Hardware/Software Co-design Methodology and DSP/FPGA Partitioning: A Case Study for Meeting Real-Time Processing Deadlines in 3.5G Mobile Receivers

Michael Brogioli, Predrag Radosavljevic and Joseph R. Cavallaro
Department of Electrical and Computer Engineering
Rice University
Houston, Texas 77005
Email: {brogioli,rpredrag,cavallar}@ece.rice.edu

***Abstract*— This paper presents a DSP/FPGA hardware/software partitioning methodology for signal processing workloads. The example workload is the channel equalization and user-detection in HSDPA wireless standard for 3.5G mobile handsets. Channel equalization and user-detection is a major component of receiver baseband processing and requires strict adherence to real time deadlines. By intelligently exploring the embedded design space, this paper presents a hardware/software system-on-chip partitionings that utilizes both DSP and FPGA based coprocessors to meet and exceed the real time data rates determined by the HSDPA standard. Hardware and software partitioning strategies are discussed with respect to real time processing deadlines, while an SOC simulation toolset is presented as vehicle for prototyping embedded architectures.**

I. Introduction

Heterogeneous architectures are increasingly being used in the embedded domain, typically consisting of one or more DSP cores, simpler microcontroller and possible FPGA computing engines providing low cost, high performance reconfigurable functionality [5]. The DSP provides the flexibility and programmability, while the FPGA provides the performance of an all hardware solution.

Next-generation mobile wireless communications systems are a driving force behind this innovation in high–performance, low–cost hardware. Heterogeneous, embedded DSP/FPGA architectures are being developed in order to meet the real–time requirements and performance expected from these ubiquitous mobile devices. In designing these systems, task partitioning and hardware/software co–design present a challenge in alleviating computational bottlenecks in a traditional all software DSP based solution.

This paper shows how a hardware/software prototyping for heterogeneous DSP/FPGA devices can facilitate highly efficient real-time embedded architectures. Using a defined set of criteria for hardware/software partitioning between DSP and FPGA, in conjunction with iterative DSP/FPGA hardware simulation, strict real time deadlines in the mobile wireless communications arena are met and exceeded. As a motivating case study indicative of many embedded wireless and media workloads, high data rates defined by 3.5G HSDPA wireless standard [1], [2] are achieved and exceeded via hardware/software partitioning of the mobile receiver baseband algorithm, and heterogeneous design space exploration in the DSP/FPGA embedded arena.

II. Background

In recent years, modern embedded systems have grown to be vastly more complex than a simple microcontroller and a small amount of local memory. This has resulted in a number of open ended problems with respect to how the designer should jointly specify hardware and software partitionings, and how system architects design their systems.

A. Hardware/Software Codesign for Embedded Architectures

There have been a number of uses of FPGA based computing elements as an attempt to provide high performance reconfigurable computing at the fine grained and coarse grained levels. Systems such as Chimaera and other fine grained uses of FPGAs have made attempts to provide extensible and reconfigurable ISAs in general purpose processors [6]. Other architectures employing FPGAs at a coarse grained level, offloading larger elements of computation in the input application onto an FPGA based array [10]. Rather than partition workloads at the fine grained or coarse grained level, this work shows how application partitioning at the application task level across multiple FPGAs and DSPs can achieve real-time deadlines in computationally bottlenecked embedded systems.

B. Channel Equalization in 3.5G Wireless Systems

Code Division Multiple Access (CDMA) has been the driving force behind the second and third generation wireless cellular technology, and is used as the multiple–access technology of choice by many cellular standards (UMTS, CDMA2000). Extensions to these for data services have been standardized as the High Speed Downlink Packet Access (HSDPA) standard. [1], [2]. It is assumed the HSDPA downlink transmission with single transmit and single receiving antenna. The transmission rate of 3.84 MChips/sec is defined [2] which determines the real-time processing requirements on the receiver end. Current research has indicated that channel equalization and user-detection is the most complex part of the receiver baseband signal processing chain and continues to be a bottleneck for high-data rate CDMA-based systems [7], [8].

