

Technology Driven DSP Architecture Optimization within a High-Level Block Diagram Based Design Flow

Dejan Marković¹, Brian Richards², Robert W. Brodersen²

¹Department of Electrical Engineering, University of California, Los Angeles

²Berkeley Wireless Research Center, University of California, Berkeley

Abstract—In the process of designing adaptive wireless communication systems, the design cycle traditionally requires re-entering a design at various abstraction levels, constraining the implementation choices and increasing time-to-market. An approach to use a unified design description for algorithm verification and architecture exploration is presented. The proposed graphical block-based design entry and re-targetable design flow provide the ability to track technology features in the process of architecture selection. Using this approach, word length optimization, register retiming, and data-stream interleaving are applied to square-root algorithm to reduce power and area. The flow is demonstrated on a complex multi-antenna signal processing. An example of Sobel edge detection illustrates the integration of IP macros and hardware emulation.

Index Terms—Integrated circuit design, circuit synthesis, design methodology, architecture, adaptive signal processing, square-rooting, matrix decomposition.

I. INTRODUCTION

This paper proposes a system design methodology that improves communication between algorithm developers and hardware designers, with the goal of reducing design time, while encouraging design improvements. Traditionally, algorithm designers often work with Matlab or C to validate their ideas, without feedback about the practical feasibility of a system. Chip designers then make a best effort to re-enter the design using Hardware Description Languages (HDLs), often fundamentally changing the algorithm. The result must then be validated by the algorithm designer, leading to multiple re-coding and verification of the design, significantly increasing the development cycle.

The work by Davis et. al. [1] demonstrated the feasibility of Matlab/Simulink [2] as the design environment for dedicated signal processing systems. The authors of [1] proposed a corresponding automated hierarchical ASIC synthesis methodology. This approach was augmented by FPGA emulation capability within the same framework [3]. An algorithm-to-architecture optimization methodology for reduced power and area was later integrated within the flow and used in [4]. This paper is an overview of the current design flow.

II. SIMULINK-BASED DESIGN FLOW

A unified Matlab/Simulink design environment, which is widely adopted by the algorithm designers, is proposed for hardware design. Using this approach, a design needs to be

entered only once. The environment enables both algorithm verification and hardware emulation, and also provides an abstract view of the design architecture. Finally, it allows FPGA-based ASIC testing [4], again using the same input description.

The design methodology is based on a Simulink hardware library block-set, which has readily available basic arithmetic operators such as add, multiply, shift, mux, register, and so on. These blocks have a notion of hardware parameters such as word length and latency, which are used for initial hardware emulation on an FPGA based platform [5, 6]. This block-based representation of an algorithm is an abstraction well-suited for architectural exploration.

Once bit-true and cycle-accurate behavior is described at the Simulink level, behavioral HDL is produced which allows algorithm mapping onto an FPGA for hardware emulation. The same description is also used for ASIC implementation, as shown in Fig. 1. The ASIC flow takes block connectivity from Simulink and generates the ASIC HDL for the blocks, using an in-house tool [3]. The tool also runs an initial synthesis and HDL simulation to verify functional equivalency between the two hardware descriptions. Mapped HDL can then be synthesized into a GDSII format using a foundry-provided backend synthesis tool flow.

With some architectural feedback from the underlying technology such as speed, power, and area of the building blocks, the architecture can be optimized in Simulink. With

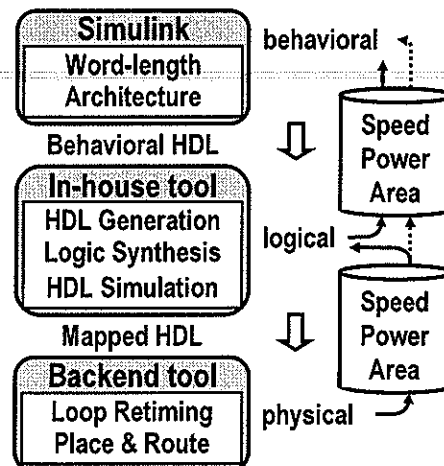


Fig. 1. Simulink-based design flow.

proper block characterization, no additional iterations through the design flow are needed. Using the flow shown in Fig. 1, it is straightforward to map a fixed architecture to a target technology. Leveraging this flow, architectural trade-offs can then be explored, allowing the designer to minimize power and area for a given technology, for a specified throughput constraint for an algorithm.

