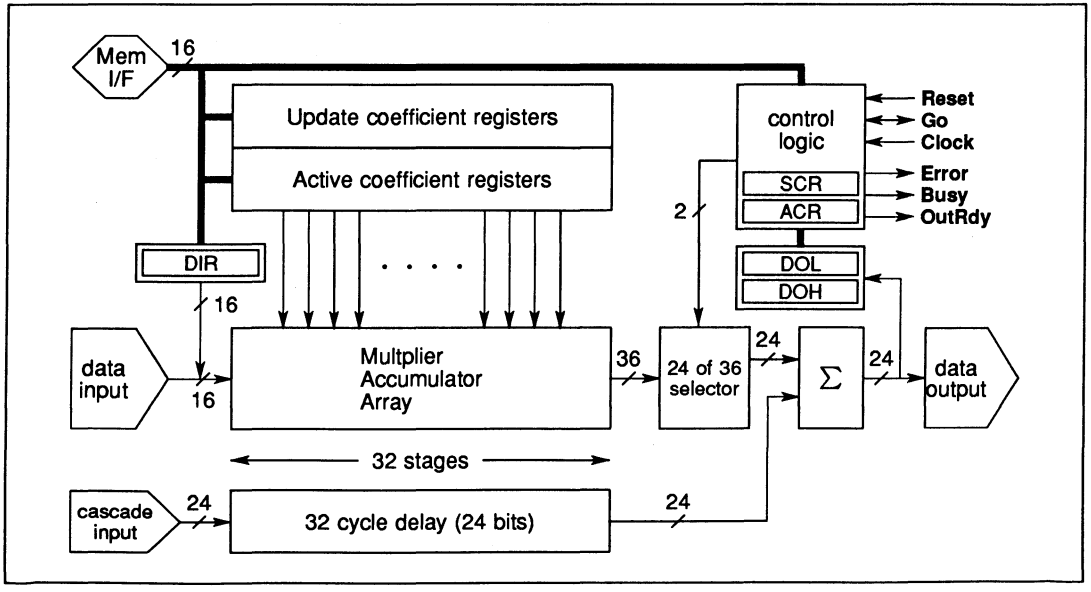


Figure 10.7 User's model of the IMS A100

In order to correlate two sequences $x(k)$ and $y(n)$, the samples of one of the two signals, say $x(k)$'s, should be stored in one set of the IMS A100's coefficient registers. These samples should be loaded from left to right with the last sample of $x(k)$ stored in the coefficient register associated with the last multiplier. If the reference waveforms $x(k)$ is less than 32-samples long, any unused left-most coefficient registers should be set to zero. For a 30-sample reference signal, this allocation is depicted in figure 10.8.

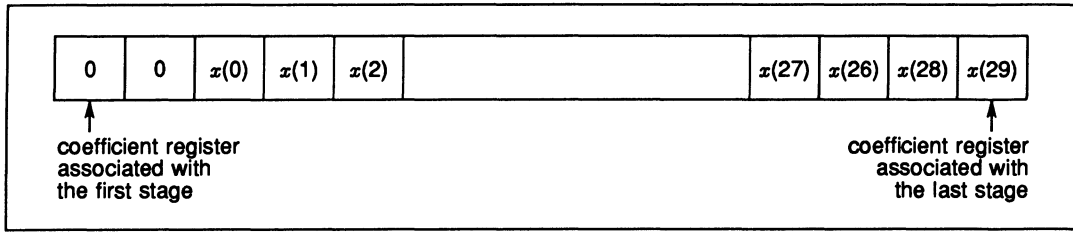


Figure 10.8 Example of the reference signal allocation for a 30 point correlator

The samples of the other signal $y(n)$ are then applied to the input of the IMS A100. As shown earlier the output sequence would correspond to the cross-correlation function of the two signals. If the two signals $x(k)$ and $y(n)$ are to be convolved rather than correlated, the reference signal $x(k)$ should be loaded in the coefficient registers in the opposite direction. The register allocation for a 30-point convolver is shown in figure 10.9. As discussed in the data sheet, the IMS A100 processor has two sets of coefficient registers. At any instant in time one set of coefficients is applied to the multiplier array, whilst the other set can be accessed via the IMS A100 memory interface. For correlations (convolutions) dealing with real signals, one set of these coefficients would be sufficient. The second set can be used to hold a different reference signal and if necessary the function of the two memory banks can be interchanged by performing a write operation to the 'Bank swap' bit of a control register. Such an operation would initiate the correlation (convolution) of the input signal with the second reference waveform. The existance of the two coefficient register sets and the continuous bank-swap mode allows the IMS A100 to perform complex (correlation)convolution, where both the reference and the input signal have real and imaginary components. This configuration is discussed in the application note 'Complex Processing with the IMS A100'.

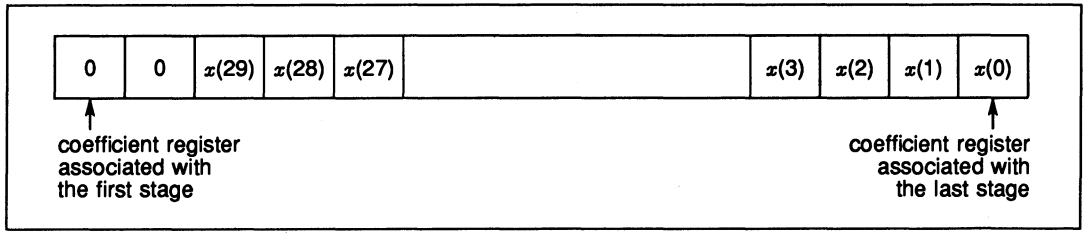


Figure 10.9 Coefficient register allocation for a 30 point convolution

For the IMS A100 the data-word length is 16 bits whilst the coefficient-word length can be programmed to be 4, 8, 12 or 16 bits. The maximum data throughput (the sampling rate) is a function of the coefficient size. Table 10.1 relates the coefficient size to the maximum sampling rate and indicates the effective number of multiply-and-accumulate operations per second in each case. The last column shows the effective number of multiply-and-accumulates when four devices are cascaded.

The strength of the IMS A100 can be appreciated by comparing the effective number of multiply-and-accumulations/sec with that of multiply-accumulator IC's which range from 5–10 million/sec.

As discussed in the A100 data sheet, in order to preserve complete numerical accuracy no truncation or rounding is carried out on the partial products in the multiply-and-accumulation array. The output is thus calculated to 36 bit precision which ensures no overflows. A barrel-shifter at the output of the multiply-and-accumulate array allows 24 bits from these 36 bits to be selected (sign-extended if necessary) and rounded for output. This selection can be programmed via a control register. The programming details can be found in the IMS A100 data sheet.

Coefficient word size (bits)	Sampling rate (MHz)	Effective number of multiply-and-accumulates (Millions/second)	
		Single A100	Four A100s
4	10	320	1280
8	5	160	640
12	3.3	106	424
16	2.5	80	320

Table 10.1

The architecture of the IMS A100 has been designed to allow large numbers of these devices to be cascaded for correlations (convolutions) involving more than 32-stages, the devices can be cascaded while preserving a high degree of accuracy and without the need for any external components. This is made possible by incorporating on chip a 32 stage, 24-bit wide, shift register and a 24-bit adder which combines the output of the barrel-shifter with that of the 32-stage shift register (see figure 10.7).

The IMS A100 chips can be cascaded by simply connecting the output of the each device to the cascade input of the following device. The input is common to all cascaded devices. The effect of such an arrangement is that the output of the first device is delayed by 32 cycles, before being added to that of the next device. Figure 10.10 illustrates how, for example, a 64-point correlator can be implemented using two IMS A100 devices. The allocation of the reference signal samples is also indicated in this illustration. In this arrangement the barrel-shifter in each device acts as a data scalar (with rounding). The cascading process can be considered as a block-floating point operation where the common exponent is determined by the extent of the shift carried out by the barrel-shifter. With this cascading technique a very high degree of accuracy is preserved because the output scaling is only performed after every 32-multiply and accumulation stages and not at any intermediate stage.

For convolution purposes the reference signal should be loaded into the coefficient stage in the opposite direction to that shown in figure 10.10.

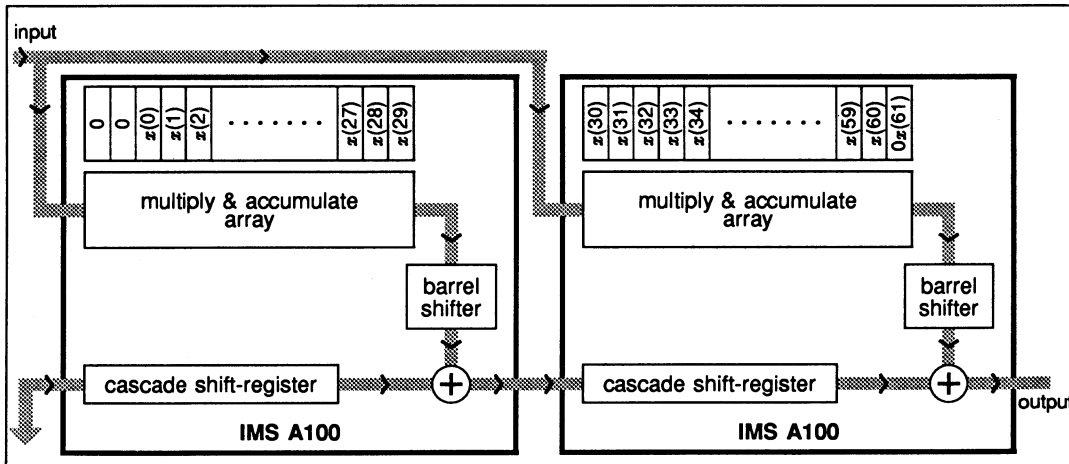


Figure 10.10 Cascading two IMS A100 devices to obtain a 64 point correlator

A very important feature of the IMS A100 transversal filter is that the part is fully memory mappable. Apart from the two coefficient memory banks, which can be accessed via the IMS A100 standard memory interface, the input and output of the device are also accessible from the same interface. This feature allows the part

to be used either with its input and output data communicated through the dedicated ports or through the memory interface. The latter options allows the device to be easily interfaced to a host processor and used as a high speed peripheral. The status and control registers of the IMS A100, accessible via the memory interface, provide full control of the part by the host processor.The memory interface can also be used as a facility for system diagnostics, as the host processor can act as a watch-dog in systems involving arrays of IMS A100's. Full specification of the IMS A100, its status and control registers and its standard memory interface are detailed in the data sheet available from INMOS.

10.6 Decomposition of long correlations and convolutions

A single IMS A100 device is effectively a 32-tap correlator (convolver) in which the samples of the two signals to be correlated can be expressed in upto 16-bit words. As described earlier, one method to deal with correlations/convolutions involving more than 32 points is to use several cascaded devices to achieve a longer correlator/convolver. For such an arrangement, and with 16-bit coefficients, tha data rate can be as high as 2.5 Million samples/sec.

Alternatively it is possible to use various decomposition techniques to partition a long correlation/convolution into a number of shorter ones, which can then be carried out by a single or a small number of IMS A100 devices. The host machine would merely combine the results from these short correlations/convolutions to obtain the overall result. The advantage of this approach, compared to using single MAC based processors, is a significant reduction in the required memory bandwidth. This is why even a medium-speed general purpose microprocessor can achieve a very high performance when combined with the IMS A100.

A simple way to decompose a long correlation/convolution of length N, between waveforms x and y, is to break up one of the waveforms, say x, into consecutive blocks of 32 sample. Each one of these blocks can then be correlated/convolved with the whole of the waveform y by loading each block into the IMS A100 coefficient registers, and using y as the input sequence. The output from these correlations/convolutions can then be combined by displacing each partial result by 32 samples, with respect to the previous one, and performing an addition operation. Note that the coefficient registers, containing blocks of waveform x, need only be updated once every time the whole of the waveform y is fed through the device, resulting in a significant saving in the memory bandwidth. The block size of 32, suggested above, whould mean that a single IMS A100 would be sufficient. However processing speed can be improved by using cascading devices to perform these partial correlations/convolutions. With suitable memory mappings, hosts such as INMOS transputers can use their on-chip DMA engine to feed the IMS A100 devices with the samples of the waveform y.

A more complicated decomposition technique, to be described here, is based on the multidimensional index mappings (references 1 & 2). These techniques are applicable to cyclic convolutions/correlations. However all convolutions/correlations can be made cyclic by adding zero terms to the end of the data blocks. As an example, consider the following cyclic correlation:

$$C(k) = \sum_{n=0}^{N-1} x(k+n)y(n) \tag{9}$$

where the indices are evaluated modulo N. The arrays C, x, and y can be mapped into multidimensional arrays C', x', and y', the requirement being that the mapping should be one-to-one and cyclic in at least one dimension. The map, in general, can assume many different forms, but the one particularly useful is the linear form. For a simple two-dimensional decomposition such a map would be of the form:

$$n = (M_1 n_1 + M_2 n_2) \ mod \ N. \tag{10}$$

Note that n is evaluated modulo N, making the map cyclic in n. In order for this map to be unique and one-to-one, the mapping constants M_1 and M_2 must satisfy certain conditions. These conditions are summarised in section 6 of the IMS A100 Application Note 2 which is available from INMOS and will not be repeated here.

As an example let us map the arrays in equation (9) into two-dimensional matrices of dimensions N_1 and N_2 where $N = N_1 \times N_2$, we can use the mapping given by equations (10) for n and k giving

$$C(M_1k_1 + M_2k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2}^{N_2-1} x(M_1k_1 + M_2k_2 + M_1n_1 + M_2n_2)y(M_1n_1 + M_2n_2) \tag{11}$$

or

$$C'(k_1, k_2) = \sum_{n_1}^{N_1} \sum_{n_2}^{N_2} x'(k_1 + n_1, k_2 + n_2)y'(n_1, n_2) \tag{12}$$

This is now a true two-dimensional convolution which can be made cyclic along n_1 if M_1 is made a multiple of N_2, and/or cyclic along n_2 if M_2 is made a multiple of N_1. With these conditions, inspection of equation (12) shows that the long N-point circular correlation can be performed by N_1^2, N_2-point correlations or N_2^2, N_1-point correlations. This involves correlating each row (or column) of the matrix y' with all the rows (or columns) of the matrix x'. These short circular correlations can be efficiently performed by the IMS A100, with the host merely adding partial results. The approach is particularly efficient as it is possible to load one row (or column) of the matrix y' into the coefficient memory of the device and to feed all the rows (or columns) of the matrix x' successively to the input of the device to obtain partial results, for the elements in the matrix C'. The fact that with this algorithm, the coefficient memories need only be updated occasionally (once every time all the elements of the matrix x' are fed into the device) results in an impressive reduction in the memory bandwidth requirement. This is why, even with a general purpose microprocessor, as the host, very impressive perfomance can be achieved.

In the example given here, we concentrated around a two-dimensional mapping. It is important to realise that the same decomposition concepts can be extended to more dimensions. The easiest way to see this is to start with a two dimensional decomposition and then partition the rows of the two-dimensional matrices further. For example if

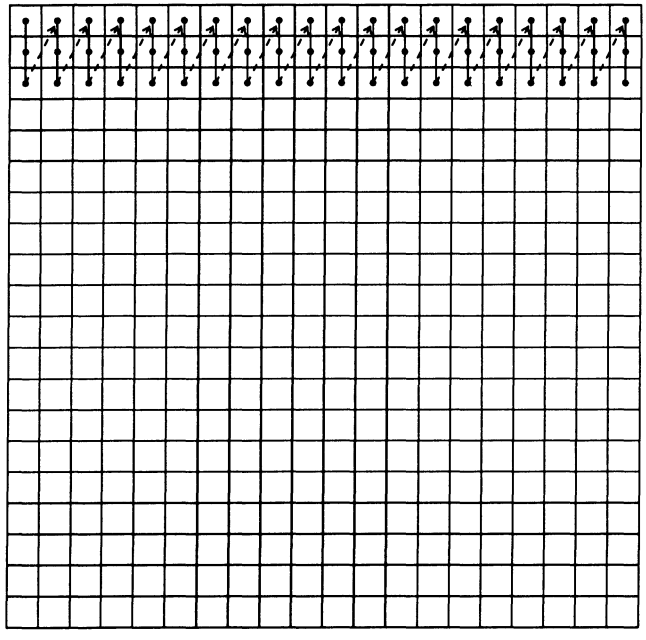
$$N = N_1 \times N_2 \times N_3$$

the original N-point correlation can be carried out via N_3^2, $N_1 \times N_2$-point correlations. However, each one of the $N_1 \times N_2$-point correlations can further be decomposed, as before into N_1^2, N_2-point correlations.

10.7 2-D image convolutions with the IMS A100

Many applications including image processing require 2-D convolutions and correlations. Such operations are needed in image filtering, edge detection, etc. There are many ways that the IMS A100 can be used to speed up these operations. This section gives an example of how the device can be used to perform 3×3, 5×5, or larger convolutions.

Figure 10.11a shows a 20×20-pixel image which is to be convolved with the 3×3 reference matrix given by figure 10.11b. One way to achieve this is to load the reference matrix, as shown in figure 10.11c, in one of the IMS A100 coefficient register banks, and sequence the image data through the device as shown by the arrowed path in figure 10.11a. In this way every third output sample of the IMS A100 would correspond to a valid filtered pixel for the second row of the image. To proceed, the same sequence, moved down by one row, is then passed through the device which provides the filtered results for the nest row and so on. A single IMS A100 can deal with reference matrices as big as 5×5.



(a) 20×20 pixel image, arrows show the required data sequencing

a	d	g
b	e	h
c	f	i

(b) 3×3 convolution matrix

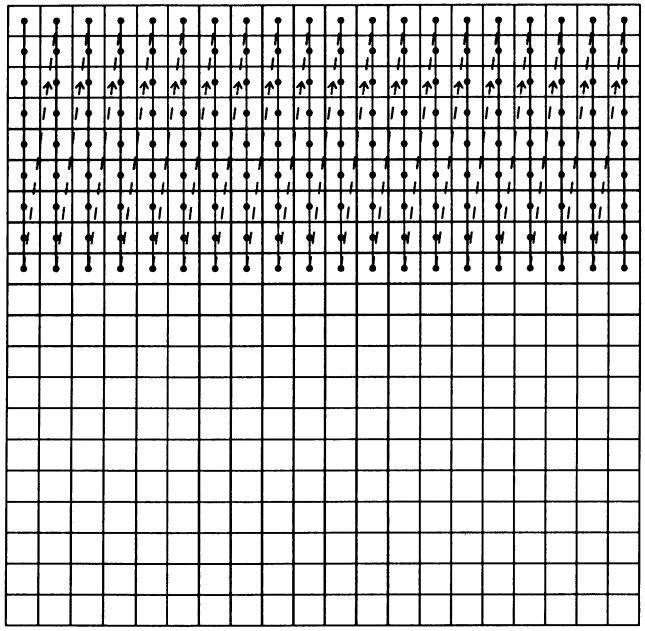
0	0	0	0	.....	0	0	a	b	c	d	e	f	g	h	i

(c) Coefficient register allocation for the 3×3 convolution

Figure 10.11 Example of a 3×3 image convolution/correlation with the IMS A100

An alternative arrangement which gives a better throughput is one where, as shown in figure 10.12a, 7 zeroes are inserted in the IMS A100 coefficient registers (between terms corresponding to the columns of the reference matrix). The data sequencing would be as shown in figure 10.12c, where ten pixels from a given column are fed through the device before moving to the next column. In this scheme the first nine rows of the image are filtered in one scan, with 80% of the output data samples being valid. (Note that, using a single device, the number of inserted zeroes can be increased from 7 to 11, allowing 13 image rows to be filtered in each scan.)

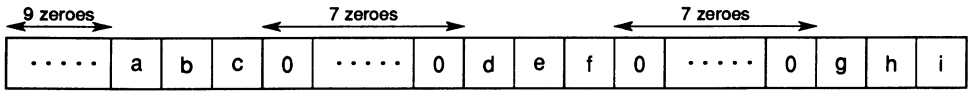
The examples given here are just a small subset of possible arrangements. Remembering that the IMS A100 devices can be cascaded or used in parallel, numerous other implementations for image processing become possible.



(a) 20×20 pixel image, arrows show the required data sequencing

a	d	g
b	e	h
c	f	i

(b) 3×3 convolution matrix



(c) Coefficient register allocation for the 3×3 convolution

Figure 10.12 Improved version of the 3×3 image convolution/correlation with the IMS A100

10.8 Some application examples of correlation and convolution

Correlation and convolution are encountered in numerous applications of digital signal processing, this section summarises some of the application areas where these techniques are used.

10.8.1 Delay and periodicity estimation

The correlation process can be used to estimate the time delay between two similar signals. Figure 10.13 shows two signals $x(t)$ and $y(t)$ which are identical in shape but have a time delay between them. If these two signals are correlated the cross-correlation function would attain a maximum when $y(t)$ is delayed by an amount equal to the delay between the two waveforms. This is illustrated in figure 10.13c where the peak of the cross-correlation function occurs at $t = T_d$ where T_d is the delay between the two waveforms. This technique has applications in areas such as radar, sonar and medical ultrasonics where a measurement of the time delay between the transmitted signal and the return echo from an object gives an indication of the range of that object.

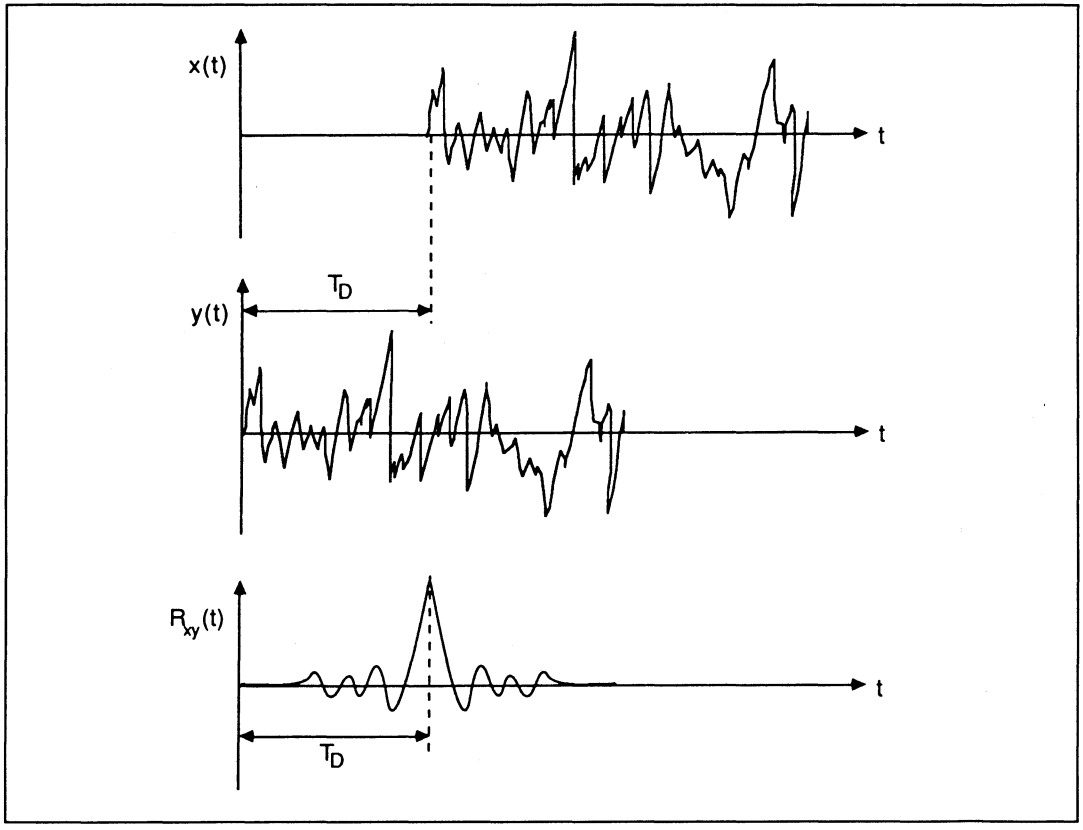


Figure 10.13 Delay estimation using correlation process

The same technique can also be used to measure the period of a repetitive signal. This can be achieved by correlating the signal with itself i.e. by calculating its auto-correlation function as illustrated in figure 10.14 the auto-correlation of a periodic signal exhibits peaks, spaced a distance, T_0, apart where T_0 is the period of the signal. One application of this technique is pitch-period measurement in speech signals. The time gap between the peaks in the auto-correlation function of a segment of speech provides an estimate for the pitch period of voiced speech.

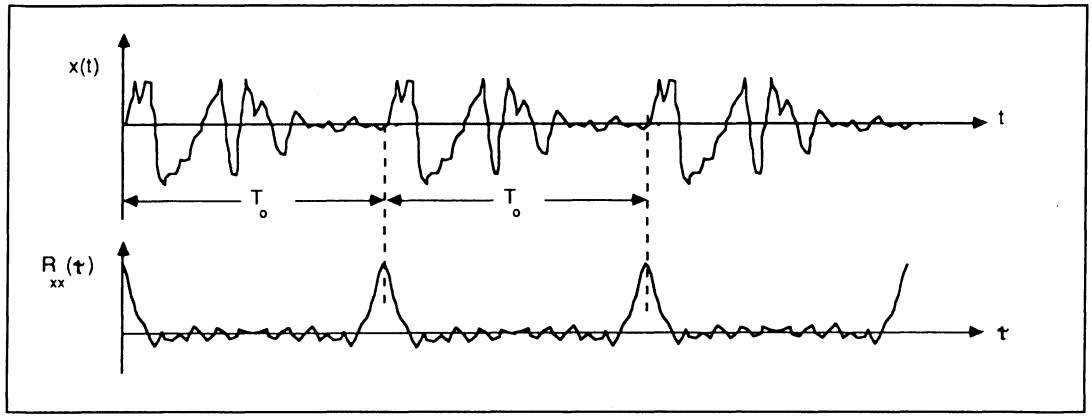


Figure 10.14 Application of correlation to the periodicity measurement

10.8.2 Noise reduction using correlation techniques

In many real-world applications the signals to be processed are immersed and possibly masked by noise. Such situations occur in noisy communication channels, long-range radar and sonar systems. In such cases correlation techniques can be used to extract and detect the signal from the background additive noise. This is achieved by correlating the noisy signal with a replica of the expected signal waveform. While the noise is uncorrelated with the replica signal, the signal immersed in noise will strongly correlate with the replica signal giving a large output value, well above the background noise. Mathematically this can be argued as follows (the proof here is not rigorous but does make the point):

Let the signal waveform consist of N samples values s_0, s_1, s_2, ...s_{N-1}. Suppose this signal is correlated by samples of a white noise having a standard deviation of σ_n (and variance σ_n^2). The ratio of the signal power to that of the noise prior to any processing is thus:

$$\frac{SignalPower}{NoisePower} = \frac{\sigma_s^2}{\sigma_n^2}$$

where σ_s^2 is the signal variance. Suppose we correlate this noisy signal with a replica of the original signal in an N-point correlator. At the instant when the signal waveform masked by the background noise is aligned with its replica in the correlator, the output attains its maximum. At this instant the amplitude of signal component at the output of the correlator would be

$$S_{out} = s_0^2 + s_1^2 + s_2^2 + s_3^2 + ... + s_{N-1}^2 \simeq N\sigma_s^2 \tag{13}$$

The corresponding output signal power would thus be:

$$Output\ Signal\ Power = N^2\sigma_s^4. \tag{14}$$

The noise would also be modified by the operation of the correlator. In this case each output noise sample is equal to the sum of weighted input noise samples-the weighting coefficients being, of course, the samples of the reference signal. Hence each output noise sample is equal to the summation of N independent random numbers having standard deviations $s_0\sigma_n$, $s_1\sigma_n$, $s_2\sigma_n$, ..., $s_{N-1}\sigma_n$. Since variances are additive in this case, the variance of the output noise samples is therefore

$$\sigma_{n_{OUT}}^2 = s_0^2\sigma^2 + s_1^2\sigma^2 + ...s_{N-1}^2\sigma^2 = N\sigma_n^2\sigma_s^2 \tag{15}$$

The ratio of the output signal power to that of the output noise is thus

$$\left(\frac{S}{N}\right)_{OUT} = \frac{Output\ Signal\ Power}{\sigma_{n_{OUT}}^2} = \frac{N^2\sigma_s^4}{N\sigma_n^2\sigma_s^2} = N\frac{\sigma_s^2}{\sigma_n^2} = N\left(\frac{S}{N}\right)_{INPUT} \tag{16}$$

This indicates that the correlation process improves the signal to noise ratio by

$$C_G = 10\log_{10}(N) \quad dB's \tag{17}$$

which is defined as the 'Correlator Gain'.

10.8.3 Pulse-compression

Another application of correlation is in radar and sonar systems where pulse compression techniques are used to improve range resolution of the systems. In many active sonar and pulsed radar systems, a short pulse is transmitted followed by a listening period that represents a 'look' in the range dimension. The two way propagation duration, i.e. the time it takes for the pulse to travel to a target and back gives an indication of the range of that target. The range resolution i.e. the shortest distance between two targets that can be resolved, is equal to $\frac{\tau c}{2}$ where τ is the pulse duration and c is the speed of wave propagation in the medium. For example for a radar system a $10\mu s$ pulse corresponds to a range resolution of 1.5km. Better range resolutions necessitate a shorter pulse. Unfortunately the transmitted pulses cannot be made too short. This is because most systems are peak-power limited and a shorter pulse means less signal power which in turn can severely limit operational range of the system.

Pulse compression techniques allow a radar or sonar to utilize a long pulse to achieve large radiated energy, but simultaneously to obtain the range resolution of a short pulse. This is accomplished by using a coded signal instead of a simple CW pulse. At the receiver the returned signal is correlated with a replica of the coded transmit signal. The returned signal would only correlate heavily with the replica for a short time, corresponding to when the echoes are aligned with the replica. This results in a narrow pulse appearing at the output of the correlator, everytime a match occurs. A signal that is commonly used in pulse-compression techniques is the lineal FM signal. An example of such a signal is depicted in figure 10.15a. The autocorrelation of such a waveform is shown in figure 10.15b. Note that the autocorrelation function has a narrow peak at the origin, with small side lobes elsewhere, i.e. the initially long FM pulse is 'compressed' into a narrow pulse after the autocorrelation process.

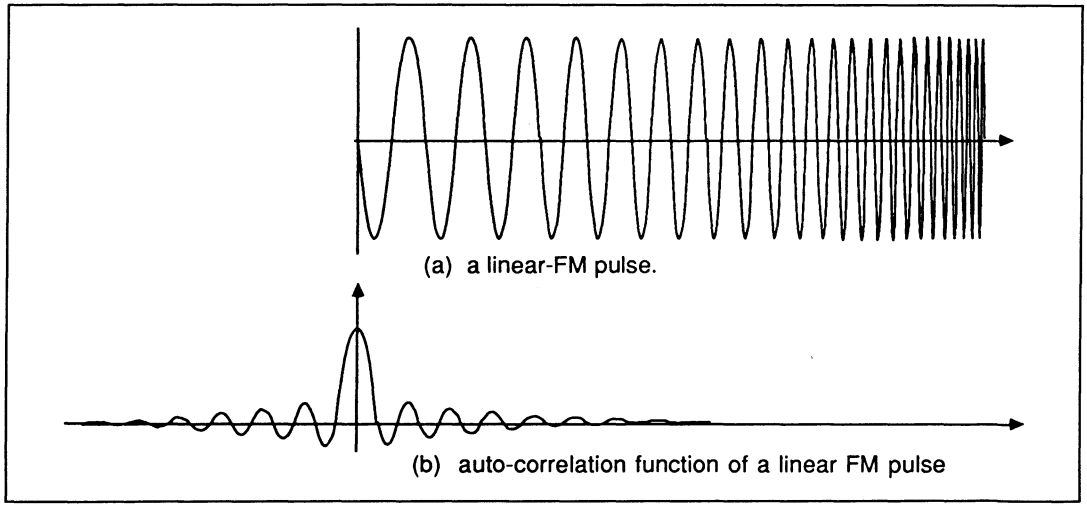


Figure 10.15 Pulse compression using a coded signal

It can be shown that the degree of compression is equal to BT where B is the bandwidth of the coded pulse and T is its duration. The effective pulse duration, as far as the range resolution is concerned, will thus be:

$$Effective\ Pulse\ Duration = \frac{T}{BT} = \frac{1}{B} \tag{18}$$

If the $10\mu s$ pulse in the previous example is coded in such a way that its bandwidth becomes 5MHz, the effective pulse duration would be:

$$\frac{1}{5 \times 10^6} = 0.2\mu s$$

This corresponds to a range resolution of 30 metres.

10.8.4 System identification using correlation

Another important application of cross-correlation is in the use of random-noise test signals to identify the impulse response of a system. For a system with an unknown impulse response $h(t)$, the output $y(t)$ is related to an input $x(t)$ by

$$y(t) = \int_{-\infty}^{+\infty} h(u)x(t-u)dt \tag{19}$$

The cross-correlation between the input $x(t)$ and the output $y(t)$ is defined by

$$\begin{aligned} \Phi_{xy}(\tau) &= \lim_{T\to\infty} \frac{1}{T} \int_0^T x(t)y(t+\tau)dt \\ &= \lim_{T\to\infty} \frac{1}{T} \int_0^T x(t) \int_{-\infty}^{+\infty} h(u)x(t+\tau-u)dudt \end{aligned} \tag{20}$$

Using simple mathematical manipulation it can be shown that

$$\Phi_{xy}(\tau) = \int_{-\infty}^{+\infty} h(u)\Phi_{xx}(\tau-u)du \tag{21}$$

i.e. The cross-correlation between $x(t)$ and $y(t)$ is the convolution of the impulse response $h(t)$ with the auto-correlation of the input signal.

If the input signal consists of broad-band white noise then its auto-correlation function, $\Phi_{xx}(\tau)$, would be an impulse (since a noise signal only correlates with itself at zero delay, $\tau = 0$). Referring to equation (21) it therefore follows that for broad-band noise input, the output $\Phi_{xy}(\tau)$ would be a direct measure of $h(\tau)$ since

$$\Phi_{xy}(\tau) = \int_{-\infty}^{+\infty} h(u)\delta(\tau-u)du = h(\tau) \tag{22}$$

Figure 10.16 illustrates the technique.

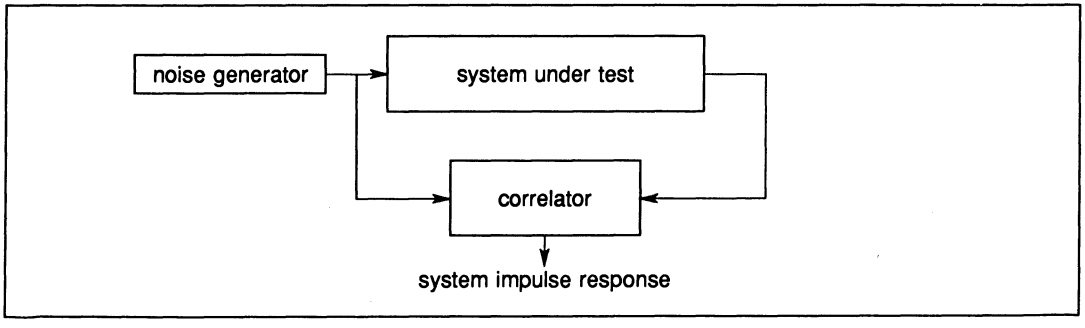


Figure 10.16 System identification using correlation

10.8.5 The Discrete Fourier Transform (DFT)

DFT has application in many signal processing areas, including speech processing, radar, sonar, image processing and control. This transform has often been performed using Cooley and Tukey radix-2 FFT algorithm (reference 3). This algorithm reduces the number of multiplications compared to direct evaluation of the DFT at the expense of complicating the required indexing.

