Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 14952, Pages 1–25
DOI 10.1155/ES/2006/14952

Rapid Industrial Prototyping and SoC Design of 3G/4G Wireless Systems Using an HLS Methodology

Yuanbin Guo,[1] Dennis McCain,[1] Joseph R. Cavallaro,[2] and Andres Takach[3]

[1] *Nokia Networks Strategy and Technology, Irving, TX 75039, USA*
[2] *Department of Electrical and Computer Engineering, Rice University, Houston, TX 77005, USA*
[3] *Mentor Graphics, Portland, OR 97223, USA*

Received 4 November 2005; Revised 10 May 2006; Accepted 22 May 2006

Many very-high-complexity signal processing algorithms are required in future wireless systems, giving tremendous challenges to real-time implementations. In this paper, we present our industrial rapid prototyping experiences on 3G/4G wireless systems using advanced signal processing algorithms in MIMO-CDMA and MIMO-OFDM systems. Core system design issues are studied and advanced receiver algorithms suitable for implementation are proposed for synchronization, MIMO equalization, and detection. We then present VLSI-oriented complexity reduction schemes and demonstrate how to interact these high-complexity algorithms with an HLS-based methodology for extensive design space exploration. This is achieved by abstracting the main effort from hardware iterations to the algorithmic C/C++ fixed-point design. We also analyze the advantages and limitations of the methodology. Our industrial design experience demonstrates that it is possible to enable an extensive architectural analysis in a short-time frame using HLS methodology, which significantly shortens the time to market for wireless systems.

1. INTRODUCTION

The radical growth in wireless communication is pushing both advanced algorithms and hardware technologies for much higher data rates than what current systems can provide. Recently, extensions of the third generation (3G) cellular systems such as universal mobile telecommunications system (UMTS) lead to the high-speed downlink packet access (HSDPA) [1] standard for data services. On the other hand, multiple-input multiple-output (MIMO) technology [2, 3] using multiple antennas at both the transmitter and the receiver has been considered as one of the most significant technical breakthroughs in modern communications because of its capability to significantly increase the data throughput. Code-division multiple access (CDMA) [4] and orthogonal frequency-division multiplexing (OFDM) [5] are two major radio access technologies for the 3G cellular systems and wireless local area network (WLAN). The MIMO extensions for both CDMA and OFDM systems are considered as enabling techniques for future 3G/4G systems.

Designing efficient VLSI architectures for the wireless communication systems is of essential academical and industrial importance. Recent works on the VLSI architectures for the CDMA [6] and MIMO receivers [7] using the original vertical Bell Labs layered space-time (V-BLAST) scheme have been reported. The conventional bank of matched filters or Rake receiver for the MIMO extensions was implemented with a target at the OneBTS^{TM} base station in [8] for the flat-fading channels [2, 3]. However, in a realistic environment, the wireless channel is mostly frequency selective because of the multipath propagation [9]. Interferences from various sources become the major limiting factor for the MIMO system capacity. Much more complicated signal processing algorithms are required for desirable performance.

For the MIMO-CDMA systems, the linear minimum mean-squared error (LMMSE) chip equalizer [10] improves the performance by recovering the orthogonality of the spreading codes, which is destroyed by the multipath channel, to some extent. However, this in general sets up a problem of matrix inversion, which is very expensive for hardware implementation. Although the MIMO-OFDM systems eliminate the need for complex equalizations because of the use of cyclic prefix, the data throughput offered by the conventional V-BLAST [2, 7, 8] detector is far from the theoretic bound. Maximum-likelihood (ML) detection is theoretically optimal, however, the prohibitively high complexity makes it not implementable for realistic systems. A suboptimal QRD-M symbol detection algorithm was proposed in [5] which approaches the ML performance using limited-tree search.

However, its complexity is still too high for real-time implementation.