III. SOC Simulation Infrastructure

All simulations and DSP/FPGA system prototyping was done using the Spinach SOC simulation environment for prototyping heterogeneous embedded architectures [3], [4]. Spinach is a library of composable, user configurable software modules for rapidly prototyping simulators for embedded heterogeneous architectures.

Spinach is built upon a SystemC like infrastructure where all module communication is transaction based at the bit–level, on discrete clock cycle boundaries within the simulated system. Spinach is intended for modelling architectures that include heterogeneous computing environments, as well as varying system topologies and timing semantics.

In designing a heterogeneous DSP/FPGA embedded architecture with Spinach, users prototype the system with a library of existing simulator software modules that have a 1:1 mapping to hardware components. The library of simulator module building blocks contains bit-true, cycle accurate Texas Instruments DSPs,

MIPS microcontrollers, and interconnect modules such as busses, bridges and crossbars with user definable bandwidth, throughput latencies, and arbitration policies. Memory modules such as SRAM, DRAM, memory controllers, caches and cache controllers are user configurable, and simulate bit true cycle accurate contents for processors, instruction/data memory, etc. FPGA modules are designed such that clock rates, gate counts and functionality are user definable. FPGA modules have support for internal dual ported RAM arrays that can be targeted by intelligent DMA engines for data transfer. While the functionality implemented by an FPGA module is user definable, timing information and gate counts can be verified on real hardware, and configured via high level parameters at runtime. This functionality is discussed further in Section V.

Intra–module communication between simulated hardware components occurs over abstracted hardware-like module I/O ports, upon discrete clock cycle boundaries at the bit level. For instance, in a DMA transaction from local on-chip data memory to FPGA dual ported RAM arrays, transactions on the simulated memory bus occur st 32/64/128/256 bit resolution upon discrete clock cycle boundaries, where one device attached to the bus wins arbitration each clock cycle and transmits a maximum of the aforementioned bit width data. All data is coupled with timing information, and thus overall system performance between DSP and FPGA for a given system topology can be quantitatively measured. Further information on usage, functionality, and model of computation for Spinach can be found in [3], [4].

IV. Methodology

In partitioning any workload across a heterogeneous system comprised of reconfigurable FPGA computational accelerators, programmable DSPs or programmable host processors, and varied memory hierarchy, a number of criteria must be evaluated to determine whether a given task should execute in software on the host processor or in hardware on FPGA, as well where in the overall system topology each task should be mapped. It is these sets of criteria that typically mandate the software partitioning, and ultimately determine the topology and partitioning of the given system. Section IV-A below discusses the key criteria in hardware/software codesign for embedded architectures.

A. Hardware/Software Partitioning Criteria

- Spatial locality of data is one concern in partitioning a given task. The ability to access data in a particular order efficiently is of great importance to performance. Issues such as latency to memory, bus contention, and DMA transfer times to local compute element all need to be taken into consideration.
- Data level parallelism is important as many signal processing applications exhibit a large amount of both instruction and data level parallelism, often far more than available DSP functional units can handle efficiently [9]. FPGAs can exploit these computational bottlenecks with parallel functional units and local block data RAM.
- Computational complexity of the application can bounds the programmable DSP core, creating bottlenecks in software performance. Mapping bottlenecks in the application from software implementation executing on host processor to an FPGA implementation can alleviate bottlenecks on the host processor and permit extra cycles for additional computation or algorithms to execute in parallel.
- Task level parallelism can effect partitioning as well. Applications contain multiple tasks that can execute concurrently, but have a limited amount of instruction or data level parallelism within each unique task [12]. If one or more task contains enough instruction/data level parallelism to exhaust the available host processor resources, it can be considered for partitioning to FPGA.