Other algorithms have also been developed which allow the evaluation of the DFT via correlation (convolution) techniques (references 1, 2, 4, & 5). The IMS A100 device can be used to perform high speed DFT's based on these convolutional algorithms. Using the IMS A100 as a peripheral to a general-purpose microprocessor converts a slow host into a high-performance DFT processor. A separate application note available from INMOS describes how these algorithms can be implemented using the IMS A100 devices.

10.9 References

1 Burrus C.S., 'Index mappings for multidimensional formulation of the DFT an convolution', *IEEE Trans. on ASSP*, Vol.25, June 1977, pp 239-242.

2 Burrus C.S., Parks T.W., 'DFT/FFT and convolutional algorithms-theory and implementation', Wiley-interscience Publication, New York 1985.

3 Cooley J.W., Tukey J.W. 'An algorithm for the machine calculation of complex Fourier series', *Math. Comput.*, Vol.19, pp 297-301, April 1965.

4 Rader C.M., 'Discrete Fourier transforms when the number of data samples is prime', *Proc. IEEE,* Vol.56, pp 1107-1109, June 1968.

5 Rabiner L.R., *et al*, 'The chirp z-transform algorithm and its application', The Bell System Technical Journal, May-June 1969, pp 1249-1292.

complex (I&Q) processing with the IMS A100

11.1 Introduction

Complex processing, involving in-phase and quadrature signal components, is necessary in many signal processing applications. This type of processing is needed in cases where the phase of the signal has significant impact on the processing outcome. For example consider a simple demodulator as shown in figure 11.1. The incoming tone, $(A\cos\omega t)$, is demodulated by mixing it with a local oscillator having the same frequency. The output of the mixer is low-pass filtered to yield the final result. If there is a phase difference Φ between the incoming tone and the local oscillator signal, the output would be proportional to $\cos\Phi$. This indicates that the output of our simple demodulator is strongly dependent on the relative phases of the incoming and local oscillator signals. For the worst case of $\Phi = \frac{\pi}{2}$, the output would become zero! This means that the relative phase shift of the local oscillator can be quite disasterous.

Let us now perform the same demodulation using complex processing. The input signal can be represented in its complex form consisting of real and imaginary parts i.e.

$$x(t) = Ae^{-j\omega t} = A[\cos(\omega t) - j\sin(\omega t)] \tag{1}$$

Mixing the signal with a complex version of our local oscillator signal i.e. $A^{j(\omega t+\Phi)} = \cos(\omega t+\Phi) + j\sin(\omega t+\Phi)$ yields:

$$Ae^{-j\omega t}e^{j(\omega t+\Phi)} = Ae^{j\Phi} = A\cos\Phi + jA\sin\Phi \tag{2}$$

Note that this output is complex and contains both phase (Φ) and amptitude (A) information. The amptitude can be extracted by taking the modulus of the output i.e.

$$amplitude \;=\; \sqrt{(A\cos\Phi)^2 + (A\sin\Phi)^2} = |A| \tag{3}$$

The above example illustrates how complex processing can be used to preserve both phase and amptitude information in a simple demodulator.

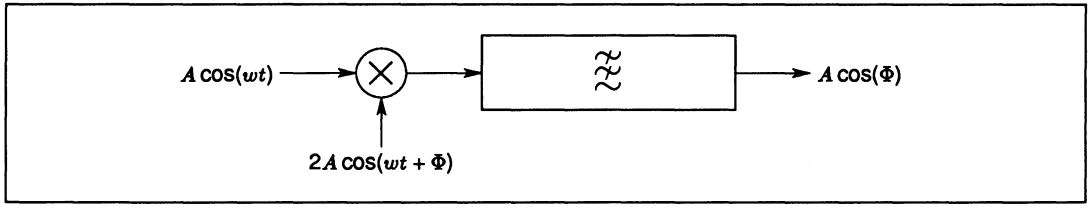


Figure 11.1 Simple demodulator

Similar phase related problems arise in correlation and convolution evaluations where complex processing becomes necessary for preserving the integrity of signals. This application note describes how on-chip facilities of the IMS A100 transversal filter can be used to perform complex correlation, convolution and filtering.

As described in the data sheet, the IMS A100 transversal filter incorporates two sets of coefficient memories (figure 11.2), each containing 32 16-bit words. At any instant one set of coefficients is applied to the multiply-accumulate array, whilst the other set can be accessed via the IMS A100 standard memory interface. The function of the two memory banks can be interchanged by performing a write operation to the 'Bank Swap' bit of a control register.

This allows the new set of coefficients to be used in the computation at the beginning of the next cycle. In this operation once the two memory banks are interchanged, the 'Banks Swap' control bit is reset by the device. No more interchanges are performed unless the bank swap control bit is again set by the host.

There is another control bit in the static control register of the IMS A100, that when set continuously interchanges the two memory banks at the beginning of each and every computation cycle. When this mode is set, alternate coefficient memory banks will be used for even and odd computation cycles. This mode is particularly suitable for implementing complex data processing. The following two sections describe how this continuous-swap mode can be employed to perform complex convolutions and correlations using the

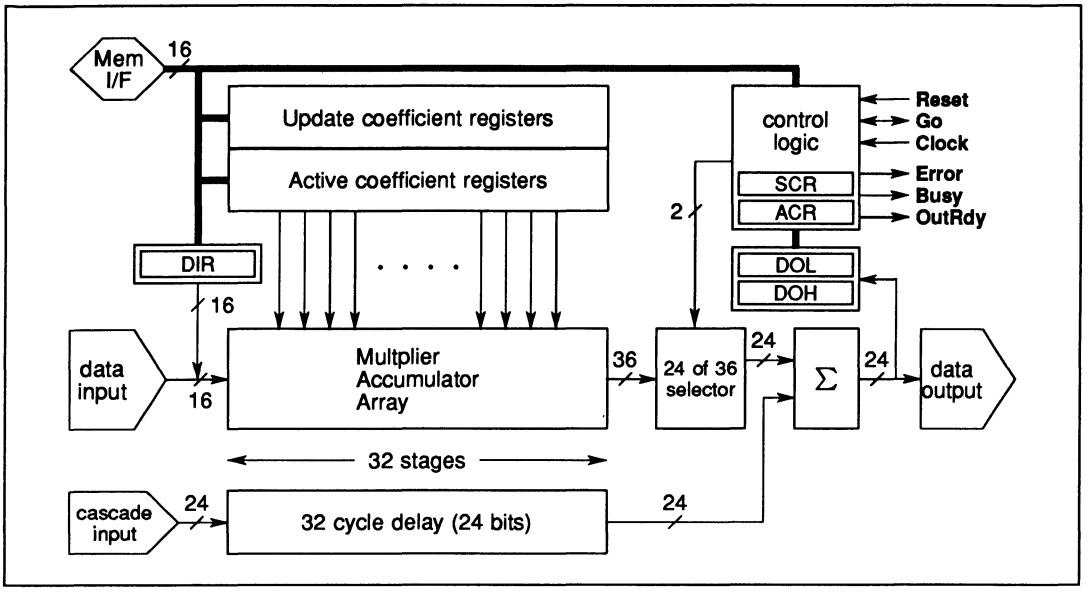


Figure 11.2 User's model of the IMS A100

IMS A100 transversal filters. A separate application note, available from INMOS, deals with the correlation and convolution concepts and their implementation using the IMS A100 device. Readers unfamiliar with the IMS A100 and its implementation of correlation and convolution functions are advised to refer to that application note before reading the following sections.

11.2 Complex correlation

The complex correlation between two signals r and s is very similar to real correlation (refer to the application note entitled 'Correlation and convolution with the IMS A100') with the difference that one of the two signals has to be complex conjugated first i.e.

$$R_{rs}(m) = \frac{1}{N}\sum_{k=0}^{N-1} r^*(k)s(k+m) \tag{3}$$

or

$$R_{rs}(m) = \frac{1}{N}\sum_{k=0}^{N-1} r(k)s^*(k+m) \tag{4}$$

where * indicated complex conjugate operation and both waveforms r and s can be complex.

Let us now investigate how the IMS A100 can perform this function. As shown in figure 11.2, the computational core of the IMS A100 contains an array of 32 multiply-and accumulators. In order to simplify the explanation of complex processing, let us consider a simple five-stage transversal filter as shown in figure 11.3. Once you have understood how such a simple structure can be used for complex correlation, it should be easy to extend the idea to larger correlations sizes involving one or many cascaded IMS A100 devices.

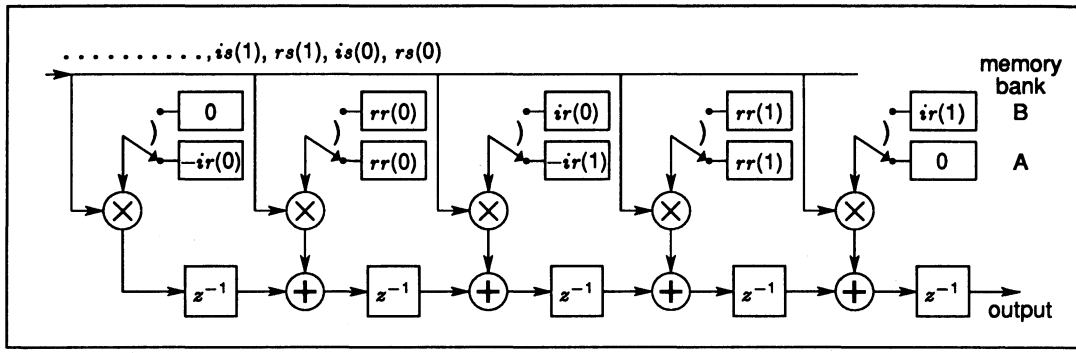


Figure 11.3 Two-point complex correlator based on the IMS A100 architecture

Suppose we want to perform a two-point complex correlation between a reference signal and a sequence of complex input samples. Let us denote the two complex samples of the reference signal with

$$r(0) = rr(0) + j\ ir(0) \quad and \quad r(1) = rr(1) + j\ ir(1)$$

where rr and ir indicate real parts and imaginary parts of the reference signal respectively.

Assume the input sequence is the following set of complex samples:

$$s(0) = rs(0) + j\ is(0),\ s(1) = rs(1) + j\ is(1),\ \ldots\ldots\ldots, s(n) = rs(n) + j\ is(n), \ldots\ldots\ldots$$

where $rs(n)$ and $is(n)$ indicate real and imaginary parts of the nth input sample, $s(n)$, respectively.

The 5-stage transversal filter shown in figure 11.3 can be used to correlate these two sequences. The reference signal samples are first complex conjugated i.e.

$$r^*(0) = rr(0) - j\ ir(0) \quad and \quad r^*(1) = rr(1) - j\ ir(1).$$

These samples of r^* are then allocated to the two coefficient memory banks as shown in figure 11.3. It can be seen from this diagram that both coefficient stores contain real and imaginary samples of the reference signal.

Assume that the input sequence is sampled into the correlator, with the real part followed by the imaginery part of each input sample. i.e. the input to the correlator is

$$rs(0),\ is(0),\ rs(1),\ is(1)\ , rs(2)\ , is(2)\ \ldots\ldots\ldots\ldots$$

where $rs(0)$ is the first input sample. Also assume that we have selected the continuous-swap mode so as the memory banks A and B are swapped every time a new input is sampled. (On the IMS A100, you can select this mode by writing to a control register). Assuming that the coefficient bank 'A' is selected for the first input sample, B for the second and so on, you should be able to convince yourself that the output sequence for the arrangement in figure 11.3 is as shown in table 11.1. Note that in this example it is assumed that the correlator is cleared first by writing several zero's to the input.

Sample number	Input sample	Output sample value		
1	$rs(0)$	0	$\Rightarrow$	0
2	$is(0)$	$rs(0) \times rr(1) + is(0) \times ir(1)$	$\Rightarrow$	Real part of $s(0) \times r^*(1)$
3	$rs(1)$	$-rs(0) \times ir(1) + is(0) \times rr(1)$	$\Rightarrow$	Imag. part of $s(0) \times r^*(1)$
4	$is(1)$	$rs(0) \times rr(0) + is(0) \times ir(0) +$ $rs(1) \times rr(1) + is(1) \times ir(1)$	$\Rightarrow$	Real part of $s(0) \times r^*(0) + s(1) \times r(1)$
5	$rs(2)$	$-rs(0) \times ir(0) + is(0) \times rr(0)$ $-rs(1) \times ir(1) + is(1) \times rr(1)$	$\Rightarrow$	Imag. part of $s(0) \times r^*(0) + s(1) \times r^*(1)$
6	$is(2)$	$rs(1) \times rr(0) + is(1) \times ir(0) +$ $rs(2) \times rr(1) + is(2) \times ir(1)$	$\Rightarrow$	Real part of $s(1) \times r^*(0) + s(2) \times r(1)$
7	$rs(3)$	$-rs(1) \times ir(0) + is(1) \times rr(0)$ $-rs(2) \times ir(1) + is(2) \times rr(1)$	$\Rightarrow$	Imag. part of $s(1) \times r^*(0) + s(2) \times r^*(1)$
8	$is(3)$	$rs(2) \times rr(0) + is(2) \times ir(0) +$ $rs(3) \times rr(1) + is(3) \times ir(1)$	$\Rightarrow$	Real part of $s(2) \times r^*(0) + s(3) \times r(1)$
9	$rs(4)$	$-rs(2) \times ir(0) + is(2) \times rr(0)$ $-rs(3) \times ir(1) + is(3) \times rr(1)$	$\Rightarrow$	Imag. part of $s(2) \times r^*(0) + s(3) \times r^*(1)$
10	$is(4)$	$rs(3) \times rr(0) + is(3) \times ir(0) +$ $rs(4) \times rr(1) + is(4) \times ir(1)$	$\Rightarrow$	Real part of $s(3) \times r^*(0) + s(4) \times r(1)$
11	$rs(5)$	$-rs(3) \times ir(0) + is(3) \times rr(0)$ $-rs(4) \times ir(1) + is(4) \times rr(1)$	$\Rightarrow$	Imag. part of $s(3) \times r^*(0) + s(4) \times r^*(1)$

Table 11.1 Output sequence for figure 11.3

The last column in table 1 expresses the output sequence in terms of the complex input and complex reference samples. Examination of the output sequence would indicate that alternate samples correspond to real and imaginary parts of the expected correlation function. The arrangement for the two point correlator of figure 11.3, can be generalised to N-point complex correlation. Figure 11.4 illustrates the allocation of a reference signal to the coefficient memories of the IMS A100 for a 15-point complex correlation. The 15 complex samples of the reference signal are represented by:

$$r(n) = rr(n) + j\ ir(n) \qquad for \quad n = 0 \rightarrow 14$$

where $rr(n)$ and $ir(n)$ are the real and imaginery parts of the nth sample of the reference waveform. Similar to the 2-point complex correlator described earlier, the correct operation is achieved if each input sample is supplied to the chip with its real parts followed by its imaginery parts. The coefficient memories, of course, should be set to the continuous-swap mode.

In general for an N-point correlator realized with the IMS A100 chip, the first sample will always be zero (see table 1). The following $N-1$ output-sample pairs (real and imaginary parts) correspond to partial results for the following complex correlation coefficients:

$$R_{sr}(-(N-1)),\ R_{sr}(-(N-2)),\ .....,\ Rsr(-1)$$

and these will be followed with fully formed correlation coefficients:

$$R_{sr}(0),\ R_{sr}(1),\ Rsr(2).....$$

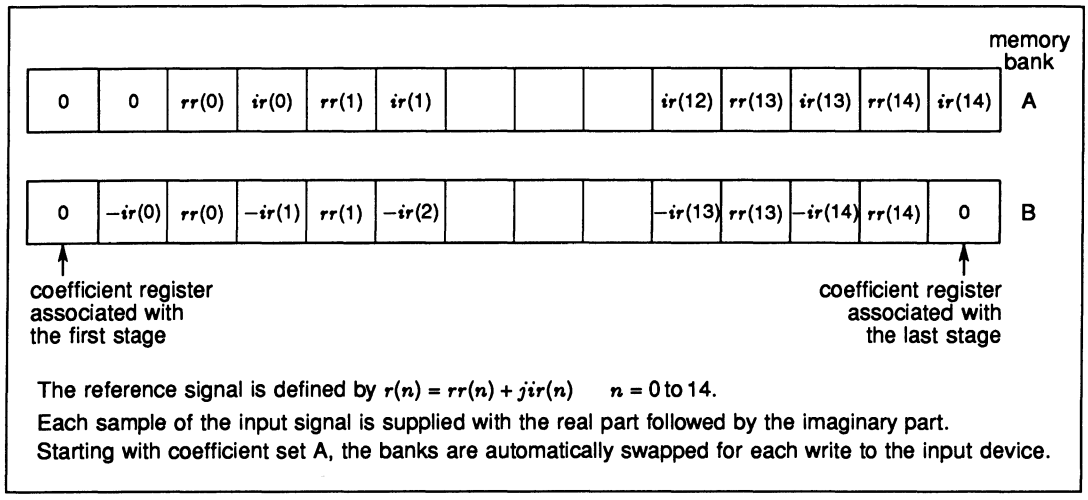
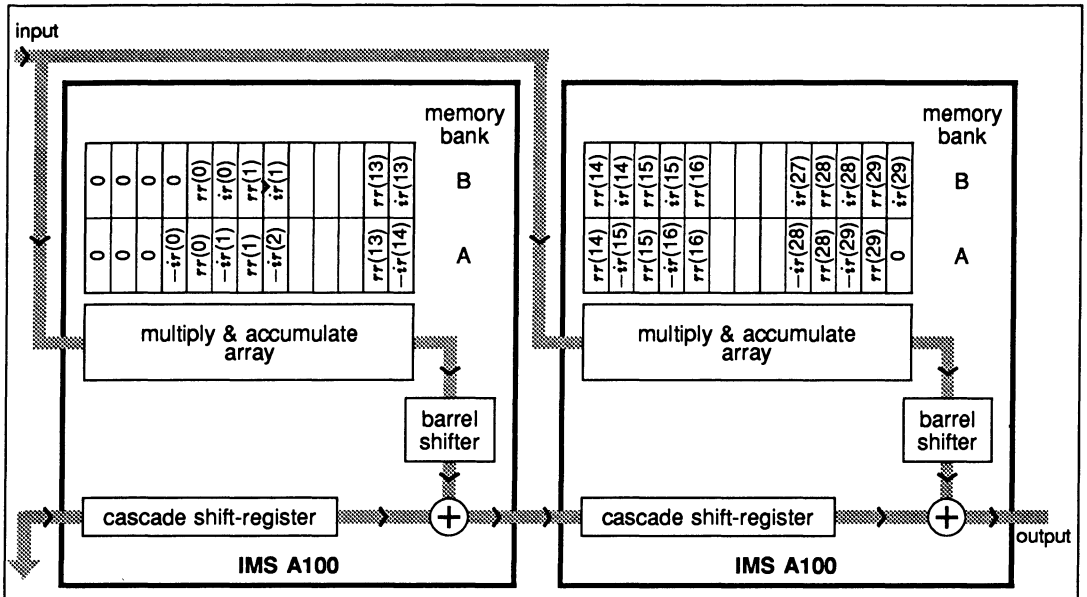


Figure 11.4 Example of reference signal allocation for a 15 point complex correlation using the IMS A100

Complex correlators involving more than 15-points can be implemented either by cascading several IMS A100 devices or alternatively by using mathematical decomposition techniques to convert a long correlation into several short ones which can then be evaluated using a single device. Although the latter approach would require fewer devices, the processing rate would be less than the cascade arrangement.

The IMS A100 can be cascaded without any external components to achieve correlators involving large number of correlation points. As an example, figure 11.5 illustrates how a 31-point complex correlator can be made up by cascading two IMS A100 devices. The allocation of a complex 30-point reference signal to the coefficient memories is also shown in figure 11.5. The input sequence, having a format described earlier, is supplied to both devices.



Allocation of the reference signal samples for a 30 point complex correllation is illustrated.

Figure 11.5 Cascading two IMS A100 devices to obtain a 31 point complex correlator

11.3 Complex convolution

The convolution process is closely related to that of correlation. In order to convolve two signals, one of the signals is time reversed and the second signal is then correlated (without complex conjugate operation) with this time reversed waveform i.e.

$$C_{rs}(m) = \frac{1}{N}\sum_{k=0}^{N-1} r(k)s(m-k). \qquad (5)$$

The process of convolution is what happens in filters where the output corresponds to a convolution of the input signal and the impulse response of the filter. This is equivalent to correlating (without conjugate operation) time-reversed version of the impulse response with the input sequence.

The IMS A100 transversal filter can be used to perform complex convolution between a reference signal

$$r(n) = rr(n) + jir(n) \qquad for \;\; n = 0 \rightarrow N-1$$

and an input sequence

$$s(0) = rs(0) + j\,is(0),\; s(1) = rs(1) + j\,is(1),\; \ldots\ldots\ldots, s(n) = rs(n) + j\,is(n), \ldots\ldots\ldots$$

Figure 11.6 illustrates how the samples of a reference signal should be loaded in the coefficient memories for a 15-point complex convolution. In a similar fashion to the complex correlator implementation, the waveform to be convolved with this reference is applied to the input of the IMS A100 with the real part followed by the imaginary part. The coefficient memory banks should be set to the continuous bank-swap mode as before.

Again several IMS A100 devices can be cascaded to implement longer complex convolvers.

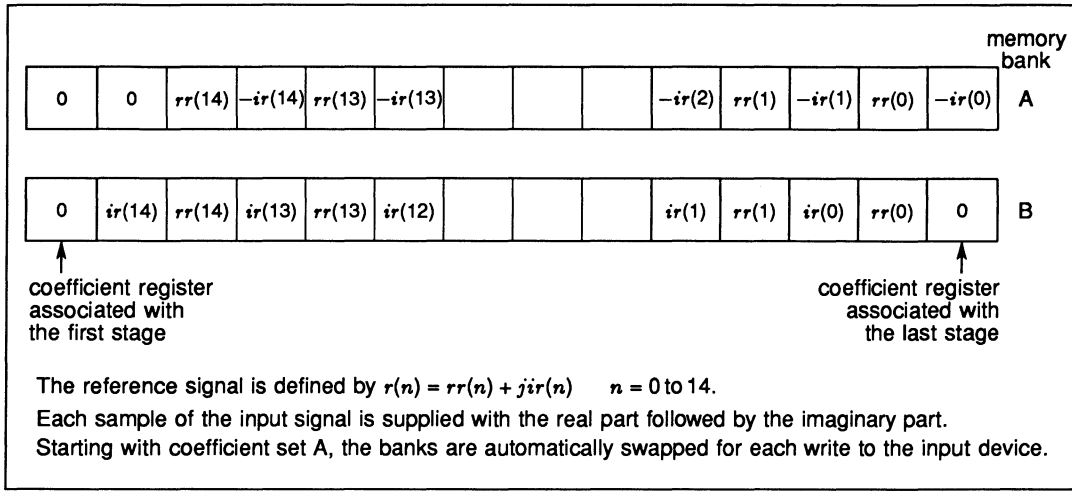


Figure 11.6 Example of reference signal allocation for a 15 point complex convolution (filtering) using IMS A100

hardware considerations with the IMS A100

12.1 Introduction

The design of modern high speed digital systems is a considerable challenge. If the designer is using unfamiliar new products which themselves are complex *VLSI* devices then this task can become very difficult. This application note will help those who wish to build systems using the IMS A100 cascadable signal processor.

12.1.1 Scope of the document

This document should be read in conjunction with the device specification [1]. The device specification is a short, precise, and minimal description of the IMS A100. The following document gives a more detailed description of the function of the device, along with many hints for designing the device into a circuit. Specific hardware designs are also given, which may help the designer further.

12.1.2 Document summary

In section 12.2 a description of the IMS A100 device is given, with a particular emphasis on the operation and timing constraints of the various inputs and outputs.

In section 12.3 smaller systems using a few IMS A100 devices are considered.

Section 12.4 describes techniques which allow large and very large systems to be designed without loss of throughput.

Section 12.5 describes a method which allows faster data rates to be achieved, by operating several IMS A100 devices in a parallel configuration.

Section 12.6 gives some suggestions for debugging and fault finding hardware after it has been built.

12.2 The IMS A100 Device

This section gives a functional and parametric description of the device, and should be read in conjunction with the IMS A100 data sheet. The data sheet is a necessary description for design with the IMS A100. The following expands on some of the device mechanics, which are described in the data sheet.

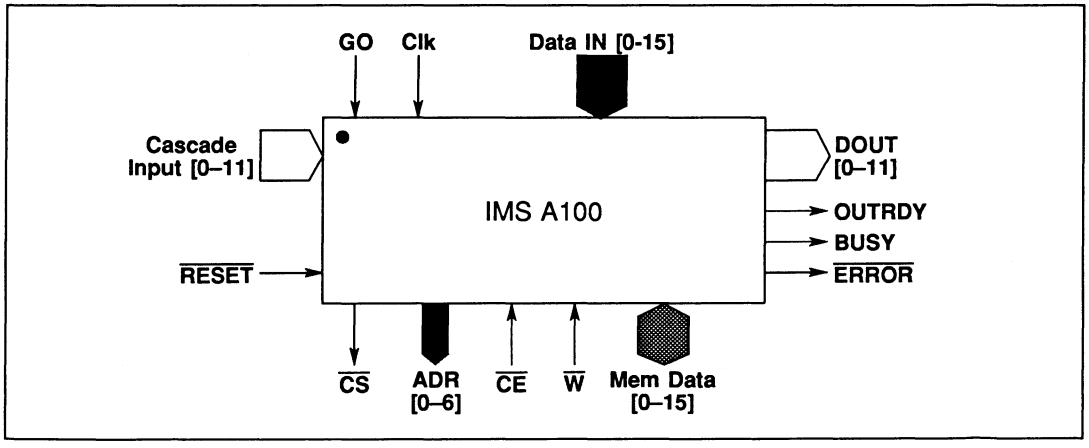


Figure 12.1 IMS A100 device schematic

12.2.1 Pin description and constraints

The description of the various pins of the device is split into a description of power supply pins, asynchronous pins, synchronous pins and control pins.

Power Supply

All power supply pins must be connected to the correct polarity of supply for the device to operate correctly. The supply must be decoupled by a capacitor with a value of 100nF or more which is suitable for high frequencies (e.g. multi-layer ceramic). One or more should be mounted as close as possible to each device and the lead lengths of the capacitors should be minimised. The device is designed to operate with a single supply of between 4.5 and 5.5volts.

Synchronous Input/Output

The synchronous pins of the device are *CLOCK, GO, OUTRDY, DataIn*[0..15], *DataOut*[0..11] (multiplexed) and *CascadeIn*[0..11] (multiplexed).

The clock

The *CLOCK* input pin requires a non-standard input signal of >4.0volts for a high level and <0.5volts for a low level. The waveform needs to be monotonic between these two levels. Details of how this is achieved are given in later sections of this application note. In general, a CMOS level clock driver with proper termination will be needed. The *CLOCK* pin forms a large capacitive load (12pF typical, 15pF max.) which needs to be considered when designing the clock driving system.

The *CLOCK* is the source of synchronisation between cascaded IMS A100 devices; *GO, DataIn*[0..15] and *CascadeIn*[0..11] all sample in response to it. The output signals *DataOut*[0..11] and *OUTRDY* are also timed from the clock. When the IMS A100 devices are programmed to operate in the *normal* mode the timing constraints associated with the transfer of data between adjacent cascaded IMS A100 devices is less rigorous than in 4 bit or *fast* modes. If the devices are being used in *fast* output or 4 bit modes it is important to keep the timing skew of the clock between devices to a minimum. In practice, it is not a good idea to buffer the clock between devices in these modes. If a *master* generated *GO* pulse is being used, a common clock is recommended. The maximum clock frequency for an IMS A100 is normally 20.8MHz, (a 30MHz version device is also available) but the device is fully static and will therefore operate at any frequency below this. It is also possible to start and stop the clock, provided no single phase becomes shorter than the minimum indicated in the data sheet.

GO

The *GO* pin initiates a compute cycle of the IMS A100 and synchronises the devices in a multiple IMS A100 system. The *GO* pin is sampled on every rising edge of *CLOCK* when the IMS A100 is idle, and no computation cycle is in progress [1]. When a '1' is sampled, a computation cycle is started, and the *DataIn* pins or data input register *DIR* is sampled on the next rising edge of *CLOCK*. The *GO* pin will not be sampled again until it is possible to commence another cycle. It is therefore possible to leave the *GO* pin at a '1' level following the initial clock edge, if the maximum data throughput is necessary. The number of clock cycles between successive *GO* samples and the result appearing at the output, are dependant on the coefficient word length setting [1]. *CascadeIn*[*0..11*] is also sampled following a *GO* signal.

For the *GO* pin to be an input, the IMS A100 is set to be a *slave* by setting *SCR[0]* to a '0'. However, the IMS A100 can be programmed to provide a *GO* signal for itself and for other IMS A100s by setting *SCR[0]* to a '1'. This causes the device to send a signal from it's *GO* pin in response to a value being written to it's *DIR* register. The falling edge of this signal indicates when new data can be safely written to the IMS A100s in the system. This feature is particularly useful in small systems where a microproccessor is being used to provide data. It should be noted that the *master* IMS A100 is only designed to drive itself plus another 3 devices (maximum load < 20pF) when operating at the maximum clock rate. However, at slower clock rates more devices can be added in line with the following table.

Max clock Freq.	Max no of *slaves*	Length of filter
20 MHz	3	128
17.5 MHz	4	160
16 MHz	5	192
15 MHz	6	224
10 MHz	10	352
5 MHz	30	992

It is possible to buffer the *GO* signal from the *master* IMS A100 providing the buffer is fast enough. The propagation delay for such a buffer with a 5pF input capacitance as a function of the IMS A100 clock period is given below.

$$Tp_{buffer} < Tclk - 46ns$$

An alternative method of buffering the *GO* signal is discussed in the section of this note dealing with large systems.

Data input bus

This 16 bit wide *DataIn[0..15]* provides high speed data to the IMS A100s when *SCR[1]* is programmed to '0'. Usually, *DataIn[0..15]* will be common to all the IMS A100s in a given cascade. The data is sampled on the rising edge of the clock following the acceptance of a '*GO* sample' by the devices. If this bus is being used to provide data, the IMS A100 must be in *slave* mode and cannot be used as a *master*. Each pin represents a capacitive load of about 5pF.

Data output bus

The 24 bit result from the IMS A100 is multiplexed through *DataOut[0..11]* as two 12 bit words, the least significant word being first. The most significant word follows and remains on the pins until the next least significant word is available. The timings of the signals are dependant on the coefficient word length and the *normal/fast* setting as defined in the *SCR*. In the 4 bit and *fast* modes the least significant word is only available on one rising edge of *CLOCK*, with the most significant word being sent immediatly following the same edge. In the 4 bit mode running at full speed, every rising edge is used to both latch the output data into the *CascadeIn[0..11]* pins, and to cause the output data to change to the new value. The advantage of the *fast* output mode is that the complete 24 bit output word is made available at the earliest possible time, whereas the *normal* mode delays the most significant word slightly. This eases the timing constraints of the circuitry sampling the output data. All devices in a given cascade must be set to the same coefficient word length and *fast/normal* option. The output drivers used on the data output pins are designed to drive small loads (e.g. 2 TTL inputs or about 15pF) with a 20MHz *CLOCK* in the fast or 4 bit modes. Even in the *normal* mode the load should not exceed 30pF on these pins.

Output ready signal

The output ready pin *OUTRDY* is provided to indicate when the two 12 bit output words from the data output pins are valid. It can be used to demultiplex the output into registers, and also indicates when the data has been stored in the data output registers *DOL* and *DOH*. The falling edge of the *OUTRDY* signal indicates that the least significant word on the output is valid, whilst the rising edge indicates that the most significant word is valid and that the DOL/DOH registers contain the new output data value. As in the case of *DataOut[0..11]* the timings of this signal are dependant upon the coefficient word length and the *fast/normal* mode setting. Again, the timing constraints are eased when the device is operating in the *normal* mode on 8 bit, 12 bit and 16 bit coefficient sizes. In the *fast* or 4 bit modes the *OUTRDY* signal is triggered by the falling edge of *CLOCK* following the rising edge of *CLOCK* which changes the output data. The *OUTRDY* pin should have a similar loading to the data output pins for optimum timing, and this should not exceed the limits set for the data output pins.

The *OUTRDY* signal can be used to supply a clock for a D/A converter which, if it uses less than 12 of the available 24 bits, will only require one of the two 12 bit words, thereby avoiding the need for demultiplexing logic. When demultiplexing is required, it can be achieved using two sets of edge triggered latches (e.g. *74ACT374*) which are clocked by *OUTRDY* and it's inverse (figure 12.1). It is suggested that any ex-

ternal logic associated with the *DataOut[0..11]* and *OUTRDY* pins be of a fast *TTL* compatible *CMOS* logic type (i.e. *FACT*) in order to minimise loadings.

Cascade input port

The cascade input port allows multiple IMS A100 devices to be cascaded together in a chain. Like *DataOut[0..11]*, *CascadeIn[0..11]* is 12 bits wide and two words are used to form a 24 bit word with the least significant word being sampled first. The cascade input timings are given in the device specification, but it should be noted that the *OUTRDY* signal does **not** normally coincide with the sampling of *CascadeIn[0..11]*. The cascade input of the first device should be grounded unless data is to be supplied to it.

Memory interface asynchronous input/output

The memory interface is the asynchronous part of the system. It is designed, as far as is practical, to appear as a memory mapped peripheral. To achieve this there are chip select, chip enable, read/write and address and data bus signals, which will now be described.

Chip select pin

The chip select pin $\overline{CS}$ has to be pulled low (active) at the appropriate time for the memory interface to be enabled. This pin is usually connected to part of an address decode system.

Chip enable pin

The chip enable pin $\overline{CE}$ is pulled low (active) to enable the memory interface, after the address, write enable $\overline{W}$ and chip select $\overline{CS}$ signals have been set up.

Read not write

The read not write signal $\overline{W}$ defines whether a given cycle is reading from or writing to the IMS A100 memory. This signal should not be changed whilst $\overline{CE}$ is low.

Memory address bus

This 7 bit wide port *ADR[0-6]* is used to address the IMS A100 memory. A memory map is given in the IMS A100 specification showing locations of the two coefficient banks and control registers. The *TCR* register, located at decimal address 68, will default to all zeros on power up or in response to a $\overline{RESET}$ signal. However if it is disturbed, by for example a system memory test, it should be written back to all zeros. Failure to do so may result in unpredictable results.

Memory data bus

The 16 bit wide memory data port *DATA[0-15]* handles both input and output data to and from the IMS A100 and is used to program the two banks of coefficient registers and the control registers. When writing to a coefficient register, the memory interface is transparent while $\overline{CE}$ is low. Writing to the active coefficients whilst the IMS A100 is running a computation cycle may cause an incorrect transient coefficient to be used. Using the update registers followed by a bank swap avoids this problem.

When the $\overline{CE}$ or $\overline{CS}$ are high the data pins are tri-state. The output stages associated with the data pins are current limited and may be loaded by more than the 30pF specified for the timings given in the specification provided the $\overline{CE}$ pulse length is increased. The table below gives an indication of the length $\overline{CE}$ pulse required for a series of loads.

Capacitive load on data bus	$\overline{CE}$ pulse length
300 pF	50 ns
100 pF	80 ns
300 pF	170 ns
1000 pF	500 ns

Each Data pin represents a maximum load of about 7pF when tri-stated.

System control

The IMS A100 is controlled by 3 signals, $\overline{RESET}$, $\overline{ERROR}$ and *BUSY* which will now be described.

Resetting the device

To reset the IMS A100 control logic pull the $\overline{RESET}$ pin low for at least 200ns followed by two cycles of the input clock. The reset function on the IMS A100 only resets the control registers to their default values. It does not change the values of the coefficients, clear the data path or reset the error flags if errors are still present. There is a power on reset signal ORed with the $\overline{RESET}$ pin circuitry which requires an adequate voltage on both the power supply and the internal clocks before it allows the internal reset signal to fall. For this reason the *CLOCK* pin must be exercised at or following power up before the control registers are programmed. A resistor (e.g. 33kΩ) from the $\overline{RESET}$ pin to VCC together with a capacitor (e.g. 10μF) to *GND* is usually sufficient to provide an adequate signal. The pin may be connected directly to *VCC* provided you are sure that your system power supply is monotonic on power up, as the power on reset circuit will only operate once.

