A PROGRAMMABLE ACCELERATOR FOR NEXT GENERATION WIRELESS COMMUNICATIONS

Karim Mohammed, Babak Daneshrad

Henri Samuelli School of Engineering and applied science, UCLA
UCLA Elec. Engr. BOX 951594, 6731G BH, Los Angeles, CA, 90095-1594, USA
phone: +1-310-382-7775, email: kabbas@ee.ucla.edu

ABSTRACT

We present a MIMO decoder accelerator architecture that offers versatility and re-programmability while maintaining a very high performance-cost metric. The accelerator is meant to address the MIMO decoding bottlenecks associated with the convergence of multiple high speed wireless standards onto a single device. It is scalable in the number of antennas, bandwidth, modulation format, and most importantly, present and emerging decoder algorithms. It features a Harvard-like architecture with complex vector operands and a deeply pipelined fixed-point complex arithmetic processing unit. Memory allows efficient access to operands in matrix form, while a custom state machine enhances performance in light of OFDM. The accelerator shows an advantage of up to 3 orders of magnitude in power-delay product for typical MIMO decoding operations relative to a general purpose DSP.

1. INTRODUCTION

Two prominent trends in wireless communication are the use of multi input multi output (MIMO) processing, and orthogonal frequency division multiplexing (OFDM) to improve data rate and reliability [1]. All trends point to the convergence of multiple MIMO-OFDM standards on a single platform. This motivates an accelerator-like approach to efficiently deliver on the computation-intensive elements of the system. The MIMO decoder is one such component. MIMO processing is computationally intensive due to the need to invert a channel matrix with very low latency. Moreover over time, systems are expected to incorporate a higher number of antennas and more advanced algorithms. Analogous to the use of Viterbi accelerator engines [2] in today's cellular systems; a MIMO decoder accelerator that is programmable in bandwidth, number of antennas, decoder algorithm and modulation format will greatly facilitate the adoption of multi standard, MIMO based solutions. Such an accelerator engine could also greatly accelerate the adoption of MIMO communications on software defined radio (SDR) and cognitive radio (CR) based platforms [3][4]. MIMO decoding is essentially an inversion of a complex matrix channel. This can be achieved using a variety of algorithms with a range of complexity and performance. The choice of algorithm and antenna configuration depends on the expected channel conditions, power budget, available resources, and throughput requirements. MIMO decoders also require a long design cycle if they are to be optimized to the target platform.

Traditionally matrix inversion is simplified by using one of a number of matrix decompositions to transform the channel matrix into a more invertible form [5]. The decomposition usually involves regular arrays (systolic arrays) of processing elements (often CORDIC processors) [6]. QR decomposition leading to an MMSE solution is the traditional approach, but Singular Value Decomposition (SVD) is also efficiently implemented on systolic arrays [7][8]. Systolic arrays deliver a quick, efficient implementation of simple algorithms such as MMSE, but they do not offer an easy tradeoff in cost/performance. Other implementations use custom arithmetic datapaths to deliver optimized solutions for specific algorithms [9]. In this paper we present a MIMO decoder accelerator architecture. The accelerator allows the programmer to define and implement MIMO decoders at will. The accelerator has a processor-like architecture with most of the controls derived from a memory-stored program. The processing core is designed to support a range of complex operations necessary to enable the realization of major MIMO decoding algorithms. This architecture does not benefit from the regular, application specific flow of regular arrays. Nor can it rely on platform or technology specific optimizations as a main driver of high performance. The MIMO accelerator departs radically from a conventional processor in several areas, which deliver an improvement in performance over general purpose processors reaching three orders of magnitude. The accelerator core accepts very wide complex matrix operands and produces complex matrix results. The high access rate required to support this is made possible by a memory map that exploits the matrix/vector nature of the operands in MIMO decoding. The memory map is augmented by sorting circuits at the inputs and outputs of memory that allow the programmer to redefine input and output order without using extra processing cycles. The processing cycle uses properties of OFDM decoding to optimize its flow, and through the use of pre-decoded instructions and proper compiler positioning of critical control signals, the accelerator ensures that the processing pipeline is continually engaged. A programmable dynamic scaling circuit automatically handles intermediate wordlength issues for high dynamic range operations. This allows us to use fixed point processing units which substantially increases the performance of the processing pipeline over a floating point implementation.

