Arithmetic Processor Design for the T9000 Transputer

Simon Knowles
Inmos Limited, 1000 Aztec West, Almondsbury BS12 4SQ, England

Abstract

This paper describes aspects of the arithmetic algorithms, architecture, and VLSI engineering of the 64-bit floating-point unit of the T9000 Transputer. The unit is fully conformant to the IEEE-754 floating-point arithmetic standard, and has been implemented in a 1 μm, triple-metal CMOS technology. The 160,000 transistor design performs addition in 40 ns, double precision multiplication in 60 ns, and double-precision division or square root in 300 ns. It will sustain 17 MFlops on the Linpac benchmark, yet occupies less than 15 mm^2 of silicon – about 8.5% of the die area of T9000.

1 Introduction

T9000 is the first of a new generation of high-performance Transputers – general purpose RISC microprocessors tailored to the needs of multiprocessor systems [Inmo91]. Design objectives for T9000 were to provide upward compatibility with the instruction set of earlier 32-bit floating-point Transputers such as the T805, whilst achieving at least an order of magnitude improvement in performance, in Mips and MFlops terms. Naturally, this performance goal demanded a completely new internal machine architecture, including a new Floating-Point Unit (FPU) and a new Integer Unit. Both arithmetic units employ essentially the same techniques for performing each of the basic arithmetic functions. In this paper we discuss specifically the FPU, where the longer wordlength and higher performance requirements generally make the design problems more acute.

One of the key factors in achieving good floating-point performance from a microprocessor is close coupling of CPU and FPU, removing the area and latency overheads inherent in a separate coprocessor design. Currently, most high-performance floating-point microprocessor designs favour on-chip integration of the FPU, and this was regarded as a high priority for T9000. Because T9000 also integrates a large unified cache, a super-scalar CPU pipeline and powerful inter-processor communications hardware, the available silicon area for the FPU was small, even given the rather large die size (approximately 10$\times$18mm). The remarkable density of a new CMOS process technology called 'ELITE' has enabled us to achieve the target floating-point performance with a 160,000 transistor design in about 8.5% of the T9000 die area.

The number of 50 MHz clock cycles required for the key floating-point arithmetic operations, add:multiply:divide, is 2:2:7 for single-precision and 2:3:15 for double-precision. These relative performance figures represent a compromise between typical relative instruction frequency and silicon area investment necessary to achieve a given level of performance for each operation. Microcoding is used to reduce the hardware overhead for less performance-critical operations such as type conversions and DeNorm handling. Although square root is not generally seen as a high-value operation, we have extended the divider unit to incorporate a fast hardware square root capability because the additional silicon cost is minimal and the performance benefit in specific applications is substantial.

The next Section introduces the top-level structure of the FPU, and illustrates how a stack-based programmers model relates to the physical implementation of the FPU as a stage in the T9000 pipeline. Section 3 describes the strategy for meeting the requirements of the IEEE-754 Standard for floating-point arithmetic. The internal architectures of the three floating-point execution units are detailed in Sections 4, 5, and 6, and Sections 7 and 8 discuss aspects of implementation technology, performance, and design methodology.

2 Overview of the Floating-Point Unit

The datapath architecture of the T9000 FPU is illustrated in Figure 1. There are five major components. Three of these are the floating-point execution units – an adder/subtractor, a multiplier, and a divider/rooter. The fourth unit is a simple microsequencer, built around a microcode ROM of about 12000 bits. Several of the less performance-critical operations, such as type conversions and operations involving denormalized values, are microcoded to reduce hardware overhead in the execution units. The fifth unit is a function called the VCU which simplifies the handling of certain special values required by IEEE-754 (see Section 3).

