A Low-Power Multithreaded Processor for Baseband Communication Systems

Michael Schulte[2], John Glossner[1,3], Suman Mamidi[2], Mayan Moudgill[1], and Stamatis Vassiliadis[3]

[1] Sandbridge Technologies
1 North Lexington Ave.
White Plains, NY, 10512,
USA

jglossner@sandbridgetech.com
http://www.sandbridgetech.com

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

{schulte,mamidi}@engr.wisc.edu
http://mesa.ece.wisc.edu

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

s.vassiliadis@its.tudelft.nl
http://ce.et.tudelft.nl

Abstract. Embedded digital signal processors for baseband communication systems have stringent design constraints including high computational bandwidth, low power consumption, and low interrupt latency. Furthermore, these processors should be compiler-friendly, so that code for them can quickly be developed in a high-level language. This paper presents the design of a high-performance, low-power digital signal processor for baseband communication systems. The processor uses token triggered threading, SIMD vector processing, and powerful compound instructions to provide real-time baseband processing capabilities with very low power consumption. Using a super-computer class vectorizing compiler, the processor achieves real-time performance on a 2Mbps WCDMA transmission system.

1 Introduction

General purpose processors have utilized various 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) processors, where multiple hardware thread units simultaneously issue multiple instructions per cycle, have been deployed [1]. These techniques have produced performance increases at high complexity and power dissipation costs.

In the embedded DSP community, power dissipation and real-time processing constraints have typically precluded general purpose microarchitectural techniques. Rather than minimize average execution time, embedded DSP processors often require the worst case execution time to be minimized in order to satisfy real-time constraints. Consequently, VLIW or statically scheduled microarchitectures with

architecturally visible pipelines are typically employed. Unfortunately, exposing pipelines may pose interrupt latency restrictions, particularly if all memory loads must complete prior to servicing an interrupt. Furthermore, on-chip memory access in DSP systems has traditionally operated at the processor clock frequency. Although this eases the programming burden and allows single cycle on-chip memory accesses, it often restricts the maximum processor clock frequency.

In this paper, we describe a compound instruction set architecture and an ultra low power multithreaded microarchitecture, in which multithreading is utilized to reduce power consumption and simplifying programming. We also describe a non-blocking fully interlocked pipeline implementation with reduced hardware complexity that allows the on-chip memory to operate significantly slower than the processor cycle time without inducing pipeline stalls.

2 Sandblaster Processor Design

Sandbridge Technologies has designed a multithreaded processor capable of executing DSP, embedded control, and Java code in a single compound instruction set optimized for handset radio applications [2-4]. The Sandbridge Sandblaster design overcomes the deficiencies of previous approaches by providing substantial parallelism and throughput for high-performance DSP applications, while maintaining fast interrupt response, high-level language programmability, and low power dissipation.

Fig. 1 shows a block diagram of the Sandblaster microarchitecture. The processor is partitioned into three units; an instruction fetch and branch unit, an integer and load/store unit, and a SIMD vector unit. The design utilizes a unique combination of techniques including hardware support for multiple threads, SIMD vector processing, and instruction set support for Java code. Program memory is conserved through the use of powerful compounded instructions that may issue multiple operations per cycle. The resulting combination provides for efficient execution of DSP, control, and Java code.

2.1 Processor Pipeline

Processor pipelines for one particular implementation of the Sandblaster DSP are shown in Fig. 2. The pipelines are different for various operations. The Load/Store (Ld/St) pipeline is shown to have nine stages. The first stage fetches and 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 is written back to the register file. Once an instruction from a particular thread enters the pipeline, it runs to completion. It is also guaranteed to write back its result before the next instruction from the same thread tries to read the result.

