VLSI DESIGN
2001, Vol. 12, No. 3, pp. 333–348
Reprints available directly from the publisher
Photocopying permitted by license only

CADRE: A Low-power, Low-EMI DSP Architecture for Digital Mobile Phones

MIKE LEWIS* and LINDA BRACKENBURY†

AMULET Group, Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, United Kingdom

(Received 20 June 2000; In final form 3 August 2000)

Current mobile phone applications demand high performance from the DSP, and future generations are likely to require even greater throughput. However, it is important to balance these processing demands against the requirement of low power consumption for extended battery lifetime. A novel low-power digital signal processor (DSP) architecture CADRE (Configurable Asynchronous DSP for Reduced Energy) addresses these requirements through a multi-level power reduction strategy. A parallel architecture and configurable compressed instruction set meets the throughput requirements without excessive program memory bandwidth, while a large register file reduces the cost of data accesses. Sign-magnitude representation is used for data, to reduce switching activity within the datapath. Asynchronous design gives fine-grained activity control without the complexities of clock gating, and gives low electromagnetic interference. Finally, the operational model of the target application allows for a reduced interrupt structure, simplifying processor design by avoiding the need for exact exceptions.

Keywords: VLSI; Low-power; Mobile; DSP; GSM; Asynchronous

1. INTRODUCTION

The market for mobile communication devices, particularly mobile phones, has grown at a phenomenal rate and is still expanding rapidly. Part of this rapid growth can be attributed to the decrease in price of the handsets, to the point that mobile network operators are able to actually give away handsets, recouping the cost in the revenue gained from contract fees and call costs. The low unit price makes this market fiercely competitive, and manufacturers are continually vying with one another to find new features which give their phones an edge over those of their rivals. However, one factor dominates when distinguishing between phones: the size and weight of the handset. This is largely controlled by the trade-off between battery size and battery lifetime, which

*Corresponding author. Tel.: +44 161 275 3531, e-mail: lewism@cs.man.ac.uk
†e-mail: lbrackenbury@cs.man.ac.uk

itself is controlled by the power consumption of the circuitry within the phone. The requirement for extended battery lifetime with reduced battery size makes mobile phones a key application for low power VLSI design.

Modern cellphones are based on digital communication protocols, such as the globally accepted GSM protocol. These require complex control and signal processing functions, with the phones performing filtering, error correction, speech compression/decompression, protocol management and, increasingly, additional functions such as voice recognition and multimedia capabilities. This processing load means that the digital components of the phone consume a significant proportion of the total power. The bulk of the remaining power is used for radio transmission, which is fixed by the distance to the basestation and the required signal-to-noise ratio. The required power will decrease as the number of subscribers increases and cell sizes decrease to compensate. Also, mobile communication devices will increasingly be used as part of local wireless communication networks such as the Bluetooth wireless LAN protocol [1], where the transmitted power is very low. It is therefore clear that the key to reduced power consumption for both current and future generations of mobile phone must be found in the digital subsystems.

These digital subsystems are typically based on the combination of a microprocessor coupled by an on-chip bus to a digital signal processor core. The microprocessor is responsible for control and user-interface tasks, while the DSP handles the intensive numerical calculations.

An example of a current part for GSM systems is the GEM301 baseband processor [2] produced by our collaborator Mitel Semiconductor, which contains an ARM7 microprocessor coupled to an OAK DSP core as shown in Figure 1. A study of the literature for this product revealed that, within the digital subsystem, the DSP is responsible for approximately 65% of the total power consumption when engaged in a call using the GSM half-rate speech compression/decompression algorithm

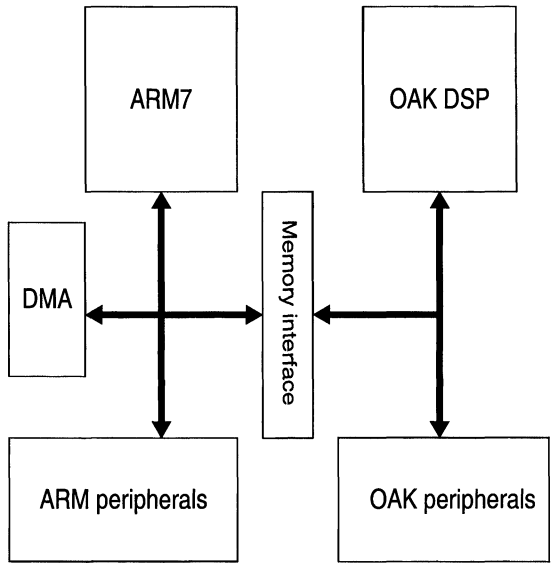


FIGURE 1 Structure of GEM301 baseband processor.

or *codec* (the term half-rate deriving from the level of compression, such that two speech channels can fit into a single transmission timeslot).

It can be expected that this proportion of the total power consumption will increase in future generations of GSM chipsets as the complexity of coding algorithms increases. For this reason, it would appear that the most benefit can be gained by reducing the power consumed by the DSP core. To address this area, the CADRE asynchronous digital signal processor is being developed to meet the requirements for performance, power consumption and EMI. Based on current throughput, it is expected that the next generation of mobile phone chipsets will require a throughput of greater than 100MIPS from the DSP. We have chosen a target performance of 160MIPS for the new design, which is intended to meet the requirements for this application comfortably and represents an approximately fourfold increase in speed over the OAK DSP in the GEM301 chip. CADRE uses fixed point arithmetic with a precision of 16 bits and an accumulator precision of 40 bits. The GSM standard specification requires only 32 bits of precision for accumulated results, but an additional 8-bit guard portion for the accumulators

simplifies program design by allowing up to 128 summations before overflow can occur.

2. SOURCES OF POWER CONSUMPTION

The power consumed by CMOS circuitry is dominated by that due to switching activity [3]. The dynamic power dissipated at any node can be calculated by the well-known equation $P = f((1/2)CV^2)$, where C is the node capacitance, V is the supply voltage and f is the rate of switching at that node. This expression leads to three main strategies for reducing the power consumption.

2.1. Reducing the Supply Voltage

The first method of reducing power consumption is to reduce the supply voltage, which has the greatest effect due to the quadratic term. However, this also has the effect of reducing the operating speed of the circuits and reducing noise margins. These effects become extreme as the supply voltage approaches the threshold voltages of the MOS transistors and, in addition, the static leakage current increases. Leakage current is a serious issue for battery powered systems where large amounts of time are spent in an idle state.

In applications where demand is variable, varying the power supply so that the data is ready 'just in time' can be a very effective means of minimising the average power [4], although changing the operating speed is complex for clocked systems. Where demand is fixed, the supply voltage should be set as low as possible while still achieving the required performance.