These high-complexity signal processing algorithms give tremendous challenges for real-time hardware implementation, especially when the gap between algorithm complexity and the silicon capacity keeps increasing for 3G and beyond wireless systems [11]. Much more processing power and/or more logic gates are required to implement the advanced signal processing algorithms because of the significantly increased computation complexity. System-on-chip (SoC) architectures offer more parallelism than DSP processors. Rapid prototyping of these algorithms can verify the algorithms in a real environment and identify potential implementation bottlenecks, which could not be easily identified in the algorithmic research. A working prototype can demonstrate to service providers the feasibility and show possible technology evolutions [8], thus significantly shortening the time to market.

In this paper, we present our industrial experience in rapidly prototyping these high-complexity signal processing algorithms. We first analyze the key system design issues and identify the core components of the 3G/4G receivers using multiple-antenna technologies, that is, the MIMO-CDMA and MIMO-OFDM, respectively. Advanced receiver algorithms suitable for implementation are proposed for synchronization, equalization, and MIMO detection, which form the dominant part of receiver design and reflect different classes of computationally intensive algorithms typical in future wireless systems. We propose VLSI-oriented complexity reduction schemes for both the chip equalizers and the QRD-M algorithm and make them more suitable for real-time SoC implementation. SoC architectures for an FFT-based MIMO-CDMA equalizer [4] and a reduced complexity QRD-M MIMO detector are presented.

On the other hand, there are many area/time tradeoffs in the VLSI architectures. Extensive study of the different architecture tradeoffs provides critical insights into implementation issues that may arise during the product development process. However, this type of SoC design space exploration is extremely time consuming because of the current trial-and-optimize approaches using hand-coded VHDL/Verilog or graphical schematic design tools [12, 13].

Research in high-level synthesis (HLS) [14–16] aimed at automatically generating a design from a control data flow graph (CDFG) representation of the algorithm to be synthesized into hardware. The specification style of the first commercial realization of HLS is a mixture of functionality and I/O timing expressed in languages such as VHDL, Verilog, SystemC [17], Handel-C [18], or System Verilog. While the behavioral coding style appears more algorithmic (use of loops for instance), the mixture of such behavior with I/O cycle timing specification provides an awkward way to specify cycle timing that often overconstrains the design. This specification style was introduced by Knapp et al. [19] and was the basis for behavioral tools such as *Behavioral Compiler* introduced in 1994 by Synopsys, *Monet* introduced by Mentor Graphics in 1997, *Volare* from Get2Chip (acquired in 2003 by Cadence), *CoCentric SystemC Compiler* introduced in 2000 by Synopsys, and *Cynthesizer* from Forte (based on SystemC [17]). The first three tools were based on VHDL/Verilog. All but *Cynthesizer* are no longer in the market. C-Level's HLS tool (no longer in the market) used specifications in a subset of C where pipelining had to be explicitly coded. Celoxica's HLS tool was initially based on cycle-accurate Handel-C [18] with explicit specification of parallelism. Their tool is now called *Agility Compiler* and it supports SystemC. *BlueSpec Compiler* targets mainly control-dominated designs and uses System Verilog with Bluespec's proprietary assertions as the language for specification. Reference [20] presented a Matlab-to-hardware methodology which still requires significant manual design work. To meet the fast changing market requirements in wireless industry, a design methodology that can efficiently study different architecture tradeoffs for high-complexity signal processing algorithms in wireless systems is highly desirable.

In the second part, we present our experience of using an algorithmic sequential ANSI C/C++ level design and verification methodology that integrates key technologies for truly high-level VLSI modeling of these core algorithms. A Catapult C-based architecture scheduler [21] is applied to explore the VLSI design space extensively for these different types of computationally intensive algorithms. We first use two simple examples to demonstrate the concept of the methodology and how to make these high-complexity algorithms interact with the HLS methodology. Different design modes are proposed for different types of signal processing algorithms in the 3G/4G systems, namely, throughput mode for the front-end streaming data and block mode for the post-processing algorithms. The key factors for optimization of the area/speed in loop unrolling, pipelining, and the resource sharing are identified. Extensive time/area tradeoff study is enabled with different architectures and resource constraints in a short design cycle by abstracting the main effort from hardware iterations to the algorithmic C/C++ fixed-point design. We also analyze the strengths and limitations of the methodology.

