Power and Area Efficient VLSI Architectures for Communication Signal Processing

Dejan Markovic, Borivoje Nikolic, Robert W. Brodersen
Berkeley Wireless Research Center, University of California at Berkeley
2108 Allston Way, Suite 200, Berkeley, CA 94704, USA

***Abstract*—A methodology for VLSI realization of signal processing algorithms for wireless communications is presented that optimizes architecture for reduced power and area. When power is limited, optimal architecture represents a point on the best power-area tradeoff curve that is obtained by balancing the algorithm throughput with the power-performance tradeoff of the underlying building blocks. Architectural optimization is done in the graphical Matlab/Simulink environment, which is also used for algorithm verification. Hardware description language produced by Simulink enables algorithm emulation on the FPGA and also serves as design entry for the chip realization. This is illustrated on complex multi-dimensional algorithms such as wideband MIMO channel decoupling through singular value decomposition (SVD) using 16 sub-carriers.**

Keywords—Circuit synthesis, design methodology, architecture, adaptive signal processing, matrix decomposition, MIMO systems.

I. Introduction

The growing demand for data-centric wireless connectivity has inspired the development and realization of complex signal processing algorithms such as those used in multiple-input multiple-output (MIMO) communication. At the same time, complexity of the devices has been steadily growing due to a need for multi-mode, multi-standard functionality. In the past, the growth of complexity of practically implemented wireless algorithms has tracked the improvements provided through the technology scaling. The recent need to support MIMO algorithms has increased the need for more power and cost efficient radios.

The goal of implementing a communications chip is to meet the functionality, throughput and latency of the underlying standard with minimum power and area cost. Most common techniques to minimize power or area (as a dominant measure of cost) are architectural. However, these techniques have been largely heuristic, and there is no established systematic way for trading off throughput, power and area.

With technology scaling, designers have more options in selecting supply voltages and transistor thresholds in addition to various circuit design techniques such as the use of sleep modes. All these techniques present some power-performance tradeoff, which makes architecture selection more complicated and also more interesting. The goal is therefore to develop a methodology that simultaneously minimizes power and area for given throughput and latency constraints.

In this paper, various architectural techniques in the energy-area-performance space are evaluated in order to minimize power and area. Prior work applied similar techniques to simple building blocks such as FIR filters, [1], in standard VLSI design environment. The methodology presented in this paper is scalable to large degrees of complexity. This work uses the Matlab/Simulink environment familiar to both theorists and implementers, thus giving practical insight to algorithm developers as well as better understanding of the algorithms by VLSI architects.

II. Choosing an Optimal Architecture

Optimal VLSI architecture is technology dependent, which requires characterization of main functional blocks for speed, power, and area. This information is used to navigate the architectural optimization procedure that is based on balancing the algorithm throughput requirement with the capability of the underlying basic building blocks. Data throughput and latency are main constraints in chip realizations. Data throughput is interesting for optimization since, for a given architecture, the throughput can be related to the frequency of operation.

The key information that provides basis for optimization is technology specific energy-delay tradeoff in datapath logic as shown in Fig. 1. This tradeoff exists because the energy needed to operate digital logic gates is related to their speed. The tradeoff is obtained by adjusting design parameters such as gate size, supply and threshold voltage. Introducing a new technology shifts the entire *E-D* tradeoff curve toward lower energy and delay. The architecture is energy optimal if its *E-D* tradeoff curve has the same slope as the underlying datapath logic. A good tradeoff point is indicated in Fig. 1. Otherwise the design would have a high cost in terms of energy or delay.

By using basic concepts of parallelism and time multiplexing an algorithm can be mapped into a range of architectures with widely varying throughput and latency. Architectural transformations such as data-stream interleaving, loop retiming and folding support more complex operations with concurrent or time-serial execution, which may also involve feedback loops.

