A FULL-RATE SOFTWARE IMPLEMENTATION OF AN IEEE 802.11A COMPLIANT DIGITAL BASEBAND TRANSMITTER

Michael J. Meeuwsen, Omar Sattari, Bevan M. Baas

Department of Electrical and Computer Engineering
University of California
Davis, CA 95616 USA

ABSTRACT

A software based IEEE 802.11a digital baseband transmitter has been implemented on a highly parallel single-chip DSP processor. The processing platform is a programmable and reconfigurable Asynchronous Array of simple Processors (AsAP) that is well matched to complex system workloads such as 802.11a. The transmitter is the first fully-compliant 802.11a software implementation, and is the first full-rate software implementation. The transmitter also complies with the high-rate portions of the 802.11g standard. It operates over all 8 data rates, includes additional upsampling and filtering functions, and sustains transmissions at 54 Mb/s on a 22-processor array—which is expected to occupy less than 20 mm^2 in 0.18 μm CMOS.

1. INTRODUCTION

Recently, there has been increasing interest in low-cost, high-bandwidth wireless LAN devices for consumer and enterprise use. The IEEE 802.11a and 802.11g standards [1] define an OFDM-based modulation scheme supporting raw data rates from 6 to 54 Mbits/s. 802.11g is a more recent standard, which incorporates backward compatibility with the lower-rate 802.11b standard and operates in a different RF band. The higher-rate portions of 802.11g utilize OFDM and are identical to the digital baseband of 802.11a.

Software radio support for these standards is desirable because it reduces system cost, and allows easy upgradability for support of future standards. Although some progress has been made toward the development of software solutions for 802.11a, no solution has yet been reported which is able to sustain the required data rates on a fully programmable platform.

We have demonstrated a fully compliant IEEE 802.11a digital baseband transmitter, implemented in software running on a novel DSP architecture. The architecture consists of an Asynchronous Array of simple Processors (AsAP) integrated onto a single chip. AsAP provides a fully programmable platform suitable for sustaining the data rates required by 802.11a. In the remainder of this paper, we first examine the current state of software 802.11a implementations. We then describe the AsAP architecture, and the implementation of the 802.11a transmitter. Finally, we consider the transmitter's performance and the runtime behavior of the processor array.

This work was supported by a UC Graduate Scholars Fellowship, a UCD Faculty Research Grant, and Intel Corporation.

2. EXISTING WORK

Software defined radio (SDR) promises flexible and reconfigurable digital communications platforms. Many contributions have been made in the development of SDR [2] [3] [4]. A major challenge recognized in this area is the development of programmable architectures with adequate throughput to support the real-time data rates required by SDR applications [3].

Current full-rate 802.11a baseband implementations use dedicated hardware (ASIC) processors [5]. This allows a highly customized data path, which provides the required throughput while maintaining a high average data rate. However, the increasing cost, long development time, and low flexibility of ASICs make programmable solutions increasingly attractive.

Software implementations of the 802.11a physical layer have been reported in the literature. Various software OFDM solutions are summarized in Table 1. All of these systems either lack full support of the IEEE 802.11a specification, or do not target a particular real-time platform.

The major challenge faced by software implementations is achieving high throughput while constrained to a sequential execution model. Current VLIW processor architectures allow exploitation of instruction level parallelism, but neglect global parallelism found at the application level. This forces the independent tasks of the baseband to execute serially when mapped onto a single processor, requiring a high number of instructions per cycle to achieve the required

Appears in *Proceedings of the IEEE Workshop on Signal Processing Systems, 2004. SIPS 2004.*

Project	Wireless Standard	Data Rate	Processing Platform	Tx/Rx
Stuber [6]	MIMO-OFDM	N/A	N/A	Tx/Rx
Bakker [7]	OFDM QPSK	8 Mbit/s	LART	Tx/Rx
Gifford [8]	OFDM DQPSK	N/A	ANSI C	Tx/Rx
Witrisal [9]	OFDM	N/A	SDR	Tx/Rx
Barton [10]	estim. IEEE 802.11a/b	N/A	BOPS DSP	Tx/Rx
Tariq [11]	partial IEEE 802.11a	1.7 Mb/s	TI DSP	Tx/Rx
This work	full IEEE 802.11a	54 Mb/s	AsAP	Tx

Table 1. Comparison of related OFDM and wireless LAN software implementations

throughput. Current DSPs do not have adequate processing power to sustain the data rates demanded by applications such as 802.11a. Work by Tariq [11] in 2002 suggests processors about 14 times faster are required for single-chip full-rate operation at 24 Mbit/s.

