Sandblaster Low Power DSP

John Glossner[1,2], Kai Chirca[1,2], Michael Schulte[1,3], Haoran Wang[1], Nasir Nasimzada[1], David Har[1], Shenghong Wang[1], A. Joseph Hoane, Jr.[1], Gary Nacer[1], Mayan Moudgill[1], and Stamatis Vassiliadis[2]

[1]Sandbridge Technologies, Inc.
White Plains, NY 10601 USA
jglossner@sandbridgetech.com

[2]Delft University of Technology
Electrical Engineering, Mathematics and Computer Science Department
Delft, The Netherlands

[3]University of Wisconsin
Dept. of ECE
1415 Engineering Drive
Madison, WI, 53706, USA

Abstract

General purpose processors have utilized complex and energy inefficient techniques to accelerate performance. In embedded DSP designs, power constraints have precluded general purpose microarchitectural techniques. Rather than minimize average execution time, embedded DSP processors require the worst case execution time to be minimized. Subsequently, Very Long Instruction Word (VLIW) processors have been employed, but architecturally visible side effects have imposed restrictions on parallelism due to interrupt and latency considerations – particularly if all loads must complete prior to servicing interrupts. In this paper, we present a low power multithreaded interlocked (transparent) microarchitecture capable of parallelizing non-associative DSP arithmetic. We describe specific memory and logic techniques for reducing power dissipation and discuss how multithreading enables low power optimization.

Introduction

The *architecture* of a computer system is the minimal set of properties that determine what programs will run and what results they will produce (1). It is the contract between the programmer and the hardware. Every computer is an interpreter of its *machine language* – that representation of programs that resides in memory and is interpreted (executed) directly by the (host) hardware. The logical organization of a computer's dataflow and controls is called the *implementation or microarchitecture*. The physical structure embodying the implementation is called the *realization*. The architecture describes what happens, while the implementation describes how it is made to happen.

Programs for the same architecture should run unchanged on different implementations. An architectural function is *transparent* if its implementation does not produce any architecturally visible side effects. An example of a non-transparent function is the load delay slot made visible in the architecture due to pipeline effects. Generally, it is desirable to have transparent implementations. Most DSP and VLIW implementations are not transparent and therefore the implementation affects the architecture.

General purpose processors have utilized transparent microarchitectural techniques such as deep pipelines, multiple instruction issue, out-of-order instruction issue, and speculative execution to achieve very high performance. Recently, simultaneous multithreading (SMT), where multiple hardware thread units simultaneously issue multiple instructions per cycle, have been deployed (2). These techniques have produced performance increases at very high complexity and power dissipation costs.

In the embedded DSP community, power dissipation constraints have precluded general purpose microarchitectural techniques. Rather than minimize average execution time, embedded DSP processors require the worst case execution time to be minimized. Subsequently, VLIW or statically scheduled microarchitectures with architecturally visible pipelines are typically employed. Exposing pipelines may pose interrupt latency restrictions, particularly if all memory loads must complete prior to servicing an interrupt. Memory access in embedded systems has traditionally operated at the processor clock frequency to ease the programming burden. This has often restricted the maximum processor clock frequency. Non-associative, saturating DSP arithmetic is also a complicating factor. In addition to prohibiting parallel execution, it is difficult for compilers to optimize.

In this paper, we explore embedded processor design power reduction techniques at all levels. First, we describe explicit architectural techniques that minimize power. Then, we discuss low-power microarchitectural techniques. After this, we present specific realization techniques for low power and describe a concrete example of these techniques used on a 32-bit low-power adder. Finally, we present some results and concluding comments.

Low Power Architecture

A. Compound Instructions

The Sandblaster architecture is a compound instruction set architecture (3). Historically, DSPs have used compound instruction set architectures to conserve instruction space encoding bits. In contrast, VLIW architectures contain full orthogonality, but only encode a single operation per instruction field, such that a single VLIW is composed of multiple instruction fields. This has the disadvantage of requiring many instruction bits to be fetched per cycle, as well as significant write ports for register files. Both these effects contribute heavily to power dissipation.

In the Sandblaster architecture, specific fields within the instruction format may issue multiple sub-operations including data parallel vector operations. Restrictions may apply if a particular operation is chosen. In contrast, a VLIW ISA may allow complete orthogonality of specification and then either in hardware or through no operation (NOP) instructions fills in any unused issue slots.