2. MINIMUM OPERATION SET

The literature is rich with alternative algorithms for MIMO decoding [6][7][9][10]. We needed to ensure that all major algorithms can be supported on the accelerator. Our approach to addressing this problem is to identify the set of primitive processing elements that form the basis of all major MIMO decoding algorithms. With such a set in hand, the realization of a specific decoder algorithm will translate into the proper sequencing of data among these primitive elements through a program. The major decoding algorithms fall into three categories: Maximum Likelihood solutions (ML) including Sphere Decoding (SD), Singular Value Decomposition (SVD) as an arithmetic aid to linear decoding or as a beamforming tool, and linear decoding algorithms such as MMSE and Zero Forcing (ZF). Matrix decomposition is critical to all these algorithms. Although there are alternative decompositions for some algorithms, QR decomposition is the most practical for hardware application.

Givens rotations are commonly used to realize QRD because they are well suited for hardware implementation using CORDIC [5]. SVD, however, requires various unitary transformations in addition to Givens rotations both to reduce the matrix to pure real and to perform the diagonalization step in Jacobi diagonalization.

The algorithms can be supported by four classes of arithmetic operations: complex multiplication, various unitary transformations, complex addition, and division. These operations spawn a large number of derivatives due to the nature of operands, for example multiplication can be of any combination of matrix and vector operands of any size, and unitary transformations may be left or right. The main difference between the accelerator and a generic vector processor is supporting data to the processor in its variant forms without loss in performance. The associated memory map is discussed in section 4.

3. PROCESSING UNIT

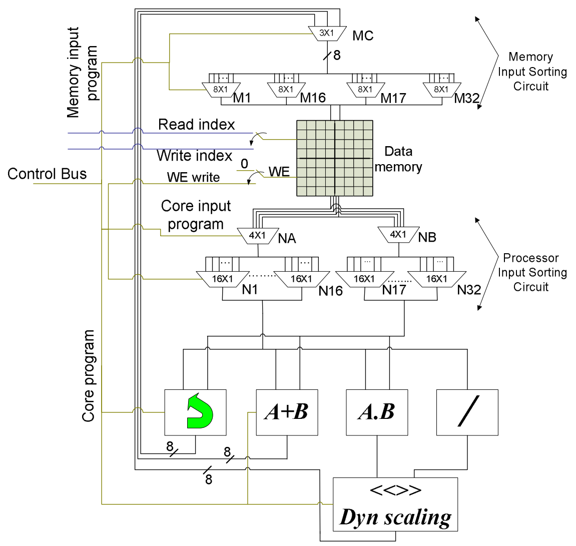

Figure 1 – Accelerator processing core, data memory, and input and output sorters.

The MIMO-accelerator (fig. 1) consists of a processing unit (fig. 2) that supports highly flexible vector coarse operations, connected to a data memory designed to utilize properties of matrix processing in order to allow flexible and highly efficient access. The processing unit (fig. 2) consists of four cores: An inner product core, a scalar division core, a coordinate rotation core, and a vector addition core. Although the design is scalable, we will discuss results for a realization optimized for 4x4 or smaller matrices. Every clock cycle the processor accepts a maximum of 32 complex inputs (necessary to support four dot products) divided into two sets/operands and produces a maximum of 8 complex outputs, corresponding to two output vectors from the addition and rotation cores.

