A General Hardware/Software Co-design Methodology for Embedded Signal Processing and Multimedia Workloads

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 hardware/software co-design methodology for partitioning real-time embedded multimedia applications between software programmable DSPs and hardware based FPGA coprocessors. By following a strict set of guidelines, the input application is partitioned between software executing on a programmable DSP and hardware based FPGA implementation to alleviate computational bottlenecks in modern VLIW style DSP architectures used in embedded systems. This methodology is applied to channel estimation firmware in 3.5G wireless receivers, as well as software based H.263 video decoders. As much as an 11x improvement in runtime performance can be achieved by partitioning performance critical software kernels in these workloads into a hardware based FPGA implementation executing in tandem with the existing host DSP.**

I. Introduction

Personal telecommunications and multimedia systems form vast segments of the embedded systems market. Variations in standards for wireless and video coding, as well as the desire for software programmability often result in applications being written in software executing on a programmable DSP core. As workloads move from wireless telecommunications to media intensive video and image processing and beyond, computational demands on the host processor are increasing [1], [4], [5], [6]. Despite increases in clock rates for modern VLIW style DSP cores, the number of parallel functional units within programmable DSP pipeline remains finite. For example, Texas Instruments TMS320C6x series DSPs have an eight-wide clustered VLIW pipeline. Even with modern code optimization techniques, the computational demands of many wireless and multimedia video kernels far exceeds the available pipeline arithmetic and logic unit (ALU) resources of todays DSP devices.

This paper presents a hardware/software co-design methodology for partitioning performance critical kernels of wireless and multimedia based workloads from software executing on DSP into a hardware based FPGA implementation. In doing so, the programmability of a software based implementation can be maintained while the speed of highly parallel hardware for computationally complex bottlenecks in the application can be achieved. Control and non–bottlenecked portions of the application execute in software on the programmable DSP core, whereas the bottlenecked data and compute intensive portions of the application are offloaded to execute in hardware within the loosely coupled FPGA. In evaluating this hardware/software partition methodology, software based channel equalization for mobile wireless receivers as well as embedded H.263 video codecs for video processing are investigated.

II. SOC Simulation Infrastructure

In order to prototype and evaluate heterogeneous DSP/FPGA based architectures, a custom simulation infrastructure is used for heterogeneous DSP/FPGA system-on-a-chip architectures [2], [3]. In using this simulation infrastructure, the user is allowed to rapidly prototype a heterogeneous DSP/FPGA based system as well as on-chip interconnects, DMA engines and peripherals in a bit–true, cycle–accurate manner. Application level software is compiled using production DSP compilers, using on-chip DMA engines within the simulation environment for data transfer between programmable host DSP data memory and the local RAM arrays of the FPGA based coprocessors. All data transfers, code execution and runtime behavior is performed in a bit-true cycle accurate manner with existing DSP and FPGA hardware. Data transfers between various modules within the simulator is performed cycle accurately on clock cycle boundaries, modelling all bandwidth contention and bus interactions on a cycle-by-cycl basis as is the case in real hardware. This inherent coupling of data flow and timing semantics, coupled with executing of bit-true production compiler generated application level executables allows the simulator to model hardware using a paradigm that mirrors real hardware execution.

As a robust toolset for architectural simulation of heterogeneous embedded computing environments, the simulator provides a large library of user modules which can be hierarchically composed to create simulators of vastly differing system topologies. Furthermore, user are free to contribute their own hardware models of proprietary or custom hardware blocks in simulation. Modules are written using C code, with all inter-module communication taking place on clock cycle boundaries over well abstracted port communication mechanisms using a pre-defined port handling API. Simulator modules include, but are not limited to the following *processing* modules: Texas Instruments TMS320C6x DSPs, MIPS R4000 series microcontrollers, fine-grained and coarse-grained FPGAa. Modules for the *memory* system modelling include: caches, cache controllers, SRAMs and controllers, DRAMs and controllers, DMA engines, point to point crossbars. Other peripherals include program loaders, Texas Instruments Code Composer Studio and MIPS gcc compiler support, program trace analyzers, memory system analyzers, etc. A fullly detailed description of the included simulator modules and peripherals can be found in [2].