B. Vector Operations

In addition to compound instructions, the Sandblaster architecture also contains vector operations that perform multiple operations in parallel. As an example, Figure 1 shows a single compound instruction with three compound operations. The first compound operation, lvu, loads the vector register, vr0, with four 16-bit elements and updates the address pointer, r3, to point to the next four elements. The vmulreds operation reads four fixed point (fractional) 16-bit elements from vr0, multiplies each element by itself, saturates each product, adds all four saturated products plus an accumulator register, ac0, with saturation after each addition, and stores the result back in ac0. The vector architecture guarantees Global System for Mobile communication (GSM) semantics (e.g. bit-exact results) even though the arithmetic performed is non-associative (4).

```
L0: lvu %vr0, %r3, 8
||  vmulreds %ac0,%vr0,%vr0,%ac0
||  loop %lc0,L0
```

Figure 1. Compound Instruction for Sum of Squares Inner Loop

C. Simple Instruction Formats

Simple and orthogonal instruction formats are used for all instructions. The type of operation is encoded to allow simple decoding and execution unit control. Multiple operation fields are grouped within the same bit locations. All operand fields within an operation are uniformly placed in the same bit locations whether they are register-based or immediate values. As in VLIW processors, this significantly simplifies the decoding logic.

D. Low Power Idle Instructions

Architecturally, it is possible to turn off an entire processor. All clocks may be disabled or the processor may idle with clocks running. Each hardware thread unit may also be disabled to reduce toggling.

E. Fully Interlocked

Unlike a VLIW processor, our architecture is fully interlocked and transparent. In addition to the benefit of code compatibility, this ensures that many admissible and application-dependent implementations may be derived from the same basic architecture.

Low Power Microarchitecture

A. Multithreading

Figure 2 shows the microarchitecture of the Sandblaster processor. In a multithreaded processor, all threads of execution operate simultaneously. An important point is that multiple copies (e.g. banks and/or modules) of memory are available for each thread to access. The Sandblaster architecture supports multiple concurrent program execution by the use of hardware thread units (called contexts). The architecture supports up to eight concurrent hardware contexts. The architecture also supports multiple operations being issued from each context. The Sandblaster processor uses a unique form of multithreading called Token Triggered Threading (T^3), which consumes much less power than simultaneous multithreading (SMT).

As shown in Figure 3, T^3 is a unique form of multithreading where each hardware context is allowed to simultaneously execute an instruction, but only one context may issue an instruction on a cycle boundary. This constraint is also imposed on round robin threading. What distinguishes T^3 threading is that each clock cycle a token indicates the subsequent context that is to execute. Tokens may be sequential (e.g. round-robin), even/odd, or based on other communications patterns. Compared to SMT, T^3 has much less hardware complexity and power dissipation, since the method for selecting threads is simplified, only a single compound instruction issues each clock cycle, and most dependency checking and bypass hardware is not needed.

B. Decoupled Logic and Memory

As technology improves, processors are capable of executing at very fast cycle times. Current state-of-the-art performance for 130nm technologies can produce processors faster than 3GHz. Unfortunately, current high-performance processors consume significant power. If power-performance curves are considered for both memory and logic within a technology, there is a region in which you get approximately linear increase in power for linear increase in performance. Above a specific threshold, there is an exponential increase in power for a linear increase in performance. Even more significant, memory and logic do not have the same threshold.

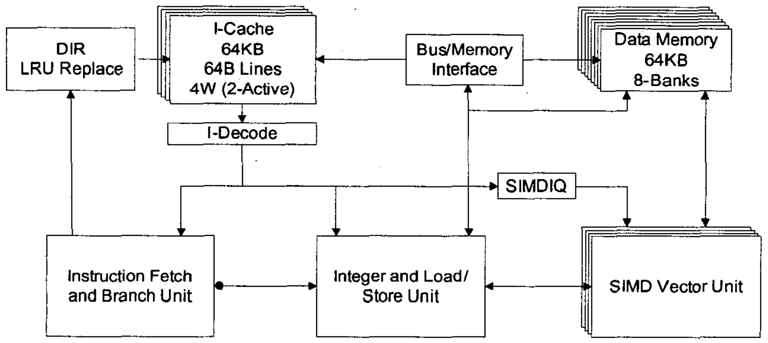

Figure 2. Sandblaster Microarchitecture

$$\begin{tikzcd} T0 \arrow[r] & T3 \arrow[r] & T2 \arrow[r] & T1 \arrow[r] & T6 \arrow[r] & T5 \arrow[r] & T4 \arrow[r] & T7 \arrow[lllllll, bend left] \end{tikzcd}$$