The addition core is a set of 8 complex adders capable of performing two vector additions per cycle. The division core is equally simple, consisting of four dividers, supporting one vector scaling per cycle. The inner product core is four complex dot product units that can perform four 4x1 dot products per cycle. The coordinate rotation core supports a variety of unitary transformations, producing two output vectors per cycle. Normally CORDIC processors [5][6][7][8] are used in unitary transformations due to their good dynamic-range properties, and the fact that they can be implemented without any multipliers. CORDIC units can be combined to perform two angle transformations on complex coordinate pairs. We use a compact realization of complex CORDIC, sometimes referred to as a super CORDIC. The traditional super CORDIC units [6] realize only complex Givens rotations. The circuits in fig. 3 are identical to this realization when the multiplexers are set to mode 0. Combined with the phase processing circuit described below, the other mode settings of the modified CORDICs allow other unitary transformations (e.g. single phase rotation on complex vectors, real-imaginary rotation on a single vector, diagonal exchange rotation, Jacobi transformation, etc.) to be performed with very little added overhead.

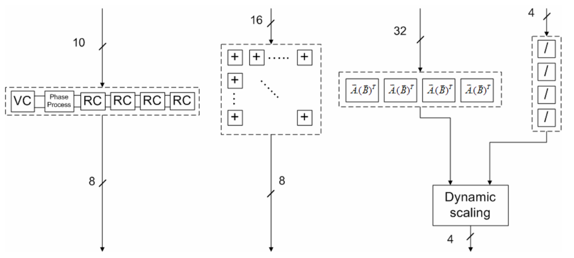

Figure 2 – Accelerator processing core. Cores from right to left: coordinate rotation, addition, inner product, scalar division.

The coordinate rotation core is derived from the traditional systolic array architecture. The rotation core is arranged as a linear array by time-sharing the traditional systolic architecture. The core delivers vector pair outputs. Since decompositions can be easily broken down into vector pair operations; this greatly simplifies programming and offers flexibility in performance-cost tradeoffs. The rotation core (fig. 2) consists of 4 super rotation (SR) CORDIC processors and one super vectoring (SV) CORDIC. Normally vectoring CORDICs generate a phase value derived from an internal microrotation vector which is then accepted and retranslated into a rotation vector in rotation CORDICs [5]. In the accelerator, however, the rotation CORDICs share identical phases, so we designed rotation CORDICs without phase translation units and instead added two common phase processing units that perform the phase interpretation for all 12 CORDIC units. This reduces the total resources of the rotation core by 8.7% while allowing the phase processors to support basic arithmetic manipulations of phases for same-cycle phase pre-processing.

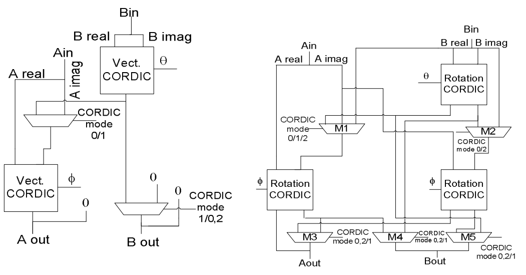

Figure 3 – Building blocks of the rotation core, left multi-mode super vectoring (translation) CORDIC, right multi-mode super rotation CORDIC. Mode 0: Complex Givens rotation. Mode 1: Real Givens rotation on two vectors. Mode 2: Single phase Givens rotation on complex vector pair.

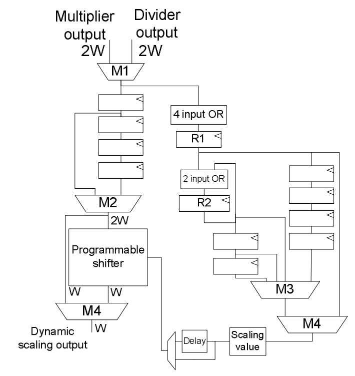

Figure 4 – Dynamic scaling circuit.