 0-8194-0694-5/91/$4.00

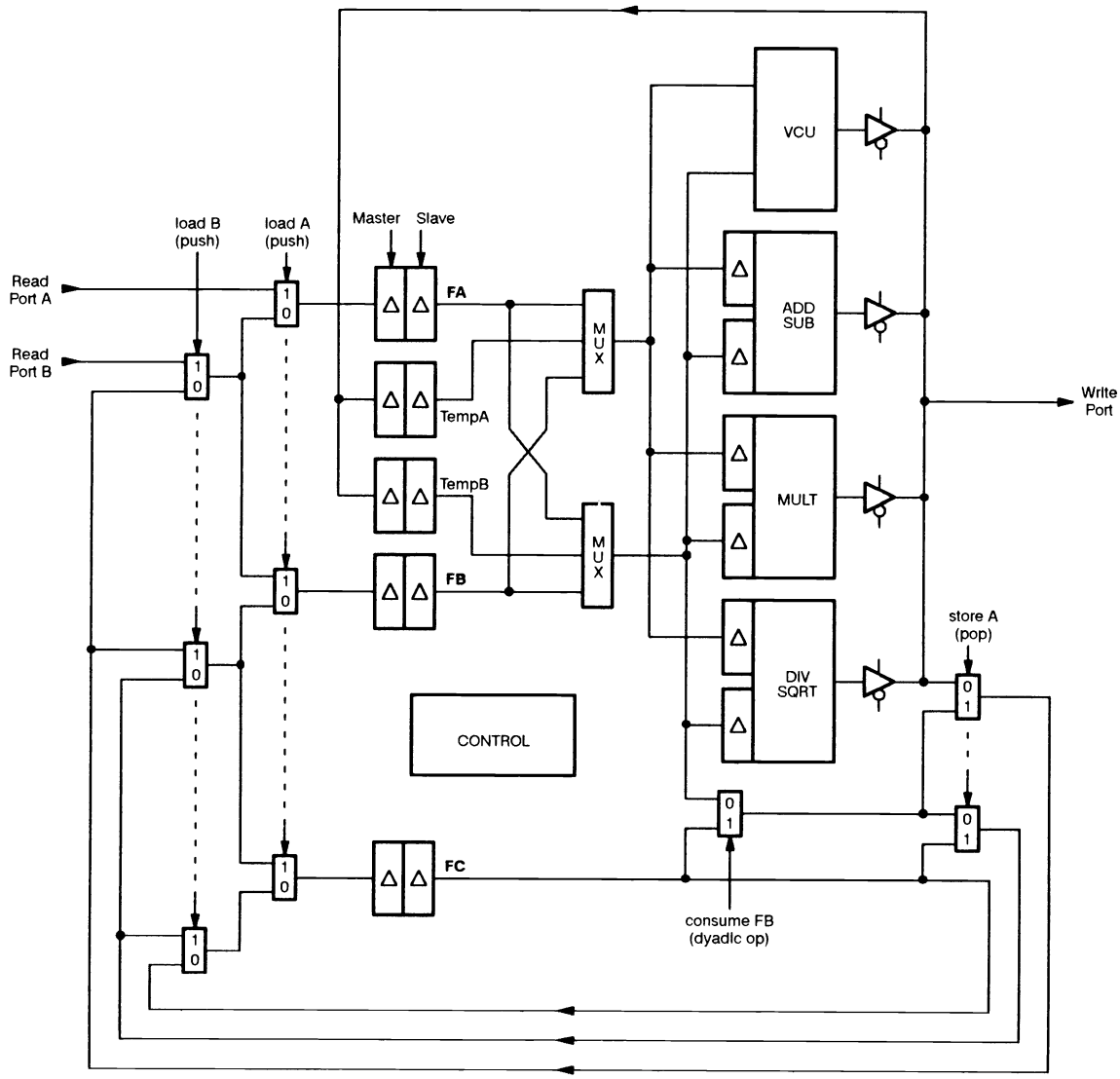


Fig. 1 : FPU Datapath Architecture

The instruction-set model of floating-point arithmetic on Transputers is based on a three element evaluation stack, represented in Figure 1 by the registers FA, FB, and FC. Within the T9000 pipeline implementation, the FPU and the Integer Unit (in parallel) form one pipe stage, with data flowing from left to right in Figure 1. Stack behaviour is emulated by the clockwise internal feedback paths and the push and pop multiplexer banks. One or two operands from the read ports may be pushed onto the stack simultaneously, and the operation is performed on data at the top of the stack (FA, or FA and FB). The contents of FB may or may not be 'consumed' (depending on whether the operation is monadic or dyadic) and the result is either returned in FA or written out of the write port, popping the stack. The anticlockwise feedback path, registers TempA and TempB, and the associated multiplexers are used under microcode control for instructions which require multiple passes through one or more of the execution units – an example of this in the handling of denormalized operands is given in Section 3.

The final point to note in Figure 1 is the arrangement of master and slave latches. In addition to the master-slave main data registers (FA, TempA, etc.), each of the execution units incorporates its own separately-controlled input slaves. This is to ensure that only the appropriate execution unit 'reacts' to new values on the operand busses, so that the peak current demand of the FPU is minimized. Functionally the slaves collocated with the masters are then redundant – they are retained to facilitate performance testing, by preventing the FPU internal buss drivers from 'stealing' cycle time from the previous pipeline stage.

3 Supporting IEEE-754

The ANSI/IEEE-754 floating-point arithmetic standard is now almost ubiquitous in the microprocessor arena, and full conformance is a commercially necessity for a general-purpose processor. As well as defining standard operand formats and arithmetic operations, IEEE-754 includes several features aimed at improving the robustness of floating-point arithmetic, such as controlled rounding, gradual underflow, and error trapping. Accommodating these special behavioural details without significantly impacting the performance available from the basic arithmetic algorithms is one of the major challenges of floating-point processor design.

There are 3 principal facets of IEEE-754 which complicate the design of a conformant floating-point unit :

- Maximally accurate rounding.
- Support for 'special' values – Denormalized numbers, Infinities, and Not-a-Number (NaN) codes.
- An error trapping mechanism, providing vectored access to user-defined error handlers.

Maximally accurate rounding can be accommodated without significantly affecting performance by incorporating some additional hardware. Examples of this will emerge in Sections 4, 5, and 6.

Handling the 'special' values is often a significant burden on the execution units because arithmetic on these values follows subtly different rules to the normal case. T9000 incorporates a separate unit, called the Vacuum Cleaner Unit (VCU), to deal with these special values. The VCU consist of a set of comparators for detecting NaNs, Infinites, DeNorms and Zeros, a constant generator, and a small amount of control logic. When execution of an instruction begins, the appropriate main execution unit operates prospectively while the VCU snoops the FPU operand busses to check the suitability of the operands. If a Zero, Infinity, or NaN operand is detected, the calculation is aborted and the VCU supplies the appropriate special result directly from its constant generator. Similarly if a DeNorm is detected, the Controller may abort the calculation. Under microsequencer control, the Adder/Subtractor will then be invoked to normalize the operand (allowing the exponent range to temporarily exceed that permitted by IEEE-754), and the operation will be restarted using this normalized value. Off-loading the handling of special values to the VCU allows the main execution engines to be optimized for performance on 'well-behaved' data. In particular it is of considerable advantage that the Multiplier and the Divider/Rooter need only deal with normalized operands.

Our implementation of IEEE-754 error handling makes use of the general error handling mechanism built into the T9000 pipeline. When an exception occurs and the appropriate trap is enabled, the pipeline is first restored to its state immediately *before* the erroneous instruction was executed, and control is then passed to the trap handler. The handler can therefore take any actions required, being supplied with the operands, the operation and (effectively) foreknowledge of the exception. Thus we have implemented a conformant trapping mechanism without any additional complexity on the part of the FPU.

4 The Floating-Point Adder

The T9000 floating-point adder performs signed addition (ie. addition and subtraction), type conversions and comparisons, and supports the multiplier and divider/rooter in the handling of denormal values as described above. The hardware overhead required to support these additional operations on the basic addition hardware is small, and the additional design complexity is minimised by microcoding. Here we describe only the mechanism for performing signed addition, which takes two clock cycles (40ns).

4.1 Basic Architecture

In the following description we use the prefix 'true' to distinguish the underlying arithmetic process from the nominal instruction, so for example (−2.37) *add* (13.4) is a true subtraction.

Conventionally, signed addition of normalized floating-point numbers is structured as a 3-step process :

1 **Alignment** : Determine which exponent is smaller. Right shift the corresponding operand mantissa by the exponent difference. Retain the larger exponent as the provisional result exponent.

2 **Signed addition** : Add or subtract the aligned mantissae (according to the operation and the operand signs), and round the result. If the result mantissa overflows (ie. is in the range 2...4), shift the mantissa one bit right and increment the exponent (note that this effectively alters the position at which rounding must be applied).

3 **Normalization** : Detect the position of the most significant '1' in the result mantissa, and shift the mantissa left until this '1' occupies the msb position. Subtract the shift distance from the exponent.