III. Example Workloads for Partitioning

In exploring the benefits of hardware/software partitioning of embedded workloads between programmable DSP cores and hardware based FPGA accelerators, two workloads were chosen. The first is a wireless channel equalizer used in high speed wireless data communication for 3.5G cellular networks supporting the HSDPA data rate of 3.84 megasamples per second. The second workload is a video decoder for the H.263 compressed video standard, operating at the CIF resolution of 240x162 pixels. These example workloads were chosen as they not only represent multiple application domains, namely wireless signal processing and multimedia content, but also because they contain various computational block common amonst many embedded signal processing workloads. Examples are: Fast

1-4244-0785-0/06/$20.00

Fourier Transform, Inverse Fast Fourier Transform, Discrete Cosine Transform, Finite Impulse Response Filtering, etc.

Figuire 1 shows the block diagram of the software only implementation of the wireless channel equalizer, whereas Figure 2 shows the block diagram of the software architecture for the H.263 video decoder. In each figure, the software's runtime profile execution executing on a dedicated DSP is shown in the corresponding block of the diagram. Looking at Figure 1, it can be seen that the dominant blocks of computation in the wireless channel equalizer are the Channel Estimation routines, Fast Fourier and Inverse Fast Fourier transforms, FIR Filtering and to a lesser extent the Despreading/Descrambling routines. Similiarly, looking at Figure 2 it can be seen that the Inverse Discrete Cosine Transform, Block Summation and Macroblock Routines, as well as the Motion Compensator dominate the majority of the software runtime.

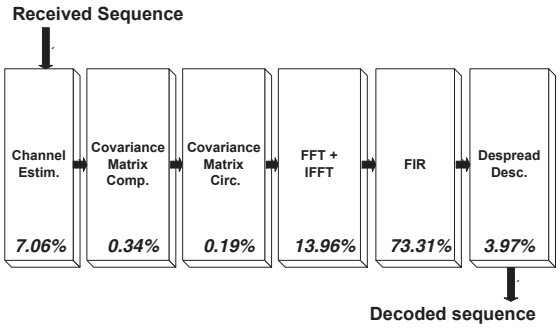

Fig. 1. Channel Equalizer Block Diagram and Software Only Runtime Profile Information

In the case of the video decoder algorithm, as depicted in Figure 2, the algorithm utilizes a feedback look for the purposes of motion compensation. It is due to this loop that although the motion compensators and block summation routines account for a significant portion of the total software runtime, larger runtimes are due to the multiple executions during feedback versus a larger level of computational complexity than the Inverse Discrete Cosine Transform for example.

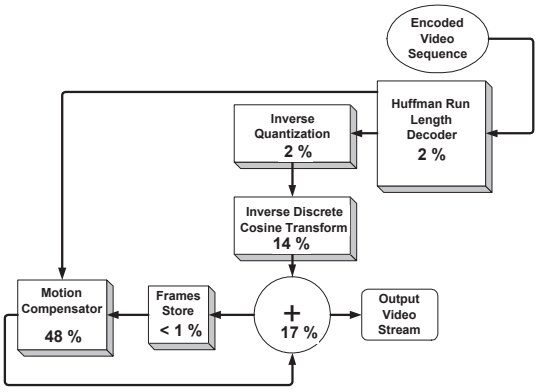

Fig. 2. H.263 Video Decoder and Software Only Runtime Profile Information

A. Workload Software Compilation

The channel equalization firmware and video decoder firmware, as described above, were aggressively compiled for the host DSP using Texas Instruments Code Composer Studio Version 2.10.0, at level three optimization with speed 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 TMS320C64x 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.