The rotation core accepts two vectors of up to 5x1 size. However, only four of the input and output pairs are significant in any clock cycle. In most cases, the rotation core rotates a pair of leading elements in the SV unit reducing one to null and calculating the phases used to achieve such a result, and identically rotates the 3 remaining pairs from the 4x1 vector input pair in 3 SR units. In other cases, however, the core is required to rotate all 4 input pairs by a phase stored in the phase processing unit or externally generated in the accelerator. The rotation core outputs are multiplexed between these two cases, resulting in a 4 element coordinate rotation output. Although this is relatively simple, it has a significant impact in light of the memory access scheme and the outputs of the remaining processing units which are limited to a maximum of two 4x1 vectors.

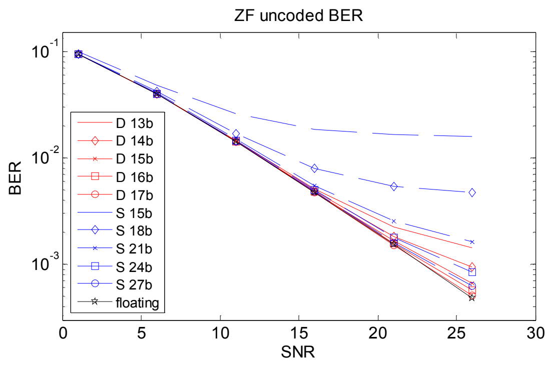

Figure 5 – BER for dynamic and constant scaling ZF.

The accelerator must support multiple algorithms and allow modifications and manipulations at will. Part of this is to provide the programmer with reasonable freedom in the number of operations that can be performed before the processor overflows. Traditionally a fixed point simulation is needed before a hardware implementation is considered. In the accelerator an algorithm is implemented by repeatedly passing operands through the processing unit (an arbitrary number of times), therefore no useful predictions can be made about appropriate intermediate precision requirements. Using very wide wordlength will cause the size of the accelerator to grow rapidly. This is exacerbated by the complex matrix/vector nature of the processing units.

Multiplication and division can result in very fast growth in the wordlength requirement. Complex vector multiplication effectively doubles the wordlength of the inputs. We use a dynamic scaling circuit to manage the precision of the multiplier and divider outputs. The circuit efficiently handles vector and matrix inputs of variable lengths over a variable number of cycles (corresponding to variable matrix sizes). The circuit (Fig. 4) accepts a vector input of size 4 (the size of the output of the multiplication and division cores). Each vector element is a complex number of size 2*W* bits per rail where *W* is the native precision of the processor. A most significant bit (MSB) mask is calculated from the inputs by passing them through OR gates. This mask is then held in R2 and further OR'ed with R1 through another set of OR gates over a variable number of cycles. The final result of the OR operations stored in R2 thus holds most significant bit information over multiple cycles. The contents of R2 are then passed to a scaling value circuit that extracts a shift value from the result. Control signals route appropriately delayed inputs to a programmable shifter where the shift value is held for a period appropriate for the matrix size while results are scaled back to *W* significant bits per rail. The dynamic scaling circuit provides programmable scaling while maintaining the throughput of the processor. The additional latency is absorbed since the latency of the CORDIC core is higher than the combined latencies of the multipliers and the dynamic scaling circuit. Fig. 5 shows simulation results for a zero forcing decoder with statically and dynamically scaled realizations. To operate within 2dB SNR of floating point performance, a circuit with static scaling needs 27 bit multipliers, while a circuit using dynamic scaling only needs 16 bits per rail.

4. MEMORY ACCESS

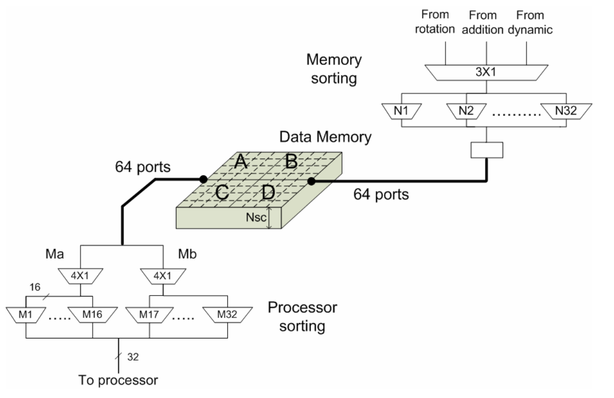

Figure 6 – Data memory map.