Figure 3. Token Triggered Multithreading, Even/Odd Sequence

For 130nm technology, the logic power-performance curve may be in the linear range until approximately 600MHz. Unfortunately, memory power-performance curves are at best linear to about 300MHz. This presents a dilemma as to whether to optimize for performance or power. Fortunately, multithreading alleviates the power-performance trade-off. The Sandblaster implementation of multithreading allows the processor cycle time to be decoupled from the memory access time. This allows both logic and memory to operate in the linear region, thereby significantly reducing power dissipation. The decoupled execution does not induce pipeline stalls due to the unique pipeline design.

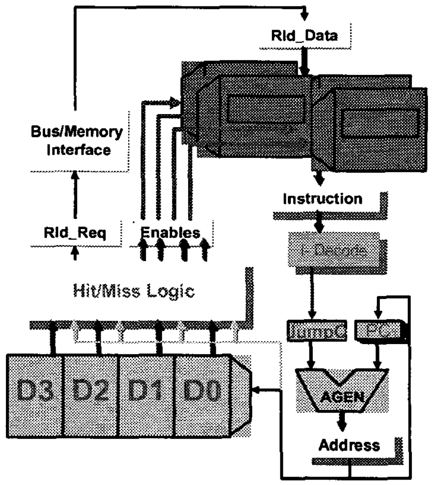

Figure 4. Cache Memory Design

B. Caches

An Instruction Cache Unit (ICU) stores instructions to be fetched for each thread unit. We use associative caches to reduce the likelihood of one thread evicting another thread's active program. In our implementation, shown in Figure 4, a Thread Identifier register (not shown) is used to select whether the line from the left or right bank is evicted. This effectively reduces the complexity of the line selection. In a 4-way set associative cache, only one additional LRU bit is needed to select which of the two lines from the left or right bank should be evicted. This approach gives a complexity of *n(n-2)/8* LRU bits for a general n-way set-associative cache. This method of using thread information and banked memory access significantly reduces the complexity of the cache logic.

C. Pipeline

The pipeline for one particular implementation of the Sandblaster DSP is shown in Figure 5. The execution pipelines are different for various functions. The Load/Store (Ld/St) pipeline is shown to have 9 stages. It is assumed that the instruction is already in the cache. The first stage decodes the instruction. This is followed by a read from the General Purpose Register file. The next stage generates the address to perform the Load or Store. Five cycles are used to access data memory. Finally, the result for a Load instruction is written back (WB) to the referenced register file location. Once an instruction from a particular context enters the pipeline, it runs to completion. It is also guaranteed to write back its result before the next instruction issuing from the same thread tries to use the result.

	0	1	2	3	4	5	6	7	8
Ld/St	Inst Dec	RF Read	Agen	Xfer	Int. Ext	Mem 1	Mem 2	Mem 3	WB
ALU	Inst Dec	Wait	RF Read	Exec1	Exec2	Xfer	WB		
I_Mul	Inst Dec	Wait	RF Read	Exec1	Exec2	Exec3	Xfer	WB	
V_Mul	Inst Dec	RF Read	MPY1	MPY2	Add1	Add2	Xfer	WB	

Figure 5. Processor Pipeline

Similarly, there are multiple (variable) stages for other execution pipelines. The integer unit has three execute stages for multiplication (I_MUL) and two execute stages for addition (ALU). The Vector unit has four execute stages - two for multiplication and two for addition.

D. Interlock Checking Hardware

Most interlocked architectures require significant interlock checking hardware and bypass logic for both correctness and performance reasons. Multithreading mitigates this effect. With the carefully designed pipeline shown in Figure 5, there is only one interlock that must actually be checked for in hardware – a long memory load or store. All other operations are guaranteed to complete prior to the same thread issuing a new instruction. This completely removes the power consuming interlock checks associated with most interlocked architectures.

Low Power Logic Design

A. Single Write Port Register Files

Having multithreading to cover the latency associated with long pipeline implementations allows the use of single write-port register files even though more than one write may occur within an instruction cycle. An important point is that the Write Back stages are staggered. This allows a single write port to be implemented but provides the same functionality as multiple write ports.