IV. Hardware/Software Partitioning Criteria

In partitioning embedded workloads from a traditional software only DSP based implementation to a heterogeneous DSP and multiple FPGA based implementation a number of criteria are considered for partitioning. The goal of the hardware/software partitioning is to alleviate bottlenecks in the input application that exceed the instruction and data paralellism exploitable by the limited resources of the DSPs processor pipeline. In migrating this computation from a software based solution in DSP, to a more explicitly parallelized implemnetation executing in an FPGA or ASIC, runtime compute bottlenecks can be alleviated that often limit the real-time performance of the software. Partitioning compute performance does not come without cost however, quite often the data transfer DMA overheads of moving the compute data from local DSP program memory to the local data memory within the coprocessor can offset any computational performance gains achieved. With this said, the specific criteria followed in this paper are:

- Spatial locality of data. The ability to access data in a particular order efficiently is of great importance to performance when DMA transferring data from local DSP data memory to FPGA RAM arrays. Issues such as latency to memory, bus contention, and DMA transfer times to FPGA compute elements all need to be taken into consideration. In cases where data is not contiguous or uniformly strided, additional overhead to arrange data before block DMA transfers can take place or data can efficiently be computed on.
- Data and instruction level parallelism. 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. FPGAs can exploit these computational bottlenecks with parallel functional units and local block data RAM, as well as the use of highly parallel multiplier and adder trees.
- Task level parallelism. Applications may contain multiple tasks that can execute concurrently, but have a limited amount of instruction or data level parallelism within each unique task. 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.

By utilizing the partitioning criteria described in Section IV in conjunction with runtime behavioral information of the workload executing in software only on a programmable DSP core, iterative partitioning of the workloads tasks and performance critical kernels from programmable software executing on the DSP to one

Simulation Parameters	*Value*
DSP Architecture	Texas Instruments TMS320C6401 Core
System Clock Rate	167MHz
Instruction Memory Bandwidth	256b on–chip
Data Memory Bandwidth	256b on–chip 32bB off–chip
Instruction Memory Size	64KB on–chip
Data Memory Size FPGA Bus Bandwidth	64KB on–chip 32 bits
DMA Bus Bandwidth	32 bits per clock cycle bidirectional
Bus Arbitration	Round Robin

TABLE I
BASELINE SIMULATION PARAMETERS

or more FPGAs can be performed. The resulting heterogeneous DSP/FPGA based system-on-a-chip architecture is then prototyped using the Spinach simulation environment as described in Section II. By iteratively partitioning the workload according to the criteria listed above in Section IV, a balance between the flexibility of a programmable DSP core and the performance of a hardware based FPGA implementation can be achieved.

Figure 3 illustrates the overall simulated system topology which is used as a platform for these experiments. For each of the workloads investigated, the programmable software portion of the algorithm executes on the programmable host DSP while portions offloaded to FPGAs for hardware based acceleration are implemented in the various crossbar attached FPGA devices as shown in Figure 3. As was mentioned previously, all DMA based data transfers are explicitly set up and controlled via software on the programmable host DSP utilizing the attached DMA engines, show in Figure 3.

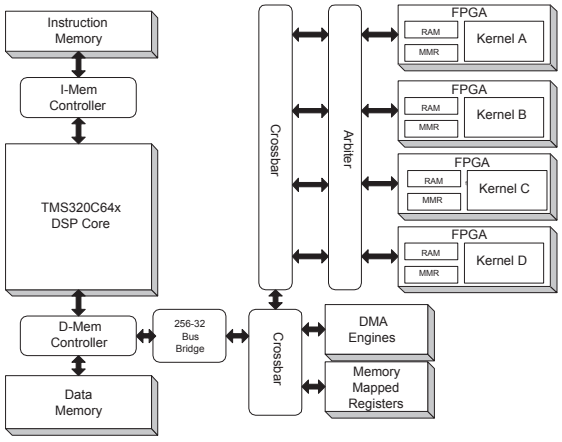

Fig. 3. Hetergeneous DSP/FPGA Architecture as Modelled with the Spinach Simulation Infrastructure

Table I shows the various architectural parameters used in the simulated system topology for both the wireless channel equalizer and the video decoder. The DSP architecture modelled is the Texas Instruments TMS320C64x series device operating at a clock rate of 167 MHz. The instruction and data memory bandwidths are 256 bits respectively, while all arbitration policies are set to round robin.