We also propose different hardware platforms to accomplish different prototyping requirements. The real-time architectures of the CDMA systems are implemented in a multiple-FPGA-based Nallatech [22] real-time demonstration platform, which was successfully demonstrated in the Cellular Telecommunications and Internet Association (CTIA) trade show. A compact hardware accelerator for both precommercial functional verification and simulation acceleration of the QRD-M MIMO detector is also implemented in a Wildcard PCMCIA card [23]. Our industrial design experience demonstrates that it is possible to enable an extensive architectural analysis in a short-time frame using HLS methodology, which leads to significant improvements in rapid prototyping of 3G/4G systems.

The rest of the paper is organized as follows. We first describe the model of 3G/4G wireless systems using MIMO technologies and identify the prototyping and methodology requirements. We then present our prototyping experience for advanced 3G MIMO-CDMA receivers and 4G MIMO-OFDM systems in Sections 3 and 4, respectively.

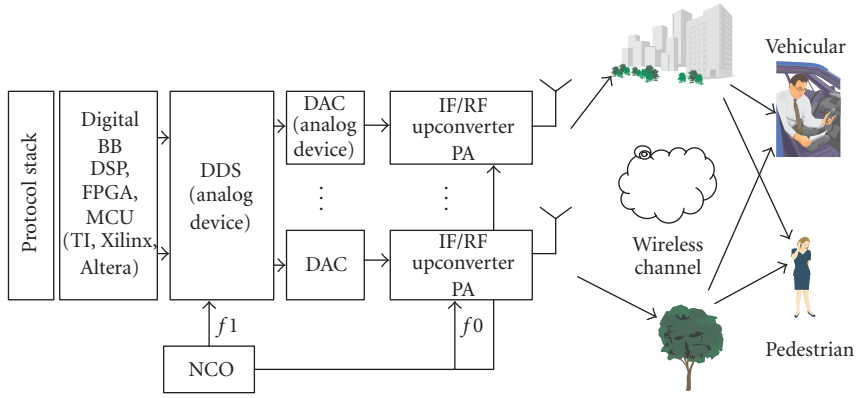

Figure 1: A realistic MIMO-CDMA transmitter block diagram with digital baseband and analog RF modules.

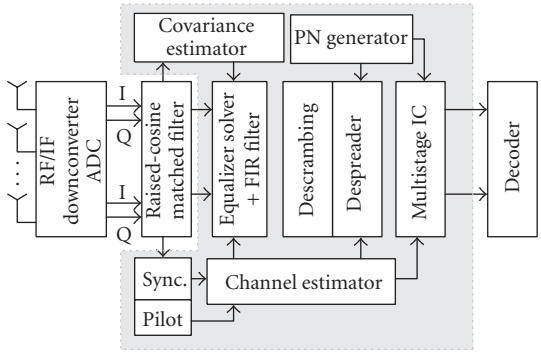

Figure 2: Advanced receiver system model for the MIMO-CDMA downlink.

The Catapult C HLS design methodology is presented in Section 5. We then demonstrate how to apply the Catapult C HLS methodology for these complexity algorithms and some experimental results in Section 6. The conclusion is given in Section 7.

2. SYSTEM MODEL AND PROTOTYPING REQUIREMENTS

2.1. CDMA downlink system model and design issues

The system model of the MIMO-CDMA downlink with M Tx antennas and N Rx antennas is described here, where usually $M \leq N$. First, the high-data-rate symbols are demultiplexed into KM lower-rate substreams using the spatial multiplexing technology [2], where K is the number of spreading codes used for data transmission. The substreams are broken into M groups, where each substream in the group is spread with a spreading code of spreading gain G. The groups of substreams are then combined and scrambled with long scrambling codes and transmitted through the mth Tx antenna. The baseband functions are usually implemented in either DSP or FPGA technologies as shown in the physical design block diagram in Figure 1. In a realistic physical implementation, the transmitter has other major modules besides the digital baseband. The protocol stack starts from the media-access-control (MAC) layer up to the network layer, application layer, and so forth. A modern implementation for a wideband system usually applies a direct digital synthesizer (DDS), for example, a component from analog devices or a digital front-end module in FPGA design. A numerically controlled oscillator (NCO) modulates the digital baseband to a digital intermediate frequency (IF). This digital IF waveform is then converted to an analog waveform using a high-speed digital-analog converter (DAC). An analog intermediate frequency (IF) and radio frequency (RF) up-converters modulate the signal to the final radio frequency. The signal passes through a power amplifier (PA) and then is transmitted through the specific antenna.