3. ARCHITECTURAL OVERVIEW

AsAP is a novel architecture aimed at overcoming the shortfalls of traditional DSPs. AsAP consists of a two dimensional array of independent processors, integrated with reconfigurable interconnect on a single chip. Key features are described below and details of the architecture were recently published [12].

Each processing element is a deeply pipelined single issue processor, supporting the types of instructions commonly found in commercial DSPs. Many architectural features are chosen to enable high clock rates. The datapath is 16-bit fixed-point and contains both an ALU and MAC unit. The processors contain small instruction and data memories, of 64 and 128 words respectively. Specialized address generation hardware is added to ease the implementation of algorithms requiring sequential, strided, or bit-reversed accesses to data memory. Each processor contains its own local clock generator, and no global frequency or phase information is shared among processors. Current estimates predict processor areas and maximum clock frequencies of 0.8 mm^2 and 1.0 GHz in 0.18 μm CMOS.

Inter-processor communication is accomplished through a reconfigurable interconnection network. Each processor may take input from two of its four nearest neighbors, and may output to any combination of these four neighbors. Asynchronous dataflow is managed by FIFO buffers at the inputs of each processor. If a processor attempts to write to a neighboring processor with a full FIFO, an output stall will occur. If a processor attempts to read from an empty FIFO, an input stall will occur. When stalled, no instructions are executed and the processor's clock can be stopped to save power.

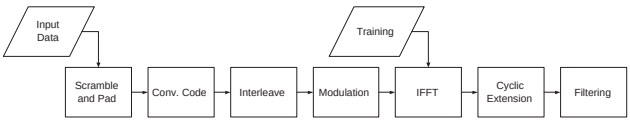

Fig. 1. Data flow diagram of primary 802.11a processing blocks

We have implemented a functional RTL model, which allows simulation of a processor array of arbitrary size. The model currently contains single cycle processors, as pipelining details will be largely driven by the results of VLSI and layout work. The program for each processor is written in AsAP assembly language. Initial instruction and data memory contents are downloaded into the processors during system startup via a common configuration bus.

4. IEEE 802.11A TRANSMITTER IMPLEMENTATION

We have implemented a digital baseband transmitter as specified in the physical layer specification of IEEE 802.11a on the AsAP platform. The transmitter has been implemented and verified to be fully compliant with Annex G of the standard [1], as well as with a software model written in Matlab.

4.1. Partitioning and Dataflow

The processing required for the 802.11a transmitter can be partitioned into a number of independent serial tasks, as summarized in Figure 1. When implementing the transmitter on AsAP, each task is mapped to a separate processor to allow parallel execution. If either more memory or higher performance is required than can be provided by a single processor, a task is mapped across multiple processors. Code for each processor is implemented independently, considering only its inputs and outputs. This makes the software development no more complicated than when writing code for a sequential machine. AsAP code may, in fact, be simpler to write because inter-process communication is handled in hardware. The final system partitioning is detailed in Table 2. The processors are integrated in a four by six array as shown in Figure 2.

The serial nature of the 802.11a transmit path dictates the dataflow among the processors. In most cases, each task takes a single input set, and produces a single output set, which is consumed by downstream processors. There are two primary exceptions to this paradigm that appear in this application. The first is when processors generate output with no input. The second is when the execution of two processors is closely coupled, requiring that a bidirectional link