The processing core is designed to accept complex matrix or vector inputs. The efficiency of the processor is contingent on a memory access scheme that allows access to any combination of matrix operands in a single cycle. Additionally to distinguish between different antenna configurations, the programmer needs to be able to define how, where, and which results are stored back to memory.

The processing core has a maximum of 32 complex operand inputs. The programmer may need up to a maximum of four 4x4 matrices to store intermediate results or observation vectors while processing the 4x4 channel matrix. If all 4 matrices are stored in a single block of memory and we rely on memory address to access elements, a serious bottleneck is created at the data bus, potentially requiring the processor to wait 40 cycles for all inputs to be registered and all outputs to be written. In OFDM, all subcarriers are processed identically and independently, so data for all subcarriers must be stored and decoded. Thus, data memory contains a number that is a multiple of the number of subcarriers. The size of memory in this case justifies splitting it into multiple blocks. If splitting is taken to the extent that each of the 64 elements of the four 4x4 matrices occupies an independent block, the processor can be clocked at its full potential. This memory map is shown in fig. 6. The 4 conceptual 4x4 matrices are labeled A, B, C, and D; and the 64 independent memory blocks are each as deep as the number of subcarriers (N_{sc}).

Exchanging a single memory block for 64 with independent address decoders introduces some challenges. Each memory location now needs a pair of indices to locate it: one to indicate its memory block; and one to index its depth, namely the subcarrier. The latter in particular can be prohibitive, needing either a very long instruction or a very complicated address decoder. However, although the processor accepts a large number of complex inputs every cycle, all elements of all operands come from the same depth (subcarrier) regardless of which of the 64 memory blocks they come from. So they all share the same subcarrier address in any given cycle. The address is provided directly by the controller as derived from relatively simple matrix index logic. Memory is clocked at twice the processor clock. Addresses are multiplexed between read and write indices, usually offset by the latency of the processing units. Write enables are multiplexed between a null word and values provided from the instruction. This allows multiple one port memories to read and write a whole 4x4 or smaller matrix every processor cycle.

Data memory provides access to all elements of a matrix in a fixed manner. The processing unit inputs are also fixed, for example the multiplication core multiplies all elements in input vector 1 with the exact corresponding elements of vector input 2, and the coordinate rotation core always considers the first element to be the vectoring element. To define which matrices or vectors are multiplied and the direction and target of coordinate rotations, the programmer has to be able to map the outputs of the data memory freely to the inputs of the processor. This is the function of the sorting circuits at the input and output of data memory. This is the function of the sorting circuits.

The sorting circuits proved to be the most resource intensive components of the MIMO accelerator. Essentially each sorting circuit consists of a collection of multiplexers equal to the number of target ports (32 for processor input sorter, 64 for memory input sorter) with a number of inputs equal to the source (8 for memory input sorter, 64 for processor input sorter). For the processor input sorting circuit this translates into 32 64x1 multiplexers, equivalent to 16,970 slices on a V4 LX200. This is roughly 150% of the total area of the coordinate rotation core or 57% of all resources used in the processor, excluding data memory.

The 32 input ports of the processor are not independent. They are divided into at most two vector/matrix arithmetic operands, each of 16 complex elements. Each operand can come from a single block (A, B, C, and D in fig. 6) in memory. This means that each set of 16 processor input ports is associated with 16 memory ports (as opposed to 64) per cycle. A first level of two 16 element wide 4x1 multiplexers (M_A and M_B in Fig. 6) is used to link each of the operands to a matrix, allowing the main sorting multiplexers (N1 through N32) to be reduced from 64x1 to 16x1. This results in a resource saving of nearly 70% over a direct multiplexing approach.