An example is loading the integer register file while performing an integer multiply. From the processor pipeline shown in Figure 5, it is apparent that the reads and writes from the register file are staggered in time. In addition, separate architected register spaces for Vector, Integer, and Accumulate operations help to reduce the number of ports. A VLIW implementation of the instruction shown in Figure 1 may take up to 10 read ports and 5 write ports for sustained single cycle throughput. Comparatively, our solution requires at most two read ports and one write port per register file.

B. Banked Register Files

Token triggered threading, which follows a permutation of even and odd thread issue policies, and the pipeline implementation shown in Figure 5 enable the use of banked register files. With this strategy, an even thread may accesses one register file bank, while an odd thread accesses the other register file bank. This effectively doubles the bandwidth of the register file, without increasing the number of ports.

C. Single Ported Memories

The same characteristics that allow banked register file operation also enable the use of single ported level-1 (L1) memories that may be banked and run at half the processor clock frequency. Since decoupled memories are highly desirable to reduce power, this provides significant overall savings.

D. Minimal Control Signals

A combination of architectural and microarchitectural techniques allows the processor to be implemented with very few control signals. Since control signals often propagate to many units they become not only a source of bugs, but also may dissipate significant power.

E. Clock Gating

Because the architecture is modular and the pipeline is deep, there is time to compute which functional units will be active for a particular thread. If a functional unit is not active in a given cycle, the clocks to that unit may be disabled. As an example, if there are no vector operations on a given cycle, the vector unit is disabled. Even within a unit it is possible to disable the clocks. For example, when performing a vector addition, the multipliers in the vector processing unit are disabled.

5. Low Power Circuit Design

The average power consumption in a CMOS circuit can be modeled as

$$P_{avg} = \alpha C V_{dd}^2 f + V_{dd} I_{mean} + V_{dd} I_{leak} \quad (1)$$

where α is the average gate switching activity, C is the total capacitance seen by the gates' outputs, V_{dd} is the supply voltage, f is the circuit's operating frequency, I_{mean}, is the average current drawn during input transition, and I_{leak}, is the average leakage current. The first term, $\alpha C\ V_{dd}{}^2 f$, which represents the dynamic switching power consumed by charging and discharging the capacitive load on the gates' outputs, often dominates power consumption in high-speed microprocessors

(5). The second term, $V_{dd} I_{mean}$, which represents the dynamic power due to short-circuit current flowing when both the PMOS and NMOS transistors conduct during input signal transitions, typically contributes 10% to 20% of the overall dynamic power (6). The third term, $V_{dd} I_{leak}$, represents the power consumed due to leakage current and occurs even in devices that are not switching. Consequently, for systems that are frequently in standby mode, the leakage power may be a dominate factor in determining the overall battery life. Since the leakage power increases exponentially with a linear decrease in device threshold voltage, leakage power is also a concern in systems that use power supply voltage scaling to reduce power.

A. Low Voltage Operation

Since the dynamic switching power, $\alpha C V_{dd}^2 f$, is proportional to the supply voltage squared, an effective technique for reducing power consumption is to use a lower supply voltage. Unfortunately, however, decreasing the supply voltage also decreases the maximum operating frequency. To achieve high-performance with a low supply voltage, our arithmetic circuits are heavily pipelined. For example, our multiply-accumulate unit uses four pipeline stages. Our unique form of multithreading helps mask long pipeline latencies, so that high-performance is achieved.

B. Minimum Dimension Transistors

Minimum dimension transistors help to further reduce power consumption, since they reduce circuit capacitance (7). Throughout the processor, we use minimum dimension transistors, unless other considerations preclude their use. For example, transistors that are on critical delay paths often need to have larger dimensions to reduce delay (8). Larger dimension transistors are also used to drive nodes with high fan-out and to balance circuit delays.

C. Delay Balancing

Gates with unbalanced input delays can experience glitches, which increase dynamic switching power and dynamic short circuit power (9). To reduce glitches, we balance gate input delays in our circuits through a combination of gate-level delay balancing techniques (i.e., designing the circuits so that inputs to a particular gate go through roughly the same number of logic levels) and judicious transistor sizing. Glitches are further reduced by having a relatively small number of logic levels between pipeline registers.