The T9000 floating-point adder derives its speed from several enhancements to this basic algorithm. Firstly we note that a long alignment shift and a long normalization shift can never both occur in the same operation, so the algorithm can be reduced to one of two possible 2-step processes : either align-add, or add-normalize. Secondly the sequential dependency between mantissa addition and normalization distance detection is removed by incorporating hardware to estimate the normalization distance in parallel with the addition (this is described in Section 4.3). Thirdly, several of the major hardware components are duplicated to allow speculative preparation of two possible data values, followed by rapid selection according to the results of other parts of the calculation. In particular, this removes rounding from the critical path.

Figure 2 illustrates the architecture of the mantissa path of the floating-point adder, which is equipped with two alignment shifters, two carry-propagate adders (which can subtract by 2's-complementing one of the operands), a normalization distance predictor, and a normalization shifter. The twin alignment shifters, driven by separate local exponent subtractors, are used to speculatively align both mantissae whilst an exponent subtractor determines the ordering of the exponents. These shifters are naturally structured as binary block shifters, so the exponent subtraction and mantissa alignment can be almost completely overlapped.

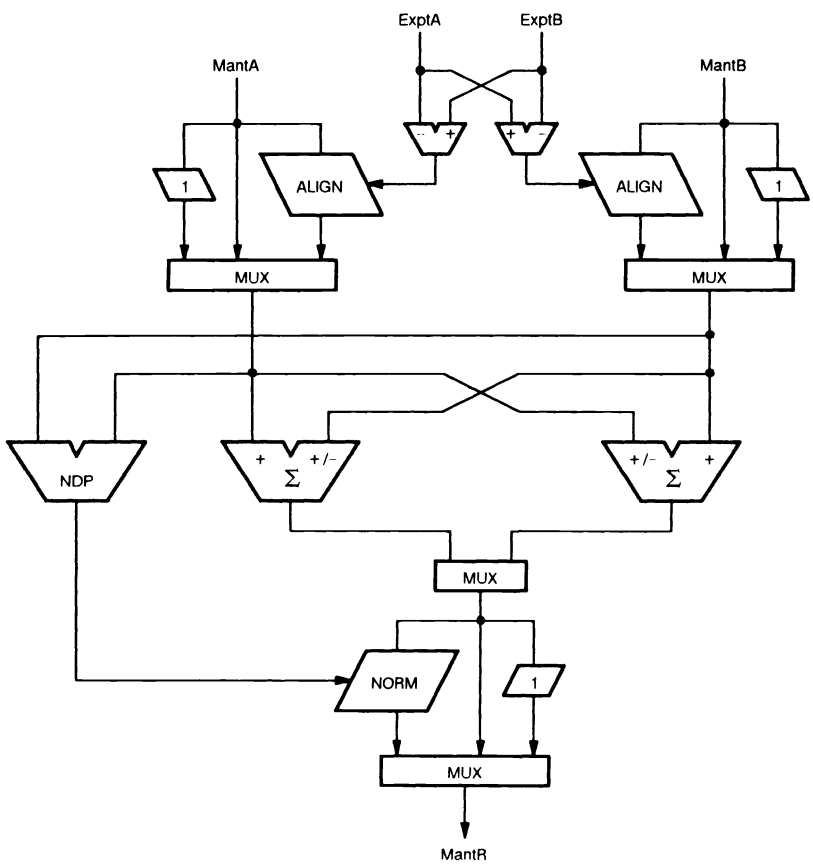


Fig. 2 : Mantissa Path of Floating-Point Adder

The unit has three data-dependent modes of operation :

1 True addition, or true subtraction with an exponent difference of more than 1. Alignment and rounding are generally required, but normalization is limited to a shift of one place. Both mantissa adders are used to form a sum, one assuming that an overflow will occur, requiring a 1-bit normalization shift, and the other assuming that this will not be the case. The correct rounded result can then be rapidly selected when the additions are complete, according to whether overflow actually occured. The main normalization shifter is bypassed.

2 True subtraction with an exponent difference of 1. The necessary alignment is limited to a single bit shift, so the main alignment shifters are bypassed. The two adders are used to form the two possible rounded results as for case 1. The normalisation distance predictor and the normalization shifter are engaged to normalise the result.

3 True subtraction with identical exponents. The alignment shifters are bypassed. Rounding will not be required, so instead of preparing two possible results according to the two possible rounding positions, the twin adders are used to form $A_{Mant} - B_{Mant}$ and $B_{Mant} - A_{Mant}$ (since we do not know which operand is larger). The correct option is then selected as the one yielding a positive result. The normalisation distance predictor and normalization shifter are engaged.

When a true subtraction is initiated, the alignment shifters are engaged according to case 1 (speculating that the exponents are distant), and the adders are engaged according to case 2 or 3 (speculating that the exponents are close or identical). When the exponent subtractors reveal the true status, either the alignment shifter outputs are ignored, or the adders are restarted with the results of the alignment as operands.

4.2 Mantissa Carry-Propagate Adders

The main 57-bit carry-propagate adders in the mantissa datapath employ two carry acceleration techniques to achieve the required performance. Firstly a 2-level skip path ([Lehm61]) allows carries to bypass groups of bits. Secondly, the basic ripple backbone on which the carry skip hierarchy is built operates as a radix-4 machine, using elemental cells which accept two adjacent pairs of operand bits. The implementation is fully static, and the same basic design is re-used in the floating-point multiplier and divider/rooter, and in the integer arithmetic unit.

The carry skip (or carry bypass) scheme may be considered a subset of carry look-ahead, in which only the group propagate terms are computed. In current VLSI technologies such architectures are particularly efficient, because they offer performance not far short of full carry look-ahead for considerably less hardware investment. This is because most of the speed advantage of look-ahead over the basic ripple structure is attributable to the acceleration of carry propagation, rather than generation or assimilation, whereas the bulk of the hardware is devoted to accelerating carry generation and assimilation, rather than propagation.

The structure of the radix-4 cells forming the backbone of the adder is illustrated in Figure 3(a). The cell effectively implements a 1-bit carry look-ahead scheme, having the advantage that the logical depth of the critical path from carry-in to carry-out is half that of a conventional radix-2 implementation. Unfortunately this advantage is offset by the reduced granularity of the resultant ripple path, which reduces the scope for optimization of the carry skip hierarchy. The rationale for adopting the radix-4 approach rests on the observation that increasing the radix of the backbone preserves regularity of layout, whereas constructing a more complex skip hierarchy does not.

The performance requirements of T9000 are such that a fairly simple 2-level skip structure can be employed, having a maximum block size of 4. The structure is illustrated in Figures 3(b) and (c).