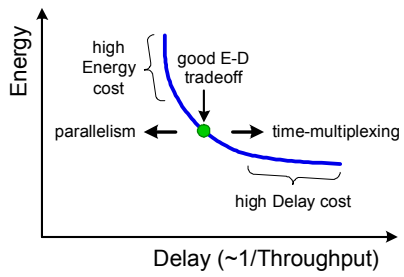

Fig. 1. Energy-delay tradeoff in digital circuits.

1-4244-0355-3/06/$20.00 (c) 2006 IEEE

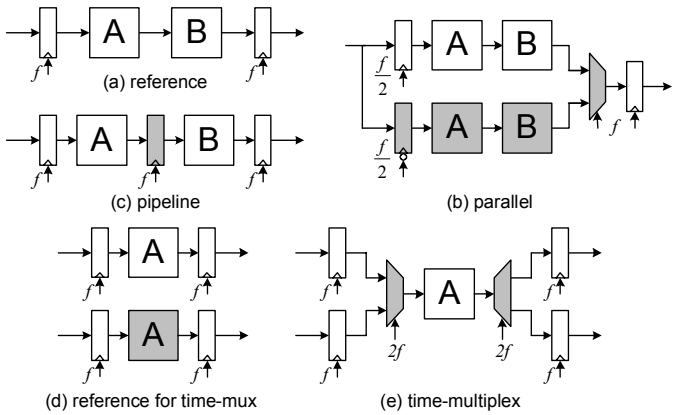

Fig. 2. Basic micro-architectural techniques: reference architecture (a), and its parallel (b) and pipelined (c) equivalents. Reference architecture (d) for time-multiplexing (e). Area overhead is indicated by shaded blocks.

A. Parallelism and Time Multiplexing

The concepts of parallelism and time-multiplexing are well-known in architecture design. Parallelism combined with adjustment in supply voltage improves the energy by slowing down the clock and distributing computation over several parallel branches. Alternatively, when using constant clock frequencies, parallelism trades off the increase in area for higher throughputs, Fig. 2(b), [2]. Time-multiplexing as shown in Fig. 2(d)-(e) does the opposite: it reduces the area by repeating computation on the same hardware unit, but needs a higher clock rate and higher supply voltage to maintain the throughput. Alternatively, it can reduce performance by maintaining the clock rate.

Similar tradeoffs as that from parallelism are available through pipelining, e.g. by inserting extra pipeline register between logic blocks *A* and *B*, Fig. 2(c). Pipelining has much smaller area overhead than parallelism, but is generally more difficult to implement since it requires design re-partitioning.

Basic energy-performance tradeoff in Fig. 1 provides good insights into techniques of introducing parallelism and time-multiplexing. Starting from a good *E-D* tradeoff point and moving toward higher speed (lower delay), we observe a steep increase in required energy. To conserve the energy, parallelism shifts the curve to the left, which can also improve the performance. Time-multiplexing reduces the area when circuit blocks can provide excess performance. It effectively shifts the curve to the right, maintaining a good *E-D* tradeoff.

B. Data-Stream Interleaving

Data-stream interleaving is another technique to improve area efficiency. In essence it is a way of time multiplexing the data. The case when the algorithm contains a recursion is analyzed as described with the simple difference equation:

$$z(k) = x(k) + c \cdot z(k-1). \quad (1)$$

The signal processing representation of the difference equation above is shown in Fig. 3(a). Algorithmic latency of one symbol period is represented with explicit register between $z(k)$ and multiply block. Symbol rate is defined by clock frequency f_{Clk}. This simple model abstracts away any

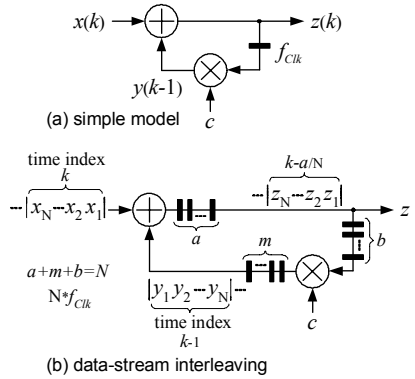

Fig. 3. Concept of data-stream interleaving (feedback example).

extra latency in *add* and *multiply* blocks. From the technology perspective, an *N* bit *add* operation is less complex in terms of the number of gates and area than an *N*-by-*N* multiplication. In order to balance the complexity of datapath logic blocks, *add* and *multiply* operations need to have different latency.