Error control

If the $\overline{ERROR}$ pin is asserted it indicates that there has been a numerical overflow in either the final adder or field selector. Bits [1-2] in the Active Control Register *ACR* indicate which error type and the $\overline{ERROR}$ pin is reset by writing a '0' to these registers. Before continuing, these error bits must be armed by writing a '1' to bits[1-2] of the *ACR*. The $\overline{ERROR}$ pin is only able to sink current to *GND* and therefore requires a pull up resistor to Vdd. Many devices can be wire ORed together to indicate an error in any one of many IMS A100 devices within a given system. The presence of an error does not affect the operation of the IMS A100 (although the results may be nonsense) and it is possible to continue to use the device without resetting the condition. Although the ACR register is reset on power up, the $\overline{ERROR}$ pin is usually set again by random numbers within the device. In order to clear the $\overline{ERROR}$ pin it is necessary to flush the system before clearing and re-arming the *ACR* registers. Flushing involves writing zeros to the data input and cascade input over 32 successive cycles.

Device busy

The *BUSY* pin indicates when the active and update coefficient registers are being or are about to be swapped. When this pin is high the coefficient registers should not be accessed. There is no guaranteed minimum duration of a *BUSY* signal since a bank swap request may be dealt with immediately. This pin operates only in conjunction with individual *bank swap* requests made via *ACR[0]* and not when the continuous *bank swap* mode is selected by *SCR[2]*.

12.2.2 Initialisation of IMS A100s

There are many ways of initialising one or more IMS A100 devices but if in doubt the following procedure is recommended. First, for a system with all devices used as *slaves* do the following operations.

1 Apply power and start *CLOCK*

2 Take the $\overline{RESET}$ pin high

3 Write all coefficients to '0'

4 Set the *CascadeIn[0..11]* of the first devices in any cascades to '0'

5 Set *GO* to '1' or provide plenty of *GO* pulses

6 Allow the system to run like this for long enough to clear out any stored junk numbers. This period will depend on the length of your filter and the frequency of your *GO* pulses.

7 Apply a $\overline{RESET}$ signal or write '0s' and then '1s'to the *ACR[1-2]* — any error signal should now disappear.

8 Set up your own *SCR*, coefficient and data values.

For a system with a *master* and *slaves* do the following.

1 Apply power and start *CLOCK*

2 Take the $\overline{RESET}$ pin high

3 Set up your own *SCR* values

4 Write all coefficients to '0'

5 Set the *CascadeIn[0..11]* of the first devices in any cascades to '0'

6 Write to the data input register *DIR* on the *master* IMS A100 to create plenty of *GO* pulses

7 Allow the system to run like this for long enough to clear out any stored junk numbers. This period will depend on the length of your filter and the frequency of your *GO* pulses.

8 Write '0s' and then '1s' to *ACR[1-2]* — any error signal should now disappear.

9 Set up your own coefficient and data values.

12.2.3 An extra selector setting using TCR

The test control register *TCR* is designed to help INMOS fully test the IMS A100. However, one of it's functions may be of interest if the output word selection field of [7-30] gives insufficient resolution. This is most likely to occur when smaller coefficient word lengths are in use. Writing a '1' to *TCR[2]* will overide the values programmed in *SCR[4-5]* to give a field selection of [−1-23] where field bit [−1] will always be '0'. The other *TCR* bits must always be set to '0' by the user.

12.3 Smaller IMS A100 systems

The techniques described in this section apply to systems employing perhaps four IMS A100 devices in a single cascade together with a small support system including a single microprocessor. Two typical systems are shown in figure 12.2.

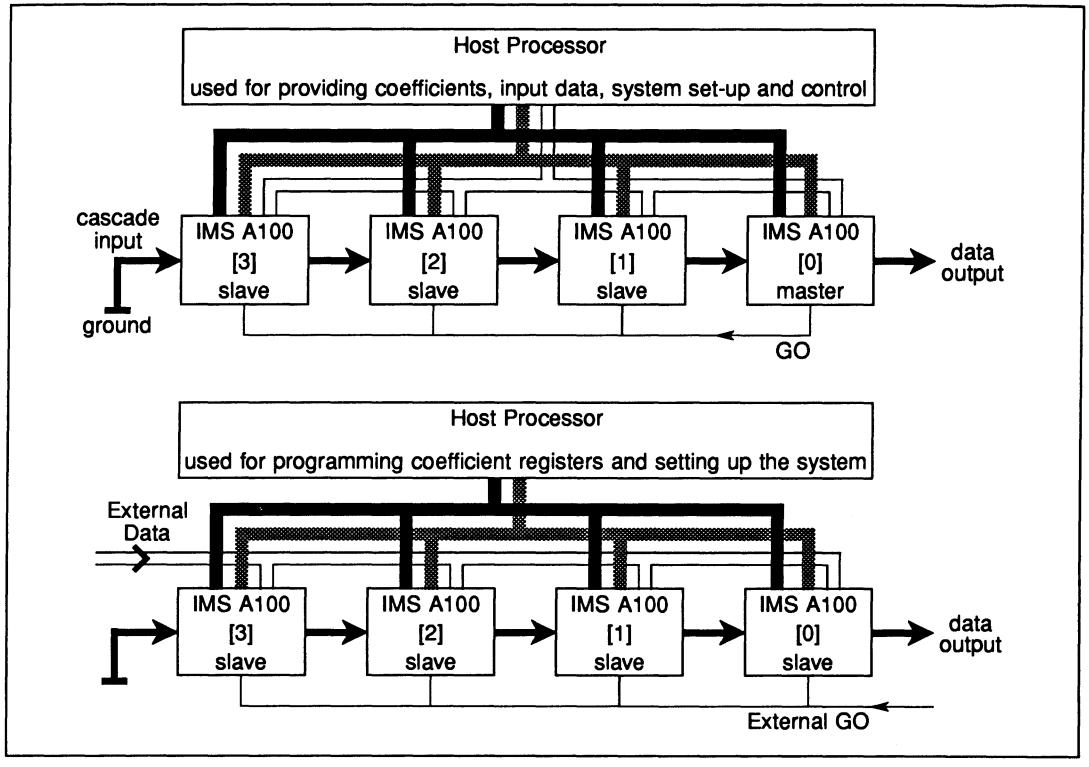


Figure 12.2 Two simple small systems

12.3.1 Board Layout Constraints

During normal operation the IMS A100 dissipates a fairly low average power (0.5W at 20MHz approx). However, due to the high degree of parallelism within the device, it requires well decoupled, low inductance connections to it's power pins. A multi-layer board with a *VCC* and *GND* plane is recommended with at least one multi-layer ceramic decoupling capacitor of 100nF or more mounted as close as possible to each IMS A100. The IMS A100 devices forming a cascade should be located next to each other with the *CascadeIn*[*0..11*] pins of the next device near the *DataOut*[*0..11*] pins of the first. Any circuitry using the *DataOut*[*0..11*] pins should be located as near to the IMS A100 as possible to avoid excessive loading. The track carrying *CLOCK* should take a direct route from one IMS A100 to the next in order to avoid excessive skew.

12.3.2 Memory Interface

The last IMS A100 in a cascade chain should occupy the lowest address space and the first in the cascade the highest. This is to maintain compatability with the addressing of the coefficient registers where *coeff[0]* resides in the lowest location within the bank.

In many applications it will not be necessary to buffer the pins associated with the IMS A100 memory interface. It is, however, necessary to confirm that this is, in fact, the case. The timings of the proposed memory interface together with the bus loadings should be checked with the IMS A100 device specification and the additional information given in Section 2 of this note. If the memory interface uses high speed buffers, some termination may be required to limit transients outside the power supply rails. In such cases 100Ω resistors in series with the offending buffer(s) are recommended.

12.3.3 Clocking

In general, the smaller the system, the easier the clocking will be. The IMS A100 *CLOCK* is not *TTL* compatable and will have to be generated by a device or devices capable of driving signals to within about 0.5volts from each power rail. The constraint that the *CLOCK* signal at the IMS A100 should be monotonic in between the high and low limits will almost certainly mean that termination will be required. The easist way of generating such a *CLOCK* for a small system is to use a *TTL* compatible *CMOS* device such as a *74ACT244* in the *FACT* family of devices. A chain of IMS A100 devices connected by a 10 thou 100Ω impedance track and driven from one end will need a terminating resistor of about 39Ω at the other (figure 12.4). The exact values of terminating component will depend on many factors and the values given here are for guidance only and some experimentation may be needed. In systems using a slow *CLOCK* rate a slower *CLOCK* edge may ease or even remove the termination constraints.

12.3.4 Data input

The input data can be provided either through the memory interface to the *DIR* register or through the 16 *DataIn*[*0..15*] pins. The main constraint on the *DataIn*[*0..15*] pins is that the data should meet the set up and hold times given in the device specification for the relevant *CLOCK* edge. In the majority of cases *DataIn*[*0..15*] will be common to all IMS A100 devices. Termination on the drivers of this bus may be necessary under certain circumstances.

12.3.5 Data output and output ready

The output data can be obtained from the *DOL/DOH* registers via the memory interface or from the 12 *DataOut*[*0..11*] pins. When the *DataOut*[*0..11*] pins are used the two 12 bit words will have to be separated in some way. In systems where less than 12 bits of the answer are required (e.g. to drive an 8 bit D-A converter) it may well be possible to discard one of the two words by choosing appropriate coefficient values and/or selector settings. The *OUTRDY* signal can be used as a basis for a *CLOCK* for a D-A converter or other circuitry but it will need buffering if the load on it becomes excessive. If the full 24 bits are required the *OUTRDY* signal can be used to *CLOCK* edge sensitive latches [1].

12.3.6 Master generated GO

The *master* generated *GO* feature of the IMS A100 was designed principly for small systems where the input data is supplied through the memory interface. On a *master* device, the *GO* pin has a dual function; first to provide a *GO* signal for all the IMS A100 devices, and second to indicate to the system when it is appropriate to write a new value to DIR and hence start another cycle. It is difficult to obtain a high throughput, compared with using *DataIn*[*0..15*], if the output is being read from the *DOL/DOH* registers, particularly in the 4 bit and 8 bit modes. Care is needed to avoid writing a new value into the *DIR* before the old value has been used, (the correct time is indicated by the falling edge of *GO*) or reading the *DOL/DOH* registers while they are being updated (the correct time is indicated by the rising edge of *OUTRDY*). In a multiple IMS A100 system using a *master* it is necessary to update the *slave DIR* registers before, or at the same time as, the *master*. It does not matter which device is the *master* but there must only be one for a given cascade. It is possible to update the *DIR* registers of all the IMS A100 devices by addressing all their *DIR* registers simultaneously by pulling all the $\overline{CS}$ pins low during the write to the *master*'s *DIR*. Alternatively, the input data can be provided to all the *slaves* via the common *DataIn*[*0..15*] which, in order to be safe, will have to remain valid until the rising edge of the *CLOCK* following the falling edge of the *master* generated *GO* signal. In this case the *SCR* registers in the *slaves* must be programmed to accept input data from *DataIn*[*0..15*] and not the *DIR* register. It is not possible for the *master* IMS A100 to take it's data from *DataIn*[*0..15*].

12.3.7 External GO

For high speed systems and for all systems where the input data is only provided via the *DataIn*[*0..15*] port a *GO* signal must be provided by the support system. In many systems where the maximum throughput is required the *GO* signal may be taken high but it is important to keep track of when *DataIn*[*0..15*] is sampled to avoid changing the input data at this time. The *GO* signal may be pulsed every N *CLOCK* cycles at or less than the maximum data rate but any attempt to pulse *GO* at a higher rate will result in a drop in speed due to some of the pulses being ignored. It is important that the *GO* signal changes outside the set up and hold times given in the device specification to avoid the risk of different IMS A100 devices in the cascade falling

out of synchronisation. If IMS A100 devices in a cascade do get out of synchronisation with each other for any reason, they will immediately resynchronise on a new correctly timed *GO* signal.

12.4 Large IMS A100 systems

This section deals with design techniques suitable for overcomming the problems raised when designing systems employing many IMS A100 devices. There is a limit to the number of IMS A100 devices that can easily be put in a single cascade without break and that limit will depend on many factors but especially board size and speed. Many of the problems are the same as those already dealt with in the previous section but more severe. Whilst with a small system it is fair to assume that every IMS A100 is on a single board, this may well not be the case with large systems. However, with care it is possible to build very large systems of IMS A100 devices with a phenomenal performance.

12.4.1 How many IMS A100 devices per board ?

In theory it is possible to put as many IMS A100 devices in a continuous chain as necessary without limit providing all the signals to and from the devices meet the specification. In practice boards have a finite size, bus capacitances build up to unreasonable values, and so on. It therefore becomes necessary to partition the problem. In practice it is possible to put 32 IMS A100 devices on a double extended Eurocard, using a 4 layer printed circuit board together with enough additional logic to allow these boards themselves to be cascaded. Such a board could be regarded as a 1024 stage subsystem. This same technique can be applied to smaller numbers of IMS A100 devices on smaller boards, although address decoding will be easier if 32, 16, 8 or 4 devices are grouped together. Whilst the data throughput of the cascade can be maintained, the speed of the memory interface will be a funtion of the loading of the data lines. For many applications this will not matter but in applications, such as fast adaptive filtering, the rate at which the coefficients can be updated may be important. It is therefore necessary to identify which aspects of performance are important, as they will have a significant effect on the way that the system is implemented.

12.4.2 Cascading boards

This section describes one way of maintaining the maximum throughput of the IMS A100 devices in a multiple board system by the use of pipelining. The general technique is to contain the timing problems to each board separately and to make inter-board communication as easy as possible. Each board has a series of edge triggered latches (e.g. *74374* devices) which latch all syncronous inputs and outputs including *DataIn[0..15]* and *GO*. In principle, all inputs are latched by a *PH1* clock which is inverted to provide a *PH2* clock to latch the outputs and to provide the clock for IMS A100s. The signal is transmitted between boards between the rising edge of *PH2* and the rising edge of *PH1* with the latches acting as drivers (figure 12.3).

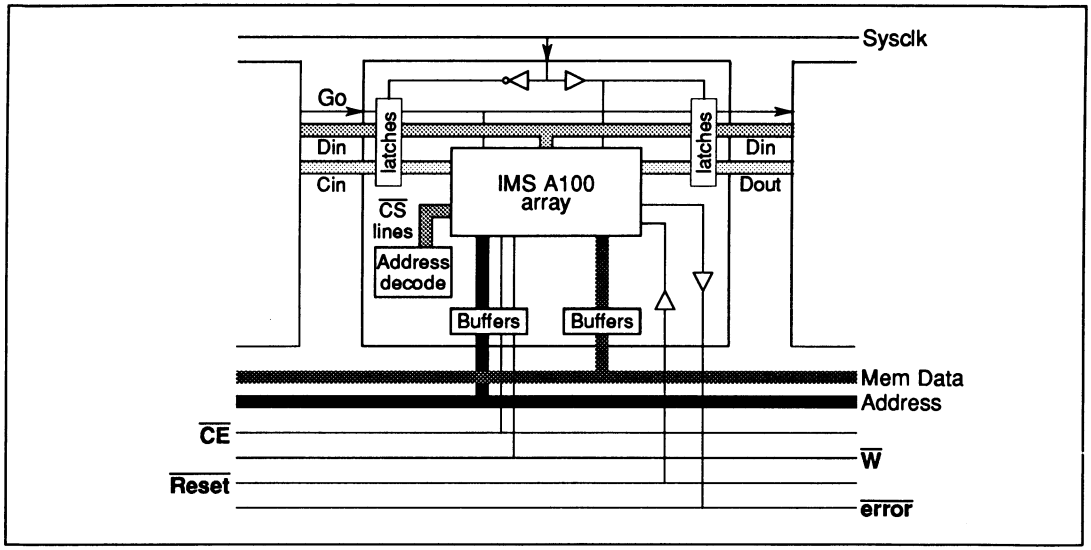


Figure 12.3 Placement of IMS A100 into large system

12.4.3 Board Design

The design of large boards in a cascade requires some care. This section considers the specific example of a cascadable board with 32 IMS A100 devices and support circuitry designed to run with a 20MHz clock with 4, 8, 12 or 16 bit coeficient word lengths. Each of these boards represents a 1024 stage filter and all inputs and outputs are latched or buffered.

Board description

To minimise the length of the connections between *DataOut*[*0..11*] and *CascadeIn*[*0..11*] of adjacent devices, it is best to arrange the IMS A100 devices in a pattern like a snakes and ladder board (figure 12.4). This has the additional advantage that common signals like *GO, CLOCK*, and the various buses may be shared between two rows of devices.

To maximise the density of devices within the board area, half of *DataIn*[*0..15*] and half of the memory data bus pass under each row of devices. The address bus and other signals pass between the first and second rows and third and fourth rows whilst *CLOCK* and *GO* pass between the second and third rows and the fouth and fifth rows respectively (figure 12.4).

The block of IMS A100 devices are mounted away from the board edge connectors. The Cascade input latches drive the top of the IMS A100 block, whilst the output data is produced at the bottom where it is latched. The input data is pipelined to drive the next board as well as providing data for the IMS A100 *DataIn*[*0..15*]. The *GO* signal is pipelined in a similar way to provide a delayed *GO* signal in step with the delayed input and output data. The system clock is used to provide *PH1* and *PH2* signals from two *CMOS* inverting buffer devices located centrally on the connector side of the board. The clock is distributed to keep the timing skew on the clock to adjacent devices to a minimum. The clock tracks driving the IMS A100 devices are terminated by 39Ω for the top track and 27Ω for the rest which are driving two rows of devices. The termination consist of a resistor in series with a 100nf capacitor to remove any *DC* path to *GND*. The tracks carrying the clock between IMS A100 devices were 10 thou wide, but the tracks from the clock driver to the beginning of the block of IMS A100 devices were of a width needed to match the terminating impedance. The *GO* signal is treated in a similar manner to the *CLOCK*, with the option of connecting termination components at the end of each track. The only signals which are not driving every device are the *DataOut*[*0..11*] to *CascadeIn*[*0..11*] links, and the $\overline{CS}$ connections. Each one of these is connected separately to the address decoder.

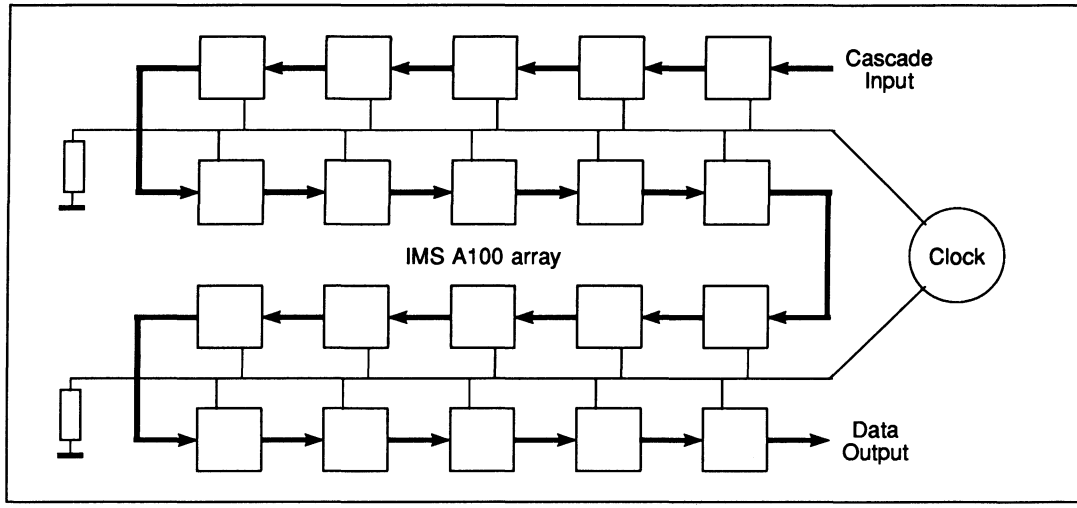


Figure 12.4 Clock distribution for large system

The memory interface buses, the $\overline{ERROR}$ and $\overline{RESET}$ functions are not latched but buffered in some way. The *ADR[0-6]*, $\overline{CE}$ and $\overline{W}$ signals are all buffered from the memory interface. The data bus passes through a bi-dirctional buffer (*74F245* in this case) with it's direction defined by the RnotW signal and with it's tri-state control connected to the board address decoder. Since there are 32 IMS A100 devices together with about 2 feet of pcb track attached to each memory data pin, the $\overline{CE}$ signal needs to have about 150ns duration.

The $\overline{RESET}$ pins of the IMS A100 devices are connected together and connected to some open collector logic plus a resistor to *VCC* and a capacitor to *GND*. This arrangement allows a $\overline{RESET}$ signal to be applied from the system but allows the board to reset itself if no such signal is applied. An LED indicates when the $\overline{RESET}$ signal is high.

The $\overline{ERROR}$ pins are also connected together with a 1KΩ resistor pulling up to *VCC*. This signal is buffered with an open collector gate to allow the boards to be wire ORed if desired. A second LED indicates if an error has occured in any IMS A100 on the board.

Memory Mapping

The exact memory map required will vary from system to system but there are one or two pitfalls which should be avoided. The coefficient registers in the IMS A100 are addressed in such a way that the last coefficient is located at address[0] and the first in location [31]. In order to be consistant with this, the LAST IMS A100 on the LAST board should occupy the LOWEST memory location. Failure to implement this will make the block moving of stored coefficient values to the IMS A100 devices less straightforward than it could have been. If the coefficient registers are to be in a continuous memory space, it is necessary to organize the memory map as follows:

```
System Address bits:     0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

IMS A100 address bits:   0 1 2 3 4                        5  6

Chip Select addresses:           0 1 2 3 4

Board Select addresses:                        0  1  2  3
```

This map assumes a 16 bit system address bus and 32 IMS A100 devices on each board. Remember that the board address space should be large enough to cover both the number of IMS A100 boards required plus any other areas of circuitry requiring address space (e.g. memory etc).

12.5 Higher Data Rates using multiple IMS A100 devices

For some applications, data rates in excess of 10 M samples/sec must be used for real time processing. Since the fundamental maximum data rate of the IMS A100 is 10 M samples/sec, this may appear to be a limiting factor. The following section describes a general method for interleaving multiple IMS A100 devices to achieve effective data rates of 20 M samples/sec and above, with little or no loss of functionality.

12.5.1 Principle of operation

Figure 12.5 shows four IMS A100 devices connected so as to provide the equivalent functionality of a 64 stage, 20 M sample/sec IMS A100. The data rate to each device is reduced by introducing a data demultiplexer, which splits the data stream into two parallel streams. This enables a reduction of input data rate to 10 M samples / sec, the maximum possible for a standard IMS A100.

The segmentation of the problem is achieved because of the transversal filter architecture of the IMS A100. For any transversal filter structure, the summation performed at any given time is as follows.

$$C_0x_0 + C_1x_{-1} + C_2x_{-2} + C_3x_{-3} + C_4x_{-4} + ... \quad (1)$$

where x_{-2},x_{-1}, x_0, ... represent successive data samples in time, and C_0,C_1,C_2, ... represent the coefficients. Equation 1 can be rewritten as follows, with the two halves of the equation performed by two separate devices. This equation may be further generalised into N segments, executed on N^2 devices.

$$(C_0x_0 + C_2x_{-2} + C_4x_{-4} + ...) + (C_1x_{-1} + C_3x_{-3} + C_5x_{-5} + ...) \quad (2)$$

Figures 12.5 and 12.6 illustrate systems exploiting equation 2 to perform 20 M sample / sec filtering. The principle is that the upper pair of devices perform the evaluation for one time period, and the lower devices perform the evaluation for the next, which gives an interleaved computation. In these figures the abbreviations *UL* refers to the upper left IMS A100, *LL* the lower left, *UR* the upper right and *LR* the lower right. The following points should be noted when observing these figures.

- The positions of the coefficients are in reverse order, and are offset between the upper and lower devices.
- One delay stage is required at the input to the *LL* device.
- Both evaluations are being performed at exactly the same time, so that the major cycles of all devices commence on the same clock edge. This is set to coincide with the time that a data sample arrives at the *UL* and *LL* devices.

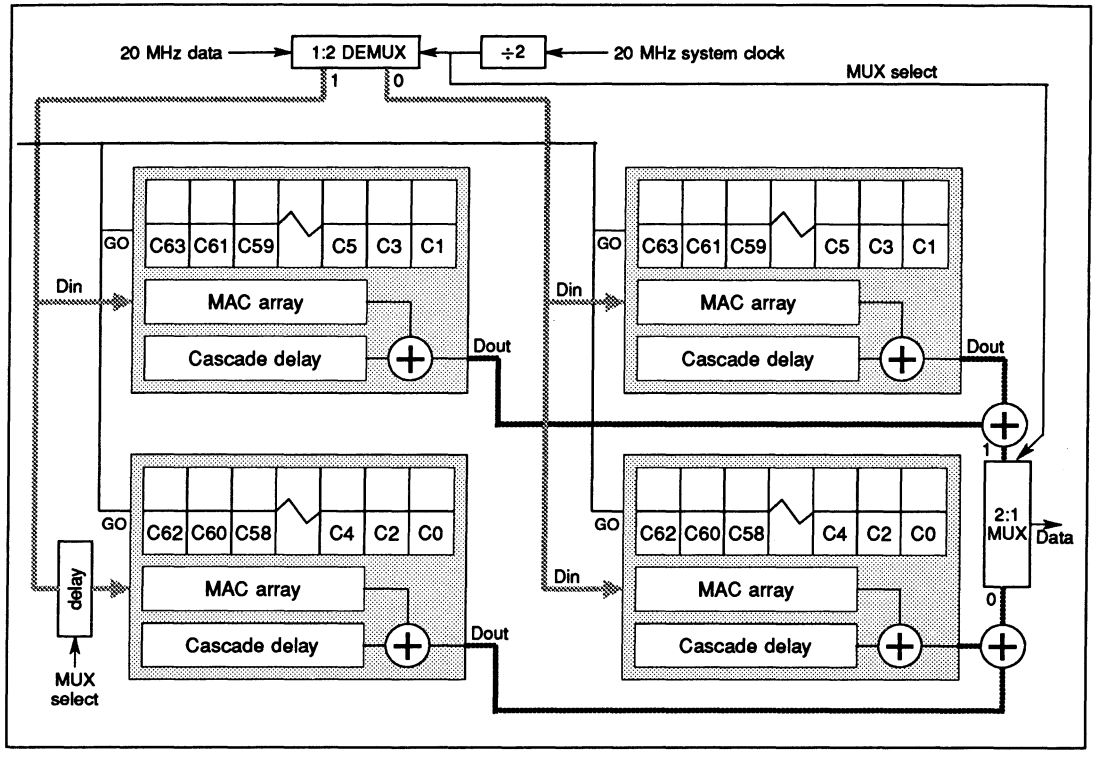


Figure 12.5 20 MHz system using external adders

12.5.2 Mechanics of Operation

To see exactly how the circuit works, consider the following sequence arriving at the data input to the demultiplexer.

$$x_0,\ x_1,\ x_2,\ x_3\ \ldots$$

Assume that x_0 is sent down bus 0, x_1 is sent down bus 1, x_2 down bus 0, and so on. Consider the major cycle that commences for all devices when x_3 arrives at the *UL* device. At that time, x_2 will be at *UR*, x_1 at *LL*, and x_2 at *LR*. For this cycle, the final output from *UL* and *UR* will be as follows.

$$(C_0x_3 + C_2x_1 + C_4x_{-1} + \ldots + C_{62}x_{-59})_{UL} + (C_1x_2 + C_3x_0 + C_5x_{-2} + \ldots + C_{63}x_{-60})_{UR}$$

whilst *LL* and *LR* will produce the following.

$$(C_1x_1 + C_3x_{-1} + C_5x_{-3} + \ldots + C_{63}x_{-61})_{LL} + (C_0x_2 + C_2x_0 + C_4x_{-2} + \ldots + C_{62}x_{-60})_{LR}$$

Thus, the output of the lower pair must be taken first through the output multiplexer, followed by the upper pair. The single delay at the input to *LL* is necessary for the organisation of coefficients. Because C_0x_n must be the last calculation, data must be presented to the device with coefficient C_0 after the device with coefficient C_1.

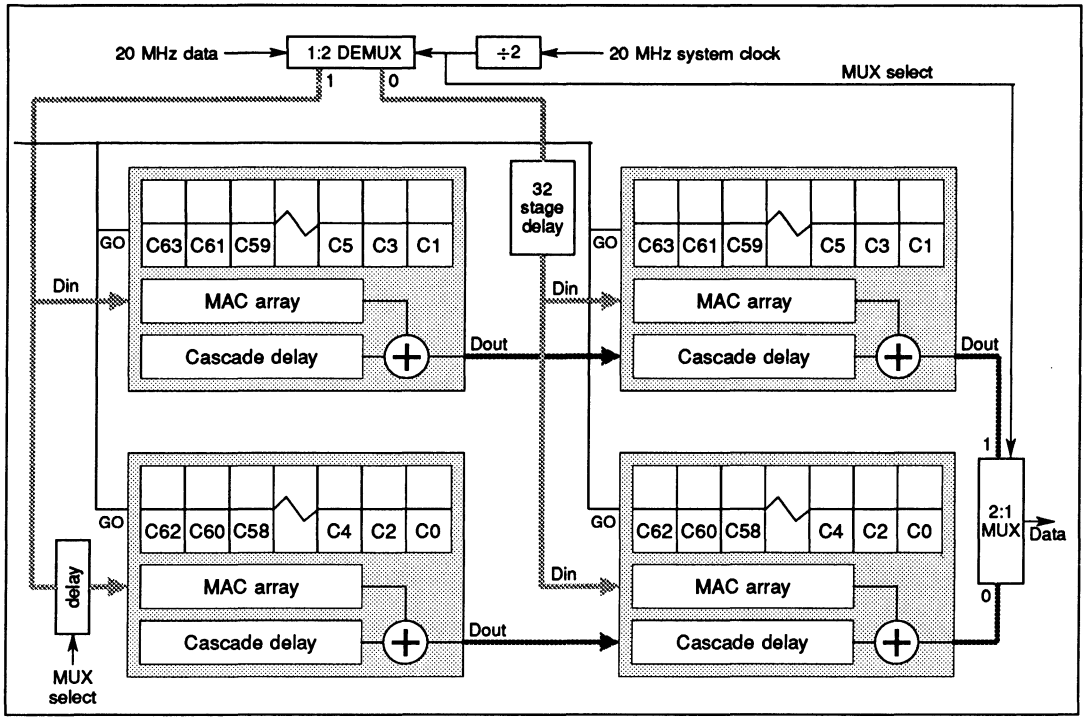


Figure 12.6 20 MHz system using cascade adders

12.5.3 Using the cascade adders

For most applications, it is more convenient to use the cascade adders rather than external devices (figure 12.6). This is because the multiplexed output of the IMS A100 causes complication. A reduction in the number of devices used, and a simplification of the design can be achieved, by using the cascade inputs of the IMS A100.

The cascade adders are used by first inserting a 32 sample delay into the path of the data for the *UR* and *LR* devices, and second, by connecting the data output of *UL* and *LL* to the cascade inputs of *UR* and *LR* respectively. This avoids the use of two accumulator devices, and simplifies circuit board layout considerably. Of the two designs this is the more elegant, and is recommended for use in practice.

12.5.4 Extensions to this technique

Once the above technique functions correctly, many extensions are possible. Some of these extensions make the design even simpler.

- **Higher speed.** By using a 3×3 or 4×4 configuration, data rates of up to 30 M samples/sec and 40 M Samples/sec respectively can be achieved. The only limitation is the speed of the demultiplexing and multiplexing logic. The minimum number of stages using this method also increases proportionally. Thus for 2×2 devices, a minimum of a 64 stage system is produced, which can only be incremented in 64 stage modules. Likewise for 3×3, the minimum number is 96 stages, and for 4×4 the minimum is 128.

- **Cascading.** Since two cascaded IMS A100 devices appear functionally equivalent to one 64-stage IMS A100, each of the four devices shown can be replaced with N IMS A100 devices to form longer

filters. The 32 stage delay would, for example, become 64 stages with two cascaded devices per location.

- **Complex Processing.** The configuration described permits complex processing, using *bank swap* as described in [5]. However, the two multiplexed data streams presented in the illustrated configurations will be the real and imaginary data streams, and the results likewise. Thus, by providing the complex input data correctly skewed in time, the multiplexers are eliminated. This results in a considerable simplification of the design.

- **Removing *UL* delay.** The single stage delay can be removed if less than 64 stages are required. This is done by having a zero coefficient in the coefficient closest to the back end (leftmost on the illustration) of the *LL* device.

- **Removing 32 stage delay.** The 32 stage delay can be eliminated, by zero filling the leftmost coefficients of *UR* and *LR* devices. This, although simplifying the circuit, may be costly, as the IMS A100 stages are being used as delay elements. The merits of this depend on the relative cost of an IMS A100 as compared with the cost of a 32 stage delay element.

12.6 Checking and debugging

This section gives some hints on how to check and debug the IMS A100 part of a system. The IMS A100 has a number of features which make it easy to test within the context of a new or unproven circuit or P.C.B. The general philosphy is to get the memory interface working and then to use the *DIR* and *DOL/DOH* registers to help find any problems. An oscilloscope is also needed to check signals like the *CLOCK*, *GO* and for checking the operation of the various busses.

12.6.1 The Memory Interface

The best way to check the function of the memory interface is to write and read values to either or both banks of coefficient registers. The correct operation of these is independant of the *CLOCK*, $\overline{RESET}$ settings or the contents of the *SCR*, *ACR* or *TCR*. The values that are written at this time are not important but writing 10101... and 01010... patterns will help locate any short or open circuits on the board. If little activity is observed from the IMS A100 then use the oscilloscope to check $\overline{CS}$, $\overline{CE}$, $\overline{W}$, *ADR[0-6]* and *DATA[0-15]* lines, and ensure the presence and correct timing of these signals. The best way to do this is to write a program on the host processor that loops continuously doing writes and reads to one IMS A100. These tests may not identify crossed data or address lines.

It is worth correcting any problems in this area before proceeding to the next series of tests.

12.6.2 Clock, GO and output ready

Before checking these basic functions it is worth reseting the IMS A100 devices either by using the $\overline{RESET}$ pin or by powering down the system. This is to ensure that earlier attempts to debug the memory interface have not accidentally written values to the *SCR*, *TCR* and *ACR* registers. The clock should be checked using an oscilloscope. Problems with impedance mismatching may cause excessive voltage overshoot and/or undershoot, or cause the clock to not meet the specification in some other way through excessive ringing. Lack of drive in the clock driver will cause poor '0' and/or '1' levels. If the clock does not quite meet the specification but is present and is a reasonable shape, it is probably worth leaving the problem until the rest of the system has been debugged.

With the clock running either pull *GO* high or provide a series of *GO* pulses. If an IMS A100 is to be used as a *master* simply pull *GO* high with a resistor for this test since all the devices are still set as *slaves*. Under these conditions the *OUTRDY* pin should be providing pulses.

12.6.3 Setting up SCR values

Before proceeding, the *SCR* registers should be set to #002 in *slave* IMS A100 devices or #003 in any *master*.

12.6.4 Checking the data path

The next step is to check the data path from the input of the first IMS A100 to the output of the last one. It is worth writing a program to display the contents of the *DOL/DOH* registers on a monitor or TV screen.