A. Block Characterization Methodology

The Simulink hardware library implicitly carries information only about latency and word length, with a sample period chosen when the design is mapped to an FPGA. For the ASIC flow, block characterization must be extended to include technology features such as speed, power, and area. The technology parameters often change with the scaling of technology, so a general and rapid methodology for block characterization is needed. The results are propagated to Simulink to provide needed information during the process of architecture selection.

A design-specific block characterization method is adopted for reduced complexity. Each new technology can be characterized for speed by performing transistor-level simulations of a simple fanout-of-four (FO4) inverter, and inferring the performance of other blocks based on these results. The goal is to obtain first-order estimates of the energy versus delay tradeoff, relative to a reference point, which is a design operating at the reference supply voltage V_{DD}^{ref} for the selected technology and operating condition. The shape of this curve is then used to estimate cycle time and energy scaling of pipeline stages for building blocks, as indicated in Fig. 2.

Once the basic energy-delay tradeoff for a pipeline stage is documented, the next step is to characterize building blocks in the latency versus cycle-time space, as indicated in Fig. 2. The goal is to assign a proper amount of pipelining to each block such that logic depth of all pipelines is balanced. Then, supply voltage scaling and gate sizing can be equally applied to the underlying pipelines to further optimize the design. Since the characterization is carried out at a fixed V_{DD}^{ref} for a standard-cell based implementation, circuit-level results are used to extrapolate the desired cycle time to V_{DD}^{opt} in order to minimize power and area. The minimization of power and

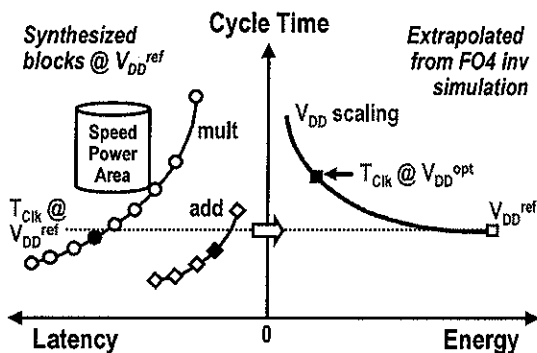


Fig. 2. Characterization methodology.

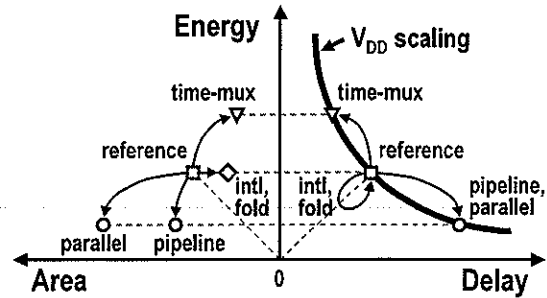


Fig. 3. Architecture transformations.

area, which will be described next, is performed under a throughput constraint for an algorithm.

B. Architecture Transformations

The results obtained from block characterization are used to guide architectural selection for an algorithm as illustrated in Fig. 3. This plot shows the energy-delay tradeoff for a simple pipeline stage, which is a fundamental component in any design. The goal is to move toward the desired point on the energy-delay line while minimizing the area of the resulting implementation.

Parallelism and pipelining relax timing constraints to reduce energy through voltage scaling at the expense of increased area. The area penalty for selecting a parallel architecture is larger than that resulting from a pipelined architecture. Time-multiplexing does the opposite: datapath logic has to operate faster to reduce area at the expense of increased energy. The energy increase comes from increased V_{DD} or upsizing of the gates. Finally, interleaving and folding [7], which involve simultaneous pipelining and up-sampling, map back to approximately the same point on the energy-delay line while reducing the area through pipeline logic sharing.

C. Verification

The final phase of the design flow is verification. Simulink is used in the entire design cycle: design entry, architecture optimization, and final ASIC verification.

The verification strategy is illustrated in Fig. 4. Behavioral HDL generated by Simulink can be used to emulate the design on the FPGA. By maintaining functional equivalency between FPGA and ASIC flows, the ASIC can be verified by real-time co-simulation with the FPGA. Test vectors are exported from

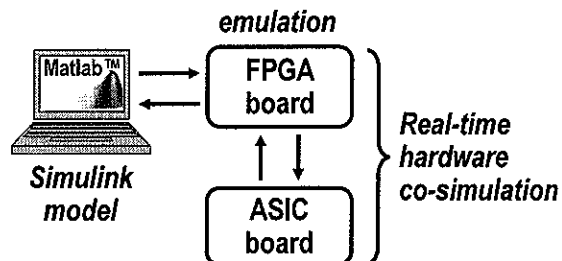


Fig. 4. FPGA-based ASIC verification.