V. RESULTS

For both the H.263 video decoder and the wireless channel equalizer workloads described in Section III, the partitioning criteria as presented in Section IV are combined with the software only implementations runtime profile performance on the programmable DSP core to arrive at a system partitioning. Figure 4 shows the resultant hardware software partitioning for the wireless channel equalizer, whereby tasks shown in white are executing in software on the programmable DSP core and tasks shown in grey are migrated into a hardware based FPGA implementation. Specifically, portions of the Channel Estimation, FIR Filtering, Despreading/Descrambling, and Fast Fourier and Inverse Fast Fourier Transforms are placed in FPGAs. In the case of Channel Estimation however, the kernel is parallelized across both the programmable DSP core and FPGA to achieve optimal parallelism.

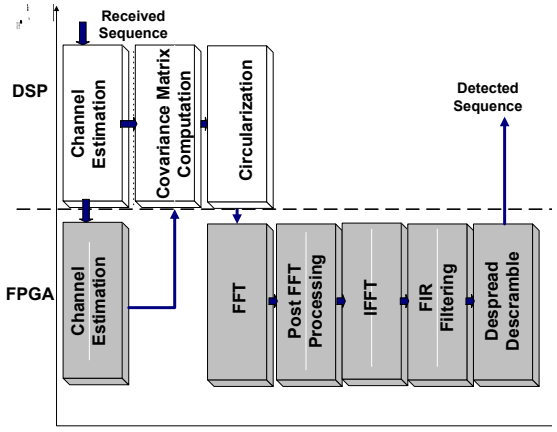

Fig. 4. Channel Equalizer DSP/FPGA Partitioning

Figure 5 shows the corresponding resultant partitioning for the H.263 video decoder. Specifically, the video block combination and summation routines are moved into FPGA based implementation, as well as the Inverse Discrete Cosine Transform and Motion Compensation routines. In both Figures 4 and 5, any functionality migrated into an FPGA requires explicit DMA transfers of compute data from local DSP program memory to the local RAM arrays of the FPGA, and back to DSP program memory upon completion of computation.

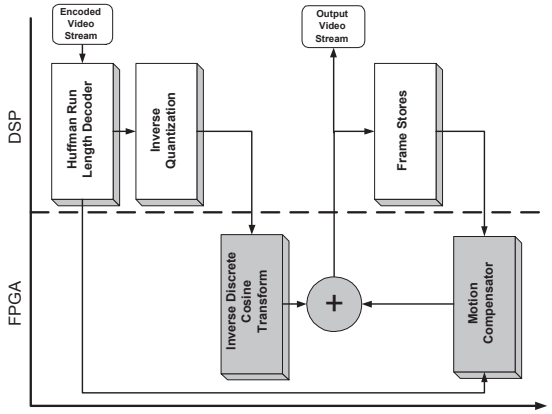

Fig. 5. H.263 Video Decoder DSP/FPGA Partitioning

In the case of H.263 video decoding, it is determined that inverse discreet cosine transform, motion compensation, and various video decoder block and macroblock routines are to be offloaded into hardware for optimal performance. It should be noted that all

computation offloaded to hardware based FPGA implementation must be controlled via the programmable DSP in terms of data transfers to local FPGA RAM arrays, as well as synchronization with loosely coupled software based tasks on the DSP. All computation within the simulated system topology is performed in a bit–true, cycle–accurate manner running compiler generated executables on DSP, or in the case of FPGA implementation, according to hardware gate counts and measured computational latencies.

Using the hardware software partitionings described above, Figures 6 and 7 show the program runtimes of the wireless Channel Equalizer as H.263 video decoder as computational bottlenecks in the software only implementation are iteratively offloaded to FPGA based implementations, using the system topology described in Figure 3. The stacked bar graphs show the component of the total runtime each kernels considered for offloading takes, as increasing amounts of functionality are partitioned into hardware.