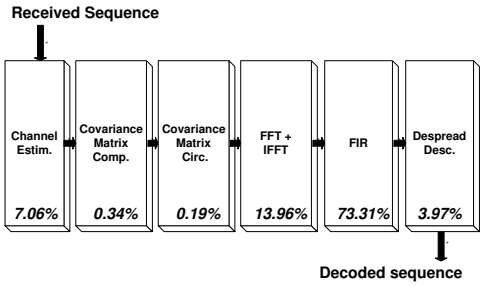

Fig. 1. Channel Equalization Block Diagram

B. Channel Equalization Workload

Total workload of channel equalization is decomposed into multiple tasks as depicted in Figure 1, where for each task the runtime profile data for software implementation on host uniprocessor DSP is listed. Channel estimation based on known pilot sequence, covariance matrix computation (only first row or column) and circularization of it, FFT/IFFT (Fast Fourier transform and Inverse Fast Fourier transform), post-processing for final computation of equalization coefficients, finite–impulse response (FIR) filtering applied on the received samples, and (despread/descramble) for recovering the user information bits.

In [14], the authors present an efficient algorithm for channel equalization in the context of 3GPP systems. By exploiting properties of the received signal statistics and incorporating the efficient time and frequency-domain processing (FFT/IFFT), they achieve a significant complexity–reduction for linear chip-level equalization (from cubic to roughly linear in terms of the length of the finite–impulse response filter used). This complexity-reduction translates directly into a faster implementation thereby enabling better receiver performance for next-generation high-data rate wireless standards such as HSDPA.

In this paper the estimation of the channel paths (120km/h Vehicular A channel with seven dominant paths is assumed) is performed by correlating pre-known pilot sequence with received data-frame. Estimated channel taps are used in time-domain to compute and circularize the first row of the receiver covariance matrix. Consequently, Linear Minimum Mean Square Error (LMMSE) equalization (which involves the inversion of the covariance matrix) can be approximated with simple FFT operations and scalar divisions [14]. Circulant approximation is possible due to the Toeplitz structure of the covariance matrix. The FFT output (after scalar divisions) represents frequency-domain equalization coefficients. Since the FIR filtering of received samples is performed in time-domain, frequency-domain filter coefficients are converted back in time-domain using IFFT operation. At the end, despread/descramble is applied in order to detect information bits of particular user. In partitioning this workload across either software programmable DSP or across configurable FPGA based computation engines, we consider partitioning at the granularity of these tasks.

C. System Modelling

Table I lists various parameters used in the base case non-partitioned architecture. The host processor in the system is the Texas

Instruments TMS320C6201 fixed point programmable digital signal processor, operating at 167MHz, running compiled code generated by Texas Instruments Code Composer Studio Version 2.10.0 as is discussed in Section IV-D. Instruction and data memory are modelled in a Harvard architecture style as per the TMS320C6201 specification, with 256–bit busses to on–chip program and data memory, supporting 256–bit throughput per simulated clock cycle. Accesses to data memory via DSP pass through the on–chip data memory controller, which also performs routing and arbitration of memory references to and from any peripheral components with a single cycle clock latency per 256–bit bus transaction.

Peripherals consist of on–chip DMA engines, memory mapped registers (MMRs), FPGA based compute engines and other peripherals. Busses for access to peripherals are considered to be fixed at 32–bits throughput per simulated clock cycle with uniform clock rate. Bridging logic is used to provide functionality between the 256–bit data memory bus and 32–bit peripheral data busses, with all references through bridging logic having a single cycle throughput latency, plus additional overhead for packet segment and reassembly according to input bus width divided by output bus width. In the instance that multiple peripherals are tied to the 32–bit data busses for access to on–chip data memory through the data memory controller, stackable MxN crossbar logic is used for memory transaction routing and arbitration, supporting flexible bit width throughput on either side with local FIFO storage, and single cycle throughput latency for arbitration winner similar to technology found in many embedded multiprocessor systems [11]. Details are listed in Table I, whereas the resulting system topology and block diagram are discussed in detail in Section V.