4.3 Normalization Distance Prediction

The long-distance normalization step has traditionally been a performance bottleneck for floating-point adders, because detection of the most significant bit of the result implies a serial scan from left to right along the result word. Since the preceding addition involves carry propagation from right to left, the two operations cannot overlap. A solution to this problem is to construct a normalization distance *predictor* (rather than *detector*) which is driven by the addition operands rather than by the result. Such a mechanism has been reported for the AT&T DSP32 signal processor [Hays85] and the IBM RS/6000 microprocessor [Hoke90].

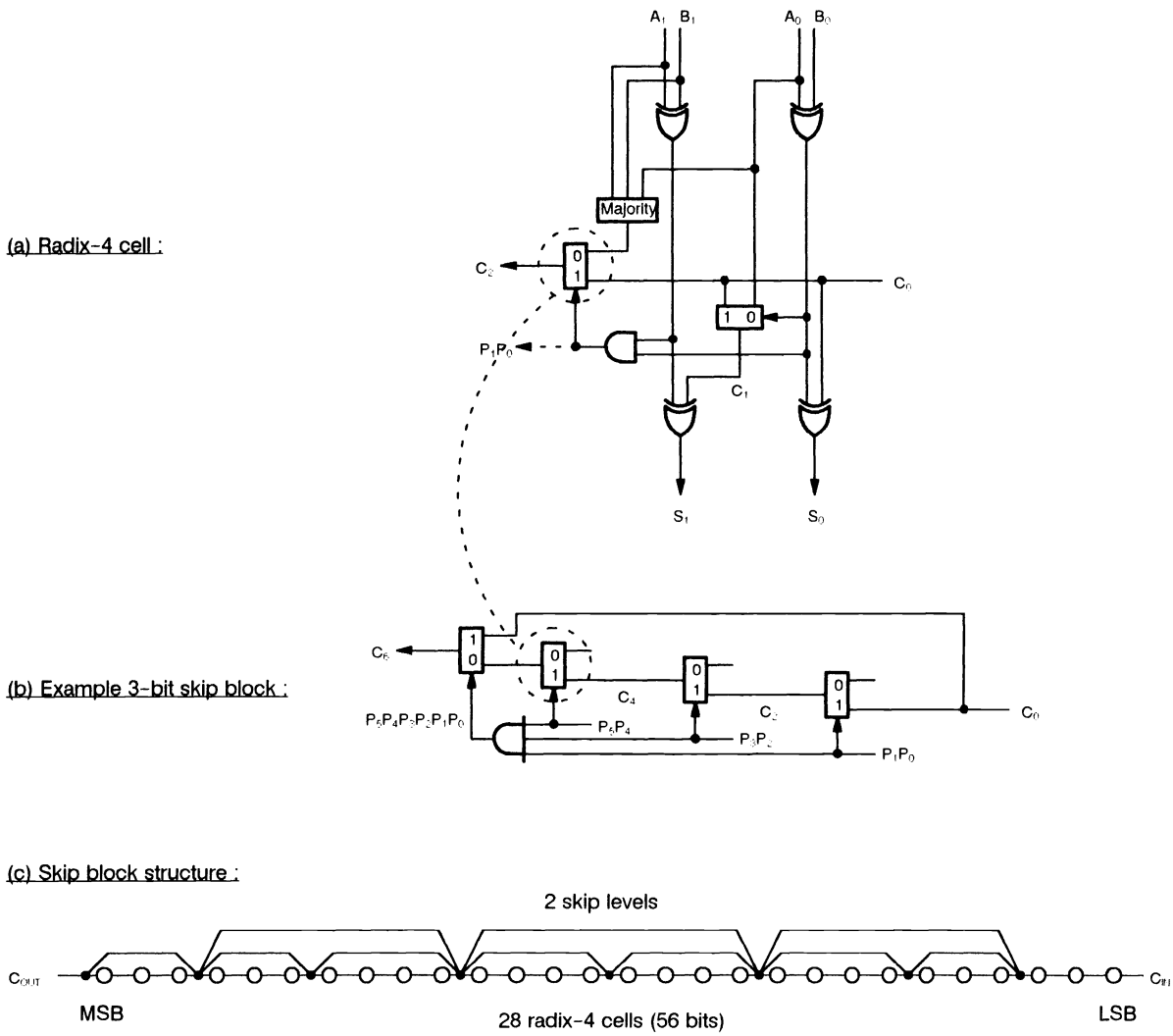


Fig. 3 : Radix-4 Carry-Propagate Adder

The T9000 normalization distance predictor (NDP) determines the normalisation distance, subject to an error of 1 bit position, in parallel with the twin carry-propagate additions. The error of one bit is easily detected once the adder results are available, and any misalignment corrected by an additional shift of one bit. To illustrate how the predictor works (space precludes a more rigourous analysis), consider the following two examples, in which the normalization point is arrowed and the operand bit pairs are annotated in terms of the conventional carry generate (G), kill (K), and propagate (P) conditions :

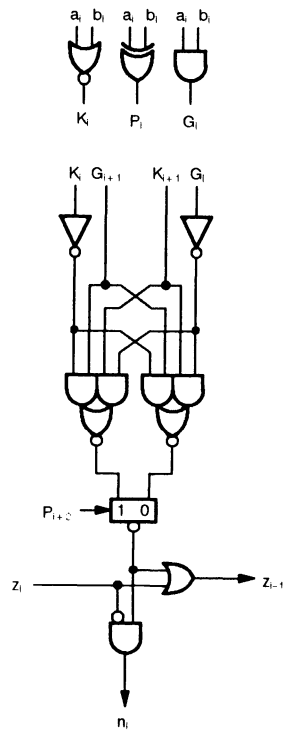
```
A = +141    00010001101          A = +141    00010001101
B = - 41    11111010111          B = -127    11110000001
            -----------                      -----------
   +100     00010100100              + 14    00000001110
            PPPGPKPPGPG                      PPPGKKKPPKG
```

In both of these examples, operand B is negative and has smaller magnitude than operand A, so the addition result will be positive. The first example demonstrates what happens in the majority of cases – scanning the operand bit pairs from left to right, P conditions are observed until the normalization point, which is marked by a G condition. The second example shows a phenomenon known as 'blocking', in which signalling of the normalization point is delayed because the G condition is followed by a string of K conditions (this is a

manifestation of the familiar 'normalization error' property of redundant numbers – because the NDP does not propagate carries it is effectively operating in the redundant domain).