2.2. Reducing Switched Capacitance

The second method of reducing power consumption is to reduce the overall switched capacitance within the circuit. This can be done to some extent by appropriate sizing of transistors within the circuit. However, as deep sub-micron technologies are adopted, an increasingly large proportion of the switched capacitance in a circuit comes from parasitic wire capacitances. The impact of this can be minimised by localising the transfer of data and avoiding long paths across a circuit wherever possible, by exploiting local storage and by appropriate algorithmic transformations [6, 7].

2.3. Reducing Switching Activity

The final method of reducing power consumption is to reduce the number of transitions at each node within the circuit. This can be done through algorithmic transformations exploiting correlations between data [3, 8, 9], by localising transfer of data [6, 7] and by eliminating redundant activity. There are a number of methods for eliminating redundant activity, such as appropriate choice of number representation [3], disabling unused subword portions of the datapath [10], and preventing the creation and propagation of intermediate values or 'glitches' [3, 11].

One potential source of redundant activity in synchronous systems is the clock signal itself, which is subject to high capacitance as it is distributed throughout the system. Clock gating to idle subsytems, or deactivating the clock altogether during idle periods, can mitigate the waste of energy to some extent, but this introduces considerable complexity in the system design. The complexity is introduced both from the need to use either software or hardware to identify idle components, and from the delay incurred in restarting a high speed clock generator or PLL clock buffer before the output stabilises.

As an alternative to clocked designs, an asynchronous (self-timed) design can be used. In an asynchronous system, operations are managed by means of local handshakes between adjacent circuits. A comparison between a conventional synchronous pipeline and an asynchronous *micropipeline* [12] is shown in Figure 2. In the synchronous system in Figure 2a, the clock controls the passage of data along the pipeline. In contrast, the

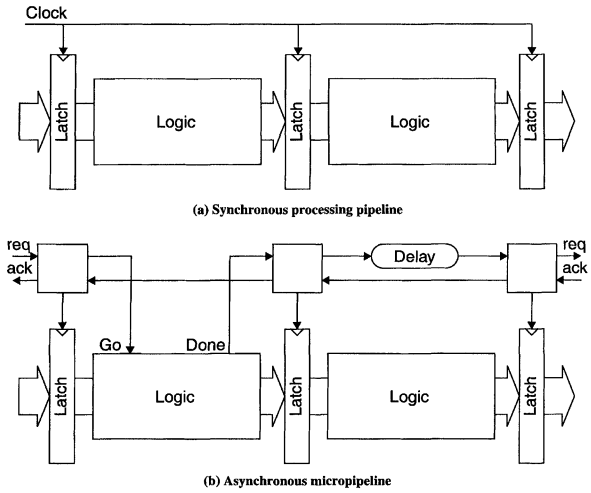


FIGURE 2 Synchronous and asynchronous pipelines.

passage of data through the asynchronous system of Figure 2b is controlled by a succession of independent *handshakes* between adjacent circuit elements, managed by the associated latch control circuits. When an element has valid data to pass on, it asserts the *request* signal. The subsequent stage indicates that it has captured the new data by asserting the *acknowledge* signal. All synchronisation activity occurs locally between adjacent stages, minimising the wiring load. The entire system can be stopped simply and rapidly by stalling the handshake process at any stage of the pipeline, and processing can resume immediately upon that stage resuming activity. This behaviour is an inherent property of an asynchronous system, and means that even the smallest period of inactivity can be exploited to reduce power consumption, without the need to stop and restart a clock.

Processing delay in the asynchronous pipeline is managed either by explicit completion signals from the datapath or by a matched delay in the request path, as shown in the figure. Whichever method is used, the delay is dependent only on the local worst-case delay as opposed to the global worst-case delay as required for synchronous systems. This means that, for example, simpler operations in a processor can complete more quickly than more complex ones.

For system-on-chip (SOC) designs, asynchronous modules offer an attractive element of composability as the interfaces to such modules are defined precisely, without reference to a global timing reference. The degree that the interface is constrained depends on the particular asynchronous design style used, ranging from the fully delay-insensitive style used by Theseus Logic Inc. [5], which is guaranteed correct by construction at the expense of using two wires for each signal, to the bundled data design style employed in this work where it is assumed that the delay in the request signal is similar to the delay in the associated data bus. This imposes more constraints on the layout, but uses half the number of wires and therefore consumes less power.

A very useful side-effect of asynchronous design is that the circuits cause inherently less electromagnetic emissions than their clocked counterparts. In the clocked circuit, the passage of data and therefore all activity is synchronised to the clock edge. This causes large current spikes at the clock frequency and large quantities of radiation at harmonics of the clock frequency. By contrast, activity in an asynchronous system is distributed in time, meaning that very little harmonic radiation is generated. It is clear that this behaviour is particularly useful in wireless systems.

2.4. Power Consumption in the DSP

The power consumption in an on-chip processing system as described here can be broken down into two main areas. The first main area is the power cost associated with accesses to the program and data memories. This is made up of the power consumed within the RAM units themselves, and the power required to transmit the data across the large capacitance of the system buses. Memory accesses can form the largest component of power consumption in data-dominated applications [6], and a study of the Hitachi HX24E DSP [13] showed that memory accesses caused a significant proportion ($\sim 20\%$) of the total power consumption

even where the activity of the system is not dominated by memory transfers.

The second main area of power consumption comes from the energy dissipated while performing the actual operations on the data within the processor core. This is made up of the energy dissipated by transitions within the datapath associated with the data, and the control overhead required to perform the operations on the data.

3. ARCHITECTURE OF THE NEW DSP

The challenge for CADRE is to meet the required throughput without excessive power consumption. An instruction rate of 160 MHz is not large when compared with current high-performance microprocessors. However, the demands of low power consumption and low electromagnetic interference mean that lower operating speeds are preferred. Meeting the required throughput at a lower operating speed necessitates the use of parallelism, where silicon die area is traded for increased speed. This allows simpler and more energy efficient circuits to be used within each processing element, and for the supply voltage to be reduced for a given throughput. If a simple proportional relationship between supply voltage and speed is assumed, doubling the number of processing elements allows the supply voltage to be halved. This reduces the energy dissipated per operation in each functional unit by four and the total power consumption by two (so-called *architecture driven voltage scaling* [3]). The practical benefit will be less than that predicted by the simple delay model, but is still sufficiently large to be worthwhile when area constraints allow it. Multiple functional units also provide flexibility for the programmer to rearrange operations so as to exploit correlations between data [3, 9, 14].