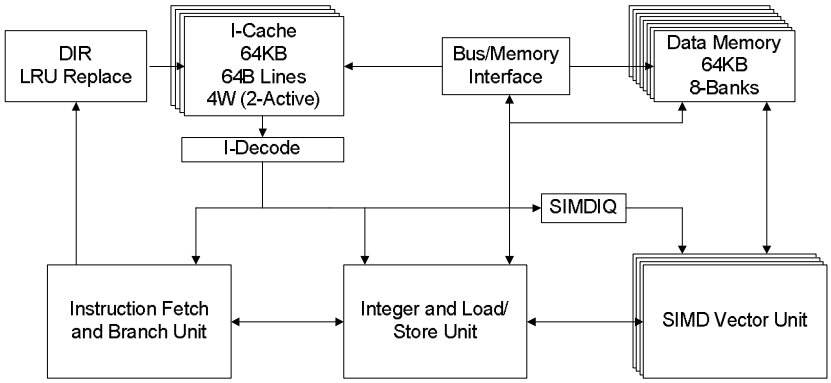

Fig. 1. Sandblaster Microarchitecture

Ld/St	Inst Dec	RF Read	Agen	XFer	Int Ext	Mem 0	Mem 1	Mem 2	WB
ALU	Inst Dec	Wait	RF Read	Exec 1	Exec 2	XFer	WB		
I_Mul	Inst Dec	Wait	RF Read	Exec 1	Exec 2	Exec 3	XFer	WB	
V_Mul	Inst Dec	VRF Read	Mpy1	Mpy2	Add1	Add2	XFer	VRF WB	

Fig. 2. Processor Pipelines

Similarly, there are multiple (variable) stages for the other execution pipelines. The integer and load/store unit has two execute stages for arithmetic and logic (ALU) instructions and three execute stages for integer multiplication (I_MUL) instructions. The Wait stage for the ALU and I_MUL instructions causes these instructions to read from the general-purpose register file one cycle later than Ld/St instructions. This helps reduce the number of register file read ports. The vector multiplication (V_MUL) has four execute stages - two for multiplication and two for addition. An additional transfer (Xfer) stage is allocated between the computation of a result and writing the result back to the register file to account for delays due to long wires in deep submicron design.

An important point is that the write back stages of the instructions are staggered. This allows a single write port to be implemented, but provides the same functionality as multiple write ports. The processor does not stall provided threads issue as even/odd pairs. This allows the register files to be banked, giving the power dissipation of a single write port, but able to completely sustain the code sequence given in Fig. 3. With the complexity of a single write port per register file, the processor can sustain more than 3.9 taps per cycle on typical DSP filters. This includes the overhead of entering and exiting the loop.

2.2 Compound Instructions

The Sandblaster architecture is a compound instruction set architecture. Historically, DSPs have used compound instruction set architectures to conserve instruction space encoding bits. In contrast, VLIW architectures are often completely orthogonal, 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 register file write ports. For example, Texas Instrument's TMS320C62x VelociTi processor fetches up to eight 32-bit instructions each cycle [5]. It has two general-purpose register files and each register file has 16 read ports and 10 write ports [5]. Both these features contribute heavily to power dissipation.

In the Sandblaster architecture, specific fields within a 64-bit compound instruction may issue multiple compound operations, including SIMD vector operations. Each field controls a different execution unit, and restrictions may apply if a particular operation is chosen. Most classical DSP instruction set architectures are compound. In contrast, a VLIW instruction set architecture may allow a completely orthogonal specification and then fill in any unused issue slots either in hardware or through no operation instructions (NOPs).

Fig. 3 illustrates the compound nature of our architecture. It shows a single compound instruction with three compound operations. This instruction implements the inner loop of a vector sum-of-squares computation. The first compound operation, `lvu`, loads the vector register `vr0` with four 16-bit elements and updates the address pointer `r3` to the next element. 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`. Further details on the `vmulreds` instruction are provided in Section 4. The loop operation decrements the loop count register `lc0`, compares it to zero, and branches to address `L0` if the result is not zero.

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

Fig. 3. A Single Compound Instruction for a Sum of Squares Loop

All the code shown in Figure 3 is encoded in a single 64-bit compound instruction. Each compound operation, including each vector operation, is specified with at most 21 bits. Like most DSP architectures, arbitrary operations are not specifiable within the same instruction. The 64-bit instruction shown in Fig. 3 may require 256 bits or more to encode on a VLIW machine. Furthermore, since the pipeline in a VLIW machine typically produces architecturally visible side effects (i.e. it is not

transparent), it may take a deeply software pipelined loop to obtain single-cycle throughput, thereby exploding the instruction storage requirements. To further distinguish our approach from VLIW and exposed pipeline architectures, each instruction is completely interlocked and architecturally defined to complete with no visible pipeline effects. This is critical for fast interrupt processing.

3 Low Power Multithreading

Multithreading is a well-known technique for hardware and software acceleration. The Denelcor HEP was designed circa 1979 [6]. In this design, multiple instructions could be simultaneously active from multiple threads. It was required that each thread complete the current instruction prior to issuing a subsequent instruction. When only one thread issues an instruction each cycle and threads progress in sequence, this is termed barrel multithreading or interleaved multithreading [7].