Simulation Parameters	*Value*
DSP Architecture	Texas Instruments TMS320C6201
System Clock Rate	167MHz
Instruction Memory Bandwidth	256b on–chip
Data Memory Bandwidth	256b on–chip 32b off–chip
Instruction Memory Size	64KB on–chip
Data Memory Size FPGA Bus Bandwidth	64KB on–chip 32 bits per clock cycle
DMA Bus Bandwidth	32 bits per clock cycle bidirectional
Bus Arbitration	Round Robin, single cycle minimum throughput latency

TABLE I
BASELINE SIMULATION PARAMETERS

Additionally, in system configurations where multiple FPGA compute elements lie on the data bus, MxN cross bar logic is used for reference routing. The MxN crossbars also contain bus arbitration logic to handle streaming references to and from multiple FPGA based compute engines and employ a round robin arbitration technique. Data memory transfers to and from the FPGA compute fabrics RAM arrays are performed via the on–chip DMA engines. DMA engines are programmed via memory mapped registers using the host DSP for control and synchronization management. It is these DMA engines, operating at 32–bit data bandwidth resolution that perform high speed block data transfers from local DSP data memory to the dual ported RAM arrays found locally on the FPGA fabrics at one 32–bit bus request per clock cycle. Upon completion of computation by the FGPA based compute fabric, it is the host DSP's responsibility to poll status registers for FPGA status and optionally program another DMA engine for data transfer of the result vectors from FPGA dual ported RAM arrays to DSP on–chip data memory.

D. Software for HSDPA Channel Equalization

The channel equalization firmware, as described in IV-B was aggressively compiled for the host DSP using Texas Instruments Code Composer Studio Version 2.10.0, at level three optimization withspeed critical performance flags turned on and aggressive inlining and loop unrolling. All computation was optimized to be 16–bit fixed point, in effect optimizing the workload for DSP performance by maximizing the functional unit utilization of the TMS320C6201 architecture. While performance critical kernels are sometimes implemented in assembly code, we have chosen to use low level optimized C code for the application software in these studies. Though in some cases assembly can be more efficient, the performance trends shown are valid not due to the absolute performance of the code but rather the fact that despite the application software efficiency there are computational bottlenecks on the DSP due to functional unit limitations.

In the case when partitioning of the workload across one or more FPGA based compute fabrics was performed, a system of macros built in to Code Composer Studio for heterogeneous system partitioning were used to program the enhanced DMA engines, FPGA based compute fabrics, and memory mapped status registers. Once code was compiled, common object file format loader utilities which are part of the simulator toolset were used to load applications into simulated system memory and begin execution. Portions of the workload chosen for offloading to FPGA are manually partitioned by the application programmer using high level macros. Functionality is then migrated from the application software into the simulated FPGA modules. Since all simulator modules are written using low level C code, users typically implement FPGA computational functionality in very low level C code that models the behavior of RTL. Timing and gate count information is then set for the simulator modules via their high level user parameters. Users can then typically use RTL/VHDL simulators or open core library specifications to obtain timing information for the modules. While it is not currently supported, support for RTL to C tools to encapsulate FPGA functionality more cleanly and automatically within the simulator is planned.

V. DSP/FPGA PARTITIONING AND SIMULATION RESULTS

In partitioning the workload from a uniprocessor DSP to a DSP and multiple FPGA based heterogeneous SOC, application performance must be investigated according to the criteria listed in Section IV-A. Figure 1 in Section IV-B showed the runtime profile data for each componenet of the channel equalization workload executing in software on the host DSP. In partitioning the input application on FPGA based accelerators, total runtime, data reuse patterns and locality are considered, as well as regular blocks of computation that map to an FPGA based implementation. Based on the runtime data in Figure 1, part of channel estimation, FIR filtering, FFT/IFFT, and despread/descramble were chosen to be offloaded to an FPGA implementation. It can be observed that the FIR filtering represents the largest workload which mainly determines maximum achievable data rate.

In order to achieve HSDPA data-rate it is necessary to process each received data-chip within time slot of approximately 260 ns [2]. Each data-chip represents four information bits based on 16 QAM transmitting modulation. For the purpose of faster computer simulations, shorter data-frames with length of 512 data-chips are used. The particular data-frame length determines total data frame processing time of 22186 clock cycles (assuming pre-determined clock frequency of 167 MHz) available to finish receiver base-band processing before the next data-frame is received.

Figure 2 illustrates final partitioning of the major computational blocks where part of channel estimation (four out of seven channel paths), FIR filtering, FFT/IFFT, and despread/descramble are fully offloaded to FPGA based co-processors while remaining base-band processing executes on the host DSP: fully partitioned workload.