Silicon die area is rapidly becoming less expensive; indeed, one of the challenges is to make effective use of the vast number of transistors available to the designer [15]. This makes parallelism and replication very attractive. Most new DSP offerings by the major manufacturers incorporate some form of parallelism, such as the LSI Logic Inc. ZSP164xx DSPs [16] with 4-way parallelism or the Texas Instruments TMS320C55x low-power DSPs [17] which feature two multiply-accumulate units and two ALUs.

3.1. Choice of Parallel Architecture

The OAK DSP core in the GEM301 baseband processor maintains a maximum throughput of approximately 40MIPS when engaged in a call using a half-rate codec. This is a uniscalar device, and so to reach the required throughput of 160MIPS we have chosen simply to increase the processing throughput by means of four way parallelism. The choice and layout of the functional units were decided upon by examining a number of key DSP algorithms [18] to see how parallelism could be exploited. To give a starting point for the instruction set, the benchmark algorithms for the Motorola 56000 DSP series [19] were chosen, as the authors have some experience with this range of processors. The chosen algorithms were FIR filters, IIR filters and fast Fourier transforms; the FIR filter and FFT will be illustrated here.

3.1.1. FIR Filter Algorithm

The first algorithm to be considered was the FIR Filter algorithm. This is expressed by the equation $y(n) = \sum_{k=0}^{M-1} c_k x(n-k)$ and there are clearly a number of ways in which this sum of products can be implemented in parallel form. The time-consuming portion of this algorithm is the succession of multiply-accumulate (MAC) operations and so, to speed up execution by a factor of four, it is necessary to have four functional units capable of performing these multiply-accumulate operations.

A simple way of distributing the arithmetic for this algorithm is to have each MAC unit process a quarter of the operations on each pass of the

algorithm, storing the partial sum in a high-precision accumulator within the unit. At the end of the pass, a final summation of the four partial sums is performed. These final sums require additional high-precision communication paths between the functional units to avoid loss of precision, and to perform the sum in the shortest possible time requires two of these pathways. The distribution of operations to the various functional units (Mac A-D) is shown in Table I.

Arithmetic operations are of the form, 'operation src1,src2,dest' where src1 and src2 are 16 or 40 bit values and dest specifies the destination accumulator. Where one of the sources is an accumulator from another functional unit, the notation mac[a−d]:src is used to indicate which functional unit and accumulator is involved. The mpy operation is a 16 × 16 bit multiply, the mac operation is a 16 × 16 bit multiply with the result being added to the destination accumulator, and the add operation is a 40 bit addition. Bold type indicates the operation in the algorithm after which the result is available.

When more than one item of new data is available at a time (such as when processing is block-based) it is possible to optimize the FIR filter algorithm to reduce power consumption, by transforming the algorithm so that more than one new data point is processed on each pass. The use of k accumulators within the functional unit allows switching activity within the multiplier and data accesses to be reduced by a factor k [21]. The transformed sequence of operations is shown in Table II. The benefit of this transformation is that correlations between both the data values and the filter coefficients can be exploited. In the new arrangement, the filter value is held constant at one input of the multiplier over four successive multiplications while successive data values are applied to the other input. This dramatically reduces the amount of switching activity within the multiplier, at the expense of requiring more instructions and more accumulator registers in each functional unit. In the design of CADRE, the trade-off between the possible power savings, size and power consumption of the accumulator bank and number of bits required to select the registers has led to the choice of 4 accumulators per functional unit.

Each functional unit now maintains 4 partial sums, one for each of the passes of the FIR filter algorithm, and these partial sums are again brought together at the end of processing. In this case, 4 high precision pathways between the functional units would be beneficial, but this represents too great an area overhead. Instead, it was noted that the summation of results across the functional units occurs in a pairwise fashion, and so it was decided to group the functional units into two pairs (Mac A and B, Mac C and D) connected by local high precision buses, with all four units connected by a single global high precision bus. As a shorthand, these buses are named LIFU1&2 (Local Interconnect of Functional Units) and GIFU (Global Interconnect of Functional Units). This arrangement, as shown in Figure 3, provides the benefits of having three high precision pathways for most operations, but incurs the area

TABLE I Distribution of operations for sample FIR filter implementation

MAC A	MAC B	MAC C	MAC D
mpy x_n, c_0, a	mpy x_{n-1}, c_1, a	mpy x_{n-2}, c_2, a	mpy x_{n-3}, c_3, a
mac x_{n-4}, c_4, a	mac x_{n-5}, c_5, a	mac x_{n-6}, c_6, a	mac x_{n-7}, c_7, a
⋮	⋮	⋮	⋮
mac x_{n-i}, c_i, a	mac x_{n-i-1}, c_{i+1}, a	mac x_{n-i-2}, c_{i+2}, a	mac x_{n-i-3}, c_{i+3}, a ($i = 4, 8, \ldots$)
⋮	⋮	⋮	⋮
mac x_{n-M+4}, c_{M-4}, a	mac x_{n-M+3}, c_{M-3}, a	mac x_{n-M+2}, c_{M-2}, a	mac x_{n-M+1}, c_{M-1}, a
–	add maca: a, a, a	–	add maca: a, a, a
–	–	–	**add macb : a, a, a**

TABLE II Distribution of operations for transformed block FIR filter algorithm

MAC A	MAC B	MAC C	MAC D
mpy x_n, c_0, a	mpy x_{n-1}, c_1, a	mpy x_{n-2}, c_2, a	mpy x_{n-3}, c_3, a
mpy x_{n-1}, c_0, b	mpy x_{n-2}, c_1, b	mpy x_{n-3}, c_2, b	mpy x_{n-4}, c_3, b
mpy x_{n-2}, c_0, c	mpy x_{n-3}, c_1, c	mpy x_{n-4}, c_2, c	mpy x_{n-5}, c_3, c
mpy x_{n-3}, c_0, d	mpy x_{n-4}, c_1, d	mpy x_{n-5}, c_2, d	mpy x_{n-6}, c_3, d
⋮	⋮	⋮	⋮
mac x_{n-j}, c_j, a	mac x_{n-j-1}, c_{j+1}, a	mac x_{n-j-2}, c_{j+2}, a	mac x_{n-j-3}, c_{j+3}, a
mac x_{n-j-1}, c_j, b	mac x_{n-j-2}, c_{j+1}, b	mac x_{n-j-3}, c_{j+2}, b	mac x_{n-j-4}, c_{j+3}, b
mac x_{n-j-2}, c_j, c	mac x_{n-j-3}, c_{j+1}, c	mac x_{n-j-4}, c_{j+2}, c	mac x_{n-j-5}, c_{j+3}, c
mac x_{n-j-3}, c_j, d	mac x_{n-j-4}, c_{j+1}, d	mac x_{n-j-5}, c_{j+2}, d	mac x_{n-j-6}, c_{j+3}, d (j = 4, 8,...)
⋮	⋮	⋮	⋮
mac x_{n-M+1}, c_{M-4}, a	mac x_{n-M}, c_{M-3}, a	mac x_{n-M-1}, c_{M-2}, a	mac x_{n-M-2}, c_{M-1}, a
mac x_{n-M}, c_{M-4}, b	mac x_{n-M-1}, c_{M-3}, b	mac x_{n-M-2}, c_{M-2}, b	mac x_{n-M-3}, c_{M-1}, b
mac x_{n-M-1}, c_j, c	mac x_{n-M-2}, c_{M-3}, c	mac x_{n-M-3}, c_{M-2}, c	mac x_{n-M-4}, c_{M-1}, c
mac x_{n-M-2}, c_j, d	mac x_{n-M-3}, c_{M-3}, d	mac x_{n-M-4}, c_{M-2}, d	mac x_{n-M-5}, c_{M-1}, d
add macb: a, a, a	add maca: b, b, b	–	add macc: a, a, a
add macd : a, a, a	–	add macd: c, c, c	add macc: b, b, b
add macb: c, c, c	**add macd : b, b, b**	–	add macc: d, d, d
–	add macd: d, d, d	**add maca : c, c, c**	–
–	–	–	**add macb : d, d, d**