For the symmetrical cases in which the addition result is negative, the behaviour differs only in that the G and K conditions are exchanged. Generally we do not know the polarity of the result in advance, because we do not know the relative magnitudes of the operands. Whatever the polarity of the result, we can determine whether a given bit position is a potential normalization point by observing 3 adjacent pairs of operand bits. Sequentially ORing these 'potential normalization point' signals from most significant to least significant will then identify the true (most significant) normalization point.

Figure 4 illustrates this scheme, and the accompanying table identifies the 27 possible combinations of G, P, and K conditions for the inspected triplet of operand bit pairs. There are 12 conditions which indicate a possible normalization point (z=1), and 9 which indicate the definite absence of the normalization point (z=0). The remaining 6 conditions can only ever occur to the right of the true normalization point, and are therefore not important.



[i+2...i]	z_i
K K K	0
K K P	1
K K G	1
K P K	x
K P P	x
K P G	x
K G K	1
K G P	1
K G G	0
P K K	1
P K P	1
P K G	0
P P K	0
P P P	0
P P G	0
P G K	0
P G P	1
P G G	1
G K K	0
G K P	1
G K G	1
G P K	x
G P P	x
G P G	x
G G K	1
G G P	1
G G G	0

Fig. 4 : Normalization Distance Predictor Cell

The ripple OR path in Figure 4 may be accelerated by any of the techniques conventionally used for carry acceleration. For example, the DSP32 [Hays85], uses a precharged pass-transistor ripple path, whereas the RS/6000 [Hoke90] uses a block look-ahead (prefix) scheme. The T9000 Transputer employs 3-level skip acceleration, in keeping with the use of carry-skip adders throughout the chip. The resultant design operates at about the same speed as the adders.

5 The Floating-Point Multiplier

The T9000 floating-point multiplier employs the widely-used techniques of Booth recoding and carry-save accumulation of partial products. Single-precision multiplications execute in 2 cycles – one for Booth recoding and carry-save reduction, and a second for carry-propagate addition, normalization, and rounding. Double-precision operations require two passes through the carry-save core, and therefore take 3 cycles.

5.1 Basic Architecture

Forming the product of two IEEE-754 mantissae A and B implies the weighted accumulation of 53 (double precision) or 24 (single precision) partial products of the form $b_i A$, where b_i is a bit of B. The well-known radix-4 Booth algorithm halves the number of partial products by recoding the n binary bits of the multiplier B into $n' = \lceil \frac{n+1}{2} \rceil$ radix-4 digits b'_i, each belonging to the digit set $[-2,-1,0,1,2]$.

The recode algorithm may be succinctly expressed as follows, from which it is easily deduced that the recode does not require carry propagation, and can therefore be very fast :

$$
\begin{aligned}
B &= \sum_{i=0}^{-(n-1)} b_i.2^i \\
&= \sum_{i=0}^{-(\frac{n-1}{2})} (b_{2i-1} + b_{2i} - 2b_{2i+1}).2^{2i} \quad ; \text{bounded by } b_1 = b_{-n} = 0 \\
&= \sum_{i=0}^{-(\frac{n-1}{2})} b'_i.4^i \quad ; \text{where } b'_i = (b_{2i-1} + b_{2i} - 2b_{2i+1}) \in [\bar{2}..2]. \qquad (1)
\end{aligned}
$$

Using such a recode reduces the necessary number of partial products $b'_i A$ to 27 for double precision and 13 for single precision operation, whilst introducing the slight complication that the partial products are now signed. If the signs are accommodated by a 2's-complement scheme then the complication of dealing with bits of negative weight is restricted to the most significant edge of the multiplier. Formation of the 2's-complement partial products is carry-free (therefore fast), requiring only bitwise AND, invert, and shift operations. Associated with each partial product is a 2's-complement correction bit which must be added at the lsb of the word. Taken together, these correction bits are treated as an additional partial product, bringing the total to 14 for single precision and 28 for double.

The structure of the carry-save accumulation core is illustrated in Figure 5. Each box in this figure represents a word-level operation, and for clarity does not show the relative shifts between words (reflecting their relative weights). Such shifts may be considered to occur in the dimension perpendicular to the page.

The core is constructed largely of carry-save adders, which are here denoted '3:2' – this is conventional multiplier notation for a device which accepts a value represented by the sum of 3 binary words and delivers the same value represented as the sum of 2 binary words. Two iterative arrays of full adders operate in parallel to reduce the partial products to a pair of carry-save values (ie. 4 binary words). Each of the arrays has double-precision width and is 5 rows deep, and can therefore accumulate 7 partial products. Up to 14 partial products can be reduced in one pass through both arrays. The array outputs are combined in a redundant 4:2 stage (actually implemented as 2 further rows of 3:2 cells – [?]), delivering a single carry-save representation of the product. A final 3:2 stage provides an additional input for feedback of a part-computed result from the previous cycle – this is used for double precision operation. Note that the partial result is assimilated before being fed back – this reduces the necessary length of the final carry-propagate addition.

The redundant accumulation architecture shown in Figure 5 attempts to balance speed against layout regularity and peak power demand. In particular it provides a high degree of serialism in the early stages of the calculation, so there is maximum opportunity for peak current spreading. The accumulator has a logical depth of 8 3:2 cells. The minimum possible depth is 6 cells, achieved for example by a Wallace tree, but the estimated peak current consumption of such a tree is twice that of the twin iterative arrays.

5.2 Rounding and Normalization

All operands submitted to the Multiplier are normalized, so the product of the operand mantissae must lie in the range [1..4). To ensure a normalized result, values in the upper half of this range require a single-bit shift. Because of the possibility of a normalization shift, the exact position of the lsb of the product is not known at