A system model for the advanced MIMO-CDMA downlink receiver is shown in Figure 2. At the receiver side, corresponding RF/IF down-converters and analog-to-digital converter (ADC) recover the analog signals from the carrier frequency and sample them to digital signals. In an outdoor environment, the signal passing the wireless channel can experience reflections from buildings, trees, or even pedestrians, and so forth. If the delay spread is longer than the coherence time, this will lead to the multipath frequency-selective channel. Significantly, more advanced receiver algorithms are required in these environments besides simple raised-cosine pulse shaping [9] because the simple pulse shaping is not enough for various channel environments. Synchronization is usually the first core design block in a CDMA receiver because it recovers the signal timing with the spreading codes from clock shift and frequency offsets.

For a CDMA downlink system in a multipath fading channel, the orthogonality of the spreading codes is destroyed, introducing both multiple-access interference (MAI) and intersymbol interference (ISI). HSDPA is the evolutionary mode of WCDMA [1], with a target to support wireless multimedia services. The conventional Rake receiver [8] could not provide acceptable performance because of the very short spreading gain to support high-rate data services. LMMSE chip equalizer is a promising algorithm to restore

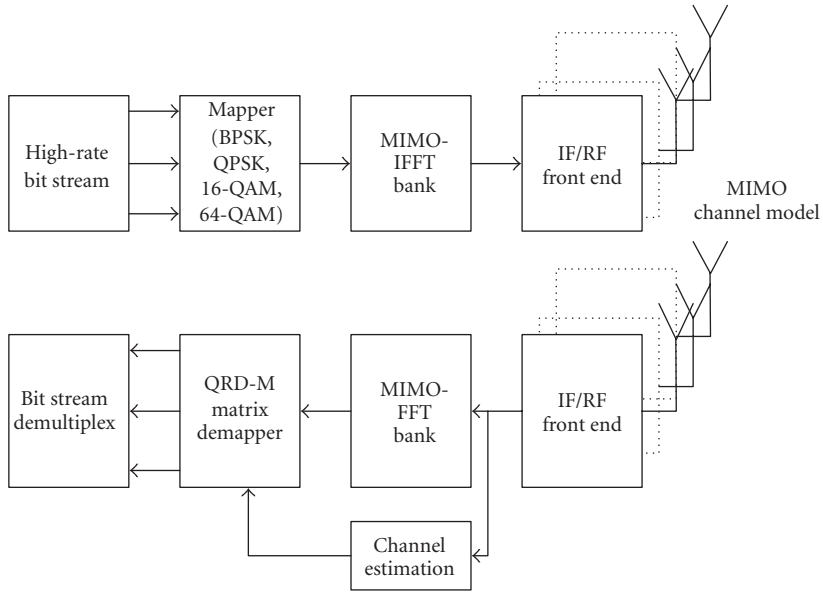

Figure 3: System model of the MIMO-OFDM using spatial multiplexing.

the orthogonality of the spreading code and suppress both the ISI and MAI [10]. However, this involves the inverse of a large correlation matrix with $\mathcal{O}((NF)^3)$ complexity for MIMO systems, where N is the number of Rx antennas and F is the channel length. Traditionally, the implementation of an equalizer in hardware has been one of the most complex tasks for receiver designs.