A more realistic model of Eq. (1) that captures the different latency of addition and multiplication is shown in Fig. 3(b). A number of *a* and *m* pipeline registers has been assigned to the adder and the multiplier, respectively, to balance the throughput by increasing latency. By going around the loop, the feedback signal $y(k-1)$ will have an increased latency of $a + m$. By increasing the clock rate by a factor $a + m$, the same algorithmic latency can be maintained. In order to fill the added pipeline with useful computation, multiple independent signal streams can be interleaved onto the same hardware. If the number of independent signals *N* is greater than $a + m$, then $b = N-a-m$ of additional pipeline registers is required to maintain the latency.

Interleaving effectively improves the area efficiency by sharing data-path logic across independent streams of data. A practical use of carrier interleaving is in multi-carrier communications, where the independent narrow-band sub-carrier streams can be time-interleaved.

C. Folding

Folding, similarly to data-stream interleaving, reduces the area. Assume serially ordered execution of some algorithmic operation *Alg* as shown in Fig. 4(a). The first block in the chain takes independent data samples $y_1(k)$, all other blocks take the result from the previous block. The concept of folding is shown in Fig. 4(b), [4]. Input to *Alg* is provided by

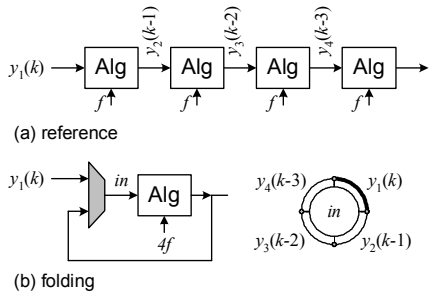

Fig. 4. Concept of folding: (a) time-serial computation, (b) operation folding. Block *Alg* performs some algorithmic operation.

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.

a multiplexer, which selects external data y_1 or internally generated results y_2, y_3, y_4. During the first quarter of the symbol period, up-sampled and interleaved data y_1 is used. The output of *Alg* is then folded over in time, back to its input, to compute y_2, y_3, and y_4. Re-ordering of y_1 samples is needed to align data at input *in*, as shown in the life-chart in Fig. 4(b). Up-sampling the clock in block *Alg* is necessary to sustain external (y_1) throughput rate.

If *Alg* is a simple feed-forward unit, no additional pipeline registers are needed. In case *Alg* block has internal feedback loops, additional pipelining is necessary to keep the internal states. This raises the issue of how to optimally distribute pipeline registers around the loop to maximize throughput, which is addressed next.

D. Loop Retiming

Loop retiming is a technique of distributing pipeline registers around recursive loops. The goal is to assign the right amount of latency to basic functional building blocks and then distribute the pipeline registers inside the blocks such that all internal datapath logic blocks lay at the same point (shown in Fig. 1) in the energy-delay space. This guarantees top-level optimality. As in the case of interleaving, additional balancing registers may be needed to ensure equal loop latency in all recursive loops.

The approach of loop retiming is illustrated in Fig. 5 on the example of iterative divider. This is a simple example with two nested loops, but the concepts are general. In a data-flow graph representation of a function, *a*, *m*, *u* are the latency of pre-characterized library blocks. This case uses adder (*a*), multiplier (*m*), and multiplexer (*u*). For each of the loops, the loop constraints are formulated as in Eq. (2).

$$\begin{aligned} L_1: &\quad m + u + b_1 = N \\ L_2: &\quad 2m + a + u + b_2 = N \end{aligned} \tag{2}$$

For a given throughput constraint, the solution of Eq. (2) is a set of positive integers (*m*, *a*, *u*) that corresponds to equal clock period, adding b_1 and b_2 balancing registers as needed to satisfy loop latency *N*. Accounting for different latency of library blocks nicely extends the retiming algorithm from [5].