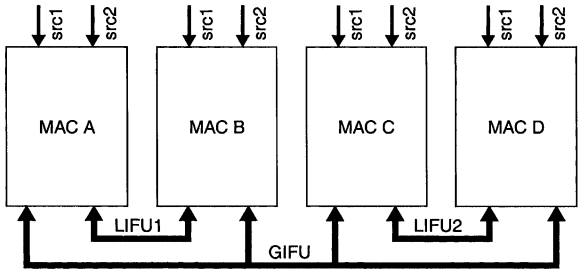


FIGURE 3 Layout of functional units.

expense of only two global pathways. Driving shorter local buses also causes less power consumption. Despite only having three pathways to perform summations over, it is still possible to keep all of the functional units occupied by interleaving the summation of the partial results with the final set of multiplications. Details of this have been omitted from Table II for the sake of clarity.

3.1.2. Fast Fourier Transform

The fast Fourier transform is actually a 'parallelized' form of the discrete Fourier transform described by the equation $X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi k(n/N)}$. The algorithm consists of a series of passes of the 'FFT butterfly' operator across the data. The butterfly operates on two (complex) data values a and b to produce two output data values A and B according to the equations $A = a + W_i b$ and $B = a - W_i b$, where W_i is the value of a complex exponential (the so-called 'twiddle factor'). The calculation of each butterfly requires a complex multiply and two complex additions. In general, the complex multiplication $W_i b$ requires four real multiply operations and two real additions, to calculate $Re(W_i \times b) = Re(W_i) \times Re(b) - Im(W_i) \times Im(b)$ and $Im(W_i \times b) = Im(W_i) \times Re(b) + Re(W_i) \times Im(b)$. Two further complex additions are then required to generate A and B, requiring four real additions in total. However, if the functional units support shifting of one of the operands, to produce a multiplication by a factor of two, then it is possible to avoid two of the final additions by using the following algorithm:

$$
\begin{aligned}
Re(A) &= Re(a) + Re(W_i) \times Re(b) - Im(W_i) \times Im(b) \\
Im(A) &= Im(a) + Im(W_i) \times Re(b) + Re(W_i) \times Im(b) \\
Re(B) &= Re(a) - Re(W_i) \times Re(b) + Im(W_i) \times Im(b) \\
&= 2 \times Re(a) - Re(A) \\
Im(B) &= Im(a) - (Im(W_i) \times Re(b) + Re(W_i) \times Im(b)) \\
&= 2 \times Im(a) - Im(A)
\end{aligned}
$$

TABLE III Distribution of operations for FFT butterfly

MAC A	MAC B	MAC C	MAC D
move a_{1r}, *a*	*move* a_{1i}, *a*	*move* a_{2r}, *a*	*move* a_{2i}, *a*
move a, b	move a, b	move a, b	move a, b
mac w_{1r}, b_{1r}, a	mac w_{1i}, b_{1r}, a	mac w_{2r}, b_{2r}, a	mac w_{2i}, b_{2r}, a
mac $-\mathbf{w_{1i}}$, $\mathbf{b_{1i}}$, **a**	**mac** $-\mathbf{w_{1r}}$, $\mathbf{b_{1i}}$, **a**	**mac** $-\mathbf{w_{2i}}$, $\mathbf{b_{2i}}$, **a**	**mac** $-\mathbf{w_{2r}}$, $\mathbf{b_{2i}}$, **a**
add 2b, **−a**	**add 2b**, **−a**	**add 2b**, **−a**	**add 2b**, **−a**

A natural way of performing these calculations within the functional units is to use them in pairs, to perform the complex operations for two butterflies simultaneously. The mapping of the FFT butterfly is shown in Table III. This mapping requires two write ports to the accumulator bank in each functional unit, so that the moves can take place in parallel with the operations (with read-before-write sequencing being enforced within the functional units). The italicised move operations only require a separate instruction on the first FFT butterfly of each pass, as they can take place in parallel with the final add of the accumulators when a number of butterflies are being performed in succession. A full implementation of this algorithm can perform 4 complex FFT butterflies with 6 parallel instructions, with all of the functional units fully occupied throughout.

3.2. Choice of Number Representation

It is well known that sign-magnitude representation of signed binary numbers can cause less switching activity than two's complement number representation in systems such as DSPs where the data shows correlation between successive values or when low-amplitude signals are being processed [3, 10]. The difference in switching activity is due to activity in the redundant sign bits required to represent small negative numbers in two's complement representation. However, sign-magnitude arithmetic requires somewhat more complexity in the arithmetic circuits, particularly in order to ensure that the result of a subtraction always has a positive mantissa.

In order to investigate this trade-off, models of DSP datapaths using both sign-magnitude and 2's complement arithmetic were written. Experimental studies performed with these models executing a simulated low-pass FIR filter algorithm on speech data showed that the sign-magnitude datapath exhibited significantly less switching activity, between 20%–40% as counted at the module interfaces. The extra complexity to implement sign-magnitude arithmetic is restricted to a minimum-geometry portion of the datapath within the adder, and so has little effect on the power consumption. Sign-magnitude arithmetic has been used within CADRE, as the reduced switching activity due to the data representation affects power consumption throughout the system. This is particularly significant when the large capacitance of system buses to memory is considered [3].