Processor	Functions Performed
Pad	Generates PLCP header; pads and scrambles data.
Scrambler	Generates a pseudo-random scrambling sequence.
Conv. Code	Performs rate=1/2, k=7 convolutional coding.
Puncture	Punctures coded data based on the data rate.
Interleave1	Applies the first interleaving permutation.
Interleave2	Applies the second interleaving permutation.
Mod. Map	Modulates the data using the appropriate rate-dependent modulation scheme.
Training	Generates short and long training sequences.
Pilot Insertion	Prepends training sequence. Inserts pilot signals onto the specified sub carriers.
IFFT BR	Reorders the input using bit-reversed addressing.
IFFT Mem	Performs address generation for the IFFT, provides data to butterfly processors, and writes results back to memory. This processor is replicated to allow multiple stages to be computed in parallel.
IFFT BF	Computes complex radix-2 butterflies. Twiddle factors are chosen to implement an IFFT.
IFFT Output	Reorders the IFFT output into proper order.
GI/Window (Real)	Performs cyclic extension, windowing, and concatenation for the real part of the data.
GI/Window (Imag)	Performs cyclic extension, windowing, and concatenation for the real part of the data. Two processors are required for GI/Windowing, because the working data set will not fit in a single processor's memory.
Upsample/FIR	Performs upsampling by two, and applies a parameterized Nyquist filter.
Output Sync	Synchronizes the data stream to the application's output data rate. This processor is clocked by an externally-supplied 80 MHz clock.

Table 2. Distribution of computation tasks in the AsAP 802.11a transmitter implementation

be established between them. The global dataflow through the array is indicated in Figure 2 by the large grey arrows, while the smaller arrows indicate all communication that takes place between processors. The required processor interconnection results in one processor that cannot be used. This processor is turned off to save power.

The major advantage of AsAP over conventional software platforms is its capability to operate on data as it passes through the system, rather than storing large amounts of data, and performing tasks sequentially. This feature effectively creates an application level pipeline, allowing the serial tasks of the transmitter to take place in parallel. As data flows through the system, it is operated on as soon as possible. In addition, only data that is required for the pending computation is required to be stored by any one processor.

This approach is highly data driven, which presents a challenge when control information is required to be communicated among processors. In the 802.11a physical layer, there are two control data required for packet transmission:

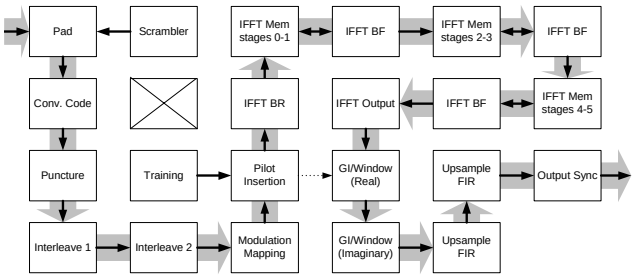

Fig. 2. Block diagram of tasks and dataflow of the 22 processors used for the 802.11a implementation. The processor marked with an "×" is unused and powered down.

the transmission rate and the data length. We solve this problem by preceding the data input to each processor with the appropriate rate and length data. Each processor is programmed to read, and propagate this data appropriately before beginning computation on a particular packet. This solution is effective because the deterministic nature of the algorithms makes it easy to differentiate control words from data words by their location in the data stream.

4.2. Modularity and Code Reuse

The AsAP architecture lends itself to code reuse. The 802.11a transmitter implementation contains two general signal processing blocks that do not require any special modification to operate in the transmitter system.

The FIR modules contain code to upsample their inputs by a factor of two, and apply a parameterized low-pass filter. Although not required by the standard, we cascade two of these upsampling filters at the output to ease analog filtering requirements.

The IFFT required by the 802.11a transmitter is implemented with an eight-processor FFT block. The twiddle factors of the block are chosen to implement an inverse Fourier transform. The algorithm uses eight processors for increased throughput. Due to the small memories in each processor, two processors are required for the core computation of a 64-point FFT. By adding four additional processors, we are able to pipeline the computation of consecutive transforms, computing two butterfly stages of each FFT in a single butterfly-memory processor pair.

To avoid modifying the FFT block to handle the control data mentioned above, we have selected a topology that allows the FFT to be bypassed temporarily, passing the needed control data to the GI/Windowing processors without passing through the FFT. This link is shown by a dotted arrow in Figure 2.