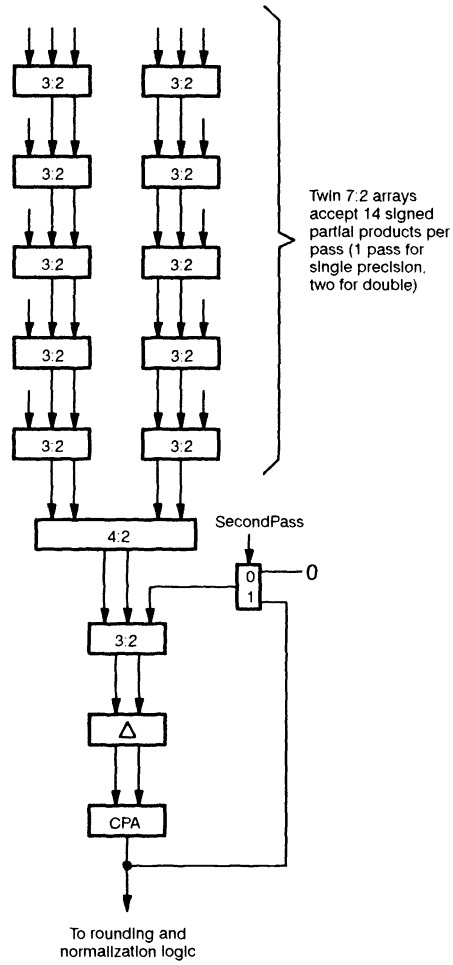


Fig.5 : Multiplier Partial Product Accumulation

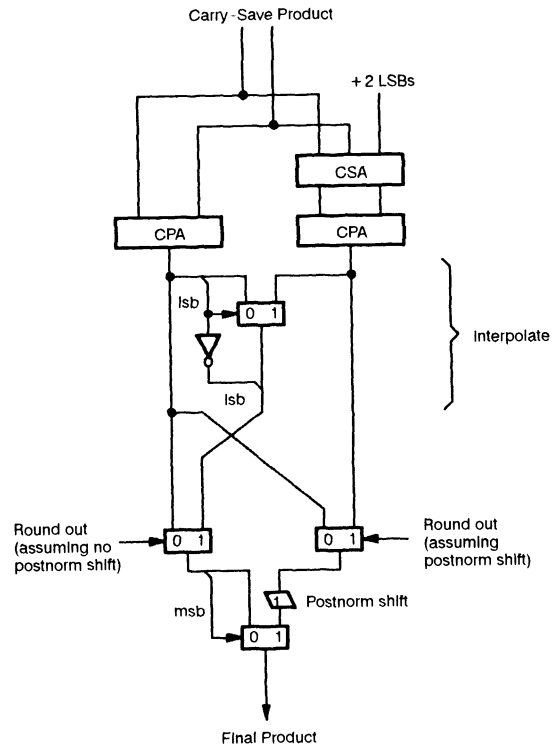


Fig. 6 : Multiplier Interpolative Rounding and Postnormalization

the time of assimilation. This suggests that rounding cannot be incorporated as part of the assimilation, but demands a second, subsequent, carry propagation.

To achieve high speed multiplication, rounding and postnormalization must be tightly coupled with the assimilation of the carry-save product. Since there are only a small number of alternatives for the final product at this stage, we can accelerate the process by computing all possible values simultaneously during the assimilation phase, and then selecting the correct alternative. There are 4 possible results, for example for double precision :

Round in,	not shifted :	$P[0..\overline{52}]$.
...	shifted :	$P[1..\overline{51}]$.
Round out,	not shifted :	$P^{+}[0..\overline{52}]$.
...	shifted :	$P^{++}[1..\overline{51}]$.

Here P is the product delivered by the accumulator, and P^{+} and P^{++} denote P incremented by one or two lsbs (of P) respectively. To simultaneously compute these three values it is only necessary to provide carry-propagate adders for the extrema P and P^{++}. The intermediate value P^{+} can then be *interpolated* very rapidly according to the following table, in which $\hat{P}$ and $\hat{P}^{++}$ are the values P and P^{++} without their lsbs, and '.' means concatenation :

$$\begin{array}{lll} \underline{P} & \underline{P^+} & \underline{P^{++}} \\ \hat{P}.0 & \hat{P}.1 & \hat{P}^{++}.0 \\ \hat{P}.1 & \hat{P}^{++}.0 & \hat{P}^{++}.1 \end{array}$$

The resultant combined structure for interpolation, rounding, and postnormalization is illustrated in Figure 6 (a simplified version of this structure is actually implemented in T9000). The interpolation is performed by the first bank of multiplexers and associated inverter. The twin multiplexer banks at the second level perform rounding under the control of the appropriate least significant bits of the product – the right-hand bank on the assumption that a postnormalization shift will be necessary, and the left-hand bank on the assumption that it will not be necessary. Finally the third level of multiplexers may perform a postnormalization shift, depending on the size of the left-hand rounded result.

6 The Floating-Point Divider/Rooter

For reasons of economy, the majority of integrated floating-point units constructed to date have not incorporated dedicated hardware units for division and square root. Instead, most have employed microcode control of the floating-point multiplier and adder to implement approximation algorithms such as the Newton-Raphson iteration. The first notable devices to incorporate a separate execution unit dedicated to division were probably the MIPS R3010 coprocessor [Rowe88] and the Weitek 3364 [Birm88]. Both of these processors implement the 'SRT' algorithm [Robe58], generally the fastest of the direct algorithms. The T9000 floating-point divider/rooter is likewise based on a radix-2 version of the SRT algorithm. Division, square root, and remainder operations are mapped onto the same core architecture, but here we discuss principally division.

Space precludes a proper explanation of the SRT algorithm, and the unfamiliar reader is referred to [Nadl56] and [Robe58]. The division iteration is of the form :

$$P_i = 2P_{i-1} - q_i D \tag{2}$$

where P_i is the partial remainder (initially the dividend), D is the divisor, and q_i is the quotient digit. At each iteration, the new quotient digit is chosen so as to reduce the magnitude of the partial remainder and thus refine the current estimate of the quotient. The great strength of the SRT algorithm lies in the use of a redundant representation for the quotient. This allows a valid quotient digit to be selected from just an *estimate* of the current partial remainder, which in turn allows the partial remainder reduction to be executed redundantly. This avoids the necessity for carry/borrow propagation along the full width of the remainder word during reduction, and thus allows the iteration to proceed very rapidly.

We employ a quotient radix of 2, so q_i is chosen from the redundant digit set $[-1, 0, 1]$. The partial remainders are represented in borrow-save form, and each iteration uses a row of full subtractor cells to implement the reduction operation. To allow the subtractors to add (when $q_i = -1$), each row is also able to 2's-complement the divisor operand under the control of the quotient digit. The core implements four such rows to yield four bits of the quotient per clock cycle.

The most significant edge of the mantissa core is illustrated in Figure 7(a). Two features are particularly worthy of note. Firstly, selection of the quotient digit depends only on the most significant 3 digits of P_i (ie. 6 binary wires in the chosen borrow-save format). Secondly, 'compression' stages are incorporated between each borrow-save subtractor and the next quotient digit selector cell. This is because the newly-formed partial remainder, being formed redundantly, may have rather more non-zero leading digits than strictly necessary. For example, 00001 = 00$1\bar{1}$ = $1\bar{1}1\bar{1}1$. We must explicitly *compress* the upper bits of the newly formed partial remainder to prevent this wordlength growth.