All coefficients should be set to '0' together with *CascadeIn[0..11]* of the first device. A *GO* signal is now needed which is provided by the support logic or by writing to the *DIR* of an IMS A100. The method depends on the system under test, which will either be a *master* generated *GO* or externally generated *GO*. The value written to a *master* IMS A100 should not effect either its output or the contents of its DIR/DOL regisers, since all the coefficients are set to '0'. If there is the option of placing a value on *CascadeIn[0..11]* of the IMS A100, that value should appear in the *DOL/DOH* register. For a single IMS A100 the value will be delayed by 32 cycles of *GO*, and for many devices the delay will be a multiple of 32 cycles.

The following tests are valid for all coefficient word lengths, although only the least significant 4 bits will be used. The source of the *GO* signal is unimportant and the devices can be set for *fast* or *normal* output mode. However, *SCR[4-5]* should be set to '0' to select the *[7-30]* field, and the answers will then be the same as those given below. The values of coefficients and data and expected are given as hexadecimal numbers.

Set *CascadeIn[0..11]* on first device to	#000000
Set *DIR* on all devices to	#1002
Set all active coefficients to	#0004

If a *master* IMS A100 is used, continuously write #1002 to it's *DIR* register. Otherwise, apply a continuous or frequent *GO* signal, while writing data to the *DataIn[0..15]* port of all the IMS A100 devices. For the first 32 cycles of every device the result in the *DOL/DOH* register should increase linearly, in steps of #4008. The result will be split between the DOL and DOH register so that for the whole result #4008 *DOH* = #0004 and *DOL* = #0008.

1st cycle in the cascade	*DOH*=#0000	*DOL*=#4008
2nd cycle in the cascade	*DOH*=#0000	*DOL*=#8010
3rd cycle in the cascade	*DOH*=#0000	*DOL*=#B018
4th cycle in the cascade	*DOH*=#0001	*DOL*=#0020
5th cycle in the cascade	*DOH*=#0001	*DOL*=#4028
7th cycle in the cascade	*DOH*=#0010	*DOL*=#8030
7th cycle in the cascade	*DOH*=#0010	*DOL*=#B038

This test can be repeated using the *DataIn[0..15]* port instead of the *DIR* registers by setting the *SCR* values in all *slave* IMS A100 devices to #000. If the *GO* signal is generated from a *master* IMS A100 it will still be necessary to write #1002 to the *DIR* register repeatedly. The input value #1002 can be changed to other values to check other bits in the data path. Once the cascade path has been filled, the answers should be stable, and any variation indicates a problem somewhere.

Now that the devices have been exercised, it should be possible to remove any error indication from the $\overline{ERROR}$ pin by writing '0s' to *ACR[1-2]* followed by writing '1s' to arm the register.

The above tests only check the memory interface and the data path through the IMS A100 devices. However, with these working, the debugging of the rest of system is made easier. Once all this works most of the system will be fully functional.

12.6.5 Fault finding guide

It is assumed throughout that power has been applied to all the *VCC* and *GND* pins correctly. If it has not, expect very unpredictable behaviour and/or possible damage to the devices.

When a problem is encountered it is often worth varying the power supply voltage or changing the clock frequency. This will often indicate the nature of the problem by showing if it is due to timing or perhaps noise. The following checks may also help to diagnose the problem.

- If there is response from the memory interface check the following:
    - $\overline{CS}$ is low (when it matters)
    - $\overline{CE}$ is pulsing
    - The addresses are valid
    - $\overline{W}$ is working
    - Any memory bidirectional data buffers are working in the right direction.
- If there is no *GO* signal from a *master* IMS A100 check the following. The clock has to be present and the $\overline{RESET}$ pin high before the SCR, ACR or TCR registers can be written to.
    - There is only ONE *master.*
    - There are no shorts on the *GO* track.
    - *SCR[0]* is set to '1'.
    - *TCR* is set to all '0s'.
    - The clock is present.
    - $\overline{RESET}$ is high.
- If there is no *OUTRDY* signal check the following.
    - *GO* signals are present on some rising clock edges.
    - *TCR* is set to all '0s'.
    - There are no shorts on the *OUTRDY* track.
    - $\overline{RESET}$ is high.
- The answers are wrong, which could be almost anything. However, the following checklist should diagnose the problem.
    - The *SCR* registers are set to the correct value.
    - The *ACR* registers are set to the correct value.
    - The *TCR* registers are set to all zeros.
    - All IMS A100 devices are in the same output mode. (*SCR[10]*)
    - There is only one *master.* (*SCR[0]*)
    - The output word selection is sensible. (*SCR[4-5]*)
    - The data input source is correct. (*SCR[1]*)
    - The coefficients word lengths are right. (*SCR[8-9]*)

- The coefficients are stored in the right bank.
- *DOL* and *DOH* are read in the right order.
- The memory data and address lines are in the right order.
- *OUTRDY* is not inverted wrongly in any external logic.
- 4, 8 and 12 bit coefficients are written into the least significant bits of the 16 bit wide coefficient registers.
- Input data is valid when sampled.
- The order of the coefficients has not been mangled by the memory map.

12.7 Conclusions

This application note deals with the hardware aspects associated with the IMS A100. Other application notes produced by INMOS relating to the design of systems employing the IMS A100 are given in the bibliography.

This document describes how system hardware with few to very many IMS A100 devices can be designed and debugged. The information from 3 separate designs of board, designed by INMOS engineers, has been accumulated in this document. These boards include the IMS B009 (a plug in card for the IBM PC having 4 IMS A100 devices), a high performance FFT/convolution module board with 4 IMS A100 devices and a large system board with 32 IMS A100 devices.

12.8 References

1 *IMS A100 data sheet*, INMOS Limited, Bristol.

2 *Digital filtering with the IMS A100*, Application note 1, Hossein Yassaie, INMOS Limited, Bristol.

3 *Discrete Fourier transform with the IMS A100*, Application note 2, Hossein Yassaie, INMOS Limited, Bristol.

4 *Correlation and convolution with the IMS A100*, Application note 3, Hossein Yassaie, INMOS Limited, Bristol.

5 *Complex (I & Q) processing with the IMS A100*, Application note 4, Hossein Yassaie, INMOS Limited, Bristol.

image processing with the IMS A100

13.1 Introduction

13.1.1 The aims of this document

The IMS A100 performance makes the real time processing of digital images a practical possibility. This document is a practical guide, which explains how the device is used to process digital images. The processing done by the IMS A100 will be some form of feature extraction, such as line, corner or edge detection. Feature extraction is often the first stage in the analysis of an image. Further analysis of an image, for example, deciding that a group of features in an image is a vehicle number plate, is a higher level function, beyond the scope of this document. This application note describes the following.

- The operations of filtering and edge detection of a picture or image using a technique of 2-dimensional convolution are explained. Some simple filter types including edge detection and contrast enhancement are described.
- The use of the IMS A100, to perform the 2 dimensional convolution, in order to process an image is described. This shows the simplicity of use of the IMS A100 in this particular application.
- The estimation of performance and cost, for processing an image using the IMS A100 is described. Several possible systems consisting of IMS A100 devices are given, to illustrate how easily the cost and performance may be controlled, by using different numbers of devices, and by altering the complexity of the system.
- The processing of images at real time speeds (20 frames per second) is described, and a hardware implementation of this is given. This shows the high performance possible using the device.

13.1.2 Document structure

The remainder of section 13.1 gives an introduction to signal processing, and shows the position of the IMS A100 within the field of signal processing, and more specifically its capabilities for the processing of digital images.

Section 13.2 gives gives a practical explanation of some of the concepts of image processing. Included is an explanation of how filtering and edge detection of a picture operates, and how this may be applied to the IMS A100.

Section 13.3 gives two possible systems which may be constructed using the IMS A100, from a medium performance system to a very high performance system which will operate at real time speeds. Included in this section is a description of how the performance of a prospective system may be estimated by trading off performance, complexity and cost.

Section 13.4 concludes and summarises the findings of this application note.

Section 13.6 gives an implementation of 2-D image processing using the IMS B009, running on an IBM PC. This is included as a practical illustration of the techniques described in section 13.2 and 13.3.

13.1.3 An overview of signal processing

Signal processing is an area of engineering which fills many people with dread. This is not entirely surprising when one considers both the theoretical and practical aspects of the subject. On the one side there are the mathematical algorithms required to solve even the simplest problem. This has long been regarded as the territory of academics and not to be tackled by the average engineer. On the other side there is the circuitry required to implement these algorithms. Historically, systems have often required many complex circuits, with system design requiring a knowledge of analogue design, and also, in the more recent past, digital design.

Not surprisingly, there are very few scientists in the world with the knowledge or experience required to deal with all the aspects of signal processing design. Signal processing design now covers both analogue and digital design from the low end audio spectrum (40 KHz) through the video spectrum (100 MHz) to the top end of the radio spectrum (100 GHz). When signal processing in all these areas began the techniques used were

purely analogue. The power of digital signal processing now approaches the top end of the video spectrum. Although it is not yet possible to process pictures the size of a TV screen in real time, it will undoubtedly become possible within the next decade. One of the main applications of the IMS A100, as described in this document, is the processing of pictures in real time.

In the radio frequency (RF) spectrum, specialised devices are used as the first stage processing elements. These devices use components such as wave-guides, to give the necessary processing bandwidth (GHz). The fastest devices use materials such as Gallium Arsenide, often super-cooled to improve its performance. However, these devices are expensive and their use is avoided if possible. The information extracted by these devices from a signal may be used by todays digital devices operating at speeds approaching 100 MHz. In the future, todays digital devices may improve to a level where they encroach on the radio spectrum. However, it is likely that RF devices will always be required as the front-end processing elements at these high frequencies. The reason for this may remain that it is impossible to either sample or synthesise an analogue signal at speeds in excess of 100 MHz, without resort to cost prohibitive technology.

13.1.4 Analogue and digital conversion

Signal processing techniques in both the Audio and Radio spectrum are advancing both theoretically with the development of new algorithms, and practically with the increase in the level of integration of integrated circuits. Wherever possible, the new levels of integration in conjunction with efficient digital algorithms are used, so that problems which were previously solved using analogue design are now solved using digital design.

Of course, it is nearly always necessary to communicate with the real world using analogue signals, so analogue to digital (A-D) and digital to analogue (D-A) converters are a necessity. This is why so much work is done to increase the speed and accuracy of the conversion which must ultimately limit the speed of the complete system.

The current range of A-D and D-A converters on the market can sample at up to 100 MHz. As might be expected, the limiting speed depends very much on the required accuracy, with slower conversion required to get improved accuracy. Of course, there is little point being able to do digital processing faster than the analogue conversion devices, so that in practice, the performance of conversion devices and digital processing devices proceed together.

So there are fundamentally two problems which hinder DSP development, one is analogue/digital conversion and the other is the digital signal processing itself.

13.1.5 Techniques for digital signal processing (DSP)

Digital signal processing has advanced rapidly since the major semiconductor manufacturers started to tackle the problem. Since then they have attempted to cram more and more raw processing power onto a single chip. At the same time they have realised that the signal processing devices need to be integrated into an entire system. So, they have devised families of devices which, however, require some considerable expertise to use. This evolution of devices has split into two directions.

The first approach is the more complex and achieves the best performance. It often involves hardware design which is not trivial, and the systems generated will generally only perform one task. Any slight change to the task (algorithm) may require a complete system redesign, which is both lengthy and expensive. However, the performance of these so called bit-slice machines has been and still is very high and has a permanent place in the field. Bit-slice machines use dedicated multipliers, accumulators and address sequencers often with several address and data bus paths to achieve high speed.

The second approach is simpler, and more versatile. However, the performance is considerably lower than the bit-slice engines previously described. Design involves using a general purpose processor (CPU) which has dedicated instructions to perform reasonably fast multiply, divide, add and subtract operations. The CPU does this by having dedicate parallel multipliers and barrel shifters integrated on the chip. The performance limit is not so much the on-chip operation as the time required to get the data off and on chip (memory bandwidth). Possibly the best known example of a signal processing CPU is the TMS32010[1] and its derivatives the TMS32020 and TMS32030.

[1]TMS is a trademark of Texas Instruments

The previous two approaches provide solutions to a large number of signal processing problems. However, one must accept either the performance limitations of the general purpose processor or the complexities of bit-slice design. In both cases the problem is bandwidth into the basic processing element. The fundamental limit is the rate at which memory can be accessed rather than the performance of the processing element itself. If the processing performed by the basic processing element can be increased and the required memory bandwidth can be reduced, an improved performance will be immediate. The IMS A100 uses a novel architecture to achieve these aims.

The IMS A100 is a processing element with considerable processing power, yet having an interface with moderate bandwidth requirement. This is achieved by having data storage on chip, processing the data in parallel, and storing the intermediate results of calculations. The IMS A100 has also been designed to accommodate many of the commonest DSP algorithms; including the discrete Fourier transform [2], correlation and convolution [3], and digital filtering [1].

13.1.6 Overview of image processing with the IMS A100

The IMS A100 is a digital processing device at the forefront of digital signal processing performance. It is capable of processing video bandwidth signals, as well as many other types of high bandwidth signals. The maximum input sampling rate of the IMS A100 is 10 MHz, which means that it could, for example, process a [512 × 512] image at a rate of 40 frames per second. The device operates on digital data with a width of 16 bits, and will perform 80 million multiply accumulate operations per second (80 MOPS) a performance well in excess of most bit-slice machines.

The IMS A100 will perform calculations on signed 16 bit integers without any loss of accuracy or overflow, perform rounding correctly, and will also perform complex number processing [4] without any additional hardware. This makes it an extremely simple device to use in a wide variety of applications, as it deals with so many of the problems which have historically plagued signal processing design. Immense care has been taken to ensure that the device is simple to use, for example, the microprocessor interface, which can be interfaced very simply with almost any industry standard processor.

Probably the most important aspect of the IMS A100 is that several can be used in parallel, with almost no 'glue' logic. In principle, there is no limit to the number and a system with 30 devices on a single board has been shown to work well. The processing of large images at high speed requires vast processing performance, making the IMS A100 capability of being able to use many devices in parallel absolutely invaluable.

13.2 Practical methods of 2 dimensional convolution

13.2.1 2-dimensional convolution

The process of 2-dimensional convolution of an image is the action of comparing a reference template with a group of pixels, at every pixel point on an image. For example, if a [3 × 3] template were compared at every point on an image of size [5 × 5], there would be 9 valid comparison points as shown in figure 13.1. The first of these valid comparisons surrounds pixel 1, the second pixel 2, and so on. The comparison is done in practice by a number of multiply and add operations. Consider the example with the first row being compared with the template. The result of the [3 × 3] convolution for the first 3 positions will be

$$1 \quad a.? + b.? + c.? + d.? + e.1 + f.2 + g.? + h.4 + i.5$$

$$2 \quad a.? + b.? + c.? + d.1 + e.2 + f.3 + g.4 + h.5 + i.6$$

$$3 \quad a.? + b.? + c.? + d.2 + e.3 + f.? + g.5 + h.6 + i.?$$

which is a total of 9 multiply-accumulate operations for every pixel in the image. The magnitude of the image data and the magnitude and sign of the template elements determine the type of features which will be extracted from the image. Some simple templates are described later in this section.

In a real image, the magnitude of the pixels which is a measure of their blackness, is referred to as grey scale, having typically 8 bit accuracy. The alternative, which uses a single bit for each pixel, was used in the past, before digital grey scale processing was possible. Future picture processing will undoubtedly be capable of processing colour images. This is a complex field, little understood at the present time, outside

the scope of this application note.

With grey scale images it is important that the result of any image transformation yields grey scale values within the limits of the original image. This being so, the sign and magnitude of the elements of the template must be chosen with care. It may be necessary to scale and/or invert the results of an image transformation, so that the resultant image can be observed in a normal grey scale.

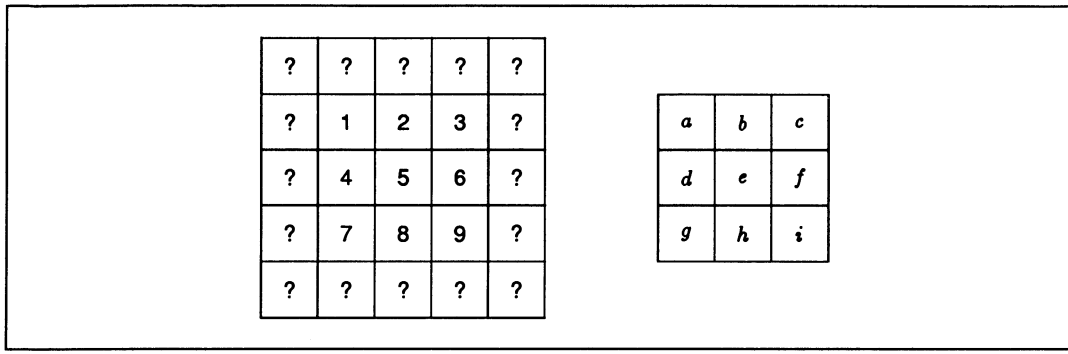


Figure 13.1 [3 × 3] convolution on a [5 × 5] image

It is usual for the template to be square, although it may be rectangular, and of any size. It is also normal when scanning a real image to traverse the picture as shown in the diagram, i.e. traversing a row, moving down, traversing again and so on until the entire image is scanned.

One point of interest concerns the outermost pixels, which represent invalid data. For a [3 × 3] template a perimeter of one pixel width is invalid, for a [5 × 5] template the outermost 2 pixels are invalid and this redundancy increases as template sizes increase. This does not matter much for large image sizes, but must be borne in mind if large templates, with small images are being used. For the remainder of this section edge effects will, for convenience, be ignored.

13.2.2 Convolution template types

Low pass filter

The effect shown in figure 13.2, is of a low pass filter. The numbers have been chosen to show the smoothing effect of the filter. Notice that this is indeed a low pass filter, and that the pixel values are changing at a frequency which is approximately the cut-off frequency of the filter. The filter has effectively changed a black and white image into a blurred grey image.

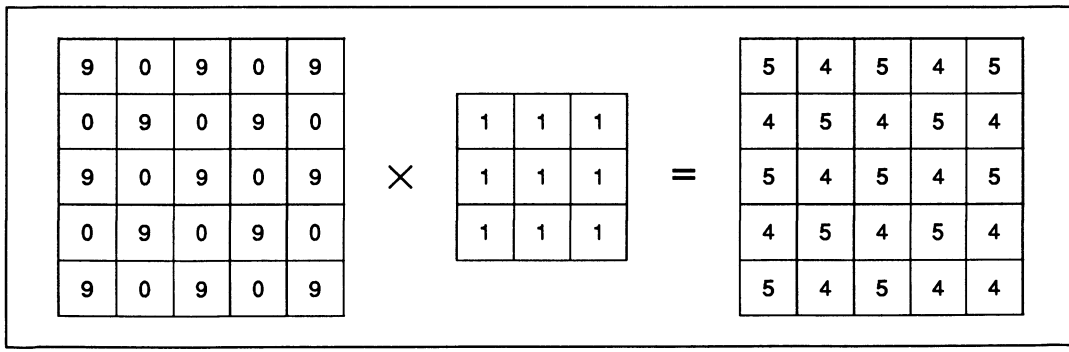


Figure 13.2 Illustration of low pass filter

If this convolution is regarded as part of a picture with a pixel rate of 5MHz the cut-off frequency, above which all frequencies are removed, would be 2.5MHz. (The figure of 5MHz has been chosen as it is the rate at which an IMS A100 can process the data.) The cut-off frequency can be reduced by making the reference template (convolution kernel) larger. For example a [9 × 9] convolution kernel would have a cut-off frequency of 870KHz.

For the low pass filter kernel no sign modification or scaling of the final image is necessary. Only when the result is outside grey scale limits will any modification be required.

Edge detection

Edge detection is illustrated below with a Sobel operator. This operator combines a vertical and a horizontal edge detector into a single Sobel operator as shown in figure 13.3.

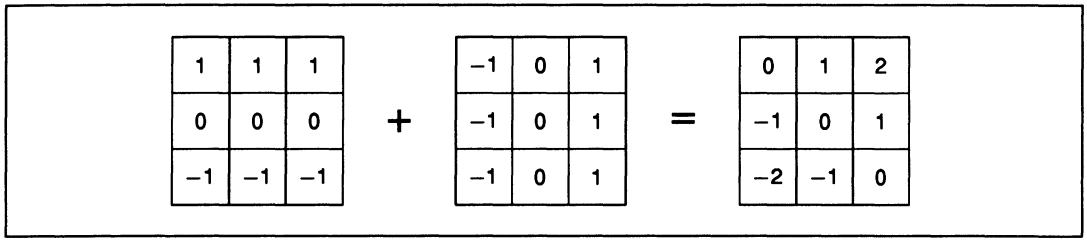


Figure 13.3 Sobel operator formation

It may be observed that the effect of applying a horizontal edge detection to an image, followed by applying a vertical edge detection to an image, and summing the results, will be exactly the same as directly applying the sobel operator to the image. This same principle of adding operators together, may be applied to many different operators with some interesting results. It is not within the scope of this application note to investigate this subject further.

The following operations, shown in figure 13.4, on part of an image illustrate the effect of the Sobel operator. It is possible to obtain similar results, by doing a vertical and a horizontal edge detection, squaring and adding the results, and taking the square root to give a result for each final pixel. This is the ideal edge detector, but the cost of squaring twice and a square root is often cost prohibitive, with the Sobel operator a very satisfactory alternative.

Figure 13.4 shows the result of 2 convolution kernels operating on the different images. This illustrates the requirement for scaling and sign inversion. Before the resultant images can be displayed all negative numbers must be sign inverted and a scaling factor of 4 must also be applied. It is interesting to note that the reason for the sign change is the direction of travel of the convolution kernel across an edge transition. Also, as will be shown later, the steepness of the edge transition is important.

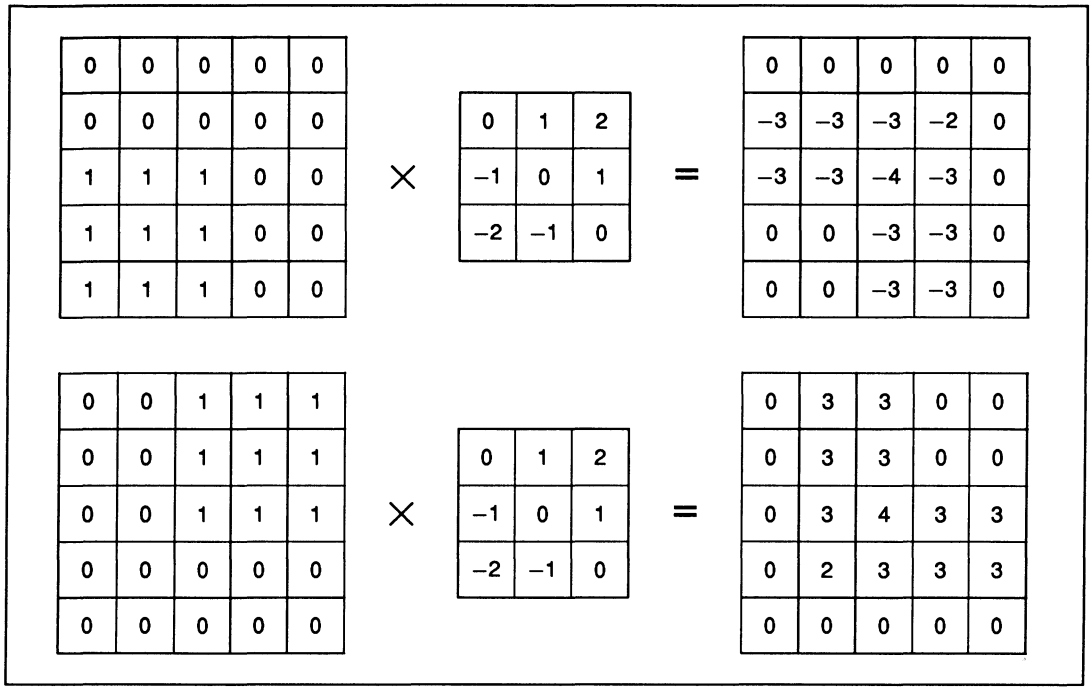


Figure 13.4 Illustration of edge detection

Laplacian filtering (edge detection)

Laplacian filtering uses a homogeneous operator, which means that it is the same in all directions. With the use of a Laplacian edge detection operator edges in all directions can be detected. This is different from the previous non-homogeneous edge detection (Sobel) operator, where edges in all directions except one can be detected.

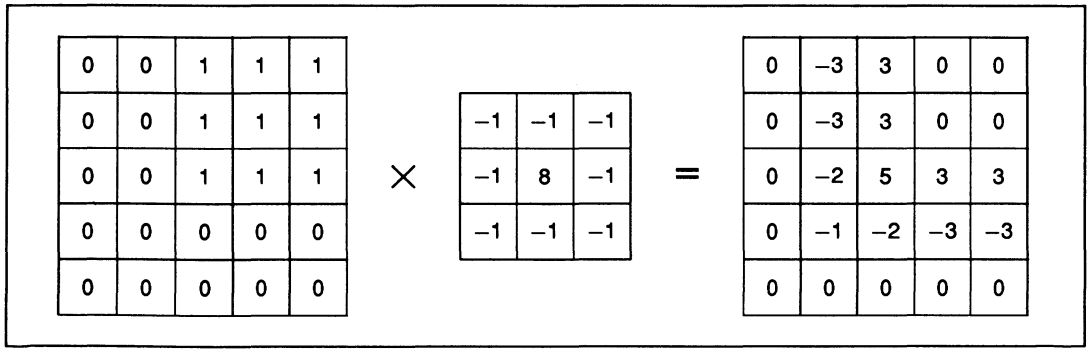


Figure 13.5 Illustration of Laplacian Filter

The effect of the [3 × 3] Laplacian operator is shown in figure 13.5. As the operator passes over an edge, the magnitude of the result increases and there is a sign change. Also, the original pixels will remain, in areas where there is no edge to be detected. In order to detect the edges, after the 2-D convolution has been done, a 3 stage process is necessary. First the result is scaled down by a factor of 9. Second, a full rectification

is done, converting all negative to positive numbers. Third, the background information is thrown away, by introducing a suitable threshold below which the result is considered to be zero.

13.2.3 Effect of template size

The previous sections have shown the effect of several [3 × 3] convolution kernels. This kernel size is very effective in many applications, while requiring moderate processing for its calculation. One of the main reasons for not using larger templates is that the processing requirement becomes excessive, mainly due to the large number of multiply operations. The advantages of large kernels are twofold, firstly the larger kernels have a filtering effect which reduces the effect of noise, and secondly the larger kernels are able to detect gradual changes in brightness across a group of pixels.

It must be remembered that single bit pixels are not used, and the pixels are represented in grey scale with range from 0 to 255. This means that in a real image, edges will span several pixels, and to detect these edges will require a larger convolution kernel.

If, for example, a [3 × 3] kernel is used to detect an edge which changes from black to white linearly over 5 pixels, then the maximum and minimum resultant pixel is 1.6 and −0.6 respectively, as shown in figure 13.6. Each pixel is represented by a single step. This result must be compared with the result in figure 13.5, where an instantaneous change between 2 pixels gives an output of −3.0 and 3.0. The results of −0.6 and 1.6 are barely enough to detect an edge.

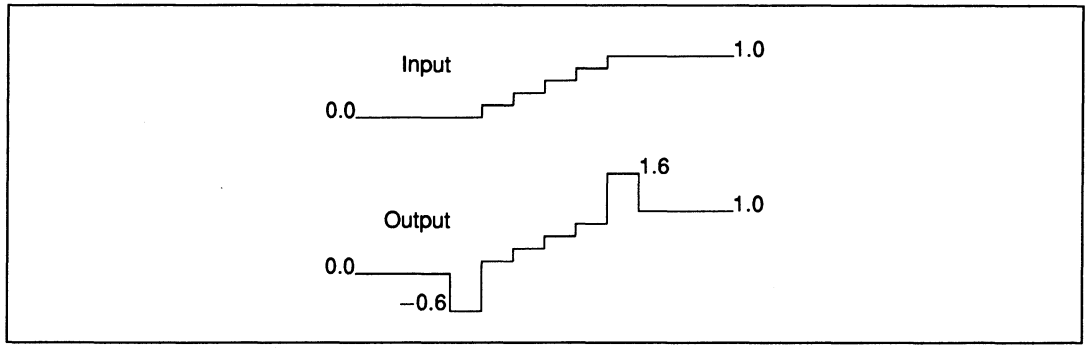


Figure 13.6 Effect of gradual edges on convolution output

13.3 Hardware requirements for 2-D convolution

Two possible hardware implementations of 2-D convolution using the IMS A100 are described in this section. Because these two implementaions use exactly the same principle of operation, the IMS A100 device, which is common to both, will first be described. The fundamental difference between the two designs is as follows. In the lower performance system all image data is transferred to the IMS A100 across a comparatively slow memory interface. In the high performance system all image data uses the dedicated input and output ports of the IMS A100. These dedicated ports permit, with the addition of some dedicated hardware, a processing rate of 5 million pixels per second.

Implementation of 2-dimensional convolution on the IMS A100 involves loading the elements of the convolution kernel into the coefficient registers, and passing the entire image through the device while storing the resultant image. To obtain maximum throughput this should be a continuous operation, and will consist of a sequence of alternate read/write operations starting at the first pixel of the first row of the image and finishing with the last pixel of the last row of the image. The two fundamental problems are to arrange that first, the convolution kernel elements are loaded into the appropriate coefficient registers, and second that the pixel data is ordered correctly both before and after processing. The required initialisations of the IMS A100 are also described.

13.3.1 The IMS A100 model

The IMS A100 model is shown in figure 13.7. The many component parts of the IMS A100 are included to add flexibility, so that many signal processing algorithms can be implemented. This means that the device can be used in many signal processing applications. The fundamental operation of the device is a high speed multiplier-accumulator, which functions as a pipeline of 32 multiplier-accumulator devices. The peripheral circuitry simplifies the use of the device.

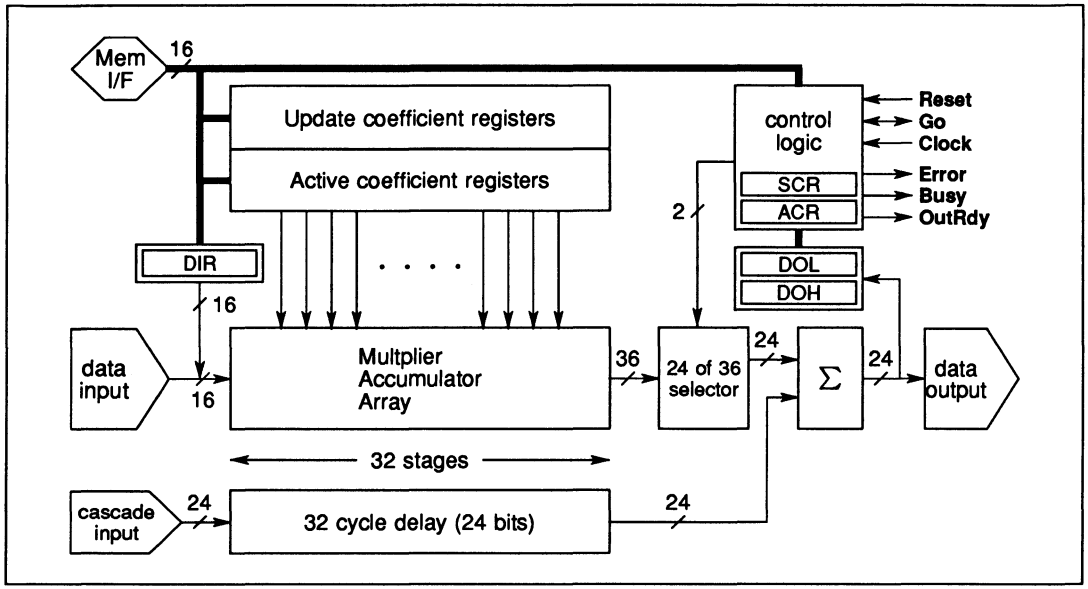


Figure 13.7 User's model of the IMS A100.

The essential elements of the device, in so far as they are important to this discussion of 2-D convolution, will now be described.

- **The multiplier accumulator array** is the powerhouse of the device. This is a 32 stage pipeline of multipliers, which multiply 32 elements of input data with the contents of the current coefficient registers in between 2 and 8 cycles. The cycle time is between $100ns$ and $400ns$, and is a function of the coefficient width. There is no loss of accuracy in this section because all calculations are at 36 bit accuracy.

- **The data input** is either from the dedicated data input port or via the data input register (DIR). The DIR can be accessed from the memory interface port, which may be connected to a microprocessor. The fastest data access is by the direct input port, with input using the DIR register being usually 2 to 4 times slower.

- **The data output** is taken from the high speed multiplexed data output port or from the data output registers (DOL and DOH), which contain the 24 bits of output data. These registers, like the DIR register, will normally be accessed much slower than the direct data output port.

- **The coefficient registers** are used to store the convolution coefficients, for which only the current coefficient registers are required. Because there is no need to bank-swap coefficient and update registers, neither the update coefficients nor the bank swap capability are ever used in this application.

- **The cascade input** is used for simple connection of devices. The cascade input port is multiplexed in exactly the same way as the data output port, so that direct connection between the two and use

of the GO signal for synchronisation are all that is required to cascade devices. For 2 dimensional convolution requirements, only one IMS A100 is required for doing a convolution with a kernel containing less than 33 elements, although more devices can be used for improved performance as will be shown later. Whenever more than one device is used the cascade input port is required.

- **The 32 cycle delay** element delays the cascade input data by exactly the same time as the multiplier accumulator array. It can be used in conjunction with the data input port to add together 2 streams of pipelined data. This is very useful, particularly for convolution requiring data partitioning. Real time 2-dimensional convolution of images using the IMS A100 requires the use of this delay element.

- **The control registers** are used to initialise the device, and for some of the working operations of the device. These are referred to as the Static control register (SCR) and the Active control register (ACR) respectively. The ACR can be altered during the operation of the device whereas the SCR cannot. The use of these registers as regards 2-dimensional convolution will be described later.

- **The output signals** are described adequately in the IMS A100 data sheet [6] and will not be described further here, except for the GO signal which is relevent to the discussion. The GO pins, of all the IMS A100 devices which are cascaded together, will be joined. GO is used for synchronisation of a cascade of devices, and is not needed in a system with only a single IMS A100, unless the cascade input of that device is used. GO is set up from the SCR to be either a master or slave, and there is never more than one master.

- **GO** is a special signal used to synchronise the cascade and data input ports. If the data input port **Din** or the cascade input port **Cin** is driven by external hardware, then all the IMS A100 devices will be set to slave mode and external hardware will be used to drive the GO pin. If neither the cascade input port or data input port are driven by external hardware, (when all data will use the memory interface) then one of the IMS A100 devices in the cascade will be configured as a master. The master which should be the **last** device in the cascade, drives the GO signal, and all the other devices synchronise their cascade and data inputs from the GO signal they receive. The GO signal master could in theory be driven by any of the devices in a cascade, and this would work for a short cascade. However, operation cannot be guaranteed, whereas an infinite length cascade will work if the master is the last device in the cascade.

13.3.2 IMS A100 initialisations for convolution

The following description summarised the initialisations of the IMS A100 devices which will be required, prior to the operation of 2-D convolution. A full understanding will require the use of the IMS A100 data sheet [6]. The settings necessary for a 2-dimensional convolution, using 8 bit grey scale data and 8 bit coefficients are described.

The coefficient size is set to 8 bits by setting bits 8 (=1) and 9 (=0) of the SCR. As 8 bit grey scale is used the top 8 data bits from either the data input port or DIR (each 16 bits wide) will be zero.

The result of the 8 by 8 multiplication will require 16 bits, and the 32 stages of accumulation will require a further 5 bits so that the final result, will require 21 bits accuracy. The result required is manipulated internally by a selector so as to be invisible to the user. The significant 8 bits of the result are obtained by setting bits 4 (=0) and 5 (=0) of the SCR, and reading data from the bottom 8 bits of the DOL register.