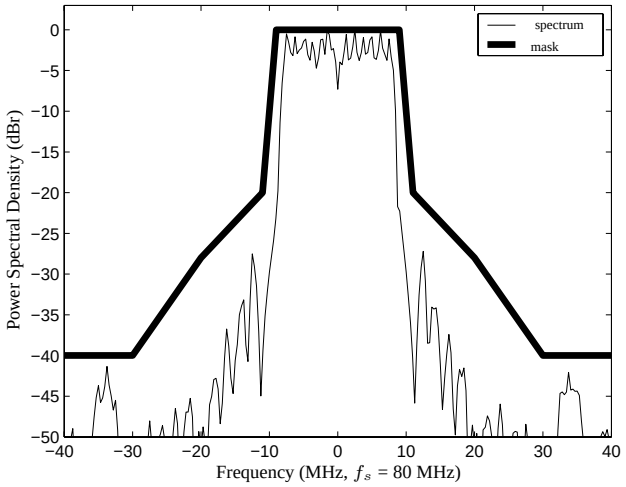

Fig. 3. Output spectrum from AsAP simulation shown with required spectral mask

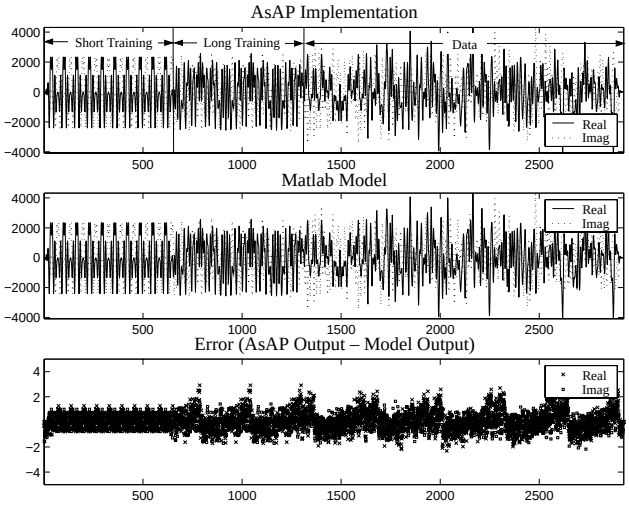

Fig. 4. Time domain waveforms from IEEE compliant AsAP and Matlab models, and difference error, for a 5-symbol packet

4.3. Diversity of Algorithmic Requirements

The algorithms that compose the transmitter are widely diverse. Block-oriented computation, such as the FFT, requires a large amount of input to be accumulated before producing any data. These algorithms often require large memories. We cope with this challenge by expanding the algorithm across multiple processors to exploit the memory of each processor. In our FFT implementation, one processor stores the data values for the current FFT stage, and resolves the complex addressing required for the algorithm. A second processor stores the twiddle factors, and executes butterflies, returning the results to the memory processor for storage. This allows us to maintain small and fast memories in each processor, decreasing cycle time and power consumption.

Other computation is not block oriented, and requires minimal data storage. Examples are the convolutional coder, and the FIR filters. Here, the input data are operated on as soon as available. When mixing block computation with computation which is not block oriented, the processors' input FIFOs help reduce the impact of data bursts at the output of block-oriented algorithms.

Bit manipulations such as scrambling and interleaving are often easily implemented in hardware, but are more challenging in software. AsAP is able to successfully handle this type of computation as well, because of its high clock rate and because other tasks can occur in parallel.

5. RESULTS

We have implemented all portions of the IEEE 802.11a physical layer digital baseband transmitter [1] on the AsAP architecture. When simulating this application, we assume that the processors in the array are pipelined to achieve a 1 GHz clock frequency. Current VLSI progress supports this assumption. All processors in the array are clocked at this frequency, but share no phase information. At this frequency, we are able to sustain transmission at 54 Mbit/s indefinitely, after an initial startup latency. In addition, we implement upsampling and filtering at the system output to meet the spectral mask requirements set forth in the standard. The resulting signal spectrum is shown in Figure 3. Our 16-bit fixed point implementation was compared against a floating point Matlab model. The AsAP-generated and reference waveforms are shown in Figure 4, along with the error in the AsAP signal. The SNR is 64.9dB. The major sources of error are the lack of rounding in the FFT and the FIRs. Rounding was not implemented to improve throughput, and reduce design time, but is an easy addition if greater precision is required.