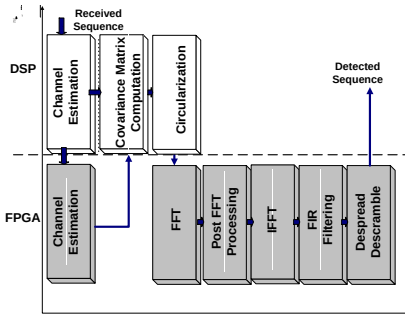

Fig. 2. Partitioning of algorithm sub-parts into DSP and FPGA

Figure 3 illustrates corresponding final system topology of the receiver base-band architecture. As it was previously described in Section IV-C, all data computed on FPGAs must be explicitly DMA'd to the FPGA's local dual ported RAM arrays via the on–chip DMA engines, which are controlled via the host DSP and memory mapped registers. Each block of computation executing within an FPGA is modelled pessimistically due to the fact that we assume a different FPGA for each task. Table II shows the percentage of total execution time for each processing part of the workload, in the fully partitioned case. It can be observed that the workloads are more balanced than before when FIR filtering was using 73.31% of the total processing runtime. Additionally, with the fully partitioned workload, the DSP is idle 70.9% of the total available time between two consecutive data frames waiting for results from FPGAs computing, or for the next data-frame to arrive. This idle time could be used to meet other real time deadlines in the system, or support additional real time operating systems.

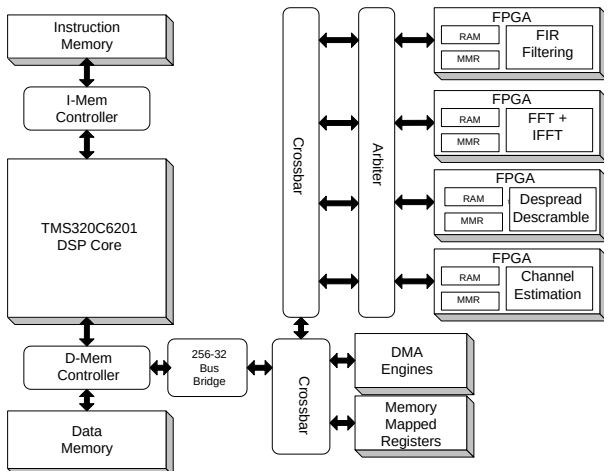

Fig. 3. SOC Block Diagram

Figure 4 shows the program runtimes for the various partitionings between software on DSP and hardware based FPGA offloading using the resulting SOC modelled in Figure 3. As it is said, the fastest runtime is achieved in the case of fully partitioned workload when FIR filtering, FFT/IFFT, despread/descramble (DD), and part of channel estimation (CE) are offloaded to the FPGA co-processors.

Program Task	*Percentage Runtime*
Channel Estimation:	47.59%
Covariance Matrix:	3.81%
Circulation of Covariance Matrix:	1.52%
FFT+IFFT:	3.91%
FIR Filtering:	38.15%
Despread/Descramble:	11.88%

TABLE II
APPLICATION PROFILING DATA: FULLY PARTITIONING CASE

Total runtime of 13476 clock cycles fits into available runtime determined by HSDPA data rate. By offloading from DSP to FPGA co-processor, FFT+IFFT runtime is reduced from 23188 clock cycles to 256 clock cycles.

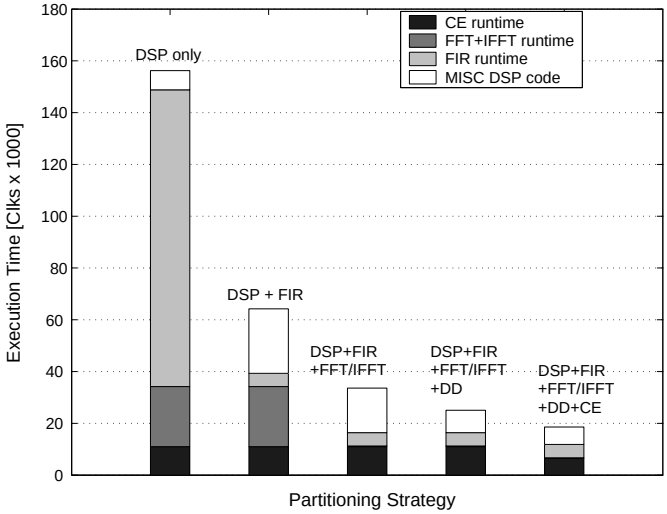

Fig. 4. Execution runtimes of algorithm sub-parts for different DSP/FPGA partitionings