In a complete receiver design, some channel estimation and covariance estimation modules are required. The equalized signals are descrambled and despread and sent to the multistage interference cancellation (IC) module. Finally, the output of the IC module will be the input to some channel decoder, such as turbo decoder or low-density parity check (LDPC) decoders. The advanced receiver algorithms including synchronization, MIMO equalization, interference cancellation, and channel decoder dominate the receiver complexity. In this paper, we will focus on the VLSI architecture designs of the synchronization and channel equalization because they represent different types of complex algorithms. Although there are tremendous separate architectural research activities for interference cancellation and channel coding in the literature, they are beyond the scope of this paper and are considered as intellectual property (IP) cores for system-level integration.

2.2. System model and design issues for MIMO-OFDM

MIMO-OFDM is considered as an enabling technology for the 4G standards. The OFDM technology converts the multipath frequency-selective fading channel into flat fading channel and simplifies the channel equalization by inserting cyclic prefix to eliminate the intersymbol interference. The MIMO-OFDM system model with N_T transmit and N_R receive antennas is shown in Figure 3. At the pth transmit antenna, the multiple bit substreams are modulated by constellation mappers to some QPSK or QAM symbols. After the insertion of the cyclic prefix and multipath fading channel propagation, an N_F-point FFT is operated on the received signal at each of the qth receive antennas to demodulate the frequency-domain symbols.

It is known that the optimal maximum-likelihood detector [24] leads to much better performance than the original V-BLAST symbol detection. However, the complexity increases exponentially with the number of antennas and symbol alphabet, which is prohibitively high for practical implementation. To achieve a good tradeoff between performance and complexity, a suboptimal QRD-M algorithm was proposed in [5] to approximate the maximum-likelihood detector. The QR-decomposition [25] reduces the K effective channel matrices for N_T transmit and N_R receive antennas to upper-triangular matrices. The M-search algorithm limits the tree search to the M smallest branches in the metric computation. The complexity is significantly reduced compared with the full-tree search of the maximum-likelihood detector. However, the QRD-M algorithm is still the bottleneck in the receiver design, especially for the high-order modulation, high MIMO antenna configuration, and large M. It is shown by a Matlab profile that the M-algorithm can occupy more than 99% of the computation in a MIMO-OFDM 4G simulation chain. It can take days or even weeks to generate one performance point. This not only slows the research activity significantly, but also limits the practicability of the QRD-M algorithm in real-time implementation. However, the tree search structure is not quite suitable for VLSI implementation because of intensive memory operations with variable latency, especially for a long sequence. Extensive algorithmic optimizations are required for efficient hardware architecture.

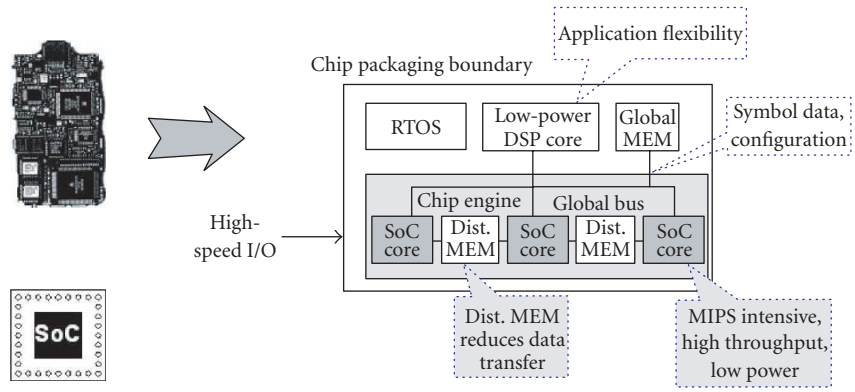

Figure 4: SoC partitioning for computational efficiency, configurability, MOPS/μW, and flexibility/scalability.

On the other hand, since there is still no standardization of 4G systems, the tremendous efforts to build a prestandard real-time end-to-end complete system still do not give much commercial motivation to the wireless industries. However, there is a strong motivation to demonstrate the feasibility of implementing high-performance algorithms such as the QRD-M detector in a low-cost real-time platform to the business units. There is also a strong motivation to shorten the simulation time significantly to support the 4G research activities. Implementation of the high-complexity MIMO detection algorithms in a hardware accelerator platform with compact form factor will significantly facilitate the commercialization of such superior technologies. The limited hardware resource in a compact form factor and much lower clock rate than PC demands very efficient VLSI architecture to meet the real-time goal. The efficient VLSI hardware mapping to the QRD-M algorithm requires wide-range configurability and scalability to meet the simulation and emulation requirements in Matlab. This also requires an efficient design methodology that can explore the design space efficiently.