More recent embodiments of multithreaded processors make use of simultaneous multithreading (SMT) [1, 7]. In this approach, multiple thread units may issue multiple instructions each cycle. When combined with superscalar techniques such as out-of-order processing, the additional hardware required for SMT is not significant. Although SMT may reduce power dissipation in superscalar processors, superscalar and SMT techniques both consume significant power [8]. They also make it difficult to determine the worst case execution time, since instructions are scheduled dynamically, rather than at compile time.

3.1 Decoupled Logic and Memory

As technology improves, processors are capable of executing at very fast cycle times. Current state-of-the-art performance for 0.13um 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 that provides 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.

For 0.13um 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, the Sandblaster implementation of multithreading allows the processor cycle time to be decoupled from the on-chip memory (e.g., cache) 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 multithreaded pipeline design.

3.2 Token Triggered Threading

Fig. 1 shows the microarchitecture of the Sandblaster processor. In a multi-threaded processor, 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 (contexts). The microarchitecture supports up to eight concurrent hardware threads. The microarchitecture also supports multiple operations being issued from each thread.

The Sandblaster processor uses a form of interleaved multithreading called Token Triggered Threading (T^3). As shown in Fig. 4, with T^3 each thread is allowed to simultaneously execute an instruction, but only one thread 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 thread that is to issue an instruction. Tokens may be sequential (e.g. round-robin), even/odd, or based on other communication 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 dependency checking and bypass hardware is not needed, as explained in the next section. Baseband processing applications have sufficient thread level parallelism to support several concurrent hardware threads. For example, our implementation of WCMDA has 32 threads that can operate concurrently.

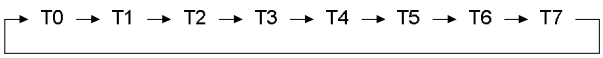

Fig. 4. Token Triggered Threading

4 Vector Processing Unit

Fig. 5 shows a high-level block diagram of the SIMD vector processing unit (VPU), which consists of four vector processing elements (VPEs), a shuffle unit, a reduction unit, and a multithreaded 2-bank accumulator register file. The four VPEs perform arithmetic and logic operations in SIMD fashion on 16-bit, 32-bit, and 40-bit fixed-point data types. High-speed 64-bit data busses allow each PE to load or store 16 bits of data each cycle in SIMD fashion. Support for SIMD execution significantly reduces code size, as well as power consumption from fetching and decoding instructions, since multiple sets of data elements are processed with a single instruction [9].

The shuffle unit transfers data between the VPEs, and is useful when implementing various DSP algorithms, such as FFTs and DCTs, which require data to be processed and then rearranged [10]. The shuffle unit reduces the number of data memory

accesses needed to perform these algorithms by allowing data to be rearranged within the VPU, instead of having to store data out to memory and then retrieve it in a different order. The reduction unit takes results from the VPEs, adds them to or subtracts them from an accumulator register file operand, and then stores the result of the reduction back in the accumulator register file. The reduction unit and accumulator register file accelerate the computation of dot products and similar vector operations.