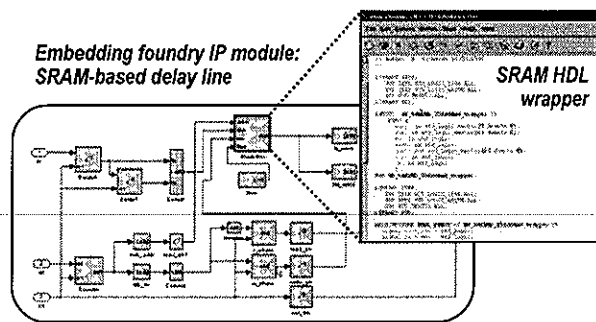


Fig. 5. Incorporating IP macros.

Simulink, and the design is programmed onto the FPGA, which not only provides input stimulus for the ASIC, but also records and analyzes output samples in real-time. Furthermore, input vectors can be programmed into the FPGA through a serial interface communicating with Matlab on a host processor. The outputs of both the FPGA and the ASIC can be conveniently stored in block-RAM memories on the FPGA, and read within the controlling Matlab environment.

III. FEATURES OF THE FLOW

In many designs, the basic library blocks may not be adequate for describing a complete system. The design flow avoids library limitations by supporting the integration of IP components for design reuse, and hierarchical synthesis for improved productivity.

A. Integration of IP Components

IP components can be embedded into a design as black-boxes. Each black-box has an HDL wrapper that makes the foundry IP look like any other library block. This is shown in Fig. 5 for the example of SRAM IP represented as black-box. The black-box primitive reads the HDL to automatically create a component. Using this approach, the IP components can be simulated together with the existing blocks, and also fully synthesized.

Figure 5 illustrates a Simulink diagram of an SRAM-based delay line. For this example, a foundry-generated SRAM block is combined with control logic to implement a parameterizable delay line.

B. Hierarchical Synthesis

If a complex block, such as the SRAM-based delay line described previously, is reused many times in a given design, the HDL can be reused. After generating the HDL for one occurrence of a design, a black-box primitive referencing the generated HDL is created, and the resulting primitive can be replicated efficiently in other designs.

Hierarchical synthesis is illustrated in Fig. 6, where the *sram_delay* block in the top diagram represents the design from Fig. 5. This block is then treated as a black-box primitive with pre-compiled HDL, as shown in the bottom diagram. Both HDL generation and synthesis tools can be run bottom-up to reduce design synthesis time.

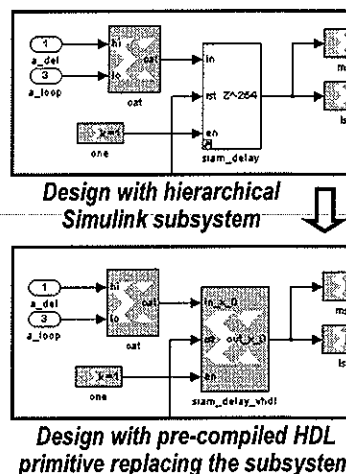


Fig. 6. Hierarchical synthesis.

IV. FLOW WALKTHROUGH: ITERATIVE 1/SQRT EXAMPLE

The Simulink design flow is illustrated on an iterative inverse square root algorithm [8], which is based on the Newton-Raphson method. In this example, it is assumed that the algorithm processes multiple interleaved streams of data.

The designer starts by entering the algorithm in graphical form in Simulink. The formula describing the algorithm shown in Fig. 7 states that solution $x_s(k)$ converges to $1/\sqrt{Z}$ when k goes to infinity. Multiply and add operations are indicated with the m and a blocks, respectively. The divide by two is realized using the shift operation performed by block s . Balancing delays b_1 and b_2 are required to support multiple incoming data streams. Parameters m , a , u , b_1 , b_2 , s also represent the latency of corresponding blocks. The parameters are resolved after word length optimization.