D. Logic Combining and Input Ordering

Dynamic and static power consumption is also reduced by utilizing a variety of specially designed complex logic cells. Our circuits include efficient complex logic cells, such as 3-input AndOrInvert (AOI), 3-input OrAndInvert (OAI), half adder, and full adder cells. Providing a wide variety of complex gates with different drive strengths, functionality, and optionally inverted inputs gives circuit designers and synthesis tools greater flexibility to optimize for power consumption. Keeping nodes with a high probability of switching inside of complex gates and reordering the inputs to complex gates can help further reduce power. In general, inputs that are likely to be off are placed closer to gate output nodes, while inputs that are likely to be on are placed closer to the supply voltage (10).

E. Low Power Adder Example

As an example of our low power circuit design techniques, we present the design and implementation of a low-power, high-performance 32-bit carry skip adder. As demonstrated in (11), carry skip adders dissipate less power than carry lookahead, conditional sum, carry select, and parallel prefix adders due to their low transistor counts and short wire lengths. Since carry skip adders have O(n) area and O($n^{1/2}$) delay, where n is the operand size in bits, they provide a nice tradeoff in terms of area and delay, along with simple and regular layout (12).

Figure 6 shows a block diagram of our 32-bit carry skip adder. In this design, four carry skip adder blocks are cascaded together to form a single 32-bit adder. The CS4F block represents a 4-bit carry skip adder, the CS6 block represents a 6-bit carry skip adder, etc. The left-most arrow on each block represents the carry out of the current block. It is connected to the carry in of the subsequent carry skip block. Inverting carry logic is used throughout the design to reduce delay and power consumption. To achieve balanced delays and reduce glitches, input bits are grouped unevenly in the carry chain and the dimensions of transistors along the critical delay path are sized appropriately. For the least significant bits, fast carry generation is more important than sum generation, since the sum bits are not on the critical delay path. To match the propagation delay of each carry in the carry chain, block P and G logic combines a small numbers of input bits, which means that block sizes are small. A structure similar to a ripple carry adder is used for the least significant bits because the ripple carry adder has good performance and power consumption when there are only a few adder input bits. The logic depth, however, accumulates to make larger block sizes advantageous at the more significant end, except for the last block, CS4L, where fast generation of the sum bits is also required.

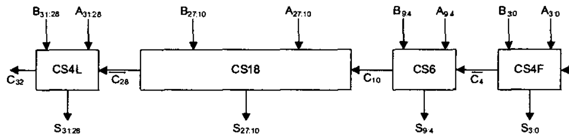

Figure 6. Block Diagram of 32-bit Low Power Adder

Figure 7 shows a block diagram of the first 4-bit carry skip adder (CS4F). The delays for signals that affect the critical path delay are shown in parenthesis next to the signal's name. For example, $P_{3:2}$ (2) indicates that the block propagate signal, $P_{3:2}$, is available after two complex gate delays. The LSB addition is implemented using a standard full adder (FA). Since the sum generation is not on the critical path here, minimal size devices are used to reduce power consumption. This FA takes one complex gate delay to generate the inverted carry out, $\overline{C_1}$. In the second FA, P and G logic operates in parallel with the carry generation in the first FA to compute inverted generate and propagate signals, $\overline{G_1}$ and $\overline{P_1}$, after one logic level. These signals are combined with $\overline{C_1}$ using a 3-input OAI cell to produce C_2 after one more complex gate delay. By matching the path lengths on all the inputs of the OAI gate, glitches do not occur on its output. This assumes the NAND, NOR and OAI gates have the same delays, which is done by properly sizing the transistors of the logic gates.

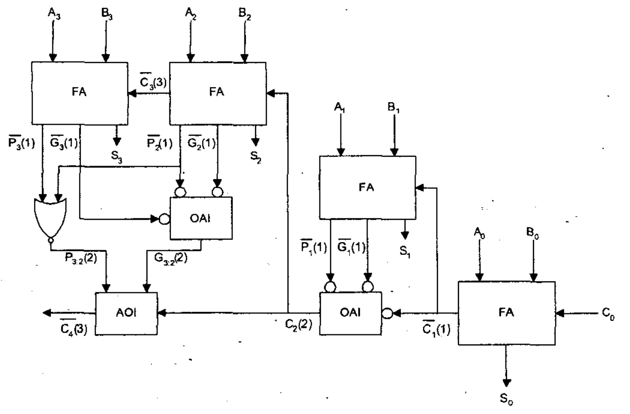

Figure 7. Block Diagram of CS4F block

The next block is a 2-bit adder with inputs $A_{3:2}$ and $B_{3:2}$, and carry in signal C_2. To match the two units of propagation delay on C_2, two generate and propagate signals are combined to form $G_{3:2}$ and $P_{3:2}$. Since all the inputs are non-inverted, an AOI cell is used to generate the inverted carry, $\overline{C_4}$. As shown in Figure 7, $\overline{C_4}$ is available after three complex gate delays and the inputs used to compute $\overline{C_4}$ are balanced.