2.3. Architecture partitioning requirement

"System-on-a-chip with intellectual property" (SoC/IP) is a concept that a chip can be constructed rapidly using third-party and internal IP, where IP refers to a predesigned behavioral or physical description of a standard component. The ASIC block has the advantage of high throughput speed, and low power consumption and can act as the core for the SoC architecture. It contains custom user-defined interface and includes variable word length in the fixed-point hardware datapath. field-programmable gate array (FPGA) is a virtual circuit that can behave like a number of different ASICs which provide hardware programmability and the flexibility to study several area/time tradeoffs in hardware architectures. This makes it possible to build, verify, and correctly prototype designs quickly.

The SoC realization of a complicated end-to-end communication system, such as the MIMO-CDMA and MIMO-OFDM, highly depends on the task partitioning based on the real-time requirement and system's resource usage, which roots from the complexity and computational architecture of the algorithms. The system partitioning is essential to solve the conflicting requirements in performance, complexity, and flexibility. Even in the latest DSP processors, computational intensive blocks such as Viterbi and turbo decoders have been implemented as ASIC coprocessors. The architectures should be efficiently parallelized and/or pipelined and functionally synthesizable in hardware. A general architecture partitioning strategy is shown in Figure 4. The SoC architecture will finally integrate both the analog interface and digital baseband together with a DSP core and be packed in a single chip. The VLSI design of the physical layer, one of the most challenging parts, will act as an engine instead of a coprocessor for the wireless link. Unlike a processor type of architecture, high efficiency and performance will be the major target specifications of the SoC design.

2.4. Rapid prototyping methodology requirements

The hardware design challenges for the advanced signal processing algorithms in 3G/4G systems lead to a demand for new methodologies and tools to address design, verification, and test problems in this rapidly evolving area. In [26], the authors discussed the five-ones approach for rapid prototyping of wireless systems, that is, one environment, one automatic documentation, one code revision tool, one code, and one team. This approach also applies to our general requirements of prototyping. Moreover, a good development environment for high-complexity wireless systems should be able to model various DSP algorithms and architectures at the right level of abstraction, that is, hierarchical block diagrams that accurately model time and mathematical operations, clearly describe the real-time architecture, and map naturally to real hardware and software components and algorithms. The designer should also be able to model other elements that affect baseband performance, channel effects, and timing recovery. Moreover, the abstraction should facilitate the modeling of sample sequences, the grouping of the sample sequences into frames, and the concurrent operation of multiple rates inherent in modern communication systems.

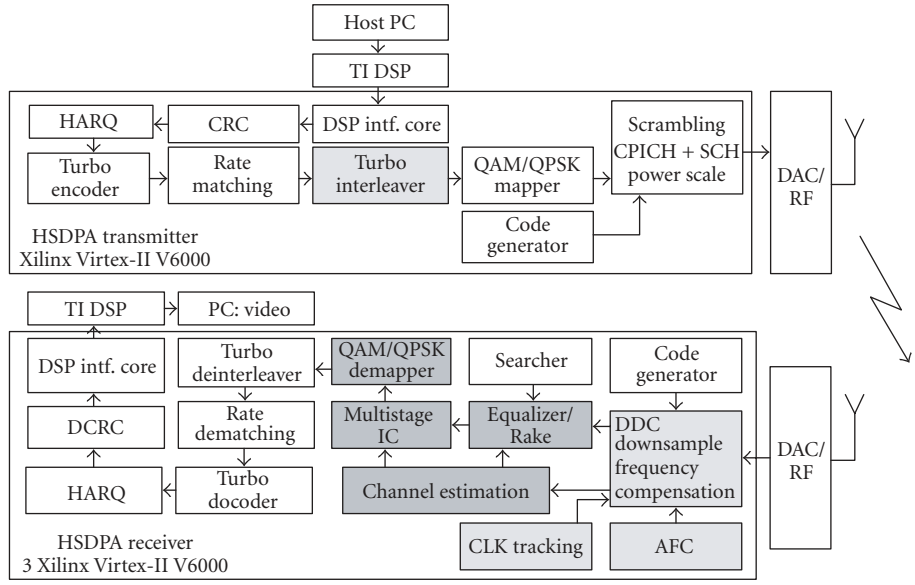

Figure 5: System blocks for the HSDPA demonstrator.