A. Word length Optimization

Word length reduction is the first step in taking a functionally validated algorithm to an area-efficient hardware implementation. This step is carried out using an in-house tool for floating-to-fixed point conversion (FFC) [9]. An FFC block is inserted at the node of interest, as shown in Fig. 7, to specify the mean-square error (MSE) due to quantization. The tool does range detection to find integer bits and applies perturbation theory to resolve fractional bits. The overall goal

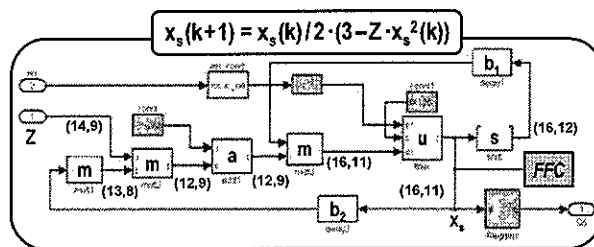


Fig. 7. Simulink entry of $1/\sqrt{()}$ formula.

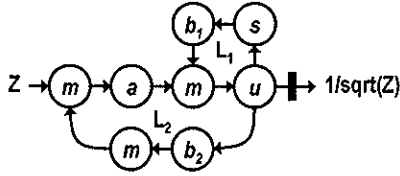


Fig. 8. DFG of $1/\sqrt{()}$ formula.

is to minimize hardware area subject to user-specified MSE criterion. The cost of hardware area is quantified in terms of FPGA resources (slices, LUTs, registers etc.), because the Simulink hardware library is already characterized for FPGA targets. The FPGA resource utilization strongly correlates with an equivalent ASIC gate complexity.

Word length optimization is based on a simulation-based input-data-dependent approach. For large systems requiring long simulation times, guard bits can be inserted. Optimal word lengths for the $1/\sqrt{()}$ algorithm, based on a sine wave input covering a full data range, are shown in Fig. 7.

B. Block Characterization

After the word lengths have been determined, a mixed-mode simulation is performed to compare the fixed and floating point outputs. Following that, blocks with pre-determined word length are characterized in the latency versus cycle time space as discussed in Section I and illustrated in Fig. 2. The characterization results are derived from a full physical synthesis for each block. Given a small variation in multiplier word lengths, characterization of a 16-bit multiplier was sufficient for the $1/\sqrt{()}$ example.

C. Loop Retiming

Characterization results are also carried over to loop retiming. The procedure first assigns latency to building blocks in Simulink using the data-flow graph (DFG) representation of an algorithm. The DFG for the $1/\sqrt{()}$ algorithm is shown in Fig. 8. Based on the DFG, the following loop constraints are formulated:

$$\begin{aligned} \text{Loop } L_1: m + u + s + b_1 &= N \\ \text{Loop } L_2: 3m + a + u + b_2 &= N \end{aligned} \quad (1)$$

where m , a , u , s are the latency of the multiplier, adder, mux, and shifter, respectively; b_1 and b_2 are balancing registers; and N is the number of parallel data streams. In this case, $N = 64$ and target clock rate $f_{\text{clk}} = 64\text{MHz}$. V_{DD} needs to be reduced to maximize energy-efficiency, so the largest values of m , u , a that satisfy Eq. (a) are selected. In this example, $m = 8$, $a = 2$, $u = s = 1$, resulting in a $V_{\text{DD}}^{\text{opt}} = 0.6\text{V}$. This operating point corresponds to energy-delay sensitivity [11] of 1.2 for a pipeline stage, resulting in a good energy-delay tradeoff. In case the resulting sensitivity was too small yielding small returns in energy for large increase in delay, time-multiplexing of the underlying blocks would have been used in order to reduce the area.

After fixing the latency of building blocks, register retiming of the building blocks is subsequently performed during the logic synthesis step. The latency assignment method ensures

that no registers are pushed around the loops. The retiming problem is therefore reduced to retiming of simple feed-forward blocks followed by an incremental compile at the top level. The design is finally carried through the physical synthesis steps to obtain final estimates for power and area: 0.18mW and 0.07mm^2 in 90nm CMOS yielding a 0.5GOPS/mW of power efficiency (op is defined as a 12-bit add equivalent operation).

IV. DESIGN EXAMPLES

More examples are presented to highlight various features of the flow including hierarchical synthesis, integration of IP macros, and FPGA emulation.