The memory input sorting circuit accepts 3*8 input busses and redistributes them over 64 memory input ports. The inputs to this circuit are results from matrix or vector operations. Similar to the processor input sorting circuit, processor outputs are divided into at most two vector outputs with 4 elements each. Each processor output is assigned in its entirety to a matrix in memory. So, it is only necessary to distribute the processor outputs over 32 ports (N1 through N32 in fig. 6) corresponding to at most two memory matrices, these ports can then be mirrored on the rest of memory without loss of generality.

5. SOFTWARE, PROGRAMMING, AND SCALABILITY

The accelerator uses an open instruction with a pre-decoded operation and control field. The addressing scheme is different from that used in traditional processors [11]. The physical addresses for all memory blocks in fig. 6 are used to index the current subcarrier. Instead, the operand is inferred from a combination of control signals in the instruction, namely controls to the sorting circuits, write enables to data memory, and a Hermitian filter control signal. This addressing scheme corresponds to the complex matrix operands in MIMO decoding. The typical processor cycle in the accelerator capitalizes on the nature of OFDM to deliver higher performance. Because an instruction is normally applied identically to all subcarriers, the instruction is fetched only once for a large number of execution cycles. By properly placing state machine control signals, the fetch state can be avoided altogether. Thus for most programs the accelerator is always executing. The accelerator is supported by a compiler that allows programs written in a custom higher level script to be translated to machine programs. The script is very similar to MATLAB and it allows the programmer to ignore the involved addressing scheme, instead defining operands and results as matrix ranges.

The architecture described above discusses a single instance of the accelerator architecture. The complete design flow allows user-directed generation of architectures by offering the size and structure of the processor, and the dimensions and access method of memory as well as wordlength; as pre-synthesis inputs to the user. These options are made through a user interface that provides real-time area, operation efficiency, latency, and dynamic performance feedback; thus allowing hardware design choices to be made rapidly before programming commences. The hardware configuration decision is also aided through an instance recognition tool that analyzes target programs for the minimum hardware configuration required.

6. RESULTS

Table 1 shows synthesis results of the accelerator and its main

TABLE 1
SYNTHESIS RESULTS. CLK IS MAX PROCESSOR CLOCK.

65nm TSMC	kGates	Power (mW)	Clk(MHz)	
Accelerator	1644	272	233	
Core	824	169	233	
V4-LX200	Slices	Multipliers	BRAM	Clk(MHz)
Accelerator	27040	48	72	209

building blocks for Virtex-4 LX200 speed grade -11, and 65nm CMOS ASIC. Results on the CMOS process are listed with (accelerator) and without (core) data memory realized as registers. Memory is realized as registers to obtain a conservative vendor-independent first order estimate of area and power. Table 2 lists cycle count results obtained from cycle-accurate fixed point simulations for different antenna/algorithm configurations for 64 subcarriers. The identity of full and singular-value-only cycle counts for 2x2 SVD is an example of the accelerator processing multiple small vectors simultaneously.

To compare the accelerator to a general purpose DSP we carried out a series of tests to measure the energy required to carry out a set of typical complex matrix operations and MIMO decoding algorithms. To quantify delay we measure the number of cycles required to run these tests on a general purpose DSP and the MIMO accelerator. We repeated the tests with different subcarrier counts to expose the effects of latency. The DSP used is a fixed point TI DSP6416 600MHz, a power number assuming 60% CPU utilization is used, and cycle counts are obtained from TI code composer studio code profiling tools. Decompositions and MIMO decoders on the DSP are not necessarily identical to those on the accelerator, instead using software friendly algorithms to reach the same decomposition. For example QR decomposition on the DSP is implemented using Gramm-Schmidt orthogonalization instead of Givens rotationns. Table 3 lists the ratio of accelerator PDP to DSP PDP for different test scenarios. In all tests the operands are complex vectors or matrices. The accelerator has a substantial advantage for all operations. Particularly for Givens rotations where the DSP expends almost half the energy (or cycles) calculating trigonometric functions. There is only a very slight trend with increasing number of subcarriers, and the trend is in the accelerator's advantage. Even when software friendly algorithms are used, the difference in performance is almost always above three orders of magnitude.