5.1. Processor Activity Analysis

A more detailed analysis of processor execution reveals some interesting insight into the AsAP architecture. Figures 5 and 6 illustrate processor activity and stall frequency for a 5-symbol packet and for a single symbol during a long transmission. Figure 5 is useful in understanding edge effects due to initial packet overhead, and Figure 6 is useful

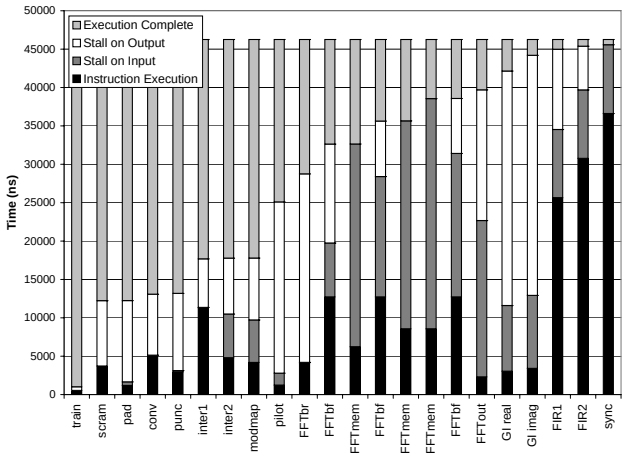

Fig. 5. Processor activity for one 5-symbol packet

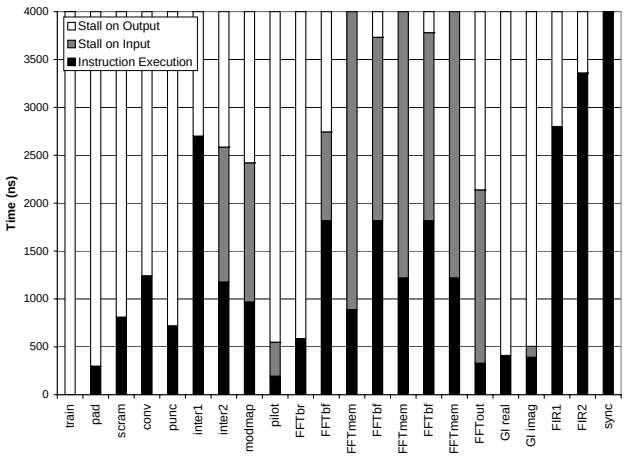

Fig. 6. Processor Activity for one 4 μsec symbol

in understanding the steady state long-packet behavior. The activity of each processor (the amount of time spent executing, instead of stalled), is indicated by the black bar in each figure. The dark gray bar indicates the time spent waiting for input to arrive, while the white bar indicates the amount of time stalled on output. In general, these stall cycles can be interpreted as slack, and any increase in execution time will reduce this stall time. The exception to this is when the input to a processor is dependent on that processor's output. This is the case in the FFT butterfly/memory pairs. The tight coupling of these processors introduces some unavoidable input stalls, which cannot be counted as slack. This coupling also explains why the FFT memory processors never stall on their outputs.

Analysis of the stall frequency and processor activity also gives insight into the bottleneck of the application. A processor will stall on output only if the downstream FIFO is full. This will occur if the source processor is too fast, or if the destination processor is not fast enough. Furthermore, a processor will stall on its input only if the source processor is not providing data at an adequate rate. The system bottleneck can be estimated by observing the stall behavior of neighboring processors. From Figure 6, we estimate two system bottlenecks due to interleaving (*inter1*) and IFFT calculation. These two algorithms are block-oriented, forcing non-block algorithms downstream to wait for data while computation is taking place.

5.2. Estimated Pipeline Impact

A deeply pipelined processor requires a careful analysis of its hazards. Processors in our current simulation model operate with an ideal CPI of one. When a pipelined implementation is considered, read-after-write (RAW) data hazards will introduce pipeline stalls, increasing the effective CPI. We estimated the frequency of RAW hazard stalls by examining dynamic execution traces of the application. We assume 6 cycle latency between operand fetch and write-back stages, and we assume that ALU results may be forwarded from the ALU output to the ALU input. For all processors in the transmitter, the RAW hazard penalty can be completely absorbed by the time currently spent stalling on input and output. Table 3 compares the estimated number of stalls due to RAW hazards against the number of cycles each processor stalls on input or output.

6. FUTURE WORK

We are currently designing a scalable full-custom CMOS implementation of the AsAP architecture and have completed layout of major functional units. Current estimates from layout and extracted layout spice simulations predict processor areas and maximum clock frequencies of $0.8\mathrm{mm}^2$ and 1.0 GHz in 0.18 μm CMOS and 0.4 mm^2 and 1.4 GHz in 0.13 μm CMOS. We recently completed a very high speed JPEG encoder implementation and are working to map other workloads, such as an 802.11a receiver, to the AsAP processor array. We are also working to develop tools and methodologies to more easily program and configure AsAP arrays.