A. MIMO Signal Processing: Hierarchical Retiming

The first example is a multidimensional signal processing algorithm for decoupling of multi-input multi-output (MIMO) wireless channels [4, 10]. This example hierarchically extends the loop retiming procedure over $1/\sqrt{()}$ example from Section III. The DFG for the algorithm is illustrated in Fig. 8, where $1/\sqrt{()}$ and div blocks indicate iterative computations from the previous level of hierarchy, level 1. At hierarchy level 2, new loop constraints are included in addition to the loop constraints from level 1, to formulate the top-level problem. The loops going over $\text{div}^{(1)}$ and $\text{sqrt}^{(1)}$ blocks take into account I/O latency of the underlying blocks.

A 4×4 MIMO algorithm that operates over 16 sub-carriers is realized in an ASIC [4]. The ASIC is optimized for 0.4V achieving 2GOPS/mW (12-bit add equivalent). The resulting V_{DD} differs from that for $1/\sqrt{()}$ due to additional top-level constraints on $1/\sqrt{()}$. The resulting energy-delay sensitivity for in this design is 0.8.

B. Sobel Edge Detector: IP Macros and FPGA Emulation

The second example, Sobel edge detection algorithm, is used to illustrate FPGA emulation capabilities. The edge detection is implemented in Simulink as shown in Fig. 9. Upon converting an RGB input stream to intensity Y , two row

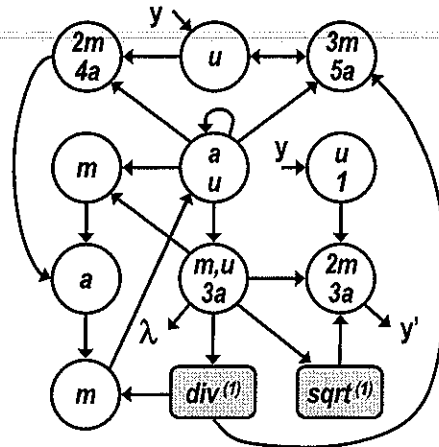


Fig. 8. DFG of eigen-mode decomposition.

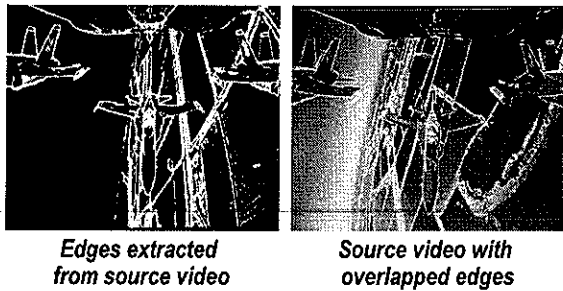


Fig. 10. FPGA emulation of Sobel edge detection algorithm.

filters based on SRAM IP are used for 3×3 convolutions. Absolute values of the horizontal and vertical edges are added for a magnitude estimate; the results are scaled and propagated through threshold circuitry to generate 1-bit edge map.

Figure 10 is a snapshot of real-time video processing that illustrates edge detection. A high-definition (HD) quality video source is processed during live view and the extracted edge map is overlaid on top of the source video (right figure). Gain, threshold, and video overlay modes are user-specified parameters.

C. Other Examples

Several other examples were also implemented using the proposed methodology. Examples range from a multi-standard FIR filter to a pulsed UWB receiver. While the MIMO signal processing was a demonstration of concept and benchmark for energy/area-efficiency of hardwired macros, the FIR filter [12, 13] quantifies the cost of flexibility for multi-standard operability. A pulse-based UWB design was also synthesized to demonstrate the ability to perform high data-rate (1GS/s) signal processing at very low voltage (target 0.35V). This is achieved using a parallel-4 architecture of the pulse-matching filter in an ASIC consisting of 200k gates. The ASIC is currently being tested.

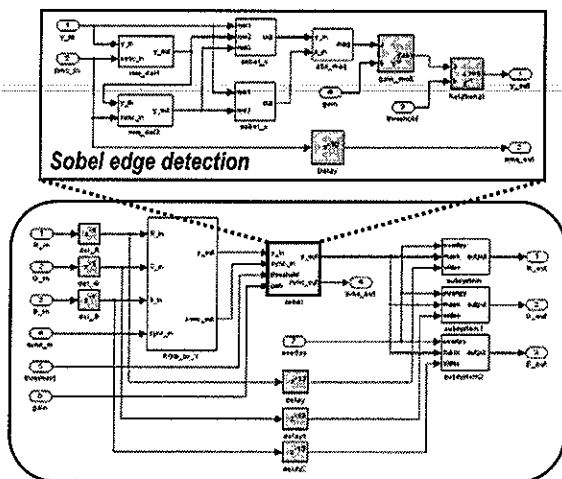


Fig. 9 Simulink entry of Sobel edge detection.