Retiming strategy can be easily expanded to traverse the layers of hierarchy. Each block is characterized with latency from its primary inputs (i) to its primary outputs (o) as well as loop constraints inside the block. This information is used to derive i/o latencies and loop constraints at the next level of hierarchy. At the top-level, loop constraints from all lower levels are thus considered.

E. Delayed Iteration

Delayed iteration occurs when majority of loops have similar latency, while only a few loops need to compute longer. Adding balancing registers to all the non-critical loops to compensate for this effect would solve the problem, but in a very power and area inefficient way. The idea is to use delayed iteration in those few loops and take delayed sample at time instant $k-1$ instead of taking it at time instant k. If this is algorithmically possible, requirements for power and area can be much improved.

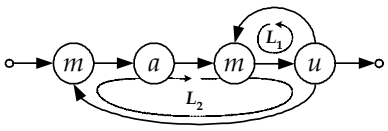

Fig. 5. Data-flow graph model of iterative division. (*m*, *a*, *u* indicate latency)

III. Simulink-based Design Framework

We use Matlab/Simulink environment to avoid design re-entry, which is a standard practice today. A design is entered in various forms by different engineering teams, resulting in heterogeneous design descriptions. Algorithm developers tend to work in Matlab environment, which has an array of built-in functions convenient for quick algorithm verification. C code is another sequential processing entry, which requires more sophisticated coding skills. Still, neither of the representations captures the architecture and information about sampling rate. The architectural description is then created by hardware designers who have to completely re-enter the design in HDL.

Matlab/Simulink design environment enables theorists and implementers to work together. An algorithm is entered only once is a graphical block form, which provides timed data-flow representation of the design and abstract view of design architecture. With technology-specific data for speed, power, and area of functional blocks, algorithm designers can explore the implementation space while remaining in Simulink environment to verify the algorithm.

For practical realization, algorithm functionality is entered using hardware-equivalent blocks from Xilinx library as shown in Fig. 6. This library has basic arithmetic operators such as *add*, *multiply*, *shift*, *mux* etc. with notion of hardware parameters such as latency and word-lengths. The Simulink description of the block interconnects is used to generate hardware description for mapping the design onto an FPGA for hardware emulation or used by ASIC place and route tools for an ASIC. Test vectors generated in Simulink are used for functional verification in both FPGA and ASIC flows.

A. "Chip-in-a-Day" Design Flow

Our "chip-in-a-day" design flow is illustrated in Fig. 7. We start with Simulink design description using Xilinx block-set. After functional verification of the design, we use an in-house tool for wordlength reduction to reduce area. The floating-to-fixed point conversion (FFC) tool minimizes hardware cost (FPGA utilization) subject to user defined performance measures such as MSE error due to quantization. Integer part is extracted from node profiling and range detection, while fractional word-size optimization is done by perturbation

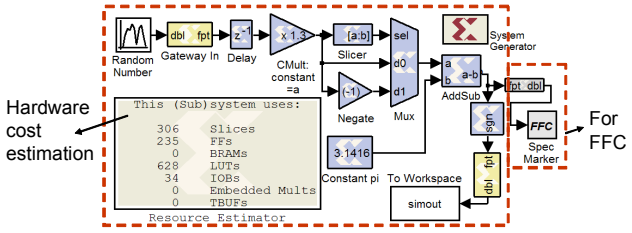

Fig. 6. Illustration of FFC-enhanced Simulink model.

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.