7. CONCLUSION

We have developed and successfully simulated a fully compliant digital baseband transmitter for IEEE 802.11a entirely in software. The transmitter implementation is fully compliant with the standard, and is able to sustain a data rate of 54 Mbit/s. The transmitter is targeted for the AsAP

Processor	Measured RAW stall estimate	IO stalls	Required % for complete coverage
train	0	4000	0
pad	188	4000	5
scram	594	3702	16
conv	162	3190	5
punc	72	2758	3
inter1	887	3280	27
inter2	153	1300	12
modmap	576	2823	20
pilot	11	3032	0
FFTbr	0	3807	0
FFTbf	709	3414	21
FFTmem	0	2183	0
FFTbf	709	3112	23
FFTmem	0	2183	0
FFTbf	709	2780	26
FFTmem	0	2183	0
FFTout	0	2780	0
GI real	195	3671	5
GI imag	238	3591	7
FIR1	0	3609	0
FIR2	0	1200	0
sync	0	640	0

Table 3. Estimated stall cycles due to RAW hazards compared to IO stalls already occuring. Coverages of less than 100% results in no additional stalls due to RAW hazards.

architecture which is currently in development. AsAP provides significant advantages over traditional programmable DSPs due to its high degree of parallelism. These include increased performance due to global pipelining, and simplified implementation due to reduced communication overhead in software. AsAP's high clock rate and small area enable high performance and energy efficiency with all the benefits of a software-programmable solution.

8. REFERENCES

[1] LAN/MAN Standard Committee of the IEEE Computer Society, "Wireless LAN medium access control (MAC) and physical layer (PHY) specifications: High speed physical layer in the 5 GHz band," in *Standard for Information Technology*. Institute of Electrical and Electronics Engineers, New York, NY, 1999.

[2] A. Haghighat, "A review on essentials and technical challenges of software defined radio," in *IEEE Military Communications Conference*, Oct. 2002, vol. 1, pp. 377–382.

[3] Z. Salcic and C. F. Mecklenbrauker, "Software radio - architectural requirements, research and development challenge," in *IEEE International Conference on Communication Systems*, Nov. 2002, vol. 2, pp. 711–716.

[4] W. H. W. Tuttlebee, "Advances in software-defined radio," *Electronics Systems and Software*, vol. 1, no. 1, pp. 26–31, Feb. 2003.

[5] J. Thomson, B. Baas, E. M. Cooper, et al., "An Integrated 802.11a Baseband and MAC Processor," in *IEEE International Solid-State Circuits Conference*, 2002, vol. 45, pp. 126–127, 451.

[6] G. L. Stuber, J. R. Barry, S. W. Mclaughlin, Y. G. Li, M. A. Ingram, and T. G. Pratt, "Broadband MIMO-OFDM wireless communications," *Proceedings of the IEEE*, vol. 92, no. 2, pp. 271–294, Feb. 2004.

[7] J. D. Bakker and F. C. Schoute, "LART: design and implementation of an experimental wireless platform," in *IEEE Vehicular Technology Conference*, Sept. 2000, vol. 3, pp. 1460–1466.

[8] S. Gifford, J. E. Kleider, and S. Chuprun, "Broadband OFDM using 16-bit precision on a SDR platform," in *IEEE Military Communications Conference*, Oct. 2001, vol. 1, pp. 180–184.

[9] K. Witrisal, K. Buke, Y. H. Kim, R. Prasad, and L. P. Lightart, "Air-interface emulation for wireless broadband communications applied to OFDM," in *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, Sept. 2002, vol. 2, pp. 1251–1255.

[10] M. E. Barton, "Baseband system design for a multimode 802.11 a/b wireless LAN adapter," in *Southeast-Con 2002. Proceedings. IEEE*, Apr. 2002, pp. 322–325.

[11] M. F. Tariq, Y. Baltaci, T. Horseman, M. Butler, and A. Nix, "Development of an OFDM based high speed wireless LAN platform using the TI C6x DSP," in *IEEE International Conference on Communications*, 2002, vol. 1, pp. 522–526.

[12] B. M. Baas, "A parallel programmable energy-efficient architecture for computationally-intensive DSP systems," in *Signals, Systems and Computers, 2003. Conference Record of the Thirty-Seventh Asilomar Conference on*, Nov. 2003.