If there is a cascade of devices the lower 8 bits of output appear on the lowest 8 bits of the multiplexed data output port, which will be connected to the cascade input of the next device in the cascade. By this means scaling is done automatically, and is invisible to the user. The whole purpose of this is that many devices can be cascaded, and appear like a single device with a number of stages which is a multiple of 32.

The remaining SCR register settings are as follows. Bank swap mode will be set to off. Data mode will be set to either input data from the DIR register or data input port depending on the application. Fast output will be set to off for this application.

The ACR will not generally be needed for this application as no bank swapping between the active and update coefficient registers is necessary. It may be necessary to examine the selector overflow and cascade adder overflow bits of the ACR should an error occur (error pin goes low).

13.3.3 IMS A100 coefficient placement and data flow

Section 13.2.2 describes some of the convolution kernels which are used to perform feature extraction and filtering on an image. The following discussion describes how these kernel elements are mapped onto the coefficient registers of the IMS A100 so that 2-D convolution is performed.

The IMS A100 can be regarded as a 32 stage multiplier accumulator with 32 constant coefficients, which will be consecutively multiplied with a stream of incoming pixels. The current coefficients are labelled from C(0) to C(31) where C(0) is closest to the output, and C(31) is closest to the input, as shown in figure 13.8.

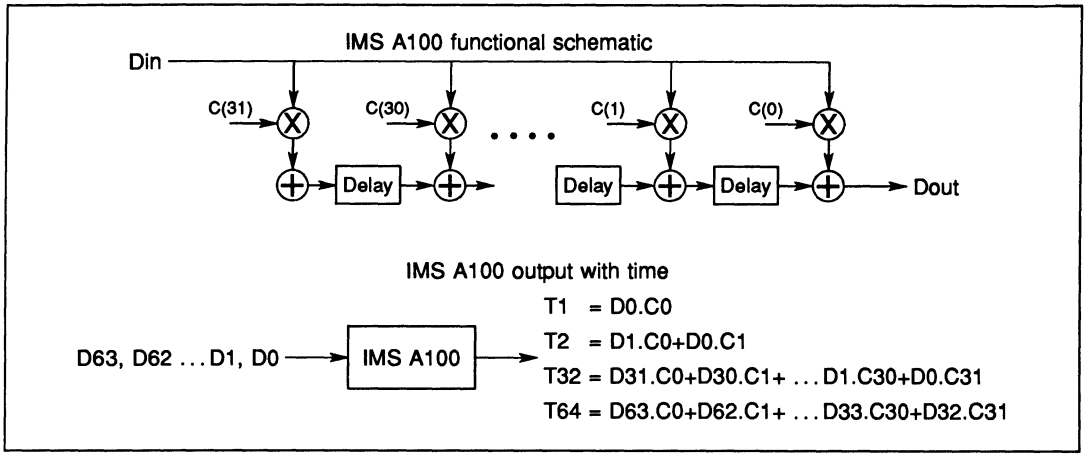


Figure 13.8 Illustration of IMS A100 pipelined calculation

In figure 13.8, pixel data presented at the Din port (or DIN register) at time 0 is referred to as D0. Immediately after data is written, at time T1, a result will be read from the Dout port (or DOL/DOH register). For the first 32 cycles (T1 to T32) of the IMS A100, partial results for data D0 to D31, and coefficients C(0) to C(31) will be output from the device. The results at time T1 and T2 are given.

From T32 onwards the device presents full results at its output, and the result at time T32 and T64 are given to illustrate this. The steady state of the device yields the accumulation of 32 multiply operations which have taken place over the previous 32 cycles. Notice also that at any instant the machine contains 32 pieces of information (state), which are the 32 partially accumulated results, as they proceed through the 32 stage pipeline.

If there is a cascade of 2 devices, there are 64 coefficients which can be referred to as C[0] to C[63]. The output from the second device in the cascade is the sum of 64 multiply operations which have accumulated over the previous 64 cycles. This principle can be extended to many IMS A100 devices, so that long multiply-accumulation operations can be done. It is essential to be able to perform long cascades so that large convolutions are possible. For example, a 128 point convolution will require 4 IMS A100 devices in cascade.

This also applies to 2-Dimensional convolution. For instance, an [11 × 11] convolution using 121 stages, will require 4 IMS A100 devices. Of course, 7 stages are not required, which means that 7 of the coefficients (C[127] to C[121]) of the first IMS A100 in the cascade will be set to zero.

As can be recalled from section 13.2, a [3 × 3] convolution requires the accumulation of 9 multiply operations. Similarly, a [5 × 5] convolution, illustrated in figure 13.9, will require 25 stages of multiply-accumulation. The only problem is that the coefficients must be loaded in the correct coefficient locations, and the input and output data must be ordered correctly, so that the IMS A100 architecture can be utilised. This is described in the following section.

13.3.4 Image scanning for a microprocessor based system

The following description will normally only apply to a system using a memory interface, for the transfer of all data to and from the IMS A100. It is perfectly possible to use the following pixel sequencing operations, for transferring data to the IMS A100 devices across the high speed data input and output ports. However, this is not advised as the sequencing operations using normal hardware are complex, but are quite easy with a microprocessor. The additional hardware could be better used for implementing an extremely high performance system, such as described later.

Image scanning for 2-D convolution implementation

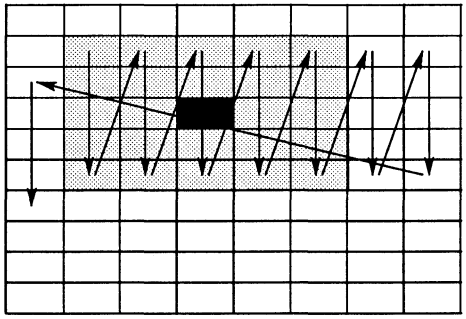
(A) Pixel array

D0	D10	D20	D30	D40	D50	D60	D70
D1	D11	D21	D31	D41	D51	D61	D71
D2	D12	D22	D32	D42	D52	D62	D72
D3	D13	D23	D33	D43	D53	D63	D73
D4	D14	D24	D34	D44	D54	D64	D74
D5	D15	D25	D35	D45	D55	D65	D75
D6	D16	D26	D36	D46	D56	D66	D76
D7	D17	D27	D37	D47	D57	D67	D77
D8	D18	D28	D38	D48	D58	D68	D78
D9	D19	D29	D39	D49	D59	D69	D79

(B) IMS A100 coefficients

C24	C19	C14	C9	C4
C23	C18	C13	C8	C3
C22	C17	C12	C7	C2
C21	C16	C11	C6	C1
C20	C15	C10	C5	C0

(C) Scanning for a single pixel



(D) Scanning for next pixel

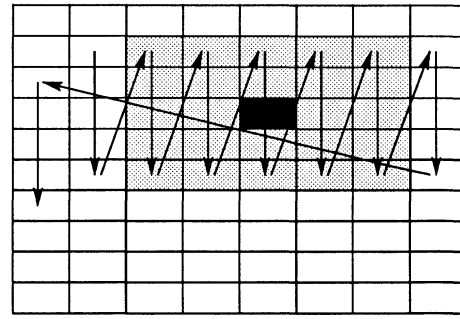


Figure 13.9 Pixel scanning and coefficient ordering

A pixel array (A) with 10 rows and 8 columns is used purely for convenience. The convolution kernel with 25 coefficients is shown in (B). The order of the coefficients is critical, starting at the bottom right and proceeding one column at a time (Remember that C0 is the coefficient of the last stage of the cascade). The scanning pattern for the image is shown in (C) and (D). The dark black squares are valid output pixels, each of which represent the convolution of 25 pixels with 25 coefficients.

If the light grey area of pixels is written to **Din** as shown in (C) the order will be D11 then D12 and so on in columns until D55 is written. Immediately after D55 is written to **Din** a valid pixel is read from **Dout**. The value of this pixel will be

D33out = C0.D55 + C1.D54 + C2.D53 + ... + C23.D12 + C24.D11

After this D51 is written followed by D52, D53, D54, D55 after which another valid pixel can be read.

D34out = C0.D65 + C1.D64 + C2.D63 + ... + C23.D22 + C24.D21

In other words, for every 5 pixels written one valid pixel is read, from the beginning until the end of the row. At the end of the row go back to the start of the row and move down a row repeating until the entire image is scanned. The net effect is a completely convolved image. This is inefficient as the entire image is effectively written to the IMS A100 **FIVE** times.

There is fortunately an optimisation which can be incorporated. The principle is that the some of the IMS A100 coefficients are set to zero, so that those stages act only to store and delay accumulated results. This is described in the following section.

Improved image scanning for 2-D convolution

Improved performance can be obtained by modifying the previous image scanning technique. The improvement is obtained not by processing the individual pixels faster, but by passing the pixels through the IMS A100 fewer times. This is illustrated in figure 13.10 below.

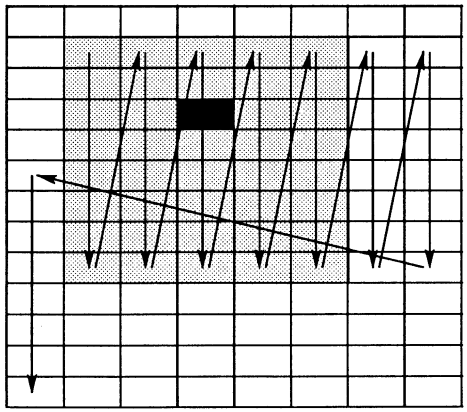
(A) Pixel array

D0	D10	D20	D30	D40	D50	D60	D70
D1	D11	D21	D31	D41	D51	D61	D71
D2	D12	D22	D32	D42	D52	D62	D72
D3	D13	D23	D33	D43	D53	D63	D73
D4	D14	D24	D34	D44	D54	D64	D74
D5	D15	D25	D35	D45	D55	D65	D75
D6	D16	D26	D36	D46	D56	D66	D76
D7	D17	D27	D37	D47	D57	D67	D77
D8	D18	D28	D38	D48	D58	D68	D78
D9	D19	D29	D39	D49	D59	D69	D79

(B) IMS A100 coefficients

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
C36	C28	C20	C12	C4
C35	C27	C19	C11	C3
C34	C26	C18	C10	C2
C33	C25	C17	C9	C1
C32	C24	C16	C8	C0

(C) Scanning for 4 pixels



(D) Scanning for next 4 pixels

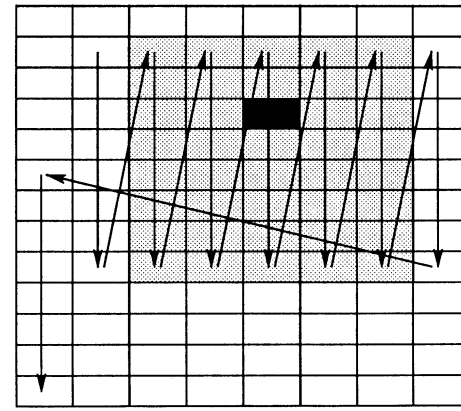


Figure 13.10 Pixel scanning and coefficient ordering – high performance

Data is written to the IMS A100 devices as before except this time data is scanning over 8 rows at a time, starting at D11 and finishing at D58. Scanning over the fifth column of 8 pixels from D51 to D58 will yield 4 valid pixels, starting at the black square.

D33 out = C0.D55 + C1.D54 + C2.D53 + ... + C23.D12 + C24.D11
D34 out = C0.D56 + C1.D55 + C2.D54 + ... + C23.D13 + C24.D12
D35 out = C0.D57 + C1.D56 + C2.D55 + ... + C23.D14 + C24.D13
D36 out = C0.D58 + C1.D57 + C2.D56 + ... + C23.D15 + C24.D14

Immediately after D33out is read from **Dout**, D56 is written to **Din**. The partially accumulated result for pixel D43out is then stored in the empty slot (The empty slot is the position in the IMS A100 which would accumulate C5 with the data at **Din**. As this coefficient is set to zero no accumulation takes place, and this stage acts as delay and storage of data only) The next pixels D56 and D57 are written, and the partially accumulated result for D44out and D45out are then stored in the IMS A100 pipeline. At this time there will be 3 partially accumulated results D43out, D44out and D45out, which will be required for processing the next column.

At the end of the column scan, after the pixel D58 is written, D61 followed by the remainder of that column of 8 pixels, are written. This yields a further 4 pixels as given below.

D43out = C0.D65 + C1.D64 + C2.D63 + ... + C23.D22 + C24.D21
D44out = C0.D66 + C1.D65 + C2.D64 + ... + C23.D23 + C24.D22
D45out = C0.D67 + C1.D66 + C2.D65 + ... + C23.D24 + C24.D23
D46out = C0.D68 + C1.D67 + C2.D66 + ... + C23.D25 + C24.D24

The scanning technique which scans across 8 rows at a time, while 4 rows of pixels are written, is 2.5 times more efficient than the previous technique, where only 5 rows are written, for a single output row. It is simple to calculate the efficiency for any number of *zeros* inserted into the coefficients of the IMS A100.

Convolution efficiency

For any system, there will be a fundamental pixel processing rate. As shown in section 13.3.5 the processing rate, writing pixels across a high performance memory interface is unlikely to be better than 4 Mpixels per second. Realistically 2Mpixels per second is a more probable performance. However, as shown above, there will be an efficiency factor, dependent upon the the convolution technique used, which will reduce the performance further. The best that can be done uses an image scanning pattern as shown in figure 13.10.

To calculate the efficiency, the number of stages and the number of zeros must be known. These calculations assume that the maximum number of zeros will be used, for whatever number of A100s are selected.

$$stages := number.of.A100s \times 32 \tag{1}$$

$$zeros := \frac{stages}{(filter.size)^2} DIV (filter.size - 1) \tag{2}$$

$$Efficiency := \frac{zeros + 1}{zeros + filter.size} \tag{3}$$

As a simple example, it is known to take $500ns$ to process a single pixel, and the efficiency is calculated at 50%. The expected processing rate will be 1Mpixels per sec.

There is an obvious trade off between the number of A100 devices used and efficiency. A small number of zeros increases efficiency greatly. However, as efficiency approaches 100% the added cost of more IMS A100 devices, to give more stages, will not give a proportionate increase in performance. No figures are given here, as it is simple to calculate the numbers, for any given application.

13.3.5 Moderate speed image convolution

A moderate image convolution rate can be obtained by using a very simple design incorporating an 8 or 16 bit processor, which controls one or several IMS A100 devices. A typical system is shown in figure 13.11. The system chosen uses an extremely high performance 16 bit processor, the IMS T212. The limiting speed of this system is either the rate at which data can be transferred across the IMS A100/IMS T212 memory interface, or the rate at which data can be transferred to and from an external system. The external system may consist of camera, frame grabbing hardware and some form of image displaying capability. For the purpose of argument it will be assumed that the IMS A100/IMS T212 memory interface is the limiting factor.

The performance of this system may be easily estimated, for whatever processor is used. This estimation assumes that the image resides in memory before processing starts, and that the data input port is not used. The resultant image will also reside in memory after processing.

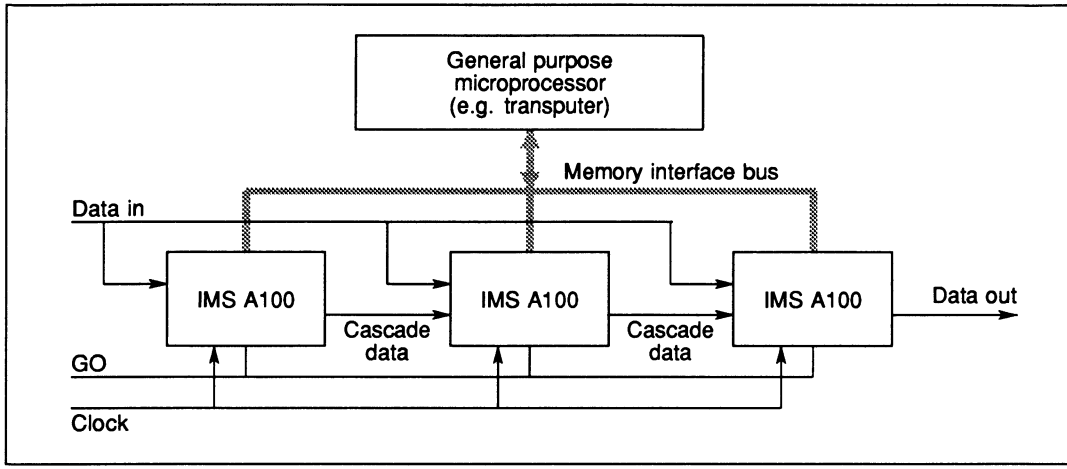


Figure 13.11 IMS A100 coefficient loading

The processing of a single pixel involves 5 steps which are in order

read from memory	= 100ns
write to IMS A100	= 100ns
IMS A100 processing,	= 200ns
read result from IMS A100	= 100ns
write result to memory	= 100ns

This shows that the time to process a single pixel will be 600ns so that for a complete picture of size [512×512] a processing rate of 6 frames per second may be possible. While this may be achievable in theory there are several problems which lead to a reduction in this performance.

- The data must be transferred both to and from the system, before and after processing. In practice this may take longer than the processing itself.
- The data must be read and then written by the processor, usually into an internal register, which will consume at least 2 extra cycles (100ns minimum)
- The data accessed by the processor will be in the form of a 2 dimensional array of pixels. The processor has to calculate the array subscripts for every pixel in the image, which will consume at least 4 processor cycles (200ns). The order with which pixels are loaded is not simply row by row and is described elsewhere. (section 13.3.4)
- The nature of the convolution algorithm means that the image may need to be split up into small blocks, which must be overlapped, to give a continuous convolution of the entire screen. This will result in an inefficiency of between 10% and 50% depending on the size of the blocks and the degree of overlap.
- The algorithm implemented on the IMS A100 has a fundamental efficiency as described in section 13.3.4. Equations are given for the calculation of this efficiency. The algorithm should be arranged so that the efficiency is better than 50%.

The effect of all these factors that a simple microprocessor based system is likely to have a processing rate of between 1 and 4 frames per second. Even this will require an high performance processor with optimised software.

13.3.6 Very high speed image convolution

In section 13.3.2 it is shown that complete images can be processed on the IMS B009 at between 1 and 4 frames per second. In appendix A, implementation of the convolution algorithm running on the IMS B009, under an IBM PC environment is less than 1 frame per second. The actual IMS A100 processing time is only 200ns per pixel, which is less than 10% of the total processing time. As the IMS A100 device is such a fast device it seems wasteful to reduce the performance to this extent. In practice, many users of the IMS A100 will wish to extract full performance which requires a hardware design.

Faster speeds require a slightly altered algorithm and dedicated hardware. The following describes a hardware implementation capable of processing speeds up to 20 frames per second. The hardware setup is shown in figure 13.12 This figure illustrates the hardware required to perform a [3 × 3] convolution on a [512 × 512] image. Larger convolutions on larger images are possible with the addition of extra hardware. For example, a [31 × 31] image convolution could be done using 31 IMS A100's and 30 sets of shift registers. Each shift register has a 512 + 32 stage delay.

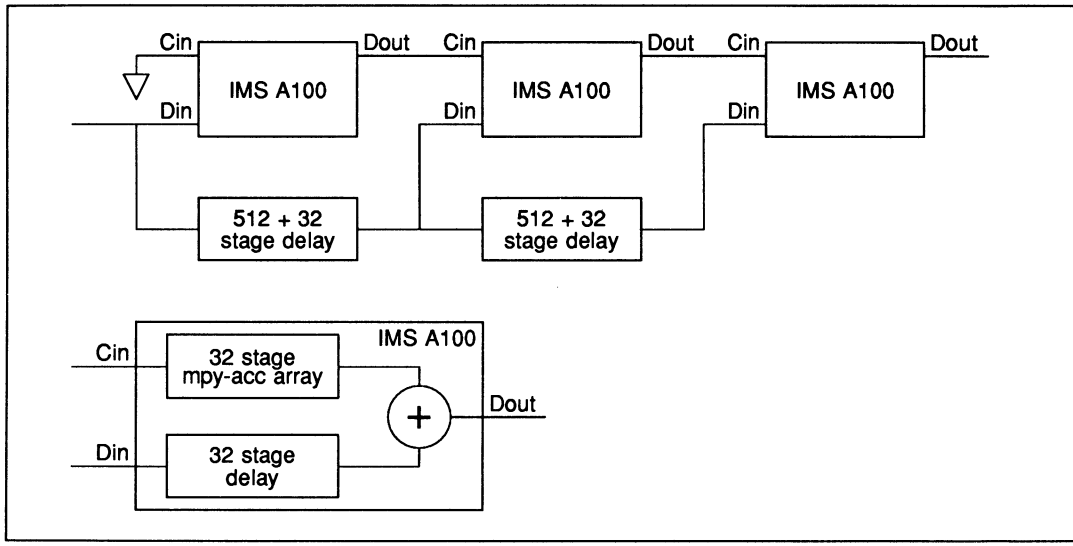


Figure 13.12 Hardware for real time image convolution

In the example shown, 2 rows of data are stored in long shift registers, while the third row enters the data input port of the first IMS A100 in the cascade. The first IMS A100 in the cascade has its cascade inputs grounded, while all other IMS A100 devices have their cascade inputs connected to the data outputs of the previous IMS A100 in the cascade.

The arrangement ensures that 2 pixels one above the other in an image are fed simultaneously into the cascade input and data input of each IMS A100. The IMS A100 devices will process one line of pixels at a time, and the entire image will eventually pass through every IMS A100. Because the system is fully pipelined this results in no performance degredation.

Each stage of the cascade requires an IMS A100 and a long shift register. The IMS A100 is like a 32 stage delay element, where the cascade input delayed by 32 stages is added to the data input after it has been through a 32 stage multiplier-accumulator array. The 32 stage delay of the IMS A100 means that the shift register delay must be a single line delay (512 pixels in this case) plus 32 stages.

The data throughput rate depends on the coefficient size selected for the IMS A100's, and is unaffected by the number of stages. If 8 bit pixels with 8 bit coefficients are selected, a data rate of 5 KHz is achieved. For a [512 × 512] image this gives a convolution time of one frame in 50 mS. This is a frame rate of 20 Hz with faster speeds achieved by selecting regions of interest or multiplexing frames between several boards.

As a further example application, it is required to pass a [31 × 31] pixel kernel template over a [1024 × 1024] image at 50 frames per second. Processing will require 310 IMS A100 devices, segmented over several boards. One configuration would use 30 [1024 + 32] delay stages, and would require 31 IMS A100's to process the image in 200 ms. Therefore 10 such boards would be required, and a means of multiplexing the individual images at a bandwidth of 50 Mpixels per second. This is mainly a problem of data distribution. The processing problem has been solved by the capability of the IMS A100.

13.4 Conclusions

This application note has shown how the IMS A100 may be used to perform fast processing of digital images. Some typical image processing functions have been described, including edge detection and filtering of a picture.

The versatility of the IMS A100 has been shown by the following observations.

- The IMS A100 is capable of processing images at real time speeds, 20 frames per second, and a hardware implementation of this has been described. This processing rate is possible because of the high speed sampling rate of 10 million pixels per second of the IMS A100.
- It is simple to build a lower performance system consisting of several IMS A100s and a microprocessor, with a capability of processing at between 2 and 4 frames per second. The simplicity of the microprocessor interface which turns the device into a memory mapped peripheral helps to achieve this.
- Identical hardware implementations can be used for different sizes and types of 2-Dimensional filters. It is therefore not difficult to modify the signal processing algorithm after the hardware design is complete. This is eased by the general capabilities of the IMS A100 for many signal processimg applications, involving the implementation of specific algorithms on the general purpose architecture.
- Even after hardware design is complete it is possible to increase or reduce performance as necessary, simply by increasing or reducing the number of IMS A100 devices in the system. This is only possible because the device is designed specifically so that several may be used in parallel.

13.5 Recent advances – the IMS A110

The IMS A100 is the first in a family of DSP devices, and the observant reader will realise that there are several inefficiencies with using the IMS A100 for 2 dimensional convolution. For this reason, INMOS have a dedicated device aimed specifically at 2D convolution applications.

The device, known as the IMS A110, has an architecture similar to that shown in figure 13.12. The device handles 8 bit data and a single device is capable of performing 7×3 image convolution at a rate of 20 M samples per second (4 times as fast as the IMS A100). Further, the line delay elements, as shown in this figure, are integrated onto the chip so that the device can process video signals **directly**, without any frame buffering. The device has several other useful features including

- **Cascadability.** It is possible to perform convolution in any multiple of 7×3, so for example 14×3, or 7×6 or 21×9 convolution is possible. No additional hardware is required for this, only a multiple of IMS A110 devices.
- **An output look up table.** At the output of the device is a look up table which may be modified across a microprocessor interface. This facilitates such useful things as dynamic range enhancement and operates at the full speed of the device.
- **A max/min register and statistics monitor post processing unit.** It can be very useful to monitor the magnitude of signals passing through the device, and this can be done without stopping processing.

The IMS A110 achieves all this by having on-chip programmable delay lines, and an on chip post-processor for data transformation, as well as the the basic 21 stage multiplier-accumulator. It is dedicated to high

speed video applications, and minimises system hardware requirements in these applications. The IMS A100 by contrast, is a general purpose device which can for example perform 2 dimensional FFT calculations, 1 dimensional FFT, convolution, correlation and complex processing on 16 bit data.

Despite the advantages of the IMS A110 in video applications, it may be practical to use the IMS A100 for large 2-Dimensional convolution sizes. For example, a 28x30 convolution would require 40 IMS A110 devices. The same could be achieved with 28 IMS A100 devices, plus line delay elements.

13.6 Implementation of convolution on the IMS B009

One possible application of the IMS A100 cascadable signal processor is the processing of two dimensional images, such as those obtained from a TV camera. The analogue signal from the camera must first undergo analogue to digital conversion, and may then be presented to the IMS A100 for processing. The size of the image used in this application note is by convenience [512 × 512] pixels. The reason why this size was selected was that a camera and frame grabber board were readily available. The IMS A100 is capable of processing any size of image including, for example, larger images such as high resolution satellite pictures of the earth.

This section shows how the IMS A100 may be used to process a [512 × 512] pixel image. The work described involves no hardware design and uses two standard boards supplied for the IBM PC. This shows how the IMS A100 can be used to perform 2 dimensional filtering, convolution or correlation.

The system, shown in figure 13.13 is composed of a camera, monitor, digitising frame grabber board and IMS B009 board. Both the frame grabber board and the IMS B009 [5] board are standard plug in boards for the IBM PC.

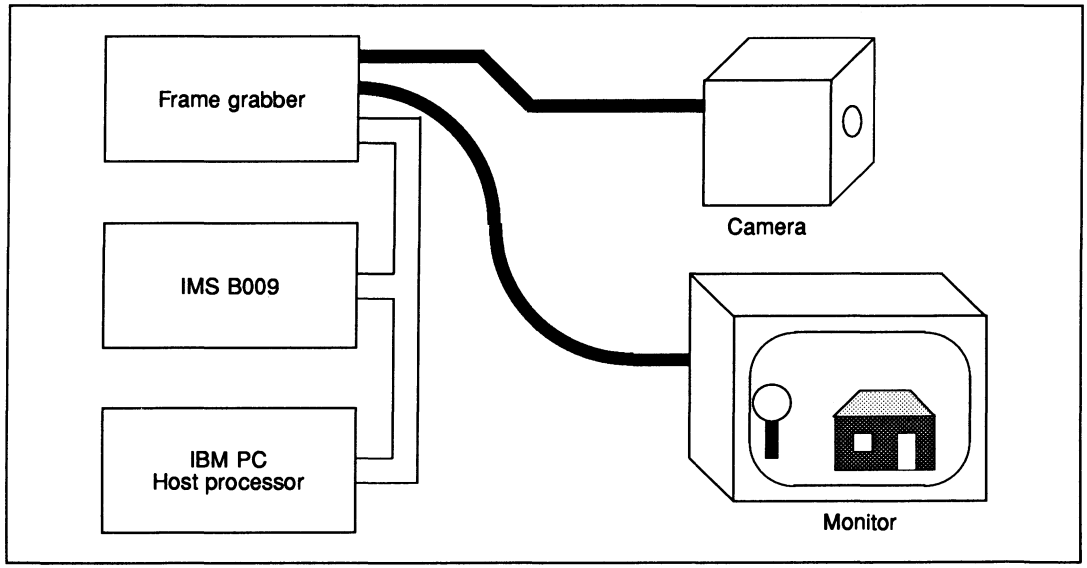


Figure 13.13 IBM PC and associated hardware

Software which controls the system runs on the IBM PC host processor and is written in Turbo Pascal, Version 3.0. This program commands the frame grabber board to grab images which are then transferred to the IMS B009 via the IBM PC host processor. The image is then transformed, using the convolution algorithm running on the IMS B009, and transferred to the frame grabber board to be displayed on the monitor. It is also possible to dump pre-processed and post-processed images to disc. This can be useful where further processing is required. The software running on the IMS B009 is written in occam, although it could also be written in C or Pascal. A knowledge of occam will be helpful for a full understanding of the implementation

of this algorithm. The program is a modified version of the IMS D703 development software [5], which is generally available for DSP development with the IMS A100.

The image transformation technique uses a 2 dimensional image convolution algorithm, and is demonstrated by performing edge detection and filtering on an image. The purpose of the computer program is only to demonstrate the technique, not to investigate the enormous number of image transformations which are possible. The program enables further investigation of 2 dimensional convolution kernels with the minimum of effort. This program is not available as a product but may be obtained by contacting the DSP group based at Bristol.

The performance of the system does not utilise the full performance of the IMS A100. There are two major reasons for this. Firstly, all the data which is processed by the IMS A100 is transferred between the processor and IMS A100, across a comparatively slow memory interface. Secondly, all the data is transferred to the processor across comparatively slow links. The net effect of these factors is to reduce the effective data rate by between 1 and 2 orders of magnitude. The bandwidth possible through the dedicated ports of the IMS A100 is 10 Mbytes/sec. This bandwidth yields a maximum frame rate of 20 frames per second, while the performance shown in the rest of this section is no better than one frame per second. A brief description of how to obtain full performance from the IMS A100 is given in section 13.3.6.

13.6.1 Frame Grabber support

The frame grabber board is a card available for the IBM PC which can perform frame grabbing operations on a video signal from an external source. The board used is a matrox PIP-1024 capable of storing a [1024 × 1024] image or 4 individual [512 × 512] images, with a resolution of 8 bits per pixel.

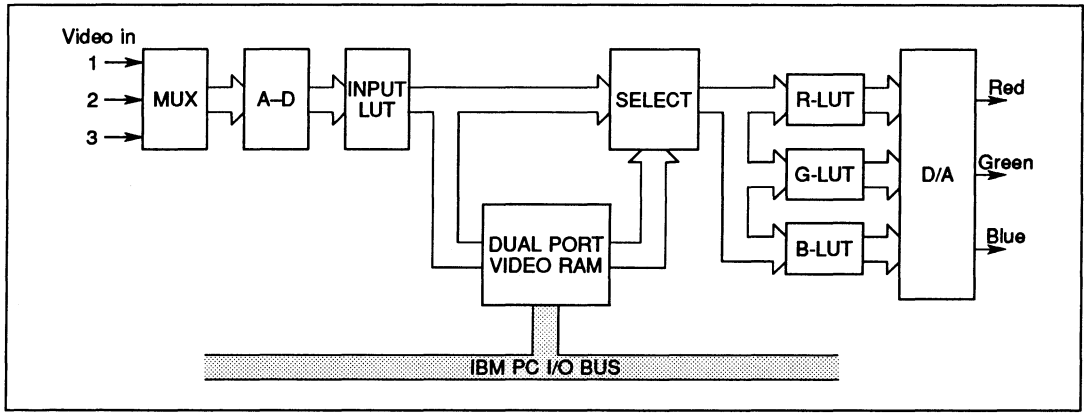


Figure 13.14 Frame grabber hardware support

The board can be used in conjunction with the IBM PC host processor to perform various signal processing functions, such as horizontal and vertical line detection, using a [3 × 3] image convolution algorithm. Using the IMS A100 devices on the IMS B009 results in a factor of 5 performance improvement. A dedicated hardware implementation using the IMS A100 will achieve at least a factor of 50 performance improvement.

The board is used to continuously grab pictures and display them, to freeze frames, grab single frames and to accept single frames from the IBM PC bus which may then be displayed. The board is controlled directly by the Turbo-pascal program which runs on the IBM PC host processor.

13.6.2 The IMS B009 hardware

An overview of the IMS B009 is shown in figure 13.15 , and a more detailed description of the key components is shown in figure 13.16 The IBM PC host processor communicates with the IMS C011 across the IBM PC data bus. Conversion into the standard inmos link protocol is performed by the IMS C011, and an optional link jumper is used to connect this link to one of the links of the IMS T414. A further fixed serial link communicates

with the IMS T212. There are several other links which may optionally connect to other parts of a system. It is possible for example to connect several IMS B009 boards together and form a larger system. Another possibility is to use several links between the IMS T414 and IMS T212 to increase communication bandwidth.

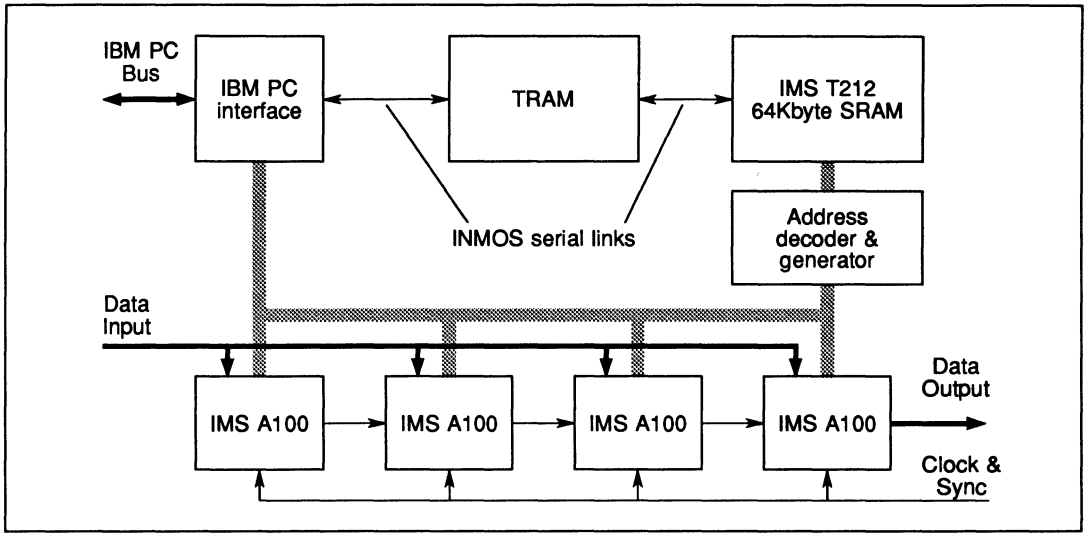


Figure 13.15 IMS B009 overview

The 64 Kbytes of SRAM act as program and data memory for the IMS T212. The high speed data in and data out interfaces to the IMS A100 cascade are available at an external connector, for maximum speed operation of the IMS A100 cascade. In this application all data input/output is performed by the IMS T212, across the slower microprocessor interface with the IMS A100.

The 4Kx12 SRAM look-up table, multiplexer and address decoder are used to speed the transfer of data during processing. The flow of data during the application of a typical signal processing algorithm will be from the frame grabber, across the IBM PC data bus, through the IMS C011 link adapter and into the memory of the IMS T414. Data is then transferred using the transputer block move engine, across the transputer link, at about 900 Kbytes per second, into the memory of the IMS T212. The data is then processed by the IMS T212 in combination with the IMS A100 cascade and the result transferred to the IMS T414. From the IMS T414 the result may be transferred back to the IBM PC host processor, either to be filed on disc or to be displayed.

The IMS B009 also has a direct interface between the IBM PC bus and the IMS A100's. This is only used when the IMS T212 is disabled. This mode of operation of the IMS B009 is not used in this application and will not be discussed further. The use of this interface is for slow access to the IMS A100 from a program written in any programming language on the IBM PC.