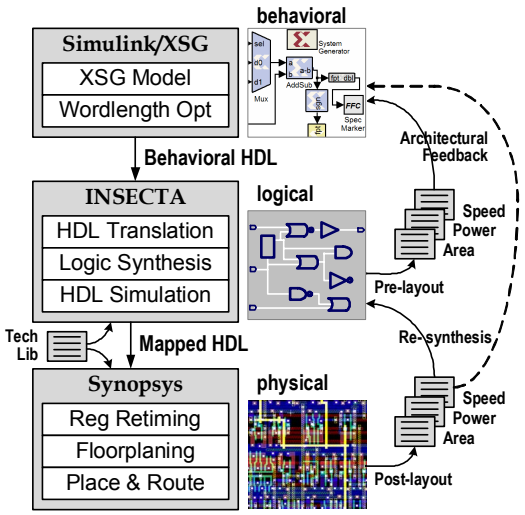

Fig. 7. Simulink-based "Chip-in-a-Day" design flow.

theory, [6]. Once bit-true cycle-accurate behavior is obtained, hardware description language (HDL) for technology mapping is then created using Xilinx System Generator (XSG) netlist utility. This HDL code can be used for FPGA emulation.

To avoid re-entry at the HDL level, an in-house tool translates FPGA dialect into format supported by commercial implementation tools (Synopsys). After logic synthesis that performs technology mapping, the resulting mapped netlist is verified for functionality using test vectors from Simulink. The netlist can be further optimized to do register retiming and logic sizing, before going to the final stage of physical layout synthesis, which is highly automated. This is where designers often stop after obtaining functionally correct behavior.

Making correct architectural decisions in Simulink relies on architectural feedback from synthesis. This is needed as early in the flow as possible to avoid unnecessary iterations. First estimates for area, speed, and power are fed back after initial logic synthesis, Fig. 7. These results can be refined if desired by more accurate estimates from physical layout synthesis. Efficient algorithm implementation is possible without requiring much post synthesis information in Simulink, with extensive characterization of basic building blocks.

B. *Characterization Methodology*

The goal in characterization is to augment XSG block-set with technology-dependent information for speed, power, and area. The complexity of basic and most commonly used library blocks is very low (granularity of *add*, *multiply*, *shift*, *mux*, *register* etc.), so full characterization over a range of latency and word-size parameters is possible.

Approach to block-level characterization is illustrated in Fig. 8 for cases of *add* and *multiply* operations. These two blocks differ in arithmetic complexity (reflected in latency), so we characterize the speed in terms of cycle time, which is a global parameter. Equal cycle time means that complexity of logic blocks between two pipeline registers has to be the same in all blocks, which allows hierarchical expansion around this block-set. Each point along the latency vs. cycle time curve is also characterized for power and area. Estimates from

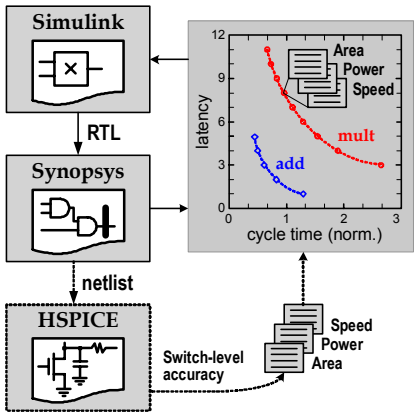

Fig. 8. Block-level characterization (*add*, *mult* examples).

physical synthesis can be further refined with switch-level accuracy, which is very time-consuming. This is usually done for just a few blocks, with results extrapolated to other points.

The Simulink block library is also characterized for area utilization of regular FPGA fabric (look-up tables, flip-flops), so users can obtain quick estimate of hardware cost in terms of FPGA resources. Translating into ASIC terminology, 10,000 FPGA slices $\Leftrightarrow$ 1mm^2 of layout area (~80% layout density) in 90nm CMOS. This relationship is obtained from linear extrapolation of area estimates for several examples that are about an order of magnitude apart from each other in terms of complexity ranging from simple arithmetic operations such as *add* and *multiply*, to complex matrix algorithms. This way, we can obtain early estimate of chip area at the Simulink level.

It is common practice in IC design to normalize silicon area to the area of a gate from standard cell library (e.g. 2-input NAND). At higher levels of granularity, we can also scale the area by the area of some basic operation such as addition.