Inserting a compressor between each pair of borrow-save subtractor stages stretches the critical path of the machine, so it is preferable to overlap compression with selection and broadcast of the next quotient digit. It is even more expedient to combine the reduction and compression of the most significant few digits into a single operation. This latter assertion hinges on the realization that such a combined reduction/compression cell requires as inputs only the top three digits (6 binary wires) of the partial remainder – *exactly the same digits as are required for quotient digit selection.* Therefore the combined cell need not wait for the quotient digit selector to decide on the reducing action (add, subtract, or do nothing) since all the necessary information is already implicitly available. Such a combined compressor/reducer cell can therefore operate autonomously from the rest of the array. This optimization is implemented in T9000 and is illustrated in Figure 7(b).

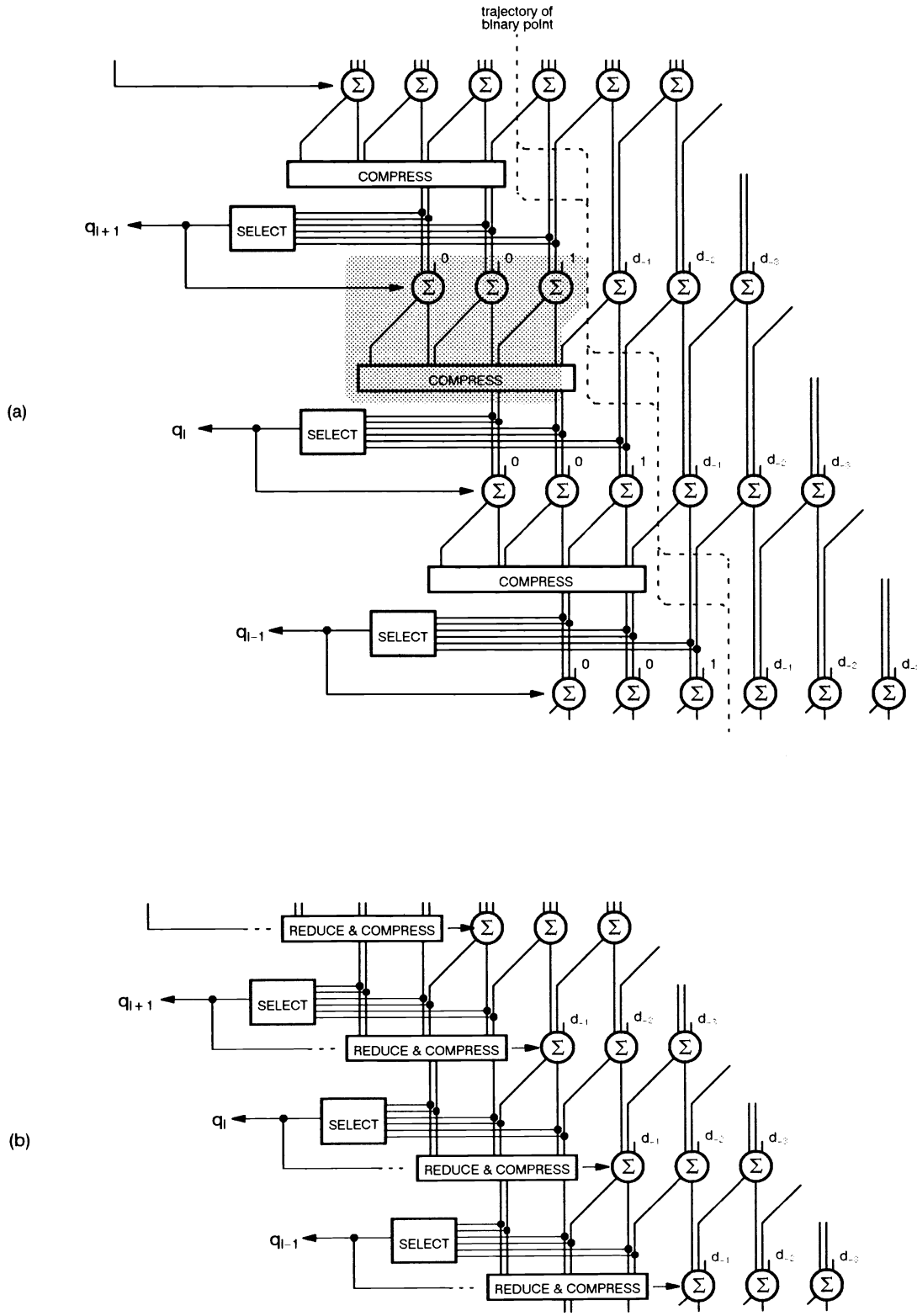


Fig. 7 : Most Significant Edge of the Divide/Root Core

The square root iteration is similar to that for division [Maje85], except that the divisor D is replaced by the current root estimate Q_{i-1} :

$$P_i = 2P_{i-1} - q_i(2Q_{i-1} + 2^{-i+1}q_i) \tag{3}$$

The required number of digits of the borrow-save partial remainder which must be inspected to determine the next root digit is again 3, and the compression scheme is identical.

The key distinction between the algorithms concerns the representation of the current root estimate Q_{i-1}. The performance advantage of the modified SRT algorithm relies on Q being computed redundantly, so to be able to accommodate Q in place of D in the borrow-save reduction stage we must be able to assimilate Q (ie. convert it into non-redundant form) at each iteration step. Conventionally this implies a full borrow propagation, but this would be prohibitively slow. T9000 uses a progressive carry assimilation (PCA) technique [Maje85] which computes a new fully-assimilated value each time a new redundant digit arrives, without the prohibitive time or complexity overhead of a discrete full assimilation.

PCA is a linearization of the 'conditional sum' addition technique, and in its most basic form requires the storage of two 'base' values at each step of the iteration. One of these values is the current result assuming that the remaining (unknown) part of the word is zero or positive overall, and the other is the current result assuming that the remaining part of the word is negative overall. Representing these two values as Q_{i-1} and Q^-_{i-1}, the algorithm for updating both values on arrival of a new redundant digit q_i is as follows (here '.' represents concatenation) :

$$\begin{array}{lcll} q_i = 1 & \Rightarrow & Q_i & \leftarrow & Q_{i-1}.1 \\ & & Q^-_i & \leftarrow & Q_{i-1}.0 \\ q_i = 0 & \Rightarrow & Q_i & \leftarrow & Q_{i-1}.0 \\ & & Q^-_i & \leftarrow & Q^-_{i-1}.1 \\ q_i = \overline{1} & \Rightarrow & Q_i & \leftarrow & Q^-_{i-1}.1 \\ & & Q^-_i & \leftarrow & Q^-_{i-1}.0 \end{array}$$

Rounding and postnormalization are handled in a similar manner to the postprocessing stage of the multiplier. There is a further complication in the divide/root case, because the algorithm may generate a negative remainder, Adjustment of the quotient (or root) and remainder values to yield a positive remainder before postnormalizing and rounding the quotient is called postcorrection. Again the time penalty of all three of these terminal operations is virtually eliminated by merging them into a single selection tree, and employing an interpolation network.