TABLE 2
CYCLE COUNT ESTIMATES FOR 64 SUBCARRIERS BY ALGORITHM AND ANTENNA COMBINATION.

QRD		Exhaustive search ML		SVD (singular values only)		SVD (full)	
2x2	512	2x2 BPSK	1280	2x2	1088	2x2	1088
3x3	896	2x2 QPSK	3200	3x3	2880	3x3	5632
4x4	1408	3x3 BPSK	2944	4x4	5568	4x4	16448
4x2	896						
6x3	1664						
8x4	2688						

TABLE 3
PDP COMPARISON WITH TI DSP6416. 600MHZ ROT.=ROTATION

	DSP PDP/Acc. PDP				Throughput	
Subcarriers	64	128	256	512	DSP (ksps)	Accel. (Msps)
Addition	143	194	236	264	1784	206
Multiplication	781	1062	1292	1452	325	206
Division	94	127	154	173	2725	206
Series	524	577	624	652	262	206
Real Rot.	4970	6577	6196	6143	77	206
Complex Rot.	5594	7374	8610	9151	51.5	206
2x2 SVD	2927	3069	3120	3146	10.7	14.5
4x2 MMSE	815	845	861	869	44	16.5
4x4 QRD	1724	1769	1793	1805	15	11.6
2x2 ML	1998	2084	2130	2154	20.8	19.3

REFERENCES

[1] G. J. Foschini and M. J. Gans, "On limits of wireless communications in a fading environment when using multiple antennas," *Wireless Pers. Commun.*, vol. 6, no. 3, pp. 311–335, Mar. 1998.

[2] M. Anders, S. Mathew, R. Krishnamurthy, S. Borkar, "A 64-state 2GHz 500Mbps 40 mW Viterbi accelerator in 90nm CMOS," *VLSI Ciruits, Digest of Tech. Papers, 2004 Symp. On*, pp. 174-175, Jun. 2004.

[3] J. Mitola III, "The Software Radio Architecture," *IEEE Commun. Mag.*, May 1995, pp. 26–38.

[4] Simon Haykin, "Cognitive radio: brain-empowered wireless communications," *IEEE J. Sel. Areas Commun., IEEE Journal on*, Vol.23, Iss.2, Feb. 2005 Pages: 201- 220.

[5] N.D. Hemkumar, "Efficient VLSI Architectures for Matrix Factorizations," Ph.D. dissertation, Rice University, Houston, TX, 1994.

[6] J. Wang, "A recursive least-squares ASIC for broadband 8 x 8 multiple-input multiple-output wireless communications," Ph.D. dissertation, Henry Samueli School on Engineering and Applied Science, University of California in Los Angeles, Los Angeles, CA, 2005.

[7] R.P. Brent, F.T. Luk, "The solution of singular-value and symmetric Eigenvalue problems on multiprocessor arrays," *SIAM J. STAT. COMPUT.*, vol.6, No. 1, Jan. 1985.

[8] R.P. Brent, F.T. Luk, C. Van Loan, "Computation of the singular value decomposition using mesh connected processors," *J. of VLSI and Comp. Systems*, Vol. 1, No.3, pp. 242-267.

[9] Hun Seok Kim, Weijun Zhu, Jatin Bhatia, Karim Mohammed, Anish Shah, and Babak Daneshrad, "A Practical, Hardware Friendly MMSE Detector for MIMO-OFDM Based Systems," *EURASIP Journal on Advances in Signal Processing*, vol. 2008.

[10] B. Hassibi, H. Vikalo, ""On the Sphere-Decoding Algorithm I: Expected Complexity," *IEEE Trans. Signal Proc.*, vol. 53, No. 8, pp. 2806-2818, Aug. 2005.

[11] F.M. Cady, "Microcontrollers and Microprocessors Principles of Software and Hardware Engineering," Oxford University Press, 1997.