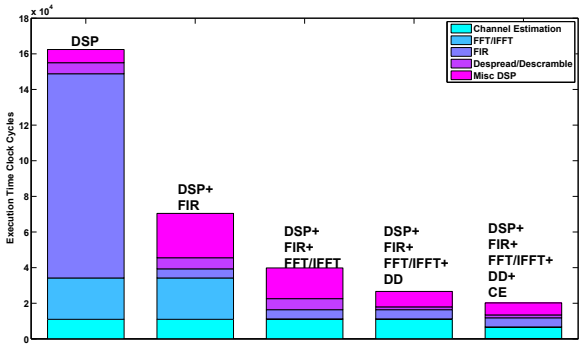

Fig. 6. Channel Equalizer Runtime vs DSP/FPGA Partitioning Strategy

Looking at Figure 6, the left most stacked bar shows the Channel Equalizer runtime performance when implemented entirely in software only using the programmable DSP core in the system. The second bar to the left shows the performance when the FIR Filtering kernel is offloaded to FPGA. Similarly, the subsequent bars show the additional offloading of the Fast Fourier and Inverse Fast Fourier Transforms, the Channel Equalization Kernel and lastly the inclusion of also the Despreading and Descrambling portion of the workload. In the fully partitioned implementation of the Channel Equalizer, shown in the rightmost column of Figure 6, there is an 11.2x improvement of the equalizers runtime performance. This includes all data transfer overheads and synchronization overheads between processing elements in the system. This actually allows the channel equalizer to process samples at a rate 64% faster than the HSDPA standard mandates, whereas the software only solution is severely bottlenecked and only processing samples at 13

Looking at the performance of the H.263 video decoder, Figure 7 shows the video decoder performance as DSP computational bottlenecks are isolated and migrated into an FPGA based implementation. In this example, however, for many of the kernels considered for offloading, the amount of parallelism implemented via the functional units within the FPGA is varied. The left most stacked bar of Figure 7 shows the composite performance of the software only implementation of the video decoder. The second to left most bar of this figure shows the performance when the block level summation and compare routines are migrated into FPGA. The block summation routines typically perform computations on 8x8 tiles of 16 bit or 8

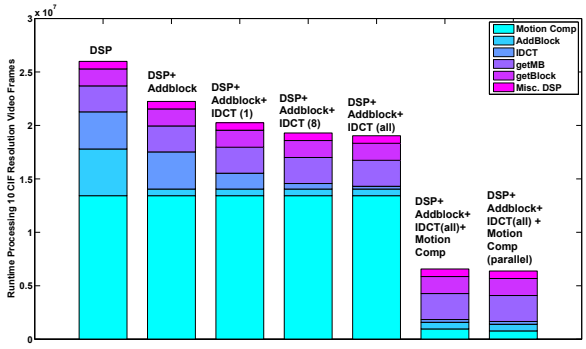

Fig. 7. H.263 Video Decoder Runtime vs DSP/FPGA Partitioning Strategy

bit values, rapidly comparing and summing them. This computation is more suited to the kind of highly parallel implementation that an FPGA can provide due to the large amount of instruction level parallelism in the kernel. Because the programmable DSP core is an 8 wide clustered VLIW pipeline, the amount of instruction and data level parallelism that can be effectively tackled in parallel is limited.

Continuing with Figure 7, the stacked bars in positions three through five show the additional offloading of the Inverse Discrete Cosine Transform in conjunction with the block summation routines. Three different implementations for Inverse Discrete Cosine Transform are shown here, namely offload the row by row, and column by column computation of the Cosine Transform to FPGA, offloading the entire row parallel computation followed by the column parallel computation, and lastly offloading the row parallel and column parallel computation into the same FPGA. The most aggressive Inverse Discrete Cosine parallelization, namely column five of Figure 7 shows the greatest improvement in performance. Interestly however, increasing the amount of parallelism available for the Cosine Transform does not drastically effect the runtime performance over the less parallel implementations. This is primarily due to two reasons: first the Cosine Transform does not account for as much of the decoder runtime as other kernels due it not being located in the motion compensation feedback loop, and secondly the data transfer overheads during DMA can shadow the gains achieved by computation alone.