The design environment must also allow the developer to add implementation details when, and only when, it is appropriate. This provides the flexibility to explore design tradeoffs, optimize system partitioning, and adapt to new technologies as they become available.

The environment should also provide a design and verification flow for the programmable devices that exist in most wireless systems including general-purpose microprocessors, DSPs, and FPGAs. The key elements of this flow are automatic code generation from the graphical system model and verification interfaces to lower-level hardware and software development tools. It also should integrate some downstream implementation tools for the synthesis, placement, and routing of the actual silicon gates.

3. ADVANCED 3G RECEIVER REAL-TIME PROTOTYPING

The advanced HSDPA receiver for rapid prototyping is the evolutionary mode of WCDMA [1] to support wireless multimedia services in the cellular devices. MIMO extensions are proposed for increased data throughput. In this section, we present our real-time industrial prototyping designs for the advanced receiver using high-complexity signal processing algorithms.

3.1. System partitioning

Because of the real-time demonstration requirement, the complete system design needs a lot of processing power. For example, the turbo decoder for the downlink receiver alone occupies 80% of the area of a Virtex II V6000. We apply the Nallatech BenNUEY multiple-FPGA computing platform for the baseband architecture design. Each motherboard can hold up to seven BenBlue II user FPGAs in a single PCI motherboard. These FPGAs include Xilinx Virtex II V6000 to V8000. Multiple I/O and analog interface cards can also be attached to the PCI card. This provides a powerful platform for high-performance 3G demonstration. We also apply TI's C6000 serial DSP to support high-speed MAC layer design.

In the transmitter, the host computer runs the network layer protocols and applications. It has interfaces with the DSP, which hosts the media-access-control (MAC) layer protocol stack and handles the high-speed communication with FPGAs. A DSP interface core in the transmitter reads the data from the DSP and adds cyclic redundancy check (CRC) code. After the turbo encoder, rate matching, and interleaver, a QPSK/QAM mapper modulates the data according to the hybrid automatic request (HARQ) control signals. With the common pilot channel (CPICH) and synchronization channel (SCH) information inserted, the data symbols are spread and scrambled with pseudonoise (PN) long code and then ported to the RF transmitter. At the receiver, the searcher finds the synchronization point. Clock tracking and automatic frequency control (AFC) are applied for fine synchronization. After the matched filter receiver, received symbols are demodulated and deinterleaved before the rate dematching. Then after a turbo decoder decodes the soft decisions to a bit stream, a HARQ block is followed to form the bit stream for the upper-layer applications. In Figure 5, we also depict other key advanced algorithms including channel estimation, chip-level equalizer, and multistage interference cancellation to eliminate the distortions caused by the wireless multipath and fading channels. The clock tracking and AFC which are slightly shaded will be used as the simple cases to demonstrate the concept of using Catapult C HLS design methodology. The darkly shaded blocks in the MIMO scenario will be the focus for high-complexity architecture design.