7 Technology and Performance

The maximum sustainable floating-point performance of T9000 is 25MFlops at 50MHz internal clock, under worst case conditions of temperature, supply voltage, and process corner. This performance is scalar rather than pipelined, and is therefore very accessible in general-purpose applications. About 17 MFlops are achieved on the Linpac benchmark. The following table summarizes the performance, area, and complexity characteristics of the three floating-point execution units :

Unit	Complexity	Area	Performance Single/Double
Adder/Subtractor	35kT	3.1mm^2	25/25 Mflops
Multiplier	44kT	3.7mm^2	25/17 Mflops
Divider/Rooter	45kT	3.7mm^2	6/3 Mflops

The device is implemented in a triple metal, single poly, 1μm p-well CMOS process. The upper metal layers are aluminium, the bottom layer tungsten, and tungsten plug vias are used. The process allows via stacking and placement on minimum metal pitch, and therefore achieves a remarkable transistor density – the execution unit cores integrate about 17000 transistors/mm^2. Apart from the microcode ROM, which is precharged, all of the circuits within the FPU use fully static N-pass or complementary gate logic.

An increasingly important side-effect of fast, dense processes is that the effects of switching current transients in the power and ground lines are particularly severe. The current generation of VLSI CAD tools do not enable realistic simulation of these effects. In CMOS, peak supply current is a more significant design issue than the thermal effect of average current, which can be more easily modelled. There are two areas of impact :

- Electromigration of the supply tracks.
- Supply voltage drops due to track and bond wire resistance and inductance, leading to circuit slow-down and potential disruption of neighbouring circuitry.

Sections 2 and 5 have highlighted two of the areas where considerations of peak supply current have had an impact on the *architecture* of the T9000 FPU – the twin-array structure of the multiplier core, and the duplicate slave latching of operands into each of the execution units. To maximize supply decoupling on T9000, vdd and ground are routed as fine two-dimensional grids throughout the chip. The availability of a tungsten interconnect layer is frequently useful here, because its electromigration characteristics are far superior to those of aluminium. Unfortunately its resistance is higher, so transient voltage drops are worse.

8 Design Methodology

The 160,000 transistor T9000 FPU is the result of about 10 man-years design and layout effort over a period of 2 years, by a team which started with 2 Engineers and peaked at 8. This is approximately 15% of the total design effort for T9000.

Simulations at both logic and circuit level, as well as all the layout, were performed using Inmos proprietary CAD tools on Transputer-based workstations. Most of the design is full custom to achieve the required speed and layout density – in particular significant effort was expended in optimizing the cores of the multiplier and divider/rooter, and the carry-skip adder design which recurs throughout the FPU. To accelerate the implementation of some of the random control logic we used the Synopsys logic synthesizer in combination with our own standard cell libraries and autorouting software. Fault cover analysis and test vector generation were assisted by an Ikos accelerator, together with several in-house tools.

A high-level model of the FPU in the parallel language 'Occam' was developed for design verification at the architecture level, used in conjunction with an augmented version of the Berkeley IEEE-754 test suite. The principal additions to the standard suite concern type conversions and the use of multiple execution units for special operand handling. The same test vectors were used to exercise the transistor-level description of the final design written in the Inmos 'Hylas' hardware description language. Comparative tests were also executed against two commercially-available floating-point processors.

9 Conclusions

An 10 man-year effort has yielded a 25 MFlop floating-point unit of 160k transistor complexity occupying less than 15 mm^2 of silicon, using a triple metal 1μm P-well CMOS process. This remarkable performance density demonstrates the significant advances in both process technology and arithmetic algorithms and architecture which have been made over the past few years.

10 Acknowledgements

The author gratefully acknowledges the enormous effort put into this project by the other members of the T9000 floating-point team, principally John Webster, Tim Johnson, Simon Cottam, and Peter Cumming. Thanks are also due to many other T9000 designers who contributed in various ways – in particular to Dave Halliwell for handling the testability issues, Jean-Luc Bauer for doing most of the pipeline interfacing work, and Richard Forsyth and Clive Dyson for holding the compute side of the T9000 project together.

References

[Birm88] **M. Birman** *et al. Design of a High-Speed Arithmetic Datapath.* ICCD, New York, pp 214-216, Oct. 88.

[Hays85] **W. P. Hays** *et al. A 32-bit VLSI Digital Signal Processor.* IEEE J., SC-20(5):998-1004, Oct. 85.

[Hoke90] **E. Hokenek, R. K. Montoye** *Leading-Zero Anticipator (LZA) in the IBM RISC System/6000 Floating-Point Execution Unit.* IBM J. R.& D., 34(1):71-77, Jan. 90.

[Inmo91] **Inmos Limited** *The T9000 Transputer Products Overview Manual.* Inmos Limited, 1000 Aztec West, Almondsbury, UK.

[Lehm61] **M. Lehman, N. Burla** *Skip Techniques for High-Speed Carry-Propagation in Binary Arithmetic Units.* IRE Trans., EC-10(4):691-698, 1961.

[MacS61] **O. L. MacSorley** *High-Speed Arithmetic in Binary Computers.* Proc. IRE, 49:67-91, Jan. 61.

[Maje85] **S. Majerski** *Square-Root Algorithms for High-Speed Digital Circuits.* IEEE Trans., C-34(8):724-733, Aug. 85.

[Nadl56] **M. Nadler** *A High-Speed Electronic Arithmetic Unit for Automatic Computing Machines.* Acta Technica (Prague), 6:464-478, 1956.

[Robe58] **J. E. Robertson** *A New Class of Digital Division Methods.* IRE Trans., EC-7(9):218-222, Sept. 58.

[Rowe88] **C. Rowen, M. Johnson, P. Ries** *The MIPS R3010 Floating-Point Coprocessor.* IEEE Micro, pp. 53-62, June 88.