3.3. Supplying Instructions to the Functional Units

Having chosen a parallel structure for the processor, the next challenge is to devise a method of supplying independent instructions to the functional units at a sufficient rate without excessive power consumption. In a general-purpose superscalar microprocessor, this task is often managed by a dedicated scheduling unit which analyses the incoming instruction stream and dispatches independent instructions to the available resources. This approach has been adopted by ZSP Corporation for the ZSP164xx DSPs. However, the scheduling unit is a complex device which consumes significant amounts of power, so for power-critical applications it makes more sense to remove

this task from the processor. Instead, the programmer (or, more often, the compiler) can group independent instructions, in advance, into a single *very long instruction word* which can be read from memory and directly dispatched to the functional units. The VLIW approach is becoming the more common method for managing parallelism in current DSPs. The main drawback with conventional VLIW is that, where dependencies exist, it is necessary to insert NOPs within the instruction word which reduce the code efficiency. This can be tackled to some extent by using variable length instructions, such as the EPIC (Explicitly Parallel Instruction-set Computing) technique [20] at the expense of greater complexity of instruction decoding. Variable length instructions of this type are employed in the Texas Instruments TMS320C55x DSPs. However, in the case of both superscalar and VLIW approaches it is necessary to fetch data from program memory at the full rate demanded by the functional units.

DSP operations tend to be characterised by regular repetition of a number of short, fixed algorithms. It is possible to exploit this characteristic to reduce the quantity of information that needs to be fetched from program memory, thereby reducing power consumption. One possible method would be to cache the incoming instruction stream, to exploit the locality of reference in the memory accesses. However, cache memory consumes a significant amount of power when searching for a hit, particularly when multiway associative caches are used. In addition, it is still necessary to fetch instructions and update the program counter at the full issue rate of the processor or to use a very wide instruction path. Instead, we propose that VLIW encodings for the required instructions can be stored, in advance, in configuration memories internal to the functional units themselves. These stored operations can then be recalled with a single word from program memory, dramatically reducing the amount of information that needs to be fetched. DSPs already exist which make use of configurable instructions, such as the Philips REAL DSP core [22] or the Infineon CARMEL DSP core [23]. However, both of these have a single global configuration memory for the entire core, which is only used for specialised instructions. The proposed scheme differs in that *all* parallel execution is performed using preconfigured instructions. To reduce the distance over which the data needs to travel, and hence the power consumption, the configuration memories are separate and located within each functional unit. Locating the memories within the functional units also increases modularity, and allows any arbitrary type of functional unit to be inserted into the architecture (although to speed design, identical functional units are being used in the prototype). In the current design the configuration memories are RAMs, allowing reconfiguration at any point in execution. For a given application, it may be desirable to turn part of this storage into ROM to encode a few standard algorithms.

3.4. Supplying Data to the Functional Units

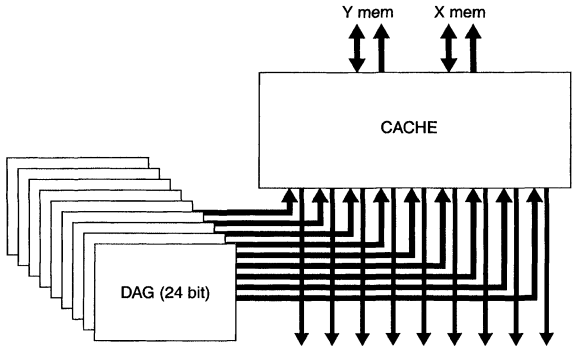
Given a parallel processing structure, and a means of supplying instructions to it, the next design issue is to supply data at a sufficient rate, without excessive power consumption. This is clearly a serious problem, as each functional unit can require two operands per operation and may also need to write data back from the accumulators, giving a total of eight reads and four write accesses per cycle.

CADRE, in common with many other current DSPs, uses a dual Harvard architecture where one program memory and two separate data memories (labelled X and Y) are used. This avoids conflicts between program and data fetches, and many DSP operations map naturally onto dual memory spaces (*e.g.*, data and coefficients for a FIR filter operation).

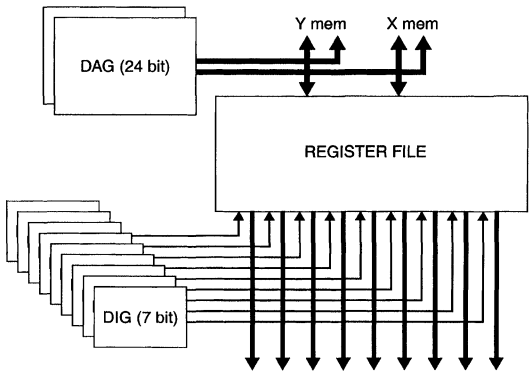
The memory hierarchy principle works well for DSPs, as many algorithms display strong locality of reference. For this reason, a large register file of 256 16-bit words was included in CADRE, segmented into X and Y register banks to match the main memory organisation. The large register

file allows for a high degree of data reuse (allowing, for instance, a complete GSM speech data frame of 160 words to be stored), and a large explicit register file offers a significant advantage over having a cache and fewer registers as is common in traditional DSP architectures.

In the programmer's models of most traditional DSP architectures, as shown in Figure 4a, operands are treated as residing within main memory and are accessed by indirect reference using address registers. These address registers must be wide enough to address the entire data space of the processor, 24 bits in this design. After each operation, it is generally necessary to update these address registers to point to the next data item. The data address generators (DAG) generally provide support for the algorithm being executed,



(a) Conventional DSP architecture



(b) CADRE architecture

FIGURE 4 Reducing address generation and data access cost with a register file.

with circular buffering or bit-reversed addressing, and therefore require complex circuitry. Even if all eight of the fetched data items reside within the cache, there is still a significant power consumption associated with these address register updates (up to eight of them), and this power must be added to that required for the cache lookups.

In the new architecture (Fig. 4b), 24-bit address registers are used only for loading and storing data in bulk between the data register file and main memory. 32-bit ports from the register bank to both X and Y memory allow up to 2 registers from each bank to be transferred simultaneously using a single address register for each bank. Once the data is loaded into the register bank, it can be accessed indirectly by means of 7-bit index registers. The 7-bit data index generators (DIG) give much faster updates at a much lower power cost than their 24-bit counterparts. Also, a multiported register file is significantly less complex and consumes substantially less power than a multiported cache memory, particularly if the cache is an associative design.

3.5. Register Bank Design

The parallel architecture implies that there must be four ports to each of the X and Y register banks, if each functional unit is constrained to using at most one item from each of X and Y data spaces. While many DSP algorithms do segment neatly into X and Y data spaces, it is desirable not to constrain the programmer in this way and to allow them each free choice of access to both the X and Y register banks. Unconstrained choice of register would require eight read ports to each of the X and Y register banks to cater for the worst case of all the functional units requiring both operands from the same bank. This is in addition to the ports required for writebacks from the functional units and loads/stores between memory and the register bank. A large, multiported register bank of this type would require a considerable area, would be slow, and would consume large amounts

of power if directly implemented, due to the high load on the ouput bus of each read port.