C. *Architectural Transformations*

Architectural optimization approach is illustrated in Fig. 9. The goal is to drive the design to a desired *E-D* tradeoff point (e.g. reference point), while minimizing the area. This is done manually by the designer using transformations shown in Fig. 9. For example, time-multiplexing saves the area, but requires increased supply voltage which results in an increase in energy per operation. Parallelism and pipelining save the energy through supply voltage scaling, but increase the area. It is important to realize that there is no unique architectural solution since energy-efficiency can be traded for area.

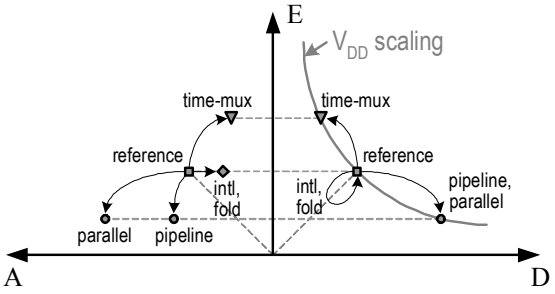

Fig. 9. Architectural transformations in Area-Energy-Delay space.

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.

When power is limited, which is often the case in practical designs, the area can be minimized as follows. Given power limit, desired power efficiency is calculated from the required amount of functionality, using a known relationship between the energy per operation and supply voltage. This determines the desired operating point on *E-D* line. Optimization procedure then applies appropriate transformations as in Fig. 9 to reach the desired point.

IV. Examples

The use of architectural techniques is illustrated on few examples. We first look at simple iterative square rooting and division, then analyze more complex vector-based operations including Grahm Schmidt orthogonalization procedure and eigen-mode decomposition.

A. Iterative Square Rooting and Division

Square rooting and division are operations common to many wireless communication algorithms, [7]. Computing the norm requires inverse square rooting, for example. More specifically, in adaptive algorithms input argument changes relatively slowly, which is often the case in wireless channels. The slow-varying condition makes the iterative approach attractive for practical realization.

Among the algorithms for iterative square rooting and division, a method based on Newton-Rhapson formulas is attractive because of its favorable convergence properties. Survey of these algorithms can be found in [8]. Equations (3) and (4) describe inverse square rooting and division:

$$x_s(k+1) = \frac{x_s(k)}{2} \cdot (3 - N \cdot x_s(k)^2), \quad (3)$$

$$x_d(k+1) = x_d(k) \cdot (2 - N \cdot x_d(k)), \quad (4)$$

where N is the input argument, x_s describes inverse square rooting, and x_d describes division. Analysis of error dynamics reveals quadratic convergence: $e_s(k+1) = -0.5 e_s(k)^2 \cdot (3 + e_s(k))$, $e_d(k+1) = -e_d(k)^2$. In implementation terms, this means that each iteration resolves two bits of accuracy, [8].

The number of iterations required for algorithms to converge is a function of initial condition and error dynamics. Table 1 summarizes convergence properties for a 25% and 50% initial error, for various accuracies of the final answer. Even for very large initial error of 50%, only five iterations are needed to achieve accuracy within 0.1%. The results in Table 1 also suggest that the error quickly decreases in every iteration. So, if the result of current iteration is taken as the initial condition for the next iteration, under slow-varying input the answer can be obtained in only one iteration.

Main architectural techniques applied in this example are loop retiming (the model in Fig. 5 illustrates division) and data-stream interleaving for multiple operands. Loop constraint N is equal to the number of interleaved operands.