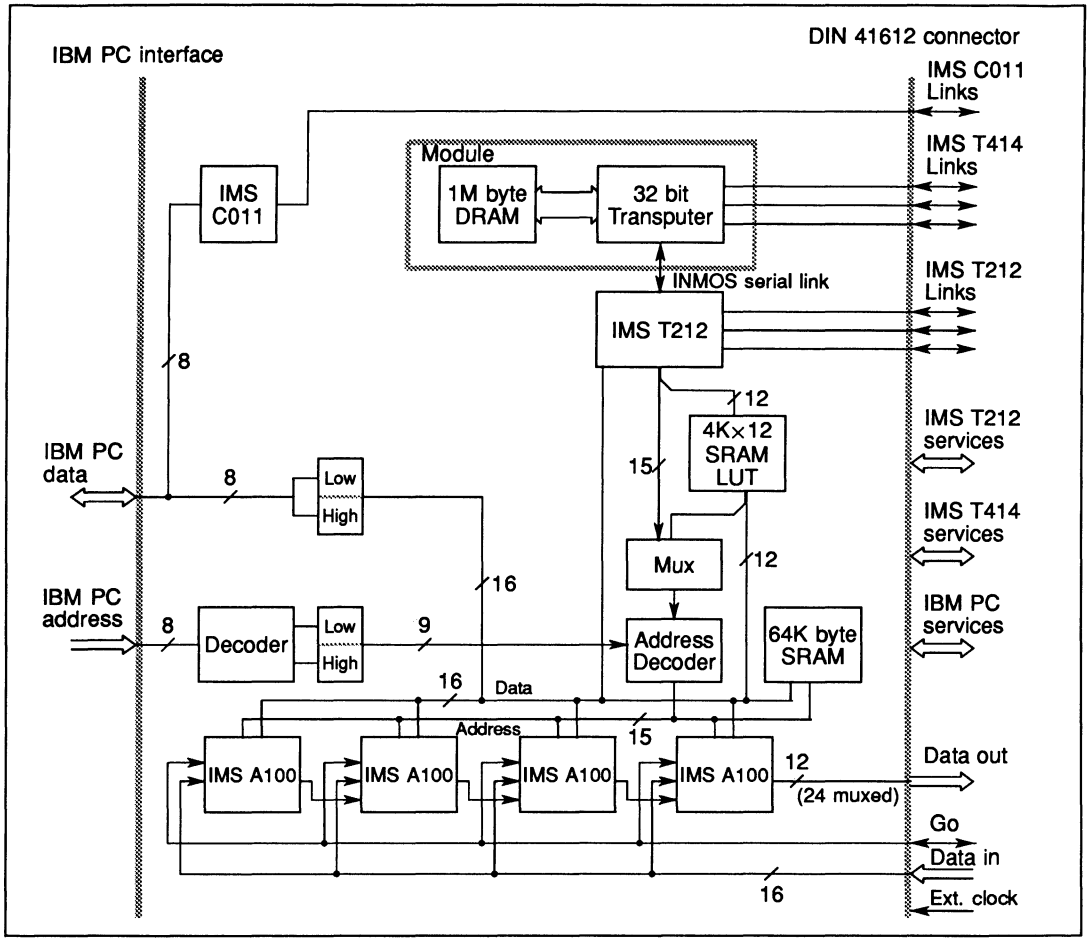


Figure 13.16 Key components of the IMS B009

13.6.3 Transputer block move capability

The transputer block move capability is used to transfer data at maximum memory bandwidth, from one position in memory to another. This following describes the mechanics of the block move, and shows how it is modified so that it may be specifically used for this application.

The technique uses hardware external to the transputer to modify memory accesses. Therefore the transputer is not in total control of what is happening during the block move. The transputer is responsible for the initialisation of the external hardware prior to execution of the block move operation. This technique, while useful for improving performance, requires caution for its use.

All transputers have dedicated hardware support for moving blocks of data from one area of memory to another. The IMS T212 doing a block move on the IMS B009 will transfer a 16 bit word from one memory location to another in 300ns, with 150ns required for the read operation and 150 ns for the write operation. One possible occam implementation is given in the code below. In this case 1024 words are transferred from

position #4000 to #5000. The compiler deals with setting up the counter and address registers.

```
[1024]INT array1:
[1024]INT array2:
PLACE array1 AT #4000 :
PLACE array1 AT #5000 :
SEQ
  ... Load array with source data
  Array1 := Array2
```

The block move capability can be used to transfer data into and out of the IMS A100 devices extremely quickly. However, the data must be in the correct order and word aligned. If processing is required to position the data correctly in memory, then the performance of the system will be impaired.

The sequence of operations required to access the IMS A100 devices from the IMS T212 is a read from memory followed by a write to the DIR [2] register, followed by a read from the DOL [3] register followed by a write to memory. There is little similarity between the simple transputer block move operation and the transfer of data into and out of the IMS A100.

The hardware required to modify the simple block move operation is an address decoder and look-up table (LUT), which are included as part of the IMS T212 memory interface. Whenever a memory address is output by the IMS T212 and the LUT is active the address is translated by the LUT. The 4Kbytes of LUT are used to convert a block of sequential addresses into an arbitrary sequence of addresses, with **no processing overhead.**

In addition to the address translation LUT the address output by the IMS T212 is decoded along with the information of a read or a write cycle. The result is used to decide if a write to the DIR register, a read from the DOL register, or a normal read/write cycle is required.

For the 2-D convolution, data is placed in the memory of the IMS T212 one pixel at a time and one line at a time. However, the pixels are loaded into the DIR register of the IMS A100's one column at a time. The address translation LUT is used to map the rows of pixel data into columns of pixel data so that the data enters the IMS A100 cascade in the correct order. In exactly the same way the output data from the IMS A100 is in columns and must be translated into rows, so that it may be redisplayed on a monitor.

The size of the LUT enables 4096 possible translations, half of which are used for the data input, and half of which are used for the data output from the IMS A100 cascade. Therefore it is only possible to have a block of data of 2048 pixels. This means that for example an image of [512 $\times$ 512] pixels must be split into, say, 128 blocks each with [16 $\times$ 32] pixels. For efficency reasons it is best to keep the blocks as close to square as possible.

The sequence of operations involved with a 2-D convolution is as follows:

1 Read from memory through the address translation LUT so that the correct pixel is accessed.

2 Write pixel to the DIR register of all the IMS A100's in the cascade.

3 Read result of the convolution from the DOL register of the last IMS A100 in the cascade.

4 Write result to memory location given by the address transtation LUT.

The result of doing this operation for every pixel in the block, is stored sequentially in memory and is then transferred across the transputer link to the IMS T414, where all the blocks are recombined into one image. The complete image is then transferred to the matrox board to show the result of the convolution.

[2] The DIR register is the Data Input Register

[3] The DOL register contains the Low byte of Output Data

13.6.4 Implementation of the 2D convolution algorithm

The convolution algorithm is implemented in occam on the IMS B009 which consists essentially of 2 processors running in parallel. The main purpose of these processors is to prepare the data for processing by the IMS A100 devices. This following description gives the details of this implementation with respect to both the hardware and software.

The IMS T414 and IMS T212

The IMS T414 module with 1 megabyte of memory is connected to the IBM PC host processor via a link adaptor. This processor runs the main program and also stores the complete image buffer and result of the convolution. Both the result of convolution and the image buffer contain 256 Kbytes of image data.

The IMS T212 handles reading and writing of data across the IMS A100/IMS T212 memory interface. All data processed by the IMS A100 passes across this interface.

The IMS T212 which is connected to the IMS T414 by a transputer link, runs several specialised procedures and has limited data space for the storage of images. Therefore, the IMS T212 will at any moment during program execution be processing only a small portion of the image. The following program runs on the IMS T414 processing individual blocks of the image one after the other. The operation is described below in pseudo-occam. Notice that the three dots at the beginning of some lines hide code within them.

```
PROC 2D.convolve(VAL [][]BYTE input.image, [][]BYTE convolved.image)
  ...  calculate block sizes, rows and columns
  ...  set up address mapper
  SEQ block.row = 0 FOR total.block.rows
    SEQ block.col = 0 FOR total.block.cols
      SEQ
        ...  Calculate new pixel coordinates of the block
        ...  Dump block of input image array to IMS T212 from IMS T414
        ...  Convert data into IMS A100 format
        ...  Flush IMS A100 cascade
        ...  Block move data through IMS A100's (DO THE REAL WORK)
        ...  Convert result into bytes
        ...  Send result from IMS T212 to IMS T414
        ...  Place result into convolved image array (final result)
:
```

The block sizes, the number of blocks in each row and the number of rows in each image are first calculated and apply for the duration of the convolution of one entire [512 × 512] pixel image. Each block is made up of a precalculated number of rows and columns of individual pixels. The maximum number of pixels in each block is 2048 pixels. This is a limitation of the address mapper which can perform a maximum of 4096 address translations, 2048 for input and 2048 for output. Without this address mapper the pixel data would need to be reordered by the transputer, which is extremely time consuming. The address mapper makes possible arbitrary address sequences, so that data in the transputers memory space may be in any arbitrary order prior to processing by the IMS A100.

The address mapper is set up once before processing the image. The code to do this is as follows:

```
SEQ
  SEQ i = 0 FOR block.size
    SEQ
      j := 2 * i
      mapper.array[j]   := i
      mapper.array[j+1] := block.size + i
  ...  write array to mapper
```

This address mapping, between the memory and the transputer, enables the use of the transputer block move engine. Under normal operation the block move engine transfers a block of data in contiguous memory to another position in memory one word at a time at maximum speed. This is considerably faster than doing individual read/write operations.

In order to use this block move facility without the address mapper the data would need to be interleaved with the result. This would lead to inelegant and inefficient software, and it is much better to have arrays in contiguous memory. The transputer is reading and writing at consecutive locations, but the external hardware is "cheating" so that the transputer is actually reading or writing at locations defined by the address mapper. The previous piece of code is used to set up this interleaved addressing.

The operation of passing data through the IMS A100's may be considered as 4 distinct actions, which use up two transputer block move cycles.

During the first block move, data is read from the address mapped memory location output by the transputer. The transputer then attemps to write this data to the block move output address. However, the external hardware recognises this cycle and intercepts it, so that data is written to the DIR register of all the IMS A100's and not to memory. During the write to DIR the address mapper is unused.

During the second block move, the transputer attempts to read from the next memory location of the input array. However the external hardware again recognises this and the data is read from the DOL register of the last IMS A100 in the cascade. The transputer then attempts to write data to the next memory location of the output array. However this is again intercepted and the data is written to the address mapped memory location output by the Transputer.

The net effect is that the image block residing in IMS T212 memory is passed through the IMS A100 DIR and DOL registers and the convolution result read from the DOL register now resides in contiguous memory ready to be transferred back across a transputer link to the IMS T414.

Performance

The performance of this setup is easily calculated as the sum of the time for two transputer block moves (2 x 300ns for IMS T212 with 1 wait state) plus the time for a single cycle of the IMS A100's (200 ns for 8 bit coefficients). This gives a total pixel transformation time of 800 ns.

To obtain the time required for the convolution of a complete image it is only necessary to calculate the number of blocks of pixels comprising a complete image. As will later be shown the number of blocks is not just a function of image size, but depends on the size of the convolution kernel, and the number of pixels in each block.

The best possible performance assumes that 256K pixels each require 800 ns of processing. This corresponds to a processing rate of 5 frames per second. In other words the performance degredation caused by using a memory interface, as opposed to using the dedicated cascade and data ports, to get the data into and out of the IMS A100 yields 25% of the available performance of the IMS A100. As will now be explained, the actual frame processing rate will be somewhat less than this, because the blocks of image data comprising the complete image must be overlapped.

Image segmentation

The image is segmented into several blocks of pixels. This is illustrated in figure 13.17. In this example each block overlaps in both the x and y directions dependent on the size of the convolution kernel. This image convolution kernel with size [5 x 5] requires an overlap of 4 pixels on each edge. If this is not done the result of the convolution of the image will have vertical and horizontal lines of incorrectly convolved data running down and across.

Overlapping of image blocks means that pixels at the edge of a block will be passed through the IMS A100 twice. This does not matter except that the time to process an entire image will be increased. Also pixels at the edge of the image must be ignored. In this example the 2 outermost pixels at the edge of an entire frame do not contain useful information.

To process a single block will require the processing of 144 pixels, only 64 of which will be valid. This represents 44% efficiency for processing the entire image. The larger the kernel or the smaller is each individual processing block, the less efficient is this technique of image segmentation. Unfortunately this technique is the best that can be done using the IMS B009, with data transfer across the relatively slow memory interface.

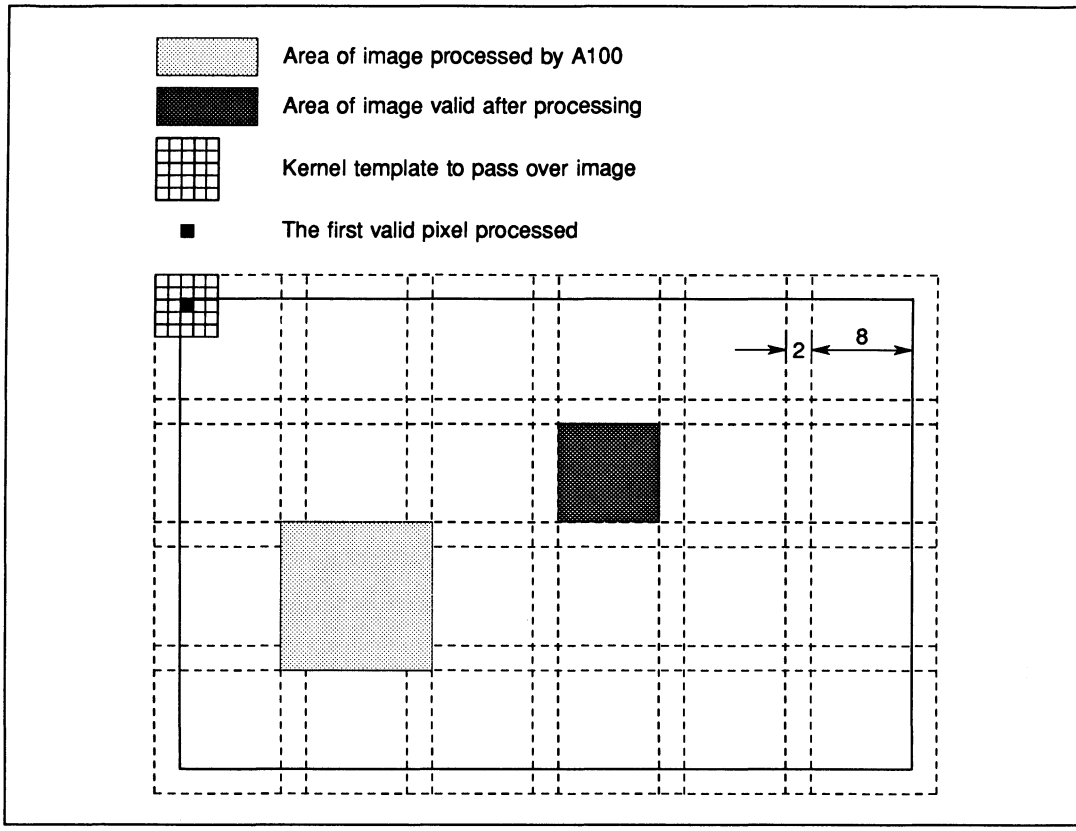


Figure 13.17 Image segmentation

Thresholding and scaling using software LUT

A software LUT is used to do scaling of the data output from the DOL register of the IMS A100. Thresholding has not been done though it is simple to add.

The data from the camera consists of [512 × 512] pixels each with 8 bit grey scale. Each pixel is operated upon by the convolution kernel, which may have negative components. This operation is done inside the IMS A100 and the result of the convolution for each pixel may be either negative or positive. Also, because the data is only 8 bits, the limits are −128 to +127. As these numbers are inappropriate for output to the monitor, scaling is applied to convert into a grey scale value between 0 and 255. However, if the values output by the IMS A100 are known to be positive, because all the kernel elements are positive, the output does not require scaling. The program enables the optional use of a predefined software LUT in order to create images with the maximum dynamic range.

The LUT facility can be used for other techniques such as non-linear scaling and thresholding. A side effect of the thresholding operation is further deterioration of performance. Each pixel must be read from memory, transformed through the LUT and written back to memory. This takes 800 ms (40-50 cycles) for the IMS T414, with all the data in off-chip memory. The operation in occam is

```
pixel[i][j] := table[pixel[i][j]]
```

Transfer of image across links

Data transfer across links involves the transfer of 512 Kbytes of data which will take approximately 500 ms using a single 20 Mbit/sec link. Also, because of the image segmentation method described earlier, the perimeters of each image block will in effect be transferred twice. This inefficiency is worse for large kernel sizes and small block sizes.

In the examples used in this application note the time taken for data transfer lies between 500 ms and 1000 ms.

13.6.5 The Demonstration Program

The following information is only relevant to users of the IMS D703 [5] software, which is available from the DSP group, based at Bristol. Readers not using this software can ignore the following.

The demonstration program can be used to execute several functions including the convolution of a single image. Images may be grabbed from the frame grabber board and processed, and the result may be displayed. Also the original images and convolution results may be stored on disc, although a lot of disc space is required at 256 Kbytes per image. An image can also be read from disc instead of from the frame grabber which is useful if images need to be processed several times. Storing away this amount of data is however quite slow. An optional post processing program is also available which transforms these grey scale images on disc into postscript format. The pictures in this document are created in this manner.

IMS D703 Development software

The program operates a modified version of the IMS D703B development software. For more information on this please consult the IMS D703 user guide and reference manual. Briefly the differences are as follows:

- The routines for accessing the matrox board are included in the software but not with the standard IMS D703.
- Routines which enable byte wide data to be transferred across the links between the IMS T212 and IMS T414 have been enabled. With the IMS D703 all data is transferred as 16 bit words both to and from the IMS T212. The 16 bit word format is directly suitable for the IMS A100's assuming the use of 16 bit coefficients. The IMS A100's process data in 400 ns in this mode.
- Because only 8 bit data is used in this application (8 bit grey scale) it would be very wasteful to transfer 16 bit words as half the data would be redundant. However, the 8 bit data must be transformed into 16 bit data with the top 8 bits held low (zero). This is because the IMS A100 does require 16 bit data even when in 8 bit coefficient mode.
- The 5 applications in the IMS D703 system have been removed, and replaced by the single application. It would be possible to add this application to the original 5 but because this application uses a lot of memory, (512 Kbytes for the image buffers alone) it was found that the 6 applications overflowed the 1 Mbyte of memory on the IMS B009-2. This may be aleviated in the future when modules with 2 mega bytes or more are available.

Injection of noise onto images

The program has a facility to add random noise onto an image. The reason for doing this is to show the benifit of large convolution kernel sizes.

Qualitatively, the effect of a large kernel is to locally average pixels surrounding a pixel point. This results in a blurring of the image but does give the effect of reducing noise. This is the analogue equivalent of a low pass filter.

Convolution kernel file

A convolution kernel file is read by the program each time it is executed. This means that many convolution kernels may be investigated without the need for recompilation of the program. An example format of this file is given below. It is only necessary to edit this file to investigate any number of convolution kernels.

The file must start with the number of convolution **KERNELS**. When the program runs, 4 kernels in this example will be sequentially executed, and the resultant image for each is displayed on the monitor. Each **KERNEL** has an optional description which is shown simultaneously while the convolution is being done. The **SIZE** of each kernel must also be given. This is used to check for correctly entered kernel elements. The **SCALE** is multiplied by each element of the kernel to give the resultant convolution kernel. The **SIGN** is used to determine the software LUT which operates on the output of the IMS A100. There are 2 possible LUTs available for the existing program, '+' which assumes that all elements in the array are positive and '–' which accepts either positive or negative integers. These LUTs are used because the IMS A100 is a 2's complement integer machine, and it is therefore necessary to know if the output from the IMS A100 requires conversion or may be assumed positive.

It is possible to add other software LUTs, for example to do nonlinear scaling and saturation control. As different convolution kernels may require different output conversions through the look-up table this attribute is made individual to each kernel.

```
KERNELS 4

KERNEL Simple filter
SIZE 3 SCALE 28 SIGN +
  ROW  1 1 1
  ROW  1 1 1
  ROW  1 1 1
FINISH

KERNEL Simple filter
SIZE 9 SCALE 3 SIGN +
  ROW  1 1 1 1 1 1 1 1 1
  ROW  1 1 1 1 1 1 1 1 1
  ROW  1 1 1 1 1 1 1 1 1
  ROW  1 1 1 1 1 1 1 1 1
  ROW  1 1 1 1 1 1 1 1 1
  ROW  1 1 1 1 1 1 1 1 1
  ROW  1 1 1 1 1 1 1 1 1
  ROW  1 1 1 1 1 1 1 1 1
  ROW  1 1 1 1 1 1 1 1 1
FINISH

KERNEL Sobel Operator Edge Detection
SIZE 3 SCALE 32 SIGN -
  ROW  -2 -1  0
  ROW  -1  0  1
  ROW   0  1  2
FINISH

KERNEL Sobel Operator Edge Detection
SIZE 9 SCALE 1 SIGN -
  ROW   -7 -7 -7  -4  -3  -4  0  0  0
  ROW   -7 -7 -7  -3  -4  -3  0  0  0
  ROW   -7 -7 -7  -4  -3  -4  0  0  0
  ROW   -4 -3 -4   0   0   0  4  3  4
  ROW   -3 -4 -3   0   0   0  3  4  3
  ROW   -4 -3 -4   0   0   0  4  3  4
  ROW    0  0  0   4   3   4  7  7  7
  ROW    0  0  0   3   4   3  7  7  7
  ROW    0  0  0   4   3   4  7  7  7
FINISH

FINISH kernel file
```

13.7 References

1 *Digital filtering with the IMS A100*, Application note 1, Hossein Yassaie, INMOS Limited, Bristol.

2 *Discrete Fourier transform with the IMS A100*, Application note 2, Hossein Yassaie, INMOS Limited, Bristol.

3 *Correlation and convolution with the IMS A100*, Application note 3, Hossein Yassaie, INMOS Limited, Bristol.

4 *Complex (I & Q) processing with the IMS A100*, Application note 4, Hossein Yassaie, INMOS Limited, Bristol.

5 *IMS D703 reference manual*, INMOS Limited, Bristol.

6 *IMS A100 data sheet*, INMOS Limited, Bristol.

7 *Index mappings for multidimensional formulation of the DFT and convolution*, Burrus C.S., IEEE Trans. on ASSP, 25:239–242, June 1977.

8 *DFT/FFT and convolution algorithms – theory and implementation*, Burrus C.S. and Parks T.W., IEEE Trans. on ASSP, 25:239-242, June 1977.

cascading IMS A110s

14.1 Introduction

The IMS A110 is a single-chip programmable and cascadable device suitable for many high speed image and signal processing applications. It consists of a configurable array of multiply-accumulators (420 MOPs), three programmable length 1120 stage shift registers, a versatile post-processing unit and a microprocessor interface for configuration and control purposes. The comprehensive on-chip facilities makes a single device capable of dealing with many image processing operations. A simplified block diagram is shown in figure 14.1.

For some applications however, the power and versatility of a single IMS A110 is not sufficient, in these cases a cascade of devices often provides a solution. The purpose of this document is to describe some of the most useful ways to cascade IMS A110s to achieve even higher performance and as such does not cover the use of the backend processor or device applications.

14.2 Operation of a single IMS A110

The A110 may be set up as either a one or two dimensional multiplier accumulator array (MAC).

14.2.1 One dimensional operation of an IMS A110

For one dimensional operation the first delay PSRc is set to some arbitrary value (normally zero) while PSRb and PSRa are set to zero. N.B. at any given point in time the first MAC stage in bank c is processing the oldest data while the last MAC stage of bank a is processing the newest data.

14.2.2 Two dimensional operation of an IMS A110

For two dimensional operation the first delay (PSRc) is again set to some arbitrary value; however, the setting of PSRa and PSRb is dependant on the line length in pixels of the image being processed. It turns out that in order to achieve a rectangular convolution window the number of delays to be programmed into PSRa and PSRb is equal to the line length in pixels plus the length of the MAC pipelines (seven stages). For example if the screen width of the image to be processed is 512 pixels then the delay to be programmed into shift registers PSRa and PSRb is 519.

N.B. normally when processing an image with an arbitrary setting of PSRc the delay (latency) through the IMS A110 causes the output image to be incorrectly aligned or skewed. This results in an apparent rotation of the output image in the horizontal plane. To correct this problem PSRc may be adjusted to introduce a suitable number of delays to shift the image into the correct position.

Typically image data is fed into an IMS A110 line by line starting at the top left and ending at the bottom right. Given this definition it may be seen that the first MAC stage in each row is processing the data nearest the left hand side of the screen (the oldest data) and that the last MAC stage in each row is processing the data nearest the right hand side of the screen (the newest data). In a similar fashion the first row is always processing the newest data (the data nearest the bottom of the screen) and the last row is always processing the oldest data (the data nearest the top of the screen). It is important to bear in mind these relationships when programming IMS A110s, otherwise the operation being performed on an image may not be what was expected.

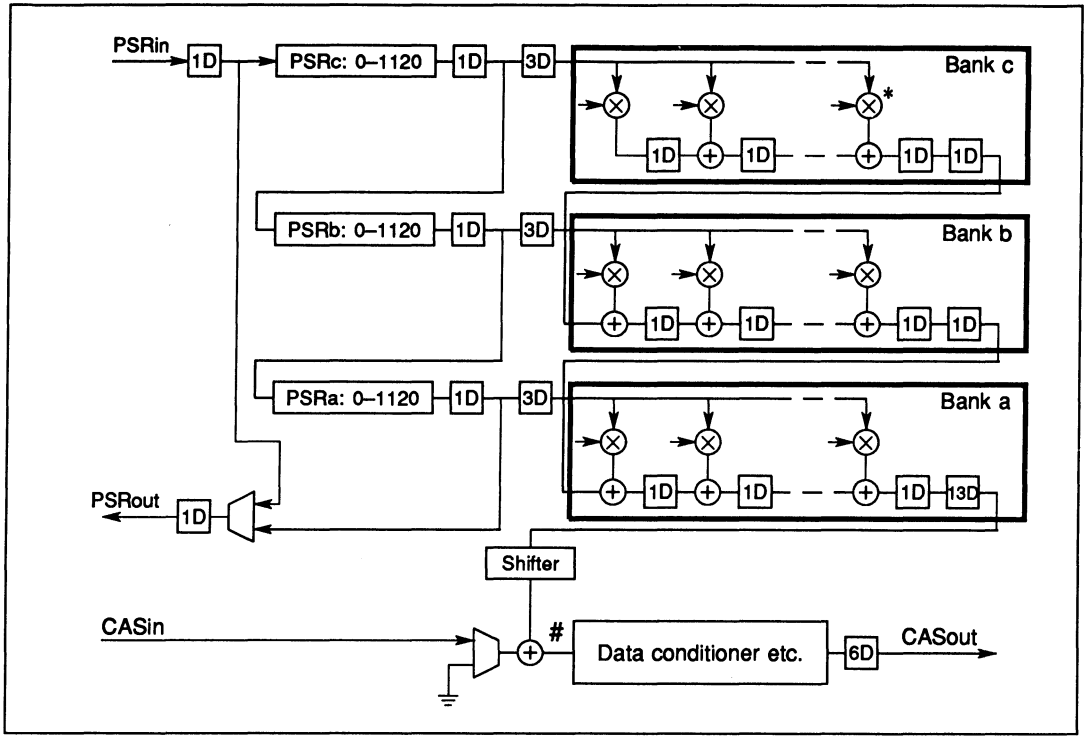


Figure 14.1 Block diagram of the IMS A110

14.3 Fundamentals of cascading IMS A110s

Consider a single IMS A110 configured to perform some task on a stream of data values. The filter kernel formed by the coefficients may be thought of as a block passing over the data. To produce bigger filters it is necessary to join a number of separate blocks together. This may be achieved by connecting together a number of IMS A110s, as shown in figure 14.2, and configuring them suitably. In order to create a contiguous filter kernel (i.e. a filter without overlap or gaps) it is essential that the route between PSRin and PSRout for each device is programmed correctly and that the internal delay lines are programmed to the correct lengths.

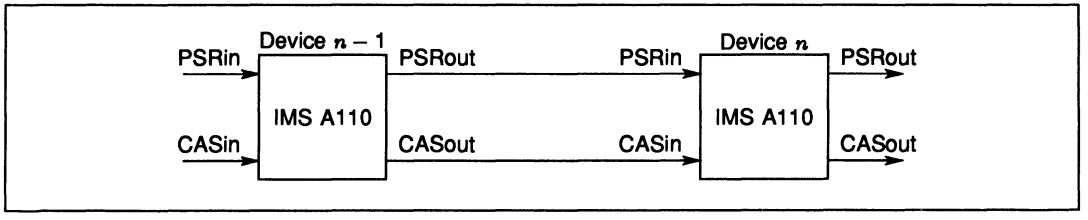


Figure 14.2 Standard connection for cascading IMS A110s

To assist in the calculation of the delays to be programmed into the programmable shift registers it is convenient to define a reference data path through the MAC of any given IMS A110. In this document, unless specified, the reference path is taken to be from the input to the multiplier marked with an asterisk (*) in figure 14.1 to the cascade adder marked with a hash (#).

In addition before embarking on any calculation it is necessary to know the following:

1 The delay between PSRin and PSRout when the data is routed directly from PSRin to PSRout without passing through the programmable shift registers. This delay is known as D_D.

2 The delay along the reference path. This delay is known as D_R.

3 The delay through the backend between cascade in and cascade out. This delay is known as D_B.

4 The locations of the other inherent delays within IMS A110s.

5 The meaning of line length, kernel width and kernel height. See figure 14.3 for a definition of these terms.

Figure 14.1 shows a functional block diagram of an IMS A110 with all the inherent delays included. From this diagram it is possible to calculate the value of the three delay constants as shown in table 14.1.

$D_D = 1 + 1$ $D_D = 2$	$D_R = (1 + 1) + (7 + 1) + (7 + 13)$ $D_R = 30$	$D_B = 6$

Table 14.1

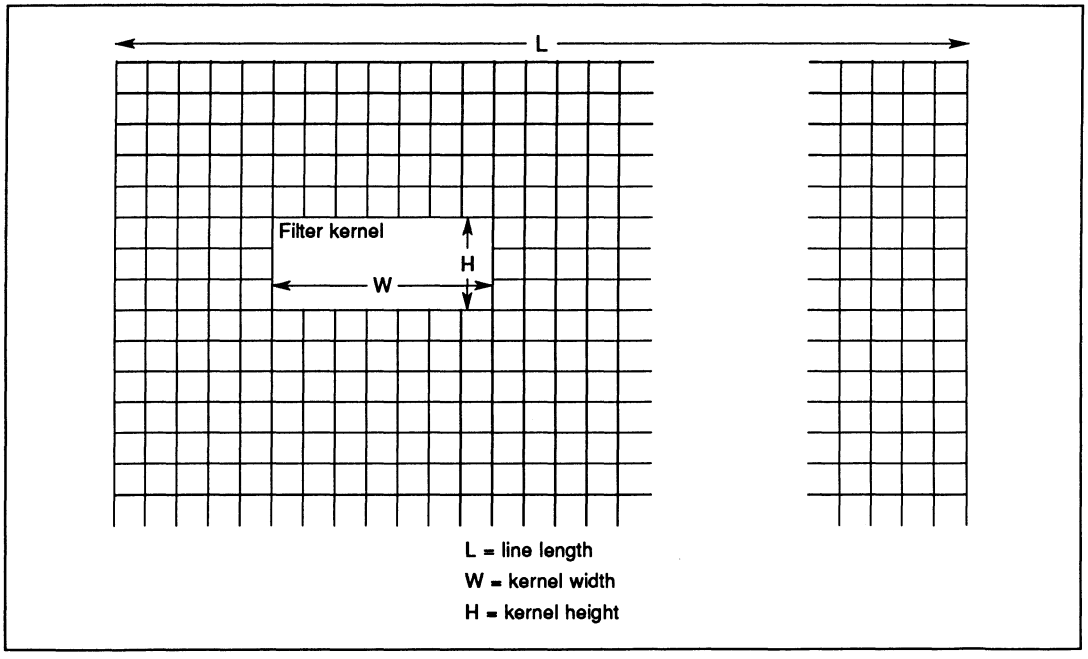


Figure 14.3 Depiction of line length, kernel width and kernel height

14.4 Cascading IMS A110s to produce long one dimensional filters

A single IMS A110 is capable of producing a one dimensional filter with up to 21 taps (shorter filters may be made by setting unrequired coefficients to zero). To create longer filters it is necessary to cascade a number of IMS A110s together. Each additional device added to the cascade gives an additional 21 taps allowing filters of almost unlimited size to be built from simple building blocks.

To develop the delays required to be set up in a one dimensional cascade the system shown in figure 14.4 will be considered. This system only contains two devices but will be examined in a general way so that rules may be developed for cascades of arbitrary length. It has already been mentioned how to set up the delays to achieve one dimensional convolution in a single device. Fortunately, in cascades of IMS A110s the data relationships within each device are the same as those which would exist inside a single non cascaded device processing the same data. Hence, in the one dimensional cascade under consideration the delays programmed into PSRa and PSRb of each device are zero.

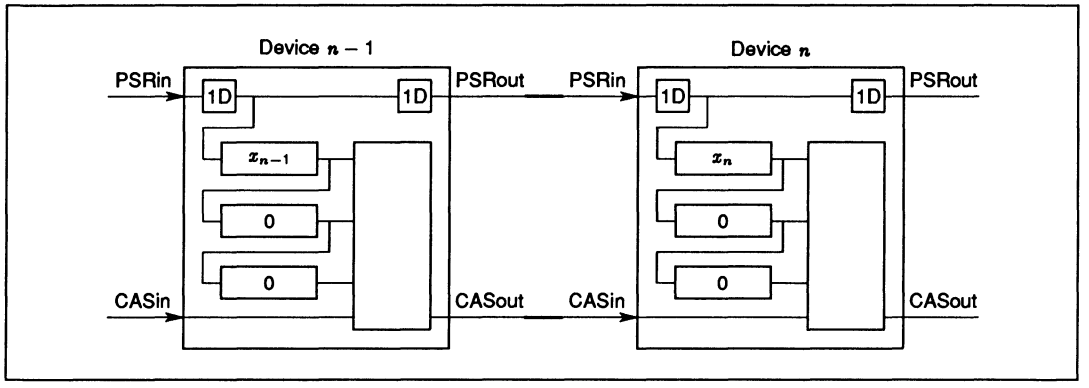


Figure 14.4 Direct data path connection for cascading IMS A110s

In order to cascade IMS A110s into long one dimensional filters the data is normally routed directly from the input to the output of each device without passing through the programmable shift registers, as shown in figure 14.4. It may be seen that each piece of data takes two routes through the cascade. One route generates partial results via the MAC of device $n-1$ and the other via the MAC of device n. These partial results are eventually combined at the cascade adder in the backend of device n. To produce the correct result it is important that these two separate data streams are aligned correctly.

Assuming that the delay in the PSRc of device $n-1$ is x_{n-1} and that the delay in PSRc of device n is x_n, it is desired to calculate the relationship between these delays for correct combination of the partial results. Consider an item of data when it reaches device $n-1$. The delay before the component due to this data, flowing via the reference path in device $n-1$, reaches the cascade adder of device n is:

$$D_{n-1} = 1 + x_{n-1} + 1 + 3 + D_R + D_B$$

$$D_{n-1} = 41 + x_{n-1}$$

Similarly the delay before the component due to this data, flowing via the reference path in device n, reaches the cascade adder of device n is:

$$D_n = D_D + 1 + x_n + 1 + 3 + D_R$$

$$D_n = 37 + x_n$$

Now, for a contiguous convolution kernel, it is desired for the results flowing via the MAC of device $n-1$ to arrive at the cascade adder of device n, 21 clock cycles behind those which have come from the other route. Hence:

$$D_{n-1} - D_n = 21$$

$$41 + x_{n-1} - 37 - x_n = 21$$

$$x_{n-1} = x_n + 17$$

This means that the PSRc of device $n-1$ must be programmed with the value which is in PSRc of device n plus a fixed constant of 17. This rule may be extended to take into account any number of devices providing that the maximum length of the delay lines is not exceeded. The PSRc of the last device in the cascade may be programmed to an arbitrary value (normally zero) providing the maximum length of the first PSRc delay in the cascade is not exceeded.