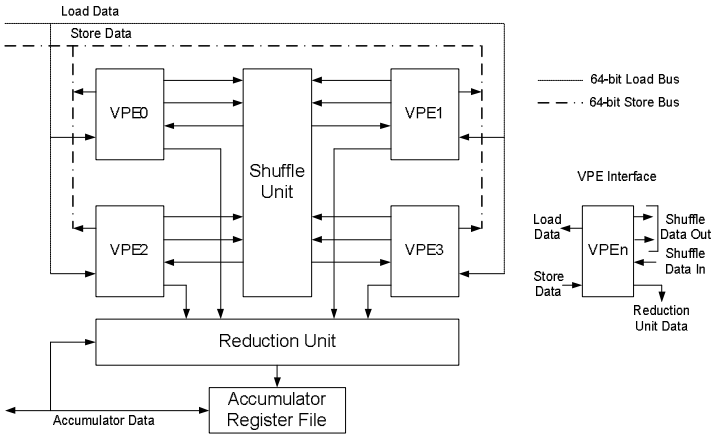

Fig. 5. SIMD Vector Processing Unit

Each VPE contains a 2-bank multithreaded vector register file (VRF), a multiply-accumulate (MAC) unit, a shifter, a compare-select unit, a logic unit, and an adder. To reduce dynamic power dissipation, clock gating is employed such that when performing an operation in a particular unit, the inputs to other units do not change. All of the VPE functional units support operations on 16-bit, 32-bit, and 40-bit operands, except for the MAC unit, which only multiplies 16-bit operands and can then add a 32-bit or 40-bit accumulator. Multiplication of numbers larger than 16 bits is not necessary for most DSP algorithms in our application domain, and support for it would lead to an unacceptable increase in area, cycle time, and power consumption. When required, 32-bit multiplications are implemented with multiple 16-bit multiplications. The MAC unit, shifter, and adder all support both saturating and wrap-around arithmetic operations. DSP algorithms in our application domain typically use saturating arithmetic with 32-bit accumulators and wrap-around arithmetic with 40-bit accumulators.

Most SIMD vector instructions go through eight pipeline stages. For example, a vector MAC instruction goes through the following stages: Instruction Decode, VRF Read, Mpy1, Mpy2, Add1, Add2, Transfer, and Write Back. The Transfer stage is needed due to the long wiring delay between the bottom of the VPU and the VRF.

Since there are eight cycles between when consecutive instructions issue from the same thread, results from one instruction in a thread are guaranteed to have written their results back to the VRF by the time the next instruction in the same thread is ready to read them. Thus, the long pipeline latency of the VPEs is effectively hidden, and no data dependency checking or bypass hardware is needed. This is illustrated in Fig. 6, where two consecutive vector multiply instructions issue from the same thread. Even if there is a data dependency between the two instructions, there is no need to stall the second instruction, since the first instruction has completed the Write Back stage before the second instruction enters the VRF Read stage.

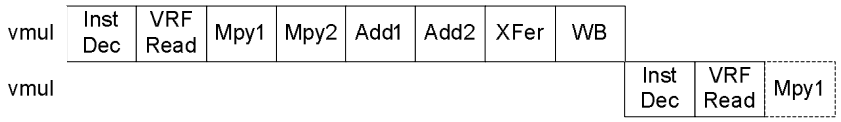

Fig. 6. Two Consecutive Vector Multiply Instructions that Issue from the Same Thread

The VRF in each VPE has eight 40-bit register file entries per thread. Since both a vector MAC operation (with three vector source operands and one vector destination operand) and a vector load or store operation (with one vector destination or source operand) can appear in the same compound instruction, a VRF designed using a standard implementation requires four read ports and two write ports.

To reduce the number of ports, the VRF uses a novel technique, which divides it into two register banks; one for even threads and one for odd threads. Register accesses by certain source and destination operands are delayed, such that in a given cycle each register file bank has at most two operands being read and one operand being written. For example, when a MAC operation and a store operation appear in the same compound instruction, the two multiplier operands are read from the VRF immediately following the instruction decode stage, but the accumulator and store operands are read one cycle later (i.e., during the Mpy1 stage). Thus, the accumulator and store operands are read from one bank of the register file, while the next instruction, which issues from a different thread, reads at most two operands from the other bank.

The reduction unit and accumulator register file are used with the VPEs to perform dot products and similar vector operations, which are required in many DSP applications. In particular, Global System for Mobile communication (GSM) standards, which are fundamental components of second and third generation cell phone technology, frequently perform dot products with saturation after each multiplication and each addition. To be compliant with GSM standards, the results produced by GSM algorithms much be identical (bit-exact) to the results obtained when the algorithms are executed serially with saturation after each operation. Since saturating arithmetic operations are not associative, most DSP processors execute saturating dot products in GSM algorithms sequentially, which degrades performance.

The Sandblaster processor executes roughly (*k/4*) vmulreds instructions to perform a *k*-element saturating dot product. This computation is similar to the sum of squares computation shown in Fig. 3, except the vmulreds instruction changes to

```
vmulreds %ac0,%vr0,%vr1,%ac0
```

and data is also loaded into `vr1`. For each vmulreds instruction, four pairs of vector elements are multiplied in parallel by the MAC units in the four VPEs. The reduction unit then adds the results from the VPEs, along with an operand from the accumulator register file, with saturation after each addition. The result from reduction unit is then written back to the accumulator register file to be used in the next instruction.

5 SDR Implementation Results

Sandbridge Technologies has developed a complete Software Defined Radio (SDR) product, including a baseband processor and C code for the UMTS WCDMA FDD-mode physical layer standard. Using an internally developed super computer class vectorizing compiler, real-time performance on a 2Mbps WCDMA transmission system has been achieved. This includes all chip, symbol, and bit-rate processing. Fig. 7 shows the performance requirements for 802.11b, GPRS, and WCDMA as a function of SB3000 utilization for different transmission rates. The SB3000 contains four Sandblaster cores and provides processing capacity for full 2Mbps WCDMA FDD-mode including chip, symbol, and bit-rate processing.