Many DSP algorithms exhibit sequential accesses to data which, when parallelized, mean that four sequential data values are required at each iteration. The other common case is that a single data value is required by all functional units (*e.g.*, when all data is to be prescaled by a quantity). It is possible to exploit these access characteristics so as to provide what appears to be a large multiported register file, but with a lower hardware cost, lower power consumption and improved access speed. This is done by segmenting each of the register banks into four sub-banks as shown in Figure 5. The four sub-banks hold sequential registers (*i.e.*, registers 0, 4, 8, ... in sub-bank 0, registers 1, 5, 9, ... in sub-bank 1, *etc.*). The structure of the register bank is shown in Figure 5. Reads and writes to the register bank are managed independently, using very different approaches. The four possible write requests from the functional units, plus the write request of any load operation from memory, arrive asynchronously. Each write request is forwarded to the appropriate sub-bank based on its register selection, where an asynchronous arbiter tree controls access to the write port of each bank.

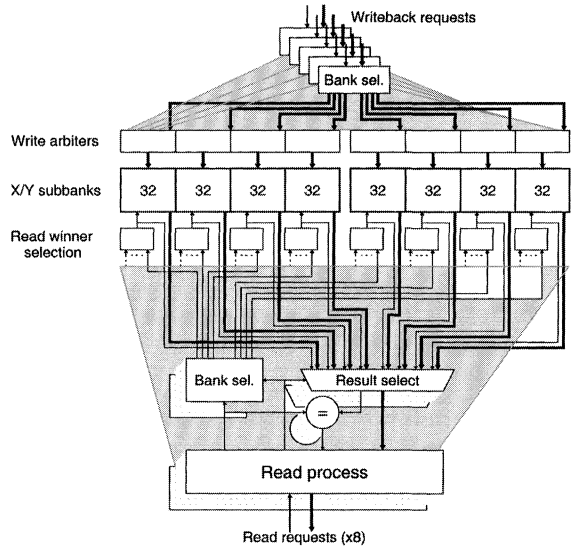


FIGURE 5 Register bank arrangement.

Asynchronous arbitration has the property of being very fast when no contention occurs, but allows requests to be safely serviced in turn if two or more requests arrive simultaneously.

Read requests arrive together from all of the functional units, and are managed by separate read processes synchronized by an overall control unit (not shown). Each active read process requests a particular register from a particular sub-bank. The read request is passed to a simple (non-asynchronous) priority arbiter at the selected sub-bank. Each of these arbiters can accept requests from any number of the read processes, and selects one request as the winner. The winning request is then granted access to read the required register from the sub-bank.

Once the read has completed, the winning register choice passes back to the read processes along with the data, and any process whose read request has been satisfied can capture the data and remove their request. The read cycle can then be repeated until all of the read requests have been satisfied.

This arrangement means that, if the read processes require sequential register numbers, each read process will request data from a different sub-bank, no contention for access will occur, and the read operation will complete in a single cycle. If a number of read processes require access to a single register then all of their requests will arrive at the same read arbiter. This causes contention, but it does not matter which of the processes wins control: as soon as the register has been read, all of the read processes see that the correct register is available and can capture the data and remove their requests.

These strategies for reading and writing mean that the common case operations will execute quickly (within the time of a read or write cycle). Where the programmer has been unable to avoid conflict, a delay of one or more read or write cycles will be incurred. However, this is not a serious penalty as the cycle time for a 32 entry single-port register file should be reasonably fast. The asynchronous nature of the design means that

the variable completion time does not cause any problems elsewhere in the design: the rest of the system can simply wait as necessary.

The proposed scheme requires a considerable amount of wiring resources to route the read and write requests to the appropriate register sub-banks. However, only one of these routes will be active for a given read so the power implications of this extra wiring are not severe, and modern multiple metal layer processes will help to mitigate the area cost. Some extra logic delay is incurred, but the logic depth is similar to that which would be required for address decoding in a 256 word register bank, and the majority of the logic delay is only incurred on the first read cycle.

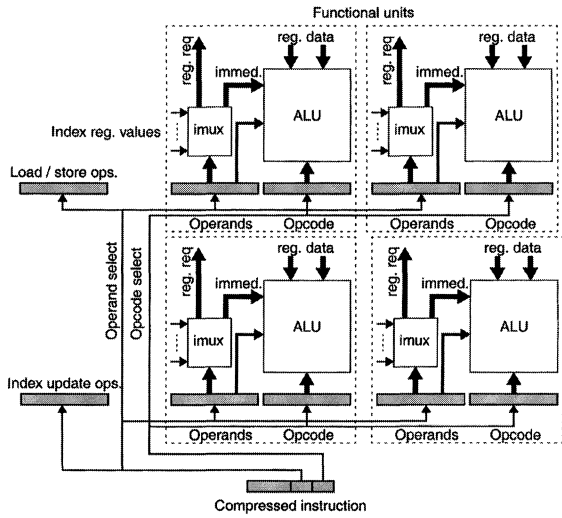


FIGURE 6 Parallel instruction expansion.

3.6. Instruction Encoding and Execution Control

The instructions for the DSP consist of 32 bit words (to match the data width of typical host microprocessors), and are split into two classes: compressed parallel instructions, or all other control and setup instructions. Control and setup instructions are responsible for tasks such as setting up index and address register values and initializing loops, after which the processing work can be done by the compressed parallel instructions without disturbance.

Compressed parallel instructions are described by a 32 bit instruction which maps onto a 320 bit long instruction word stored in 10 separate 128×32-bit configuration memories, as shown in Figure 6. Within each functional unit are two separate 32 bit configuration memories, the opcode and operand memories. The configuration words from opcode memory set up the sequence of operations to be performed by the ALU, which can consist of any combination of:

- An arithmetic operation (with the result being written to the ALU accumulators).
- A parallel move to the ALU accumulators.
- A writeback from the accumulators to the register bank.

Also, the opcode configuration word is responsible for setting up additional functions such as driving of the GIFU/LIFU.

The configuration words from the operand memory specify the source of the data for the operations in the ALU, the destinations for the operations, and the target register of any write-back. The source data for operations are selected by the **imux**, and can be either an indirect reference to the register file (using an index register), a direct reference to the register file, or an immediate value stored in the operand memory.

The remaining two configuration memories are located outside of the functional units. The first of these holds details of how the index registers are to be updated. The second specifies load or store operations to be performed in parallel with the arithmetic operations, and includes details of the address registers to be used to access memory, how the address registers are to be updated, and which register locations are to be used (specified either directly, or indirectly using an index register value).