Similar delay balancing, logic optimization, and transistor sizing techniques are used in the remaining three carry skip adder blocks; CS6, CS18, and CS4L. The CS6 block internally uses two 3-bit carry skip adders and the CS18 block uses six 3-bit carry-skip adders to balance delays along the carry chain. Although each adder block internally uses carry-skip logic, the carry out of each block uses highly optimized carry-lookahead logic to further decrease delay. In the CS4L block, a combination of carry lookahead, carry skip, and carry select logic is used to quickly produce the sum bits. With these techniques, the 32-bit carry-skip adder produces its output in just seven complex gate delays. Compared to other high-speed adder implementations, our adder decreases power consumption by reducing the number of logic levels and glitches, while using minimum dimension transistors off the critical delay paths. Further details on our adder design are presented in (13).

Results

A. 32-bit Low Power Adder

Our 32-bit adder was implemented for different technologies and process corners. All input delay balancing along the carry chain was implemented in layout. To also balance wire lengths, the carry chain is put close to the block P and G logic. Full SPICE simulation with wire load estimation was performed on the circuit. Table I shows performance results for 130nm and 90nm technologies for Slow (V_{dd} – 10%, 125C), Typical (V_{dd}, 25C), and Fast (V_{dd} + 10%, -40C) process corners. Table II shows average power dissipation results at 600Mhz for Cold (V_{dd} – 10%, -40C), Typical (V_{dd}, 25C), and Hot (V_{dd} + 10%, 125C) process corners. Going from 130nm to 90nm roughly doubles the adder's performance and decreases the adder's typical power at 600Mhz by roughly a factor of 2.4.

TABLE I. ADDER PERFORMANCE RESULTS

Process	Slow	Typical	Fast
130nm,V_{dd}=1.2V	0.675GHz	1.089GHz	1.625GHz
90nm,V_{dd}=1.0V	1.201GHz	1.997GHz	3.102GHz

TABLE II. ADDER POWER RESULTS AT 600MHz

Process	Cold	Typical	Hot
130nm,V_{dd}=1.2V	0.540mW	0.786mW	1.224mW
90nm,V_{dd}=1.0V	0.222mW	0.324mW	0.726mW

B. Processor

Figure 8 shows the power-performance characteristics of a number of modern digital signal processors. Performance is characterized as Millions of Mulitply-Accumulates per second (MMAC) plotted on a log scale. Power is characterized as milliWatts also plotted on a log scale. Normalizing lines are drawn at 1 MMAC/mW through 50 MMACs/mW. Not shown, but characteristic of nearly all previous generation DSP processors, is that the power performance metric is below 1 MMAC/mW. Newer DSPs such as Starcore SC110 and SC140, ADI Blackfin, and TI C55x have much better power efficiency. The Sandblaster core does not have significantly better power efficiency than modern cores. However, through the use of low power design techniques, the architecture provides a higher peak performance at the same power efficiency. This is important for executing baseband processing in software.

Table III, column 2 shows a breakdown of power dissipation of an RTL version of our processor that contains all low power techniques, except custom circuits and clock gating. The power estimate was computed using Sequence Design's PowerTheater. An FIR filter was chosen for execution as a worst case internal logic toggle parameter. It can be seen that the Bus Interface Unit (BIU), Clocks (CLK), Instruction Memory (IMEM), Directory Logic (DIR), Data Memory (DMEM), and Data Memory Unit (DMU) consume reasonably low percentages of power. The Branch Unit (BRU), Directory Memory (DIR MEM), Load / Integer Unit (LIU) and the Vector Processing Unit (VPU) consume the majority of power.

As shown in Table III, column 3, when clock gating is implemented to turn off non-operational units it becomes evident that the majority of power is consumed by the BRU, DIR MEM, and even more noticeably the VPU. In 90nm technology, as shown in Table III, column 4, it is evident that the BRU and VPU are key targets for custom circuit techniques. Clock gating and scaling from 130nm to 90nm technology each reduce average power dissipation by roughly a factor of two.