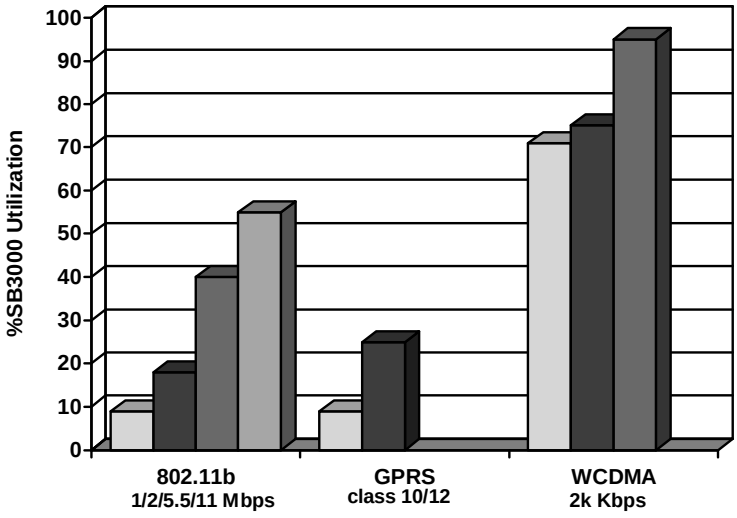

Fig. 7. Processor Utilization in Baseband Communication Systems

6 Conclusion

This paper has presented the design of a high-performance, low-power processor for baseband communication systems. The design uses a unique combination of token triggered threading, SIMD vector processing, and powerful compound instructions to provide very low power consumption and real-time baseband processing capabilities. Having validated our low power design approach with working 0.18um silicon and having implemented complete baseband processing on our core, we can provide a terminal class SDR baseband processor with power dissipation appropriate for commercial terminals.

References

1. D. M. Tullsen, S. J. Eggers, H. M. Levy: Simultaneous Multithreading: Maximizing on-chip Parallelism. 22nd Annual International Symposium on Computer Architecture (June, 1995) 392-403
2. J. Glossner, D. Iancu, J. Lu, E. Hokenek, and M. Moudgill: A Software Defined Communications Baseband Design. IEEE Communications Magazine, Vol. 41, No. 1 (January, 2003) 120-128
3. J. Glossner, T. Raja, E. Hokenek, and M. Moudgill: A Multithreaded Processor Architecture for SDR. The Proceedings of the Korean Institute of Communication Sciences, Vol. 19, No. 11 (November, 2002) 70-84
4. J. Glossner, M. Schulte, and S. Vassiliadis: A Java-Enabled DSP. In E. Deprettere, J. Teich, and S. Vassiliadis (eds.): Embedded Processor Design Challenges, Systems, Architectures, Modeling, and Simulation (SAMOS). Lecture Notes in Computer Science, Vol. 2268. Springer-Verlag, Berlin (2002) 307-325
5. N. Seshan: High VelociTI Processing: Texas Instruments VLIW DSP Architecture. IEEE Signal Processing Magazine, vol.15, no 2 (March 1998) 86-101
6. B. J. Smith: The Architecture of HEP. In J. S. Kowalik (ed.) Parallel MIMD Computation: HEP Supercomputer and Its Applications. MIT Press, Cambridge, MA, (1985) 41–55.
7. T. Ungerer, B. Robič, and J. Šilc: A Survey of Processors with Explicit Multithreading. ACM Computing Surveys, vol. 35, no. 1 (March 2003) 29-63
8. J. S. Seng, D. M. Tullsen, and G. Z.N. Cai: Power-Sensitive Multithreaded Architecture. International Conference on Computer Design, (September, 2000) 199-208
9. J. Sebot and N. Drach: SIMD Extensions: Reducing Power Consumption on a Superscalar Processor for Multimedia Applications. Cool Chips IV (April 2001)
10. R. B. Lee: Subword Permutation Instructions for Two-Dimensional Multimedia Processing in MicroSIMD Architectures. Proceedings of the IEEE 11th International Conference on Application-Specific Systems, Architectures and Processor (July 2000) 3-14
11. B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs. Oxford University Press, New York (2000)