For example consider the problem of filtering a data stream with a 50 tap filter. This could be achieved by cascading three IMS A110s. Typical delays which would have to be programmed into the devices are given in table 14.2.

	Device 1	Device 2	Device 3
PSRa	0	0	0
PSRb	0	0	0
PSRc	34	17	0

Table 14.2

14.5 Cascading IMS A110s to produce wider two dimensional filters

A single IMS A110 is capable of filtering an image with a two dimensional kernel which has a maximum width of seven cells (narrower filters may be made by setting unrequired coefficients to zero). To create wider filters it is necessary to cascade a number of IMS A110s together. Each additional device added to the cascade increases the maximum width by an additional 7 cells, allowing filters of almost unlimited width to be created.

The connections required to cascade IMS A110s into horizontal cascades may be seen in figure 14.4. It may be noted that the connections for this type of cascade are identical to those presented in section 14.4 for one dimensional cascading. The difference in function is achieved by changing the delays present in the programmable shift registers. It was mentioned in section 14.2 that for two dimensional filtering using a single device the length of PSRa and PSRb have to be programmed to the line length plus seven. Hence to ensure correct alignment of the rows of the filter in a horizontal cascade it is necessary that PSRa and PSRb of each of the devices must also be set to this value.

In order to cascade horizontally the pixel data is normally routed directly from the input to the output of each device without passing through the programmable shift registers. As before it may be seen that each item of data (pixel) takes two routes through the cascade. By assuming that the delay in the PSRc of device $n-1$ is x_{n-1} and that the delay in PSRc of device n is x_n, then the route delay equations derived are the same as those calculated in section 14.4.

$$D_{n-1} = 41 + x_{n-1}$$

$$D_n = 37 + x_n$$

Now, for a contiguous convolution kernel, it is desired for results flowing via the MAC of device $n-1$ to arrive, at the cascade adder of device n, 7 clock cycles behind those which have come from the other route. This may be achieved by ensuring that the data passing via MAC $n-1$ takes 7 cycles longer than data passing via the MAC n route. Hence:

$$D_{n-1} - D_n = 7$$

$$41 + x_{n-1} - 37 - x_n = 7$$

$$x_{n-1} = x_n + 3$$

This means that the PSRc of device $n-1$ must be programmed with the value which is in PSRc of device n plus a fixed constant of 3. This rule may be extended to cascade any number of devices providing that the maximum length of the delay lines is not exceeded. The value programmed into the PSRc of the last device in the cascade is arbitrary (normally adjusted to deskew the output image) but must not be set so high that the PSRc of the first device in the cascade exceeds its maximum.

For example consider the problem of filtering a 1024 pixel wide image with a 15×3 filter kernel. This could be achieved by cascading three IMS A110s into a horizontal cascade. Typical delays which would have to be programmed into the devices are given in table 14.3.

	Device 1	Device 2	Device 3
PSRa	1031	1031	1031
PSRb	1031	1031	1031
PSRc	6	3	0

Table 14.3

14.6 Cascading IMS A110s to produce higher two dimensional filters

The maximum height of a two dimensional filter kernel produced by a single IMS A110 is three cells. This is restricting in some applications but, may be easily overcome by cascading a number of IMS A110s into a single vertical strip. The theoretical maximum height of filter which can be created is equal to three times the number of devices cascaded. Hence the vertical filter size is limited only by the number of devices used.

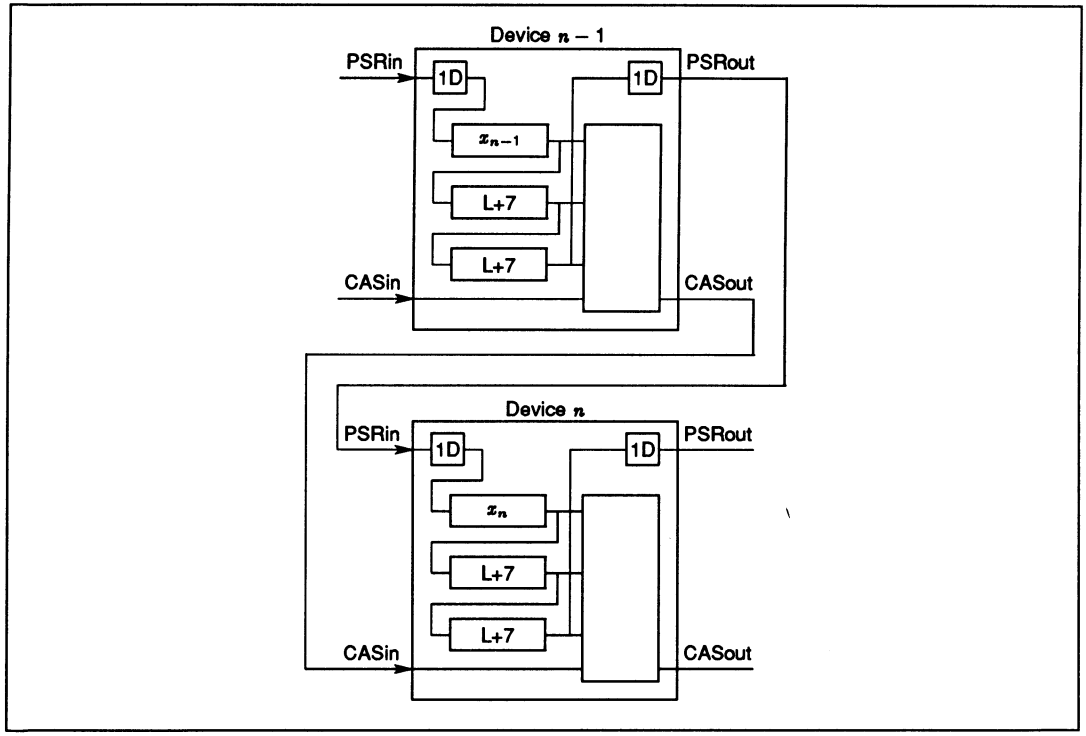


Figure 14.5 Indirect data path connection for cascading IMS A110s

To develop the delays required to be setup in a vertical cascade the system shown in figure 14.5 will be considered. This system only contains two devices but will be examined in a general way so that rules may be developed for cascades of arbitrary length. It was mentioned in section 14.2 that for two dimensional filtering using a single device the length of PSRa and PSRb have to be programmed to the line length plus seven (L+7). Obviously to ensure correct alignment of the rows of the filter in a vertical cascade it is necessary that PSRa and PSRb of each of the devices must also be set to this value.

To cascade vertically the pixel data is normally routed from the input to the output of each device via the programmable shift registers (see figure 14.5). Again it may be seen that each pixel takes two routes through the cascade. One route generates partial results via the MAC of device $n-1$ and the other via the MAC of device n.

These partial results are eventually combined at the cascade adder in the backend of device n. In order to produce the correct result it is important that these two data streams are aligned correctly.

Assuming that the delay in the PSRc of device $n-1$ is x_{n-1} and that the delay in PSRc of device n is x_n, it is desired to calculate the the relationship between these delays for correct combination of the partial results. Consider a pixel when it reaches device $n-1$. The delay before the component due to this pixel, flowing via the reference path in device $n-1$, reaches the cascade adder of device n is:

$$D_{n-1} = 1 + x_{n-1} + 1 + 3 + D_R + D_B$$

$$D_{n-1} = 41 + x_{n-1}$$

Similarly the delay before the component due to this pixel, flowing via the reference path in device n, reaches the cascade adder of device n is:

$$D_n = 1 + (x_{n-1} + 1) + (L + 7 + 1) + (L + 7 + 1) + 1 + 1 + (x_n + 1) + 3 + D_R$$

$$D_n = 54 + 2L + x_{n-1} + x_n$$

But for a contiguous convolution kernel it is desired for results flowing via MAC n to arrive, at the cascade adder of device n, three line lengths after those which have come from the other route. This may be achieved by ensuring that the data passing via MAC n takes 3L (where L is the line length in pixels) cycles longer than data passing via the MAC $n-1$ route. Hence:

$$D_n - D_{n-1} = 3L$$

$$54 + 2L + x_{n-1} + x_n - 41 - x_{n-1} = 3L$$

$$x_n = L - 13$$

This means that the PSRc of device n must be programmed with a value which is equal to the line length minus a fixed constant of 13. This rule may be extended to cascades containing any number of devices providing that the maximum length of the delay lines is not exceeded. N.B. the setting of the PSRc of the first device in the cascade is arbitrary and may be adjusted to deskew the output image.

For example consider the problem of filtering a 512 pixel wide image with a 7×7 filter kernel. This could be achieved by cascading 3 IMS A110s into a vertical cascade. Typical delays which would have to be programmed into the devices are given in table 14.4.

	Device 1	Device 2	Device 3
PSRa	519	519	519
PSRb	519	519	519
PSRc	0	499	499

Table 14.4

14.7 Cascading IMS A110s to produce wider and higher two dimensional filters

To produce filters which are both wider and higher than allowed by a single IMS A110 it is possible to cascade a number of the wider filters discussed in section 14.5 into a vertical strip.

The connections required to cascade IMS A110s into two dimensional cascades may be seen in figure 14.6. The system shown has arbitrary width but only two rows of devices allowing a maximum filter height of six cells. However the system will be examined in a general way so that rules may be developed for cascades of arbitrary height. It may be noted that across each row, except for the last device, direct connection is used between PSRin and PSRout. The last device uses the indirect route via the programmable shift registers to connect to the first device of the next row. Since each row of this cascade consists of a horizontal cascade the rules developed for the delays in such a cascade (see section 14.5) apply to each row of this larger configuration. However, the relationship between the delays in the vertical direction requires careful consideration.

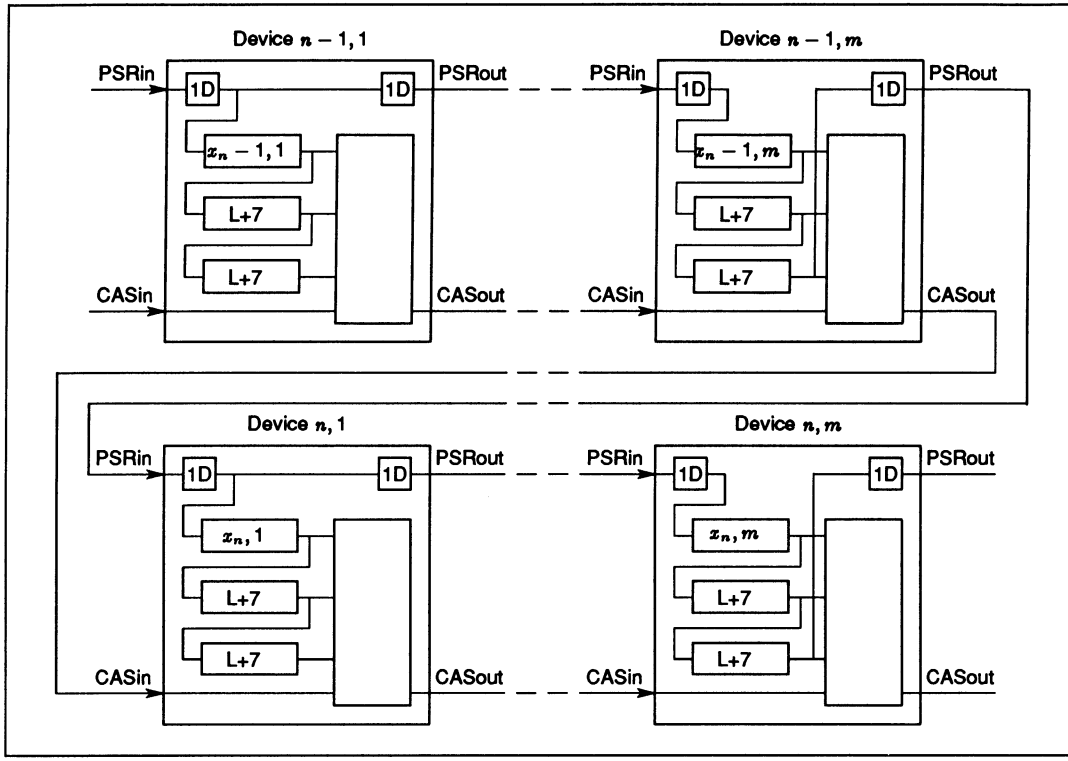


Figure 14.6 Connections for cascading IMS A110s into wider and higher 2-D filters

Assuming that the array of IMS A110s contains M devices in the horizontal direction and that the delay in PSRc of each device is as shown in figure 14.6, it is desired to calculate the relationship between these delays for correct combination of the partial results generated by each row within the cascade. Consider a pixel when it reaches device $n-1,1$. The delay before the first component due to this pixel, flowing via the reference path in device $n-1,1$, reaches the cascade adder of device $n,1$ is:

$$D_{n-1,1} = 1 + x_{n-1,1} + 1 + 3 + D_R + D_B M$$

$$D_{n-1,1} = 35 + 6M + x_{n-1,1}$$

Similarly the delay before the component due to this pixel, flowing via the reference path in device $n,1$, reaches the cascade adder of device $n,1$ is:

$$D_{n,1} = 2(M-1) + 1 + (x_{n-1,M} + 1) + (L + 7 + 1) + (L + 7 + 1) + 1 + 1 + (x_{n,1} + 1) + 3 + D_R$$

$$D_{n,1} = 52 + 2M + 2L + x_{n-1,M} + x_{n,1}$$

But for a contiguous convolution kernel it is desired for results flowing via MAC $n,1$ to arrive, at the cascade adder of device $n,1$ a period of 3L clock cycles after those which have come from the other route. Hence:

$$D_{n,1} - D_{n-1,1} = 3L$$

$$52 + 2M + 2L + x_{n-1,M} + x_{n,1} - 35 - 6M - x_{n-1,1} = 3L$$

$$17 - L - 4M + x_{n-1,M} + x_{n,1} = x_{n-1,1}$$

Now it is also known from section 14.5 that any given device in a row except the final device has PSRc programmed to 3 more than the device which follows. This leads to the following relationship between the delays programmed into the first and the last devices of the top row:

$$x_{n-1,1} = x_{n-1,M} + 3(M-1)$$

By substituting this result into the previous result gives:

$$x_{n,1} = 7M + L - 20$$

This means that the PSRc of device $n,1$ must be programmed with the value which is equal to 7 times the number of devices cascaded horizontally plus the line length minus a fixed constant of 20. This rule may be extended to cascades containing any number of devices providing that the maximum length of the delay lines is not exceeded. N.B. the setting of PSRc of the right most device in the first row is arbitrary, but is normally adjusted to deskew the output image.

For example consider the problem of filtering a 512 pixel wide image with a 9×9 filter kernel. This could be achieved by cascading six IMS A110s into a cascade containing three rows of two devices. Typical delays which would have to be programmed into the devices are given in table 14.5.

	Device 1,1	Device 1,2	Device 2,1	Device 2,2	Device 3,1	Device 3,2
PSRa	519	519	519	519	519	519
PSRb	519	519	519	519	519	519
PSRc	3	0	506	503	506	503

Table 14.5

14.8 Cascading IMS A110s to perform multi pass filtering operations

In addition to being able to cascade IMS A110s for increased filter size it is also possible to cascade devices to perform multi pass filtering operations. For example consider the problem of edge detection in a noisy image. This task is often performed in two stages the first is low pass filtering to reduce the amount of noise and the second is the edge detection operation. This complete task may be performed by cascading two IMS A110s as shown in figure 14.7. Note that only an eight bit window of CASout from the first device is connected to PSRin of the second device.

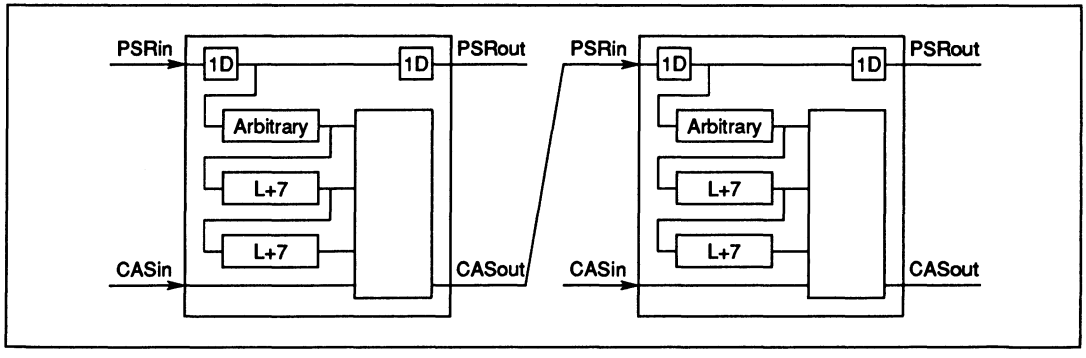


Figure 14.7 Cascading IMS A110s for multi-pass filtering

To configure such a cascade to perform the double filtering operation each device is considered separately and the delays are setup as described in section 14.2. For the example under consideration the coefficients of the first device are configured to perform the low pass filter operation while the coefficients of the second device are configured as an edge detector.

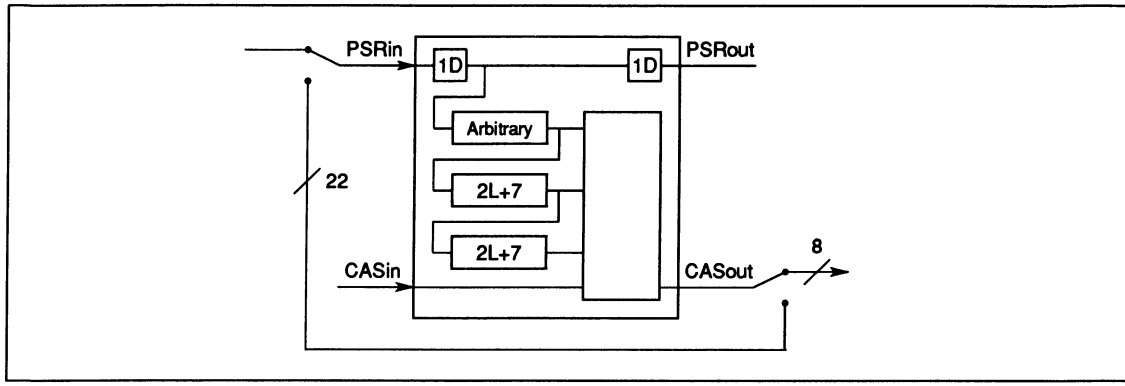


Figure 14.8 Multi-pass filtering by using feedback

This technique of multi pass filtering can obviously be extended to include more devices or it may be combined with the cascading techniques discussed in earlier sections to allow multi pass filtering with larger filter sizes.

It is possible to use a single device for multi pass filtering. This technique works by feeding back alternate cascade outputs to PSRin, and making use of bank swapping. Figure 14.8 shows the basic setup. The disadvantages of this method are:

1 The maximum data throughput is halved.

2 The maximum filter size is reduced.

3 External logic is required.

To setup such a system requires careful programming to achieve the desired result. For example consider the problem of passing the local averaging filter kernel shown below over an image twice.

1	1	1
1	1	1
1	1	1

It may be shown using similar techniques to those presented earlier that the delays to be programmed into the programmable shift registers a and b are:

$$2L + 7$$

This value is equal to twice the line length plus the length of the MAC pipelines. N.B. logical reasoning would have lead to the same result by considering that the data rate within the device is equal to twice the rate of the applied image data.

To create the correct filter kernels it is very important that the coefficient registers are programmed correctly. Each filter is programmed into one of the two coefficient banks, and every odd coefficient must be set to zero otherwise the two interleaved data streams will corrupt each other. The table below shows how the coefficients should be programmed for the example under consideration.

CR0	a	1	0	1	0	1	0	0
	b	1	0	1	0	1	0	0
	c	1	0	1	0	1	0	0
CR1	a	1	0	1	0	1	0	0
	b	1	0	1	0	1	0	0
	c	1	0	1	0	1	0	0

14.9 Cascading IMS A110s for increased data precision

In some high precision applications the 8 bit word length of a single IMS A110 is not sufficient. This section presents three techniques to overcome this problem. The first two combine IMS A110s with simple external hardware, the last one requires no external hardware but does place certain restrictions on the coefficients and the data.

14.9.1 Increasing data precision with an external 22 bit adder

The first technique makes use of an external 22 bit adder in the configuration shown in figure 14.9.

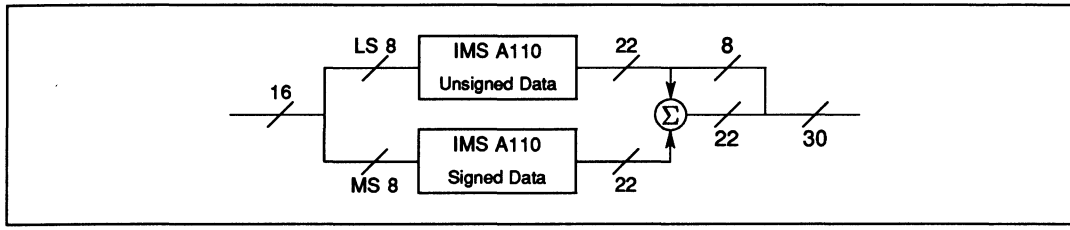


Figure 14.9 Cascade of IMS A110s for increased data precision

At the input each 16 bit input value is split into two 8 bit words one containing the least significant 8 bits and the other containing the most significant 8 bits. Each of these 8 bit data streams is fed into an IMS A110. If the data is unsigned then both of the devices must be set to unsigned data operation. However, if the data is signed then in order to correctly process the data and preserve the sign information it is necessary for the least significant byte to be processed as unsigned data and the most significant byte to be processed as signed data (see figure 14.9). This may be easily achieved by setting or clearing bit 2 of the SCR register in each IMS A110 as appropriate. The 22 bit partial results from each device are combined by making use of a 22 bit adder. This adder forms the sum of the top 14 bits of the least significant partial result and the full 22 bits of the most significant partial result to give the upper 22 bits of the final result. This is combined with the lower 8 bits of the least significant partial result to give the complete 30 bit result. See figure 14.10 for a graphical representation of this.

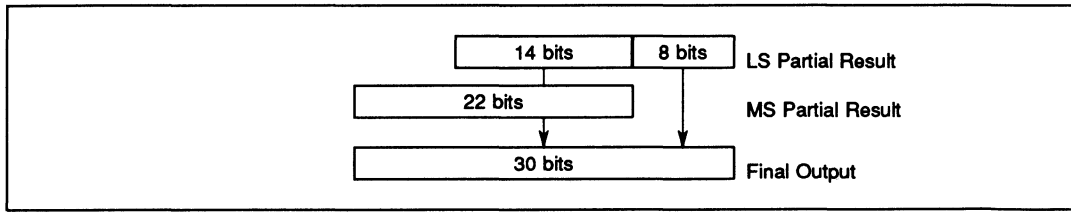


Figure 14.10 Calculation of the final output

This technique may be extended to give data precisions above 16 bits, however, such precisions are rarely used in practice. Sometimes it may be desired to combine a bigger filter size, as discussed in earlier sections, with increased precision. Such a system is simple to create and just involves replacing each IMS A110 in figure 14.9 with the appropriate cascade of devices. Similarly multi pass filtering, as discussed in section 14.8, may be combined with increased precision. This is achieved by selecting a 16 bit window from the output of the system shown in figure 14.9 and feeding this into the input of another high precision stage.

14.9.2 Increasing data precision with an external delay line

As an alternative to using an external adder it is possible to make use of the cascade adder built into each IMS A110 and an external delay line (of length D_B) as shown in figure 14.11.

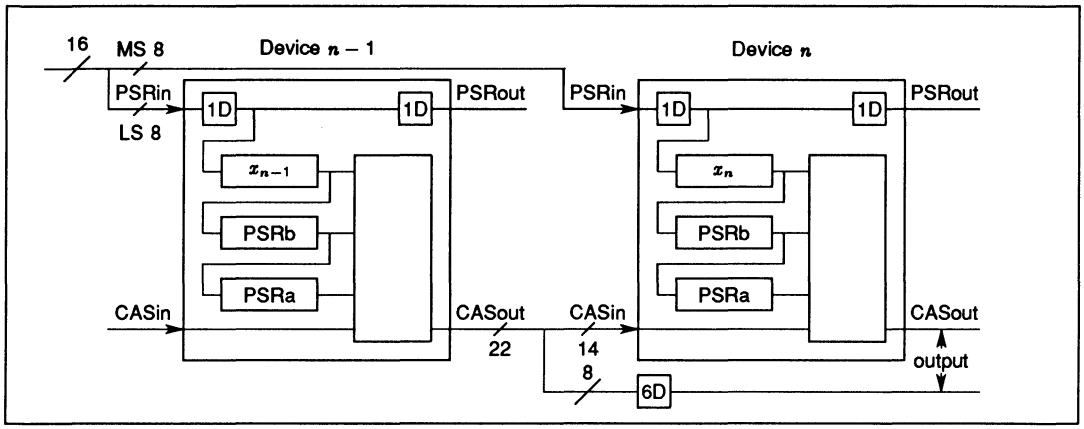


Figure 14.11 Alternative cascade if IMS A110s for increased data precision

The rules discussed earlier in this section about signed data apply equally to this configuration. This means that if signed data was being processed then the left and right hand devices in the diagram would have to be configured for unsigned and signed operation respectively. The one other consideration when increasing the data precision in this way is the number delays required in the programmable shift registers of each device.

Obviously the setings of PSRa and PSRb are not affected by the presence of another device and are setup as described in section 14.2. The setting of PSRc for each device however is important, and incorrect setting will result in erroneous calculation of the most significant 22 bits of the result.

Assuming that the delay in the PSRc of device $n-1$ is x_{n-1} and that the delay in PSRc of device n is x_n, it is desired to calculate the relationship between these delays for correct combination of the partial results. Consider an item of data when it reaches device $n-1$. The delay before the component due to this data, flowing via the reference path in device $n-1$, reaches the cascade adder of device n is:

$$D_{n-1} = 1 + (x_{n-1} + 1) + 3 + D_R + D_B$$

$$D_{n-1} = 41 + x_n$$

Similarly the delay before the component due to this data, flowing via the reference path in device n, reaches the cascade adder of device n is:

$$D_n = 1 + (x_n + 1) + 3 + D_R$$

$$D_n = 35 + x_n$$

Now for the data to be correctly aligned at the cascade adder of device n the delay along each path must be the same. Hence:

$$D_{n-1} - D_n = 0$$

$$41 + x_{n-1} - 35 - x_n = 0$$

$$x_n = x_{n-1} + 6$$

This means that the PSRc of device n must be programmed with the value which is in PSRc of device $n-1$ plus a fixed constant of 6.

Obviously this technique of increasing data precision may be extended beyond 16 bits, or may be combined with other cascading techniques to give larger filter sizes etc.

14.9.3 Increasing data precision with no external hardware

If the data and coefficients are such that only 22 bits or less are required to represent the result then it is possible to increase the data precision with no external hardware. The connections required are similar to those shown in figure 14.11. However, the 6 stage delay must be removed and the full 22 bits of CASout from the first device must be connected to CASin of the second device. To correctly sum the two contributions of the result, it is necessary to left shift the MAC output of the second device 8 places to the left. This shift is easily performed using the shifter in the second device, however, care must be taken to ensure that overflow does not occur. If such an overflow does occur then it will not be detected.

14.10 Cascading IMS A110s for increased coefficient precision

Section 14.9 described three different techniques for increasing data precision by cascading IMS A110s. In this section three very similar techniques are presented for increasing coefficient precision.

14.10.1 Increasing coefficient precision with an external 22 bit adder

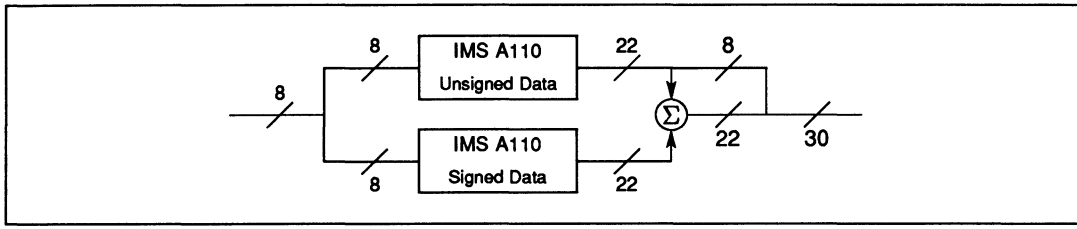


Figure 14.12 Cascade of IMS A110s for increased coefficient precision

The first method makes use of an external 22 bit adder as shown in figure 14.12. At the input each 8 bit value is fed to PSRin of both the IMS A110s. The device at the top of the diagram is programmed with the least significant 8 bits of the coefficients and the device at the bottom is programmed with the most significant 8 bits of the coefficients. If the coefficients are unsigned then both of the devices must be set to unsigned coefficient operation. However, if the coefficients are signed then in order to correctly process the data and preserve the sign information it is necessary for unsigned and signed coefficient operation to be set in the top and bottom devices respectively (see figure 14.12). This may be easily achieved by setting or clearing bit 3 of the SCR register in each IMS A110 as appropriate. The 22 bit partial results are then combined in exactly the same fashion as described in section 14.9.

As discussed for increased data precision this technique may be extended to more than 16 bits of accuracy if required, or may be adapted to make use of increased filter sizes etc. For very high precision systems increased coefficient and data precision may be combined to give very accurate results.

14.10.2 Increasing coefficient precision with an external delay line

The second method makes use of a delay line in a very similar configuration to that discussed in the previous section. A diagram showing the setup may be seen in figure 14.13.

The rules discussed earlier in this section about signed coefficients still apply in this configuration. Hence if signed coefficients are required then the left and right hand devices in the diagram have to be configured for unsigned and signed coefficient operation respectively. The calculation of the setting of PSRc for each device may be calculated in the same manner as described in the previous section. When the calculation is performed the following relationship is developed:

$$x_n = x_{n-1} + 4$$

This means that the PSRc of device n must be programmed with the value which is in PSRc of device $n-1$ plus a fixed constant of 4.

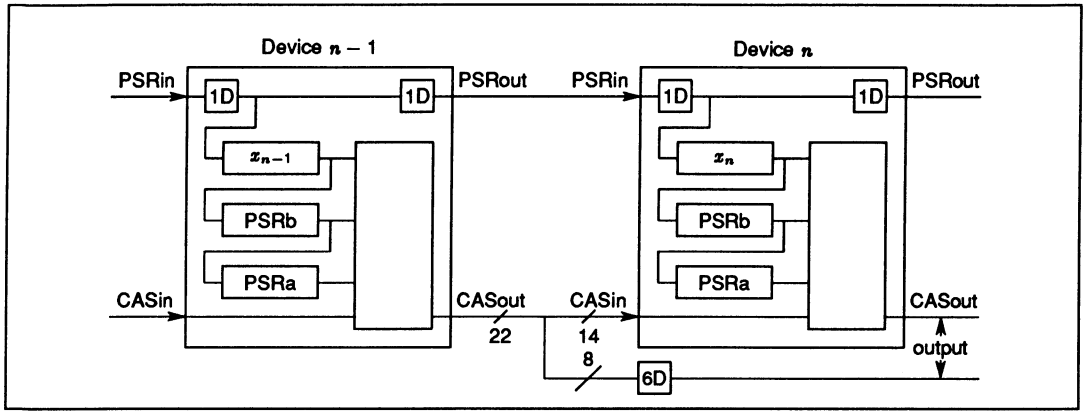


Figure 14.13 Alternative cascade if IMS A110s for increased coefficient precision

Obviously this technique may be extended for more precision or adapted using information presented in earlier sections to give increased filter size, multi pass filtering etc.

14.10.3 Increasing coefficient precision with no external hardware

It was discussed in section 14.9 how to achieve increased data precision without using any external hardware. Since exactly the same technique may be applied to give increased coefficient precision duplicate details are not given here.

14.11 Summary

This document has attempted to describe some of the many ways in which IMS A110s may be cascaded to yield even higher performance. Obviously it has not been possible to discuss every possible configuration but hopefully the examples discussed should have provided both an insight into the extensive capabilities of these devices when cascaded, and some simple rules to allow easy setting up of some of the most common forms of cascades.

the IMS A110 backend post processor

15.1 Introduction

The IMS A110 consists of a high performance configurable array of multiply-accumulators (420 MOPs), three programmable length 1120 stage shift registers and a versatile backend post processing unit. All these features are controlled from a microprocessor interface. The comprehensive on-chip facilities ensure that a single device is capable of dealing with many tasks commonly found in the fields of signal and image processing.

The backend post processing unit gives the IMS A110 a high degree of flexiblility, especially for image processing applications. This document describes by example some of the uses of the backend post processor.

Unless specified otherwise all the examples considered will be based around image processing applications with 8 bits per pixel being used to represent the image data.

15.2 Description of the backend post processor

Figure 15.1 shows the functional blocks and interconnections which are present within the backend post processor of the IMS A110. This diagram can be broken down into 4 main sections, the input block, statistics monitior, data conditioning unit and output block. A brief description of each of these major sections is given below, for full details reference should be made to the data sheet.

15.2.1 Input block (shifter, cascade adder and rectifier)

Data from the MAC array encounters the shifter when it enters the input block. The shifter is capable of up to 8 arithmetic shifts in either direction. When shifting left it is possible for an overflow to occur. Such an overflow is not detected by the device, hence it is left to the user to ensure that unintentional overflows do not occur. When shifting right rounding is applied to improve the accuracy of the device. The magnitude and direction of the shift are controlled by BCR0[5..1] as described in the data sheet.

The output data from the shifter is fed into the cascade adder. Here it is added to both the rounding bit generated by the shifter and the data applied to either the cascade input bus or zero depending on the setting of BCR0[0]. Should the result of the 22 bit signed addition be greater than $2^{21} - 1$ then a positive overflow is generated. Similarly if the result is less than -2^{22} a negative overflow is generated.

The output from the cascade adder can be optionally full or half wave rectified depending on the setting of BCR0[7..6]. The output of the rectifier drives the X bus. Note that when full wave rectification is being used and the output of the cascade adder is -2^{21} then the output from the rectifier remains as -2^{21}.

15.2.2 Statistics monitor

The statistics monitior allows the X bus to be monitored for certain conditions. Four different modes of operation are possible and these are tabulated below:

Mode	BCR1[1]	BCR1[0]
Max Register	0	1
Min Register	0	0
Overshoot Counter	1	1
Undershoot Counter	1	0

When configured to be in max register mode and the X bus exceeds the current threshold in the MMR (max/min register), then the MMR is loaded with the value on the X bus and the counter (OUC) is incremented. If the threshold is not exceeded then no action is taken. Thus assuming the MMR was initially set to -2^{21} its value at some later time is the maximum value which has appeared on the X bus in that period, and the OUC has been incremented by the number of times the threshold has been updated.