TABLE I
Convergence Speed of Iterative *Sqrt* and *Div* Algorithms

Target relative error (%)	0.1%	1%	5%	10%
e_0: 50%, # iter (sqrt / div)	5 / 4	5 / 3	4 / 3	3 / 2
e_0: 25%, # iter (sqrt / div)	3 / 3	3 / 2	2 / 2	2 / 1

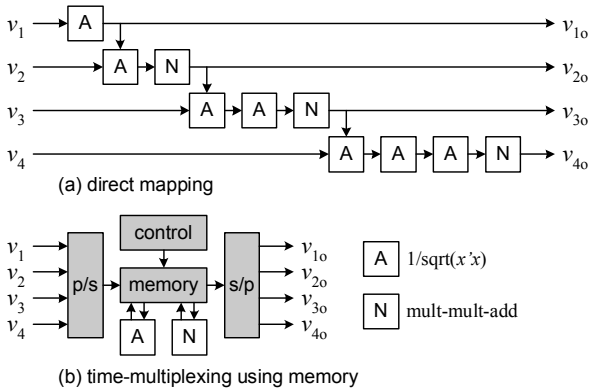

Fig. 10. Grahm Schmidt Orthodonalization: (a) direct mapped architecture, (b) memory based time-multiplexing approach.

For the case with 64 narrow-band 1MHz wide sub-carriers, required throughput is 64MHz. The total (power, area) is estimated at (180µW, 0.07 mm^2) for inverse square rooting and (120µW, 0.05mm^2) for division. This example illustrated an architecture for scalar operations. In the next example, we analyze vector arithmetic.

B. Grahm-Schmidt Orthogonalization

Another common technique in communication signal processing is the projection of a set of vectors onto an orthogonal base. A popular approach is the method of Grahm Schmidt orthogonalization (GSO). In practice, it is also possible that during adaptation, phase shifts and magnitude changes cause vectors to loose orthogonality. GSO is then periodically applied to rectify this. The concept of GSO is illustrated in Fig. 10. Two basic operations are indicated in boxes *A* and *N*. In the figure, $\boldsymbol{v}_1$-$\boldsymbol{v}_4$ are complex input vectors, and $\boldsymbol{v}_{1o}$-$\boldsymbol{v}_{4o}$ is their orthogonal form. Due to repetitive use of operations *A* and *N*, this algorithm is suitable for time-multiplexing if the required data rate is low.

Regular time-multiplexing from Fig. 2 is possible, but this would result in a sizeable interconnect overhead. Each of the lines in Fig. 10 represents a complex vector with N bits. For example, a vector of dimension 4 with 16 bits for in-phase and quadrature components would mean 4×16×2 = 128 bits, and this number is further multiplied by the level of time-multiplexing (4 for block *N*, 6 for block *A*). So, traditional time-multiplexing is not favorable in terms of interconnect complexity. To reduce wire complexity, time-multiplexing can be implemented using memory, as shown in Fig. 10(b). Only 128 bits now need to be exchanged between memory and processing blocks, reducing the routing overhead. Summary of silicon area and routing complexity is given in Table 2 for cases of direct implementation, regular time-multiplexing, and memory based time-multiplexing. Savings in routing complexity directly translate to power reduction since power spent in switching of interconnect wires is also reduced.

TABLE II
Summary of GSO Implementation Features

Architecture	Direct mapped	Time-mux (TM)	TM w/ memory
Area (silicon / FPGA)	2.6 mm^2 / 60k	1.2 mm^2 / 14k	1.0 mm^2 / 14k
Total wire length	6.6 m	3.2 m	2.2 m

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.

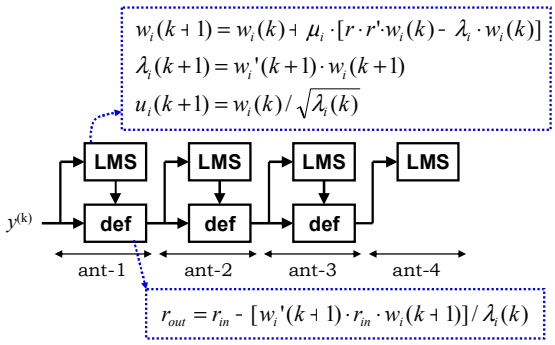

Fig. 11. Eigen-mode decomposition algorithm from [9].