Compressed parallel instructions are indicated by means of a zero in the most significant bit position, meaning that they can be rapidly identified. The instruction format is shown in Table IV. Each 32 bit parallel instruction contains two 7-bit

TABLE IV Parallel instruction encoding

Bit Position	Function
0–6	Opcode config. memory address
7–13	Operand/load-store/index config. memory address
14	Enable for load/store operations
15	Global enable of writes to accumulators
16	Global enable of writebacks
17	Enable index register updates
18–22	Condition code bits
23–26	Enable operations in functional unit 1–4
27–30	Select conditional opearation in functional unit 1–4
31	0 – indicates a parallel instruction

fields to select the configuration memory entries required for the operation: bits 0–6 select the opcode configuration memory word to be used, while bits 7–13 address the operand memory word to be used, and also which load/store and index update operations are to be performed. Splitting the configuration memory in this way allows the maximum amount of reuse for configuration memory locations; for example, many algorithms may require four parallel multiply-accumulate operations, but may require different source and destination registers for the operations.

To provide even more flexibility in operation, and to reduce configuration memory requirements still further, it is possible to selectively disable components of the stored parallel operation from within the compressed instruction word. This allows each configuration memory location to specify the maximum number of possible concurrent operations, avoiding redundancy of storage, and each algorithm can then select only those parallel components required at the time. Bits 14–17 of the compressed instruction are master enables for the load/store operations, writes to the accumulators, writebacks to the register bank and updates to the index registers; and bits 23–26 enable or disable arithmetic operations in each of the functional units. Arithmetic operations in each of the functional units can also be made conditional, using bits 27–30. Each functional unit maintains an internal condition code register, and the state of this can be tested against the condition code provided in the instruction. Conditional execution reduces the need for branch instructions, which disrupt normal pipeline operation unless expensive branch prediction is used.

3.7. Instruction Buffering

Most DSPs include some form of hardware loop instruction, allowing an algorithm to be executed a fixed number of times without introducing branch dependencies. In the CADRE architecture, this function is managed by a 32 entry instruction buffer which also manages the loop count, meaning that subsequent stages see an entirely flat instruction stream, and supports up to 16 nested loops. The highly compressed instructions mean that even fairly complex DSP kernel routines can fit within this space, and can be executed without the need to access main memory. A study of the Hitachi HX24E DSP [24] showed that power consumption could be reduced by between 25% and 30% by employing a 64 entry instruction buffer: this was sufficiently large for simple algorithms, but not for example a FFT. The compressed instructions for CADRE allow more complex algorithms to be buffered, despite the use of a smaller buffer. The use of an instruction buffer to reduce power consumption has been adopted for the new Texas Instruments TMS320C55x processors.

Apart from the looping behaviour, the buffer acts as a FIFO ring-buffer and to store prefetched instructions, meaning that the next set of instructions can be prepared while either executing the current algorithm or when waiting for new data to

arrive. Despite its complex looping behaviour, the instruction buffer consumes less power in normal operation than many conventional FIFO buffers, due to the asynchronous structure chosen [25]. The combination of the large register file and the compressed instruction buffer can massively reduce the number of memory accesses: for example, it is possible to perform a 64-point complex fast Fourier Transform with only a single pass through both the program and data memories. This represents a reduction of 86% in program memory accesses and PC updates, and a reduction of approximately 74% in data memory accesses, when compared to a conventional architecture where instructions and data are always fetched from memory.

3.8. DSP Pipeline Structure

A block-level representation of the DSP is shown in Figure 7. The fetch stage autonomously fetches instructions from program memory, from where they are passed on to the instruction buffer stage. From here, the instructions pass on to the decode stage, where the most-significant bit is examined to separate them into compressed parallel operations and control/setup instructions. Control and setup instructions are decoded and executed without further pipelining, to minimise setup latency.

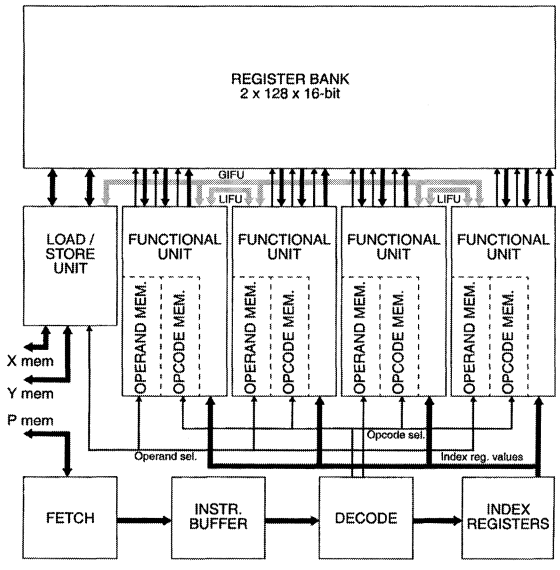


FIGURE 7 Top level architecture of CADRE.

If a compressed parallel instruction is detected, then a read is initiated in the operand configuration memories, index update memory (within the decode block) and load/store memory (within the load/store unit). The next stage of operation is for each functional unit to capture those index register values which are required for indirect references to the data registers, and to update the index register values according to the current instruction.

Once the register sources are known, each functional unit requests the specified data from the register bank. While the registers are being read, the opcode configuration memories are read to set up the operations to be performed in each functional unit, and any parallel load or store operation is initiated.

Once the register and configuration reads have completed, both the data and setup information is valid and the requested arithmetic operations, parallel moves and writebacks can be performed by the functional units.

3.9. Interrupt Support

DSP pipelines are traditionally optimized for repeated execution of small DSP kernel routines, and are less efficient at executing control-oriented code. However, most manufacturers add extra hardware to their designs, such as branch prediction, speculative execution, complex interrupt structures and support for exact exceptions, to improve the control performance and allow the processor to be used as a stand-alone device. CADRE is intended to operate in conjunction with a microprocessor, and so a considerable amount of this hardware can be eliminated by allowing the microprocessor to handle control tasks and for the DSP to operate in the role of a coprocessor. Obviating the requirement for additional hardware, through the proper allocation of tasks between the two devices in this application, contributes to lowering the overall power

consumption. The microprocessor prepares tasks for the DSP, and instructs it to perform them through a simple interrupt structure which also allows for synchronisation with data. Under normal circumstances, the DSP will only respond to an interrupt when halted, *i.e.*, when it has completed the current task. This allows the processor state to be managed without the need for exact exceptions. If necessary, the host microprocessor can issue a non-maskable interrupt, which will cause the DSP to respond immediately at the expense of losing the current processor state. Situations where non-maskable interrupts would be issued are cases when the processor has failed to complete the current task in the time available, or when an urgent event needs to be tended to, and so it is acceptable to discard the data and either repeat the operation later or forget about it.