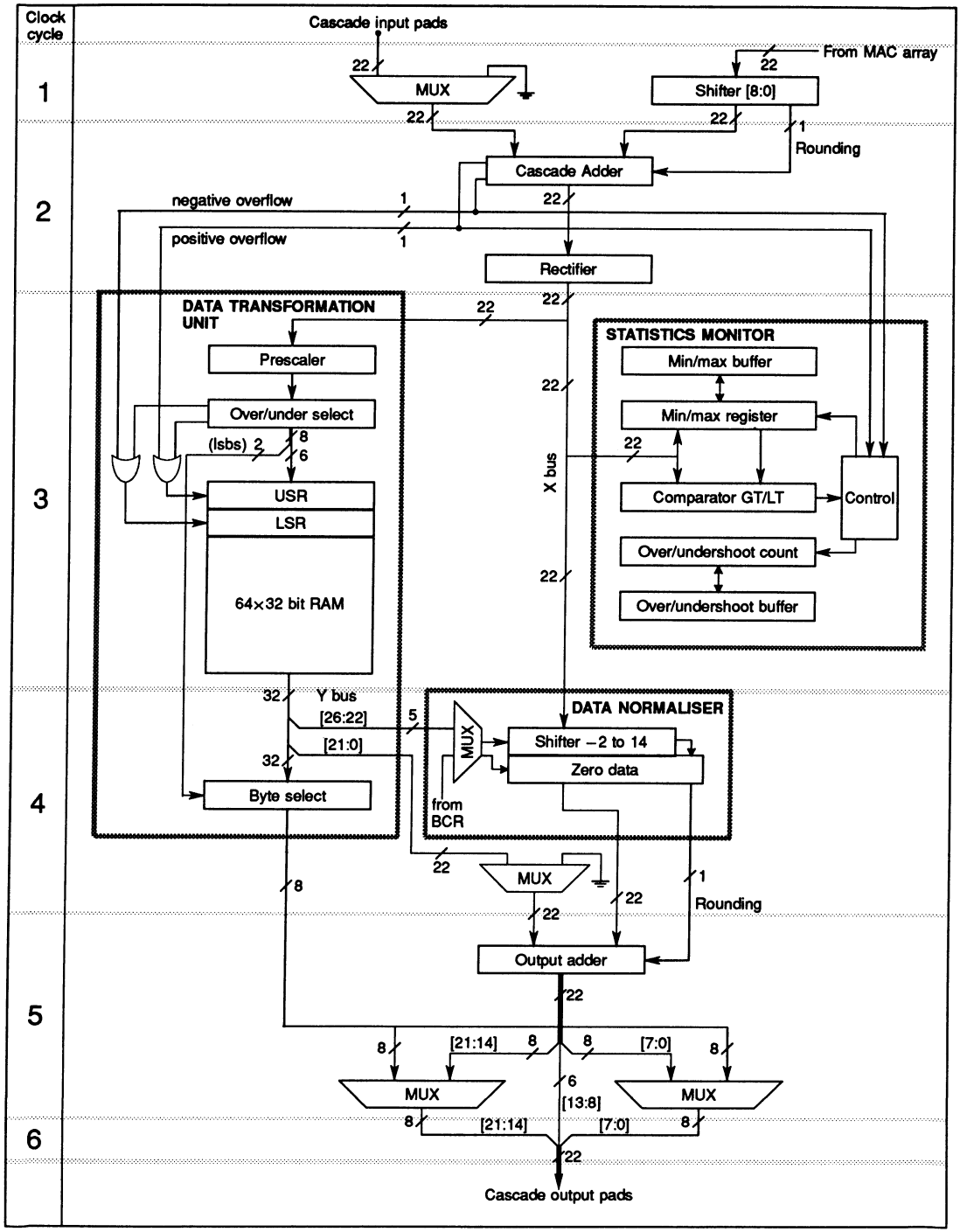


Figure 15.1 Detailed block diagram of the backend post processing unit

If configured to be in min register mode the threshold is updated and the counter incremented whenever the X bus is less than the current threshold. Note that when operating in max/min register mode if a positive or negative overflow occurs then the threshold is not updated since this could leave a misleading value in the MMR.

As an overshoot counter the statistics monitor operates by incrementing the OUC every time the value on the X bus exceeds the threshold in the MMR or if a positive overflow occurs. The OUC is unsigned and will not wrap around, thus behaving as a saturating counter. Similarly when configured to be in undershoot counter mode the OUC is incremented every time the value on the X bus is less than the current threshold.

When overflows occur this is recorded in bits 22 and 23 of the MMR. Positive overflows cause bit 22 to be set while negative overflows cause bit 23 to be set. These bits may be cleared by writing to the MMB copy location.

Direct access to the MMR and OUC via the microprocessor interface is not possible. Instead the reading and writing of these registers is performed by making use of the MMB, CMM, OUB and COU registers. Full details may be found in the data sheet.

15.2.3 Data conditioning unit (data transformation unit and data normaliser)

Data transformation unit

The data transformation unit contains a prescaler, an under/over select detector, a look up table and a byte selector. It may be used on its own to provide arbitrary data mappings of an 8 bit segment of the X bus, or in conjunction with the data normaliser to implement sophisticated dynamic range compression functions.

The prescaler allows an 8 bit field to be selected from anywhere within the 22 bits of the X bus. This 8 bit field is used as an address to the LUT. The over/under select detector monitors the operaton of the prescaler to ensure that all the significant bits and the sign of the X bus are included within the 8 bit field. If this is not the case then an overselect or underselect signal is generated depending on whether the X bus is positive or negative respectively.

The LUT consists of sixty four 32 bit words. In addition there are a further two 32 bit locations known as the upper and lower saturation registers (USR, LSR). The most significant 6 bits of the address field are used to select one of the 32 bit registers in the LUT. This 32 bit output is known as the Y bus. The least significant 2 bits of the address field are then used to control a byte select on the output. Thus the LUT may be used to provide arbitrary 8bit - 8bit data transformations.

Positive overflows on the X bus or overselects in the prescaler cause the LUT to access the USR overriding the address supplied by the prescaler. Similarly negative overflows and underselects cause the LUT to access the LSR. When such conditions occur the byte select control is also overridden thus causing the most significant byte (byte 3) of the appropriate saturation register to appear on the byte wide output of the data transformation unit.

The LUT is programmed via the memory interface. The addressing for the LUT corresponds directly to the 8 bit field, assuming that the byte selector is being used. To enable access to the LUT, USR and LSR from the microprocessor interface the LUT access control bit ACR[1] must be set to zero. This forces the Y bus to zero and causes the normaliser to be controlled by BCR3[7..3] regardless of the setting of the dynamic normalisation bit. Once the LUT has been programmed the LUT access control bit may be reset to one thus allowing the LUT to be used in the data transformation unit.

Data normaliser

The data normaliser contains a shifter followed by a zero data unit. The shifter is capable of right shifts of up to 14 bits and left shifts of up to 2 bits. Any amount of shift outside this range invokes the zero data unit which zeros the output of the data normaliser. The amount of shift is specified by one of two 5 bit sources. These are either BCR3[7..3] or bits 26 to 22 of the Y bus. The source currently selected is determined by the setting of BCR3[2].

15.2.4 Output unit (output adder and output multiplexers)

Output adder

The output adder takes one of its inputs from the data normaliser (including the rounding bit). The other input is either the least significant 22 bits of the Y bus or zero depending on the setting of BCR3[1]

Output multiplexers

The output multiplexers allow the selected byte from the LUT to be optionally selected to drive either the most or least significant 8 bits of the cascade output pins. This feature is controlled by the setting of BCR2[5..6]. Any cascade output pins not being driven by the selected byte are driven by the appropriate bits of the output adder.

15.3 Uses of the backend post processor

15.3.1 Local area averaging

Local averaging is the one of the simplest image filtering operations. A typical local averaging filter may be seen in figure 15.2. Although this filter looks very simple to implement on IMS A110s there is one slight problem and that is how to achieve the divide by nine operation. The operation is necessary to ensure that the output image data requires the same number of bits to represent it as the input data.

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Figure 15.2 Local averaging filter kernel

The IMS A110 is capable of dividing by integer powers of two. Using this capability the $\frac{1}{9}$ could be replaced with $\frac{1}{16}$. Although this would adequately restrict the magnitude of the output data a significant loss of dynamic range could occur. A better solution is to generate an approximation to $\frac{1}{9}$ in the form shown below. Where x represents the coefficient and y the number of right shifts applied:

$$\frac{x}{2^y} \approx \frac{1}{9}$$

It may be simply shown that the closest approximation which may be used with IMS A110s is:

$$x = 57$$

$$y = 9$$

By using these values the local averaging kernel to be programmed into the IMS A110 is as shown below:

$\frac{1}{2^9}$

57	57	57
57	57	57
57	57	57

Figure 15.3 Modified local averaging filter kernel

The division by 2^9 can't be performed by the shifter in the input block since it is only capable of right shifting up to 8 places. The shifter in the normaliser however is capable of right shifting the required nine places.

To configure an IMS A110 so that it performs the local averaging operation used in the above example the following values would have to be programmed into the coefficient and control registers:

Coeff Register	0	1	2	3	4	5	6
CR0a	57	57	57	0	0	0	0
CR0b	57	57	57	0	0	0	0
CR0c	57	57	57	0	0	0	0

Registers	Data msb .. lsb							
SCR	0	0	x	1	1	1	x	0
ACR	0	0	0	0	0	0	x	0
BCR0	x	x	0	0	0	0	0	1
BCR1	0	0	0	0	0	0	x	x
BCR2	0	0	0	x	x	x	x	x
BCR3	0	1	0	0	1	0	0	0

x – indicates don't care.

Exactly the same technique may be applied to other filter kernels which require an awkward division. For example the edge enhancement operation shown in figure 15.4 requires a division by 5 operation. A modified version of the kernel which may be easily implemented is shown below.

$$\frac{1}{5}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Figure 15.4 Edge enhancement filter kernel

$$\frac{1}{2^6}\begin{bmatrix} 0 & -13 & 0 \\ -13 & 64 & -13 \\ 0 & -13 & 0 \end{bmatrix}$$

Figure 15.5 Modified edge enhancement filter kernel

15.3.2 Histogram equalization

Histogram equalization is one example of the wider field of histogram modification [1]. All such operations manipulate the grey levels within an image to generate a new image with a modified grey level histogram. The histogram equalization technique attempts to manipulate the grey levels within an image so that an even spread is obtained across the entire range of intensities. Details of the technique are widely available in the technical press [1] so an in depth discussion will not be provided here.

There are two distinct stages in performing a histogram equalization the second of which IMS A110s are capable of performing. The first stage is the calculation of the transfer function which maps the original image

onto the histogram equalized image. The main computational cost involved in this stage is the determination of the original histogram. The second stage requires the implementation of the transfer function to map the grey levels in the input image to the equalized grey levels in the output image.

The transfer function is implemented by making use of the arbitrary 8bit-8bit mapping ability of the LUT present within the IMS A110. The offset of each location in the LUT may be regarded as one of the original grey levels and the value programmed into that location is the transformed grey level after equalization.

For example suppose that it was desired to use an IMS A110 to perform a histogram equalization on 8 bit image data applied to the cascade input port with the MAC coefficients programmed to zero. The table below shows the values which would have to be programmed into the main control registers. The output data would appear on the lower 8 bits of the cascade output port.

Register	Data msb .. lsb							
SCR	0	0	x	1	x	x	x	x
ACR	0	0	0	0	0	0	A	x
BCR0	x	x	x	x	x	x	x	0
BCR1	0	0	0	0	0	0	x	x
BCR2	0	1	0	0	0	0	0	0
BCR3	1	0	0	0	0	0	0	0
LUT n	D	D	D	D	D	D	D	D

x – Indicates don't care.

A – Set to 0 to program LUT, set to 1 to allow IMS A110 LUT access.

D – Program with the mapping n ⇒ D[7..0].

By modifying the transfer function programmed into the LUT many other operations are possible including thresholding and image contouring which are described in sections 15.3.3 and 15.3.7 respectively.

15.3.3 Edge detection and enhancement

Edge detection

Edge detection is a very important image processing operation since it is often the first stage in feature recognition. For example consider the vertical line detector shown in figure 15.6. This filter is actually the y component of the Söbel operator. The output ($H(x, y)$) from the filter when convolved with an image is a measure of the change of intensity in the y direction at each point.

$$G_y = \frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Figure 15.6 Y component of the Söbel operator

The output at any given point may be positive or negative depending on the direction of the intensity gradient vector at that location. Often when using such a filter to detect vertical edges only the magnitude of the gradient vector is of interest (i.e. its direction is irrelevant). The results may be modified to simply indicate the magnitude by processing the output as shown below.

$$F[x, y] = |H(x, y)|$$

The modulus operation is an ideal example of the use of full wave rectification. The tables below show the configuration of the coefficient and control registers necessary to calculate $|H(x, y)|$.

Coeff Register	0	1	2	3	4	5	6
CR0a	−1	−2	−1	0	0	0	0
CR0b	0	0	0	0	0	0	0
CR0c	1	2	1	0	0	0	0

Registers	Data msb .. lsb							
SCR	0	0	x	1	0	1	x	0
ACR	0	0	0	0	0	0	0	0
BCR0	1	0	0	0	0	1	0	1
BCR1	0	0	0	0	0	0	x	x
BCR2	0	0	0	x	x	x	x	x
BCR3	0	0	0	0	0	0	0	0

x – Indicates don't care.

Typically once an edge detection operator has been convolved with an image it is necessary to make some sort of decision based on the magnitude as to whether an edge exists at each point of the output. The method usually used is known as thresholding [1].

The threshold operation involves mapping all points with a grey level greater than a given threshold to one value (typically 255), and all other points to another value (typically 0). The lookup table as described in section 15.3.2 provides the ability to perform just such an arbitrary mapping. By modifying the control registers presented above it is possible to do not only the edge detection operation and the full wave rectification, but also to apply an arbitrary threshold all within a single device. The updated table of control registers is shown below:

Registers	Data msb .. lsb							
SCR	0	0	x	1	0	1	x	0
ACR	0	0	0	0	0	0	A	0
BCR0	1	0	0	0	0	1	0	1
BCR1	0	0	0	0	0	0	x	x
BCR2	0	1	0	0	0	0	0	0
BCR3	1	0	0	0	0	0	0	0
LUT n	D	D	D	D	D	D	D	D

x – Indicates don't care.

A – Set to 0 to program LUT, set to 1 to allow IMS A110 LUT access.

D – Set to 0 for n less than or equal to the threshold, set to 1 otherwise.

Edge enhancement

Edge enhancement is often applied to images to either counteract blurring or to produce a sharper looking image which is sometimes aesthetically more pleasing. One filter kernel which gives an edge enhancement may be seen in figure 15.5. When this filter is convolved with an image it is possible to generate not only valid positive image data but also negative values under some circumstances. One solution would be to apply full wave rectification to the result however it is generally more acceptable if half wave rectification is applied.

To implement such a filter on an IMS A110 the coefficient and control registers would have to be set up as shown in the following tables.

Coeff Register	0	1	2	3	4	5	6
CR0a	0	−13	0	0	0	0	0
CR0b	−13	64	−13	0	0	0	0
CR0c	0	−13	0	0	0	0	0

Registers	Data msb .. lsb							
SCR	0	0	x	1	0	1	x	0
ACR	0	0	0	0	0	0	0	0
BCR0	0	1	0	0	1	1	0	1
BCR1	0	0	0	0	0	0	x	x
BCR2	0	0	0	x	x	x	x	x
BCR3	0	0	0	0	0	0	0	0

x – Indicates don't care.

15.3.4 Feature recognition

By using the statistics monitor it is possible to get the IMS A110 to see if a given pattern was present within an image. To enable this process to take place a number of things have to be done:

- The MAC coefficients must be configured as a pattern detector for the pattern which is being searched for. If the pattern is large a number of devices can be cascaded [2] to achieve the required window size.
- The statistics monitor must be configured so that it is in max register mode.
- The MMR must be programmed with -2^{21} at the start of the search period (typically at the start of a frame).

As one or more images are processed the MMR register is continually updated to indicate the highest MAC output which has occured so far. When the pattern detector encounters the pattern that it is designed to search for the MAC output should generate a very large output which exceeds a given threshold. This output will be recorded in the MMR. By examining the MMR at the end of the search period (typically at the end of the frame) it is possible to see if the threshold has been exceeded. If this is the case then it is possible to say that the pattern probably occurred somewhere within the data that was processed. The setting of the threshold to achieve reliable operation requires system teaching using known sets of data.

In a similar fashion it is possible to perform feature recognition with the statistics monitor configured as an overshoot counter. In this mode of operation the detection of the desired pattern is indicated by an increase in the value of the OUC (care must be taken to ensure that it does not saturate). The method of setting the threshold at which the overshoot counter is incremented is identical to the description given in the previous paragraph. At first sight it may appear that this method enables the number of occurences of a given pattern to be counted. Unfortunately this is unlikely to be the case for the following reason.

When the pattern being searched for is encountered it is possible for the OUC to be incremented more than once. This is caused by a combination of uncertainty about the pattern and the properties of pattern detectors as decribed below:

- In a typical pattern matching application the pattern is rarely perfect. Degradations from the ideal may be caused by additive noise, distortion of the object, changing lighting conditions etc. To take this into account the threshold is normally set to a value which is low enough to increment the OUC for all likely occurences of the pattern.
- Due to the nature of pattern detectors a large output is not only generated when the detector is coincident with the pattern but quite large outputs can also be generated when it is just off centre.

The combination of these two problems means that each occurence of the pattern could increment the OUC one or more times thus damaging any indication the change in OUC could give about the number of

occurences of a pattern.

15.3.5 Changing conditions compensation

The front end of many automated image processing systems will experience slowly changing input conditions. These may occur due to changing light levels, drifting component tolerances etc. The inclusion of the max/min register modes of the statistics monitor allows the system to automatically compensate for these changes. For example consider a system which uses daylight to illuminate the field of view. As the day proceeds the output from the camera will change. By spending periods of time monitoring both the maximum and minimum levels in the data stream it is possible to adapt the system to take these changes into account.

15.3.6 Binary image processing

A binary image is one which contains only two grey levels. Typically a binary image is the result of a thresholding operation as described in section 15.3.3. By making use of the MAC and the backend it is possible to implement a wide variety of different operations some of which are summarised below:

- Isolated pixel removal – removal of all pixels which have no identical neighbour.
- Line linking – bridging of small gaps between pixels.
- Encoding according to connectivity – coding of pixels depending on their connectivity with respect to surrounding pixels.
- Binary thinning including staircase elimination – [3] [4] [5] [6] [7]
- Feature growth – opposite of the above.
- Conway's game of life – the oldest computer game known to man.

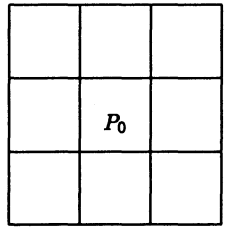


Figure 15.7 A pixel and its 8 closest neighbours

As an example of the techniques involved isolated pixel removal will be examined in more detail. Consider a pixel with its 8 surrounding neighbours as shown in figure 15.7. It is assumed that active and inactive pixels are represented by 1 and 0 respectively.

If the central pixel is in the opposite state to all its surrounding neighbours then the value of the central pixel must be toggled. In order to perform the transformation it is necessary to develop a filter kernel which will give a unique output for each of these two condition. One such kernel is shown in figure 15.8 below:

1	1	1
1	9	1
1	1	1

Figure 15.8 Filter kernel for isolated pixel removal

By programming the MAC with this kernel the outputs generated when the binary image is applied will range from 0 to 17 inclusive. The two particular cases of special interest are 8 and 9 which correspond to a 0 surrounded by 1s and a 1 surrounded by 0s respectively.

To convert from the output of the MAC to a binary image in the original format use may be made of the LUT. The complete mapping for the LUT and the setting of the main control registers for this example are tabulated below:

Coeff Register	0	1	2	3	4	5	6
CR0a	1	1	1	0	0	0	0
CR0b	1	9	1	0	0	0	0
CR0c	1	1	1	0	0	0	0

Registers	Data msb .. lsb							
SCR	0	0	x	1	1	1	x	0
ACR	0	0	0	0	0	0	A	0
BCR0	x	x	0	0	0	0	0	1
BCR1	0	0	0	0	0	0	x	x
BCR2	0	1	0	0	0	0	0	0
BCR3	1	0	0	0	0	0	0	0
LUT 0-7	0	0	0	0	0	0	0	0
LUT 8	0	0	0	0	0	0	0	1
LUT 9	0	0	0	0	0	0	0	0
LUT 10-17	0	0	0	0	0	0	0	1

x – Indicates don't care.

A – Set to 0 to program LUT, set to 1 to allow IMS A110 LUT access.

15.3.7 Multilevel thresholding – image contouring

Often it is desired to highlight a number of areas within a single image. Providing that each of the areas occupies a different region of the grey scale then this can be achieved by multi level thresholding (sometimes known as image contouring). Typically such a technique is often used in medical work. For example consider an X-Ray taken of a patient which may well contain three very distinct regions:

- Clear regions: representing bone.
- Intermediate regions: representing major body organs.
- Dark regions: representing regions where the X-Rays met little resistance.

By using the LUT to provide arbitrary 8bit-8bit data mappings as descibed in sections 15.3.2 and 15.3.3 it is possible to assign each of these three regions a separate value. As a further enhancement external hardware could be used to colour each of the three regions. Such colouring can greatly simplify the comprehension of some types of image.

15.3.8 Dynamic range compression

Consider image data which requires 12 bits to represent each pixel. If it is desired to display such an image on a system which uses only 8 bits per pixel then some form of range compression is required. One solution is to discard the lower 4 bits of each pixel. This would leave the 8 most significant bits for display. If however, the image was dark the lower 4 bits would contain a large proportion of the image data. To throw away the lower 4 bits in such a situation would almost certainly be unacceptable. A better solution in this case would be to use the nonlinear tranformation shown in figure 15.9. Using this transformation values between 0 and 63 are unchanged; values between 64 and 1023 are mapped into the range 64 to 183 and values between 256 and 4095 are mapped into the range 184 to 232.

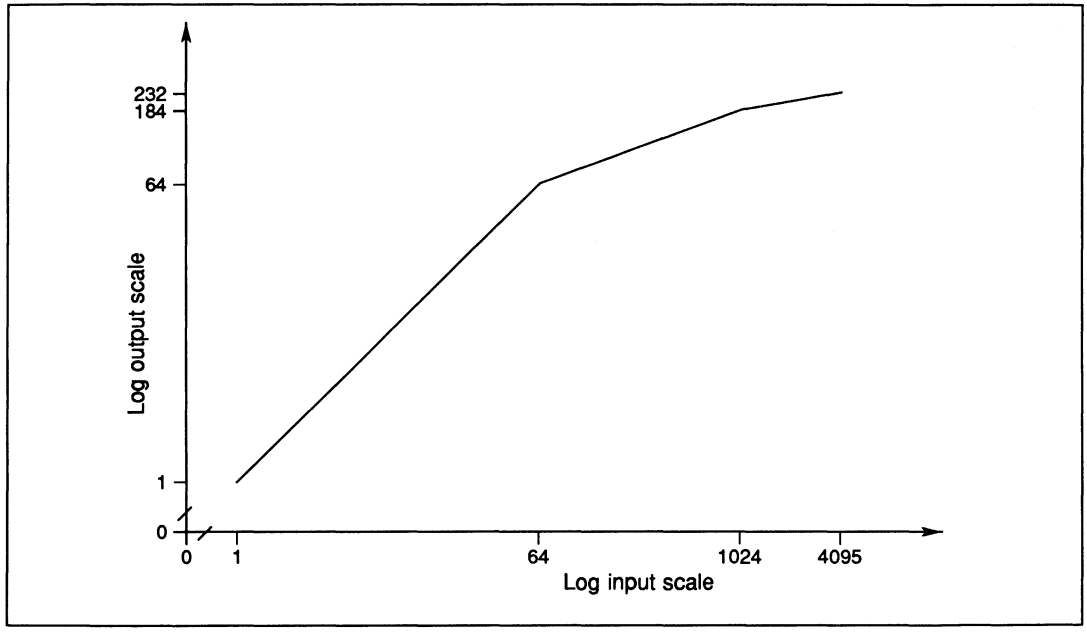


Figure 15.9 Typical dynamic range compression function

The IMS A110 is capable of performing just such a nonlinear transformation by making use of both the data transformation unit and the data normaliser. The mode of operation which is required is known as dynamic normalisation, this is selected by setting BCR3[2] (enable dynamic normalisation). In this mode the prescaler selects a 6-bit field anywhere within the X bus. This is used as an address to the LUT. Bits 22 to 26 of the output of the LUT are used to control the normaliser block so that the input to the normaliser is dynamically scaled. The output of the normaliser is then added, in the output adder, to the least significant 22 bits of the output of the LUT.

The operation can be viewed as:

$$output = (input \times scale) + offset$$

where the scale is provided by bits 22 to 26 and the offset is provided by bits 0 to 21 of the LUT.

To define the transformation function shown in figure 15.9 it is necessary to carefully calculate the values to be placed in the LUT. The first stage in this calculation is deciding which slice of the X bus the prescaler is going to select. In this example it will be set so that bits 4 through to 11 are selected. This means that bits

6 to 11 are used as the address for the lookup table. Bearing this in mind it may be seen that in the first segment of the transfer function the LUT address is zero. Since in this segment the scale is 1 (0 right shifts) and the offset is 0 the following four bytes of data must be programmed into the first 32 bit location of the LUT.

	BYTE 3	BYTE 2	BYTE 1	BYTE 0
LUT 0	00	00	00	00

The second segment of the transfer function occurs between LUT addresses 1 to 15. In this segment the gradient is $\frac{1}{8}$ (3 Right shifts). To ensure that the first and second segment line up correctly it is important to set the offset of the second segment to the correct value. It may be easily shown that in this case the offset is 56. Thus the data to be programmed into the 15 LUT locations from addresses 1 to 15 is:

	BYTE 3	BYTE 2	BYTE 1	BYTE 0
LUT 1	00	C0	00	38
LUT n	00	C0	00	38
LUT 15	00	C0	00	38

In exactly the same manner the LUT data for the third and final segment of the transfer function may be shown to be:

	BYTE 3	BYTE 2	BYTE 1	BYTE 0
LUT 16	01	80	00	A8
LUT n	01	80	00	A8
LUT 63	01	80	00	A8

The settings of the other main control registers to perform the example transform on data applied to the cascade input port are:

Coeff Register	0	1	2	3	4	5	6
CR0a	0	0	0	0	0	0	0
CR0b	0	0	0	0	0	0	0
CR0c	0	0	0	0	0	0	0

Registers	Data msb .. lsb							
SCR	0	0	x	x	x	x	x	0
ACR	0	0	0	0	0	0	A	0
BCR0	x	x	x	x	x	x	x	0
BCR1	0	0	0	0	0	0	x	x
BCR2	0	0	0	0	0	1	0	0
BCR3	x	x	x	x	x	1	1	0

x – indicates don't care.

A – Set to 0 to program the LUT, set to 1 to allow IMS A110 LUT access.

15.4 Summary

This document has attempted to describe by example some of the many ways in which the backend post processor of the IMS A110 may be used. It has only been possible to scratch the surface of a handful of applications but hopefully the examples discussed should have provided an insight into both the flexibility and capability of this section of the device.

15.5 References

[1] R. C. Gonzalez, P.Wintz – Digital Image Processing, Addison Wesley.

[2] R. Whitton – Cascading IMS A110s, INMOS.

[3] R. Stefanelli, A. Rosenfeld – Some Parallel Thinning Algorithms For Digital Pictures. Comm ACM 18, 2.

[4] H.E. Lu, P.S.P. Wang – A Comment On Fast Parallel Algorithms For Thinning Digital Patterns. Comm ACM 29, 3.

[5] C.M. Holt, A. Stewart, M. Clint, R.H. Perrott – An Improved Parallel Thinning Algorithm. Comm ACM 30, 2.

[6] R.W. Hall – Fast Parallel Thinning Algorithms: Parallel Speed And Connectivity Preservation. Comm ACM 32, 1.

[7] Z. Guo, R.W. Hall – Parallel Thinning With Two Subiteration Algorithms. Comm ACM 32, 3.

quality and reliability

The INMOS quality programme is set up to be attentive to every phase of the semiconductor product life cycle. This includes specific programmes in each of the following areas:

- Total Quality Control (TQC)
- Quality and Reliability in Design
- Document Control
- New Product Qualification
- Product Monitoring Programme
- Production Testing and Quality Monitoring Procedure

A.1 Total quality control (TQC) and reliability programme

Our objective to continuously build improved quality and reliability into every INMOS part has resulted in a comprehensive Quality/Reliability Programme of which we are proud. This programme demonstrates INMOS' serious commitment to supporting the quality and reliability needs of the electronics marketplace.

INMOS is systematically shifting away from a traditional screening approach to quality control and towards one of building in Experimental Design quality through Statistical Process Control (SPC). This new direction was initiated with a vigorous programme of education and scientific method training.

In the first year of the programme approximately 80 INMOS employees worldwide received thorough SPC training. This training has been extended to cover advanced SPC and experimental design. Some of the courses taught are listed below:

- Experimental Design Techniques
- Statistical Process Control Methods
- Quality Concepts
- Problem Solving Techniques
- Statistical Software Analysis Techniques

Today INMOS utilizes experimental design techniques and process control/monitoring throughout its development and manufacturing cycles. The following TQC tools are currently supported by extensive databases and analysis software.

1. Pareto charts
2. Cause/Effect Diagrams
3. Process Flow Charts
4. Run Charts
5. Histograms
6. Correlation Plots
7. Control Charts
8. Experimental Design
9. Process Capability Studies

A.2 Quality and reliability in design

The INMOS quality programme begins with the design of new INMOS products. The following procedures are examples from the INMOS programme to design quality and reliability into every product.

Innovative design techniques are employed to achieve product performance using, whenever possible, state of the art techniques. For example, INMOS uses 300 nm gate oxides on its high performance graphics, SRAM and MICRO products to obtain the reliability inherent in the thicker gate oxide. In addition, circuit design engineers work hand in hand with process engineers to optimise the design for the process and the process for the product family. The result is a highly reliable design implemented in a process technology achievable within manufacturing.

INMOS products are designed to have parametric margins beyond the product target specifications. The design performance is verified using simulations of circuit performance over voltage and temperature values beyond those of specified product operation, including verification beyond the military performance range. In addition, the device models are chosen to ensure tolerance to wide variations in process parameters beyond those expected in manufacture.

The design process includes consideration of quality issues such as signal levels available for sensing, reduction of internal noise levels, stored data integrity and testability of all device functions. Electro-static damage protection techniques are included in the design with input protection goals of 2K volts for MIL-STD-883 testing methods. Specific customer requirements can be met by matching their detailed specifications against INMOS designed in margins.

The completion of the design includes the use of INMOS computer aided design software to fully check and verify the design and layout. This improves quality as well as ensuring the timely introduction of new products.

A.3 Document control

The Document Control Department maintains control over all manufacturing specifications, lot travellers, procurement specifications and drawings, reticle tapes and test programmes. New specifications and changes are subject to approval by the Engineering and Manufacturing managers or their delegates. Change is rigorously controlled through an Engineering Change Notice procedure, and QA department managers screen and approve all such changes.

An extensive archiving system ensures that the history of any Change Notice is readily available.

Document Control also has responsibility for controlling in-line documentation in all manufacturing areas which includes distribution of specifications, control of changes and liaison with production control and manufacturing in introducing changed procedures into the line.

Extensive use is made of computer systems to control documentation on an international basis.

A.4 New product qualification

INMOS performs a thorough internal product qualification prior to the delivery of any new product, other than engineering samples of prototypes to customers.

Care is taken to select a representative sample from the final prototype material. This typically consists of three different production lots. Testing is then done to assure the initial product reliability levels are achieved. Product qualifications are done in accordance with MIL-STD-883, methods 5004 and 5005, or CECC/BS9000.

The initial INMOS qualification data, and the ongoing monitor data can be very useful in the user qualification decision process. INMOS also has a very successful history of performing customer qualification testing in-house and performing joint qualification programmes with customers. INMOS remains committed to joint customer/vendor programmes.

A.5 Product monitoring programme

At the levels of quality and reliability performance required today (low PPM and FIT levels), it is essential that a large statistically significant, current product database be maintained. One of the programmes that INMOS uses to accomplish this is the Product Monitoring Programme (PMP).

The PMP is a comprehensive ongoing programme of reliability testing. A small sample is pulled from production lots of a particular part type. This population is then used to create the specific samples to put on the various operating and environmental tests. Tests run in this programme include extended temperature operating life, THB and temperature cycle. Efforts are continuing to identify and correlate more accelerated tests to be used in the PMP.

A.6 Production testing and quality monitoring procedure

A.6.1 Reliability testing

INMOS' primary reliability test method is to bias devices at their maximum rated operating power supply level in a 140° C ambient temperature. A scheme of time varying input signals is used to simulate the complete functional operation of the device. The failure rate is then computed from the results of the operating life test using Arrhenius modelling for each specific failure mechanism known. The failure rate is reported at a temperature that is a typical worst case application environment and is expressed in units of FITs where 1 FIT = 1 Fail in 10E9 device hours, (100 FIT = 0.01%/1000 Hrs). The current database enables the failure rate to be valid over various environmental conditions.

The failure rate goal for INMOS products is 100 FITs or less at product introduction with a 50 FIT level to be attained within one year.

For plastic packaged product, additional testing methods and reliability indices become important. Humidity testing is used to evaluate the relative hermeticity of the package, and thermal cycling tests are used principally to evaluate the durability of the assembly (e.g. die/bond attach).

The Humidity Test comprises of temperature, humidity, bias (THB) at 85°C, 85% Relative Humidity, and a 5V static bias configuration selected to maintain the component in a state of minimum power dissipation and enhance the formation of galvanic corrosion. INMOS reliability goals have always been to meet or better the current 'industry standards' and a target of less than 1% failures through 1000 hours of THB at 90% confidence has been set.

The Thermal Cycling tests are performed from −65°C to + 150 °C for 500–1000 cycles, with no bias applied. Thermal Shock tests using a liquid to liquid (Freon) method are cycled between −55°C and + 125 °C. The INMOS Reliability qualification and monitoring goal for the above tests is less than 1% failures at 90% confidence.

A.6.2 Production testing

Electrical testing at INMOS begins while the devices are still in wafer form before being divided into individual die. While in this form, two different types of electrical test are performed.

The Parametric Probe test is to verify that the individual component parameters are within their design limits. This is accomplished by testing special components on the wafer. The results of these tests provide feedback to our wafer fab manufacturing facilities which allows them to ensure that the components used in the actual devices perform within their design limits. This testing is performed on all lots which are processed, and any substandard wafers discarded. These components are placed in the scribe streets of the wafer so they are destroyed in the dicing operation when they are not of any further use. By placing them there, valuable chip real estate is saved, thereby holding down cost while still providing the necessary data.

The Electrical Probe test performed on all wafers is the test of each individual circuit or chip on every wafer. The defective dice are identified so they may be later discarded after the wafer has been separated into individual die. This test fully exercises the circuits for all AC and DC datasheet parameters in addition to verifying functionality.

After the dice have been assembled into packages they are again tested in our Final Test operation. In a mature product the typical flow is:

- Preburn-in test
- Burn-in at 140°C
- Final test
- PDA (Percent Defect Allowed)
- Device Symbolisation
- QA Final Acceptance

The temperature setting used for hot testing is selected so that the junction temperature is the same as it would be after thermal stabilisation occurred in the specified environment. This is calculated using the hot temperature power dissipation along with the thermal resistance of the package used. All INMOS product is electrically tested and burned-in prior to shipment. Historically, the industry has selected burn-in times using the MIL Standards as a guide (when the market would support the cost) or on a 'best guess' basis dominated by cost considerations. Whereas INMOS invoke a burn-in reduction exercise to ensure the reduced time has no reliability impact.

A.6.3 Quality monitoring procedure

In the Outgoing Quality Monitoring programme, random samples are pulled from lots, that have been successfully tested to data sheet criteria. Rejected lots are 100% retested and more importantly, failures are analysed and corrective actions identified to prevent the recurrence of specific problems.

The extensive series of electrical tests with the associated Burn-in PDA limits and Quality Assurance tests ensure we will be able to continue to improve our high quality and reliability standards.

INMOS MIL-STD-883C/MIL-I-45208
MATERIAL PROCUREMENT & PRODUCT FLOW