For each hardware–software partitioning shown in Figure 4, any computation offloaded into FPGA has certain latency shown in Table III. These are effectively the ideal latencies to transfer all data at 32–bit width bus transactions, through all crossbars and arbitration logic assuming no contention for bus arbitration. Runtime latencies for data transfer may be higher due to bus contention. The compute latencies are the number of cold start clock cycles to compute on the data once it has been completely DMA transferred to the local RAM arrays on the FPGA. Each of the compute cycle latencies stated in Table III represents the number of clock cycles it takes for a given FPGA to compute on the data stored locally in its RAM arrays, assuming a non-pipelined cold start.

Table IV lists the slice count, on-chip 18 x 18 multipliers, and block RAM resources utilized in a Virtex–II Pro based device for each of the offloaded tasks in the channel equalization workload. It is again important to note that data transfer times between FPGA blocks are pessimistic as described above. Though all offloaded computation is pessimistically assumed to be in separate FPGA devices, slice counts shown in Table IV show that all functionality could be implemented in single device. The FFT engine operates in burst–mode configuration and is implemented using Xilinx's FFT core for the Virtex–II Pro FPGA [13]. The computation reads 32-bit

FPGA Coprocessor	*Compute Cycles*	Data Transfer Cycles
FIR	10*512 samples	38
FFT/IFFT	128	128
DD	1536	276
CE	4608	1034

TABLE III
FPGA COLD START COMPUTE LATENCIES AND SOC DMA TRANSFER ACTIVE TIMES

complex data points from the FPGA's BlockRAM and writes the 32–bit complex results back to the same memory location. Dual–ported BlockRAMs [13] are used to minimize access latencies. The FIR filtering is implemented using a multiply–accumulate structure while the despread/descramble is implemented with add/subtract and sign comparison operations.

Offloaded Blocks	*Slices*	*18x18 Multipliers*	*BlockRAMs*
FIR	128 (2.4%)	8 (18.0%)	2 (4.4%)
FIR+FFT/IFFT	798 (16.3%)	15 (34.1%)	5 (11.0%)
FIR+FFT/IFFT+DD	824 (16.7%)	15 (34.1%)	5 (11.0%)
FIR+FFT/IFFT+ DD+CE	856 (17.7%)	17 (38.6%)	6 (13.2%)

TABLE IV
FPGA RESOURCE UTILIZATION

It can be observed from Figure 4 that significant speedups in performance can be achieved by offloading the various tasks on FPGA co-processors. Even in the case of offloading just FIR filtering with all other functionality executing in software on the programmable host DSP, improvements on the order of about 68.5% are seen. Significant additional improvement in performance can be achieved if FFT/IFFT is offloaded on FPGA as well. At this point the DSP begins not to be the computational bottleneck in the system and we gain in overall application performance. Both FIR filtering and FFT/IFFT are computationally intensive when executed on the DSP. This is due to the iterative and repetitive nature of the computations, and the amount of live data that the DSP must attempt to keep in registers for local computation. If FIR filtering and/or FFT/IFFT are executed on the host DSP, we see that the DSP is performing poorly compared to the FPGA based implementation due to the limited size of the register files for each of its VLIW clusters. With only sixteen 32–bit registers to feed each side of the clustered VLIW pipeline, there are far more intermediate values for computation than can simultaneously fit within the register file at any given time. This results in the DSP also having to execute spill code to move temporary values from registers to memory.