V. CONCLUSION

Graphical block-based design entry restores the missing link between algorithm and circuit designers. It enables a single design entry, architecture optimization, and final hardware verification within Matlab/Simulink environment, which is widely adopted by the algorithmic designers.

Re-targetable flow enables tracking of technology features in the process of architecture selection. Characterization of design-specific blocks with respect to power, speed, and area greatly reduces design complexity. Datapath logic and block level characterization provide simple way for guiding the architecture selection in the area-energy-delay space.

Using the proposed approach, highly energy-efficient ASIC implementations are possible, as demonstrated on a multi-channel 4×4 MIMO channel decoupling algorithm achieving a 2GOPS/mW in a 90nm CMOS process.

ACKNOWLEDGMENTS

This research is supported in part from "Robust, Rapid and Wireless Chip Design" project sponsored by MARCO on contract #02919 via Carnegie Mellon University. The authors acknowledge technology support from ST Microelectronics and Xilinx, and other BWRC member companies. Kimmo Kuusilinna, Rhett Davis, Chen Chang, Pierre Yves-Droz, and Borivoje Nikolić are acknowledged for their contributions in the flow development and technical discussions.

REFERENCES

- [1] W.R. Davis et al., "A Design Environment for High-Throughput Low-Power Dedicated Signal Processing Systems," *IEEE J. Solid-State Circuits*, vol. 37, no. 3, pp. 420-431, March 2002.
- [2] <http://www.mathworks.com/products/simulink>
- [3] <http://bwrc.eecs.berkeley.edu/Research/Insecta/default.htm>
- [4] D. Marković, R.W. Brodersen, and B. Nikolić, "A 70GOPS, 34mW Multi-Carrier MIMO Chip in 3.5mm^2 ," in *2006 Symp. on VLSI Circuits Dig. Tech. Papers (VLSI'06)*, June 2006, pp. 158-159.
- [5] K. Kuusilinna, C. Chang, M.J. Ammer, B. Richards, R.W. Brodersen, "Designing BEE: A Hardware Emulation Engine for Signal Processing in Low-Power Wireless Applications," *EURASIP Journal on Applied Signal Processing*, pp. 502-513, 2003.
- [6] C. Chang et al., "Rapid Design and Analysis of Communication Systems Using the BEE Hardware Emulation Environment," in *Proc. IEEE Rapid System Prototyping Workshop*, June 2003.
- [7] K.K. Parhi, *VLSI Digital Signal Processing Systems*, New York: NY, John Wiley & Sons, 1999.
- [8] C. Shi and R.W. Brodersen, "Automated Fixed-point Data-type Optimization Tool for Signal Processing and Communication Systems," in *Proc. IEEE Design Automation Conf.*, pp. 478-483, June 2004.
- [9] C.V. Ramamoorthy, J.R. Goodman, and K.H. Kim, "Some Properties of Iterative Square-Rooting Methods Using High-Speed Multiplication," *IEEE Trans. Computers*, vol. C-21, no. 8, pp. 837-847, 1972.
- [10] A.S.Y. Poon, D.N.C. Tse, and R.W. Brodersen, "An adaptive multiple-antenna transceiver for slowly flat-fading channels," *IEEE Trans. Communications*, vol. 51, no. 13, pp. 1820-1827, Nov. 2003.
- [11] D. Marković, V. Stojanović, B. Nikolić, M.A. Horowitz, and R.W. Brodersen, "Methods for True Energy-Performance Optimization," *IEEE J. Solid-State Circuits*, vol. 39, no. 8, pp. 1282-1293, Aug. 2004.
- [12] M. Ler, *An Energy Efficient Reconfigurable FIR Architecture for a Multi-Protocol Digital Front-End*, Master's Thesis, Dept. of EECS, University of California, Berkeley, Spring 2006.
- [13] F. Sheikh et al., "Power-Performance Optimal DSP Architectures and Implementation," in *Proc. 40th Asilomar Conf. on Signals, Systems, and Computers*, Nov 2006