Looking at the right most two columns of Figure 7, in addition to the block level summation routines and Inverse Discrete Cosine Transform, two implementations of the Motion Compensation routines are added with the last one being explicitly parallelized in FPGA hardware and having a much shorter computational latency. With this most aggressive parallelization strategy, the overall runtime performance of the video decoder is increased by a factor of 4.2x. Similar to the case of the highly parallelized Cosine Transform discussed above, the aggressively parallelized FPGA based Motion Compensation routine does not afford drastic runtime improvements over the more modest implementation. The reason for this are that while the Motion Compensation routine does lie in the feedback loop of the video decoder, and does account for a significant portion of the total decoder runtime, data transfer overheads of the DMA engines overshadows the computational portion of the FPGA runtime and thus limits total improvement gains.

While the wireless Channel Equalizer saw improvement gains on the order of 11.2x, the video decoder saw much more modest

improvements in runtime performance on the order of 4.2x. The reasons for this are two fold. First, the video decoder software architecture was such that portions of the codec were not readily available for implementation in FPGA. For example, data was mapped non-contiguously in the global space or allocated on heaps. Control flow dominant portions of the code used non-well defined structure and erroneous control flow via go-to type statements. Additionally the algorithmic details made data usage patterns of the incoming frame stores non-deterministic at runtime and thus difficult to migrate into fixed buffer sized FPGAs. The final reason for the more modest gains of the video decoder were due to the input problem size. CIF resolution video is rather small in frame size compared with larger resolutions such as 16CIF or even other codec standards like H.264. While this affords a shorter simulation time and smaller memory footprint, it limits the amount of instruction and data level parallelism seen in the kernels. The overall data trends are valid for larger frame sizes and more modern standards, and in fact would be more pronounced when using a larger resolution video sequence. For the sake of keeping simulation tractable, however, smaller input data sets were used.

The improvements on the video decoder are somewhat more modest due to the initial software organization and data layout of the decoder, which in some cases requires data reorganization and manipulation before DMA transfers to FPGA for local parallel computation. Additionally, video decoding was performed on CIF video streams, which have a much smaller data footprint and fewer blocks and macro-blocks to compute on versus 4CIF and 16CIF video or video for the H.264 standard. These performance gains are impressive even for CIF resolution video frames, and will be more pronounced for higher resolution encodings such as 4CIF and 16CIF for the H.263 video standard.

VI. Conclusions

This paper shows a hardware/software codesign methodology for partitioning signal processing and multimedia applications across software programmable DSP cores and hardware based FPGA compute engines in embedded environments. By using a determined set of criteria for partitioning software based applications across one or more FPGA based compute elements, the parallel computation available in FPGA based hardware can be used to overcome limits in instruction and data level parallelism inherent with modern VLIW style DSP cores. By combining this hardware/software partitioning methodology with a prototyping environment for simulation of heterogeneous DSP/FPGA based embedded systems, performance gains on the order of 3.8x to 11x can be achieved in the embedded signal processing and multimedia domains. By intelligently exploring the design space, and using an simulator prototyping environment for heterogeneous DSP/FPGA architectures, an iterative design flow of hardware/software codesign is achieved.

Acknowledgment

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

References

[1] M. E. Al-Mualla, C. N. Canagarajah, and D. R. Bull. *Video Coding for Mobile Communications*. Academic Press, 2002.

[2] 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.

[3] 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.

[4] 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.

[5] M. Ghanbari. *Video Coding: An Introduction to Standard Codecs*. The Institute Of Electrical Engineers, London, UK, 1999.

[6] 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.