4. CONCLUSIONS

An overview has been given of a novel architecture for a low-power DSP for mobile phone chipsets. This demonstrates an aggressive multi-level low power design approach, tackling power consumption at all stages from the algorithmic and architectural down to the circuit level. The characteristics of the presented application allow for particularly dramatic reductions in accesses to both the program and data memories, and the type of data being processed allows for correlations in the data to be exploited by the use of sign-magnitude number representation and algorithmic transformations. The architecture is also scalable, as both the number of functional units, size of configuration memories, and the size of the register file can be expanded to suit a particular application, and a mixture of functional units can be used to suit. The design for the processor has been completed at the schematic level, and testing of the complete system is now under way. A set of power consumption results for key benchmarks and algorithms from the GSM protocol is soon to follow and will be reported, after which it is expected to take the design to layout.

Acknowledgements

This work formed part of the EPSRC/MoD Powerpack project, grant number GR/L27930. The authors wish to express their gratitude for this support. All manufacturers' product, trademark and service names used herein are trademarks and service marks of their respective companies.

References

[1] *Official Bluetooth web site*, http://www.bluetooth.com/.
[2] *GEM301 GSM Baseband Processor Preliminary Information*, Mitel Semiconductors, 1997.
[3] Chanrakasan, A. P. and Brodersen, R. W., "Minimizing Power Consumption in Digital CMOS Circuits", *Proc. IEEE*, **83**(4), April, 1995.
[4] Nielsen, L. S., Niessen, C., Sparsø, J. and van Berkel, K., "Low-Power Operation Using Self-Timed Circuits and Adaptive Scaling of the Supply Voltage", *IEEE Transactions on VLSI Systems*, **2**(4), Dec., 1994.
[5] Fant, K. M. and Brandt, S. A. (1996). "NULL Conventional Logic: A Complete and Consistent Logic for Asynchronous Digital Circuit Synthesis", In: *Proc. International Conference on Application-specific Systems, Architectures and Processors*, pp. 261–273.
[6] Catthoor, F. (1999). "Energy-Delay Efficient Data Storage and Transfer Architectures and Methodologies: Current Solutions and Remaining Problems", *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, **21**(2), 258–265.
[7] Danckaert, K., Masselos, K., Catthoor, F., DeMan, H. J. and Goutis, C. (1996). "Strategy for Power-Efficient Design of Parallel Systems", *IEEE Transactions on VLSI Systems*, **7**(2), 219–231.
[8] Ramprasad, S., Shanbhag, N. R. and Hajj, I. N., "Decorrelating (DECOR) Transformations for Low-Power Digital Filters", *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, **46**(6), June, 1999.
[9] Arslan, T., Erdogan, A. T. and Horrocks, D. H., "Low Power Design for DSP: Methodologies and Techniques", *Microelectronics Journal*, **27**(8), 731–744, Nov., 1996.
[10] Nielsen, L. S. and Sparsø, J., "A Low Power Asynchronous Datapath for a FIR Filterbank", In: *Proc. Advanced Research in Asynchronous Circuits and Systems*, IEEE Computer Society Press, March, 1996.
[11] Lewis, M. and Brackenbury, L., "Reconfigurable Latch Controllers for Low Power Asynchronous Circuits", In: *Proc. Advanced Research in Asynchronous Circuits and Systems*, IEEE Computer Society Press, April, 1999.
[12] Sutherland, I. E., "Micropipelines", *Communications of the ACM*, **32**(6), 720–738, June, 1989.
[13] Kojima, H., Gorny, D., Nitta, K. and Sasaki, K., "Power analysis of a programmable DSP for architecture/program optimization", In: *Tech. Dig. IEEE Symp. Low Power Electron.*, pp. 26–27, Oct., 1995.
[14] Erdogan, A. T. and Arslan, T. (1996). "Low Power Multiplication Scheme for FIR Filter Implementation on Single Multiplier CMOS DSP Processors", *Electronics Letters*, **32**(21), 1959–1960.

[15] Bindra, A., "Flexible, modular process packs more than half a billion transistors on a tiny silicon chip", *Electronic Design*, **48**(10), 26, May, 2000.
[16] *An Overview of the ZSP Architecture*, white paper available at http://www.zsp.com/, LSI Logic Inc.
[17] *TMS320C55x DSP Core Technical Documentation*, Texas Instruments Inc.
[18] Higgins, R. J. (1990). *Digital Signal Processing in VLSI*, Prentice Hall Inc., Englewood Cliffs NJ.
[19] *DSP56000 Digital Signal Processor Family Manual*, Motorola Inc., 1995
[20] Schlansker, M. S. and Ramakrishna Rau, B. (2000). "EPIC: Explicitly Parallel Instruction Computing", *COMPUTER*, **33**(2), 37–45.
[21] Sundararajan, V. and Parhi, K. K., "A Novel Multiply Multiple Accumulator Component for Low Power PDSP Design", *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, June, 2000.
[22] Kievits, P., Lambers, E., Moerman, C. and Woudsma, R. (1998). "R.E.A.L. DSP Technology for Telecom Baseband Processing", *Proc. 9th International Conference on Signal Processing Applications and Technology*, Miller Freeman Inc.
[23] *Carmel DSP Core Technical Overview Handbook*, Infineon Technologies, 2000.
[24] Bajwa, R. S., Hiraki, M., Kojima, H., Gorny, D. J., Nitta, K., Shridhar, A., Seki, K. and Sasaki, K., "Instruction Buffering to Reduce Power in Processors for Signal Processing", *IEEE Transactions on VLSI Systems*, **5**(4), 417–423, Dec., 1997.
[25] Lewis, M. and Brackenbury, L. E. M., "An Instruction Buffer for a Low-Power DSP", *Proc. Advanced Research in Asynchronous Circuits and Systems*, pp. 176–186, IEEE Computer Society Press, April, 2000.

Authors' Biographies

Mike Lewis is a Ph.D. student with the AMULET Group in the Department of Computer Science at the University of Manchester UK, and will shortly be submitting his thesis on the application of asynchronous design techniques to low power digital signal processing. Previously he received the M.Eng. degree in electronic and information engineering from the University of Cambridge, in 1997. His research interests include asynchronous processor design, digital signal processing, mobile communications, computer arithmetic and design for low power. He is a student member of the IEEE.

Dr. Linda Brackenbury has been on the academic staff at Manchester University, UK since graduating with a M.Sc. in Computer Science. She worked on the designs of the MU5 and mu5 machines designed at the University before turning her attention to taking designs down to silicon. Her current research interests are in low-power design, asynchronous systems, Viterbi architectures, and algorithms and applications for digital optics.