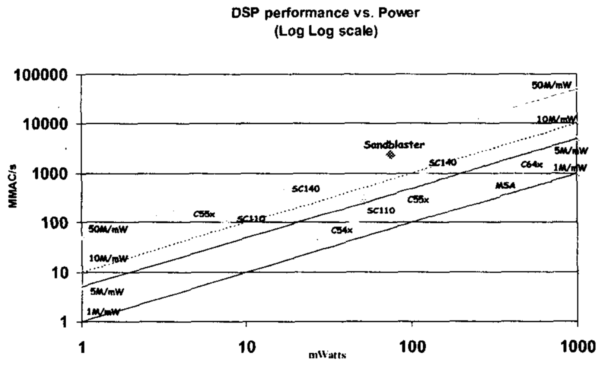

Figure 8. DSP Core Power-Performance Characteristics

TABLE III. PROCESSOR POWER PERCENTAGE RESULTS

	130nm non-gated	130nm gated	90nm gated
BIU	7%	~0%	~0%
BRU	12%	15%	21%
LIU	11%	8%	9%
VPU	26%	33%	34%
DMU	7%	4%	1%
DMEM	5%	6%	5%
DIR	6%	8%	6%
DIR MEM	12%	15%	6%
IMEM	10%	4%	4%
CLK	4%	7%	13%

Conclusions

We have presented design techniques for minimizing power dissipation in embedded DSP designs. A key difference between general purpose designs and embedded designs is a focus on deterministic behavior and minimizing the worst case execution time. In our design we use one non-deterministic technique - instruction caches. The programming advantages of an instruction cache outweigh the disadvantages.

The net effect of the low power design techniques is not to make processors that are dramatically more power efficient but rather to enable the same power efficiency at much higher performance. This reduces the number of processors required in a system and allows for efficient software implementations of communications systems.

References

(1) G. Blaauw and F. Brooks Jr., Computer Architecture: Concepts and Evolution, Addison-Wesley, Reading, MA, 1997.
(2) J. Sebot and N. Drach, "SIMD extensions: reducing power consumption on a superscalar processor for multimedia applications," presented at Cool Chips IV, April 2001.
(3) J. Glossner, D. Iancu, J. Lu, E. Hokenek, and M. Moudgill, "A software defined communications baseband design", *IEEE Communications Magazine*, vol. 41, no. 1, pp. 120-128, January, 2003.
(4) K. Jarvinen et al., "GSM enhanced full rate speech codec," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 771-774, 1997.
(5) B. Moyer, "Low-power design for embedded processors," Proceedings of the IEEE, vol. 89, no. 11, pp. 1576-1587, Nov 2001.
(6) T. Mudge, "Power: a first-class architectural design constraint," *IEEE Computer*, vol. 34, no. 4, pp. 52-58, 2001.
(7) A. Wroblewski, O. Schumecher, C. V. Schimpfle, and J. A. Nossek, "Minimizing gate capacitances with transistor sizing," *IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 186-189, May 2001.
(8) M. Borah , R. M. Owens, and M. J. Irwin, "Transistor sizing for minimizing power consumption of CMOS circuits under delay constraint," *International Symposium on Low Power Electronics and Design*, pp. 167 – 172, 1995.
(9) S. Kim, J. Kim, and S.-Y. Hwang, "New path balancing algorithm for glitch power reduction," *IEE Proceedings on Circuits, Devices and Systems*, vol. 148, no. 3, pp. 151-156, June 2001.
(10) W.-Z. Shen; J.-Y. Lin; F.-W. Wang, "Transistor reordering rules for power reduction in CMOS gates," International Asian and South Pacific Design Automation Conference, pp. 1-6, 1995.
(11) Nagendra, M. J. Irwin, and R. M. Owens, "Area-time-power tradeoffs in parallel adders," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 53, no. 10, pp 689-702, October, 1996.
(12) Guyot, B. Hochet, and J. Muller, "A way to build efficient carry-skip adders," *IEEE Transactions on Computers*, vol. 36, Oct. 1987.
(13) K. Chirca, M. Schulte, J. Glossner, S. Mamidi, and S. Vassiliadis, "A Static Low-Power, High-Performance 32-bit Carry Skip Adder", Proceedings of the Euromicro 2004 Symposium on Digital System Design, September, 2004 (in press).