C. *Singular Value Decomposition*

As a final example, we analyze architecture for estimation of indoor wireless channels through adaptive Singular Value Decomposition (SVD), [9]. The core of the SVD algorithm is in estimation of U and Σ matrices, summarized in Fig. 11. As illustrated on practical example of a 4×4 MIMO system, the algorithm extracts eigen-pairs ($\boldsymbol{u_i}$, λ_i) through successive rank reduction, where $\boldsymbol{u_i}$ is i^{th} eigen-vector, and λ_i is i^{th} eigenvalue, i=1-4. Deflation (*def*) block does successive rank reduction; LMS block computes the eigen-pairs. Each spatial sub-channel introduces a pair of LSM-*def* processing elements. Within LMS and *def* blocks there is the inverse square rooting operation from the earlier example. Step size μ_i inside the LMS block is adaptively adjusted based on estimated eigen-values, $\mu_i \propto 1/\lambda_i$. Since eigen-values are slowly changing over time, iterative division is employed for adaptive gear-shifting.

This example is an interesting case from the architectural standpoint, as discussed below. A concept of loop retiming in square rooting block is hierarchically expanded to include the whole algorithm. Narrow-band algorithm described in [9] is extended to wide-band MIMO case by introducing multiple sub-carriers. Each sub-carrier performs the same operation, so data-stream interleaving is applied to sub-carriers. In this example, the case with 16 sub-carriers is assumed (usually not all sub-carriers are needed for channel estimation; other sub-channels can be estimated using interpolation), with a 1 Msymbol/s data rate on each sub-carrier. The organization in Fig. 11 is also convenient for folding over the antennas for further area reduction as shown in Fig. 9.

By sharing logic blocks common to operations indicated in Fig. 11, the algorithm is synthesized in only 3.5mm^2 of area in a 90nm CMOS technology (for comparison, Simulink block model takes 35k FPGA slices). The arithmetic complexity of this algorithm is 70 GOPS (12-bit equivalent *add*). Estimated power consumption from logic synthesis is 21mW, with an achievable 250Mbps throughput using adaptive PSK modulation. This algorithm is relatively new, so no existing VLSI realizations exist for comparison purposes.

V. Conclusion

This paper presented architectural techniques for power and area efficient VLSI realization of signal processing algorithms for wireless communications. The choice of architecture is highly influenced by the energy-delay tradeoffs of underlying technology and data throughput of the algorithm.

Highly automated algorithm implementation is possible starting with the algorithm description in a simple graphical Matlab/Simulink environment. The algorithm can be rapidly evaluated on an FPGA or realized in ASIC. Characterization of few basic blocks from Simulink hardware library provides much needed information for architectural design early at the Simulink level, without the need for extra iterations. This provides a unified framework for algorithm developers and implementers. Several examples of varying complexity are discussed to illustrate the methodology for power and area minimization.

Looking forward, it is interesting to consider the cost of adding flexibility. Two possible directions are the following: investigate flexibility required for the execution of common operations across multiple standards, and study flexibility of having multi-functionality on a single hardware unit.

Acknowledgment

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 members of the BWRC.

References

[1] T. Gemmeke, M. Gansen, H.J. Stockmanns, and T.G. Noll, "Design Optimization of Low-Power High-Performance DSP Building Blocks," *IEEE J. Solid-State Circuits*, vol. 39, no. 7, pp. 1131-1139, July 2004.

[2] A.P. Chandrakasan, S. Sheng, and R.W. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, no. 4, pp. 473-484, Apr. 1992.

[3] D. Markovic, V. Stojanovic, B. Nikolic, 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.

[4] K.K. Parhi, *VLSI Digital Signal Processing Systems*, New York: NY, John Wiley & Sons, 1999.

[5] Y. Yi, R. Woods, L.K. Ting, and C.F.N. Cowna, "High sampling rate retimed DLMS filter implementation in Virtex-II FPGA," *IEEE Workshop on Signal Processing Systems*, pp. 139-145, Oct. 2002.

[6] 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.

[7] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*, Cambridge University Press, 2005.

[8] 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.

[9] 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.

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.