Additionally, the amount of instruction and data level parallelism in the FIR filtering and FFT/IFFT is more than the limited numbers of DSP functional units can exploit. By offloading FIR filtering and FFT/IFFT computation to the FPGAs, not only is the inherent instruction and data level parallelism exploited but we also alleviate the need for the execution of spill code and effectively use the dual ported RAM arrays in the FPGA as a large highly parallel register file. Finally, the additional speed-up required to meet HSDPA requirements is achieved if part of channel estimation is executed on host DSP while the majority of base-band processing is performed on the FPGA co-processors optimized for the particular tasks. The total speed-up of about 91.4% is achieved comparing to DSP-only solution.

VI. CONCLUSIONS AND FUTURE WORK

This paper shows how heterogeneous DSP/FPGA based embedded architectures and application software partitioning can be used to achieve real-time deadlines in modern embedded systems. A methodology for isolating computational bottlenecks in the application software running on modern DSP, and leveraging the increased parallelism of FPGAs with iterative hardware and software refinements is presented. As a case study, channel equalization in 3.5G wireless mobile receivers supporting HSDPA data rates is investigated. Through iterative hardware/software codesign and refinement, performance gains of over 90% are achieved versus traditional embedded programmable DSP based solutions.

ACKNOWLEDGMENT

This work was supported in part by Nokia Inc., Texas Instruments, Inc., and NSF under grants EIA-0224458 and EIA-0321266.

REFERENCES

[1] 3GPP Technical Report 25.848, Physical layer aspects of UTRA High Speed Downlink Packet Access, version 4.0.0, March 2001.

[2] 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Multiplexing and channel coding (TDD), version 4.2.0, March 2002.

[3] M. Brogioli and J. Cavallaro. Modelling Heterogeneous DSP-FPGA Based System Partitioning with Extensions to the Spinach Simulation Environment. In *Asilomar Conference on Signals, Systems, and Computers*, October 2005.

[4] M. Brogioli, P. Willmann, and V. Pai. Spinach: A Liberty-Based Simulator for Programmable Network Interface Architectures. In *Languages Compilers and Tools for Embedded Systems 2004*, pages 100–110, June 2004.

[5] T. J. Callahan and J. Wawrzynek. Instruction Level Parallelism for Reconfigurable Computing. In *Proc. 8th Intl. Workshop on Field-Programmable Logic and Applications*, sept 1998.

[6] J. Cong, Y. Fan, G. Han, A. Jagannathan, G. Reinman, and Z. Zhang. Instruction Set Extension with Shadow Registers for Configurable Processors. In *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays*, pages 99–106, New York, NY, USA, 2005. ACM Press.

[7] A. de Baynast, P. Radosavljevic, and J. Cavallaro. Chip level LMMSE Equalization for Downlink MIMO CDMA in Fast Fading Environments. In *IEEE Globecom*, volume 4, pages 2552–2556, November 2004.

[8] Y. Guo, J. Zhang, D. McCain, and J. Cavallaro. Efficient MIMO Equalization for Downlink Multi-Code CDMA: Complexity Optimization and Comparative Study. In *IEEE Globecom*, volume 4, pages 2513–2519, November 2004.

[9] H. C. Hunter and J. H. Moreno. A New Look at Exploiting Data Parallelism in Embedded Systems. In *Proceedings of the International Conference on Compil ers, Architectures and Synthesis for Embedded Systems*, pages 159–169, 2003.

[10] J. Lee, K. Choi, and N. D. Dutt. An Algorithm for Mapping Loops Onto Coarse-Grained Reconfigurable Architectures. In *LCTES '03: Proceedings of the 2003 ACM SIGPLAN Conference on Languages, Compilers, and Tool for Embedded systems*, pages 183–188, New York, NY, USA, 2003. ACM Press.

[11] Texas Instruments. TMS320C6000 DSP Multiprocessing With a Crossbar Switch, 2001.

[12] P. Willmann, H. Y. Kim, S. Rixner, and V. S. Pai. An Efficient Programmable 10 Gigabit Ethernet Network Interface Card. In *11th International Symposium on High-Performance Computer Architecture*, pages 96–107, February 2005.

[13] Xilinx Incorporated. Virtex-II Pro Platform FPGAs: Complete Data Sheet.

[14] J. Zhang, T. Bhatt, and G. Mandyam. Efficient Linear Equalization for High Data Rate Downlink CDMA Signaling. In *Conference Record of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 141–145, 2003.