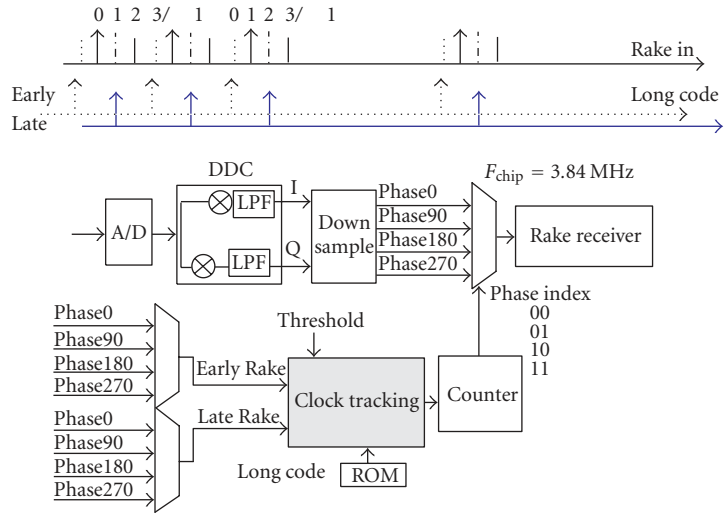

Figure 6: Clock tracking based on late-early correlation estimation in CDMA systems.

3.2. CDMA receiver synchronization

3.2.1. Clock-tracking algorithm

The mismatch of the transmitter and receiver crystals will cause a phase shift between the received signal and the long scrambling code. The "clock-tracking" algorithm [27] will track the code sampling point. The IF signal is sampled at the receiver and then down-converted with a digital demodulation at local frequency. The separated I/Q channel is then downsampled to four phases' signals at the chip rate, which is 3.84 MHz. By assuming one phase as the in-phase, we compute the correlation of both the earlier phase and the later phases with the descrambling long code according to the frame structure of HSDPA. When the correlation of one phase is much larger than another phase (compared with a threshold), it will then be judged that the sample should be moved ahead or delayed by one-quarter chip. Thus the resolution of the code tracking can be one quarter of a chip. This principle is shown in Figure 6.

The system interface for clock tracking is also depicted in Figure 6. At the downsampling block after the DDC (digital down-converter) Xilinx core, the in-phase, early, late phases are sent to both the Rake receiver and clock tracking. The long code will be loaded from ROM block. The clock-tracking algorithm computes both early/late correlation powers after descrambling, chip-matched filter, and accumulation stages. A flag is generated to indicate early, in-phase or late as output. This flag is used to control the adjustment signal of a configurable counter. The adjusted in-phase samples are then sent to the Rake receiver for detection. Thus the clock tracker is integrated with IP cores and the other HDL designer blocks (downsampling, MUX, etc.).

3.2.2. Automatic frequency control

The frequency offset is caused by the Doppler shift and frequency offset between the transmitter and the receiver oscillators. This makes the received constellations rotate in addition to the fixed channel phases, and thus dramatically degrades performance. AFC is a function to compensate for the frequency offset in the system. For a software definable radio (SDR) type of architecture, the frequency offset is computed with a DSP algorithm and controlled by a numerical control oscillator (NCO).

We apply a spectrum-analysis-based AFC algorithm. The principle is explained with the frame structure of HSDPA in Figure 7. There are 15 slots in each frame. In each slot, the first 5 bits are pilot symbols and the second 5 bits are control signals. Each symbol is spread by a 256-chip long code. So in the algorithm, we first use a long code to descramble the received signal at the chip rate. We then do the matched filtering by accumulating 256 chips. By using the local pilot's conjugate, we get the dynamic phase of the signal with the frequency offset embedded. To increase the resolution, we finally accumulate each of the 5 pilot bits as one sample. The 5-bit control bits are skipped. Thus the sampling rate for the accumulated phase signals is reduced to be 1500 Hz. These samples are stored in a dual-port RAM for the spectrum analysis using FFT. After the descrambling and matched filter, as well as accumulation, we achieve a very stable sinusoid waveform for the frequency offset signal as shown in the figure.

3.3. VLSI system architecture for FFT-based equalizer

LMMSE chip equalizer is promising to suppress both the intersymbol interference and multiple-access interference [4] for a MIMO-CDMA downlink in the multipath fading channel. Traditionally, the implementation of equalizer in hardware has been one of the most complex tasks for receiver designs because it involves a matrix inverse problem of some large covariance matrix. The MIMO extension gives even more challenges for real-time hardware implementation.

In our previous paper [4], we proposed an efficient algorithm to avoid the direct matrix inverse in the chip equalizer

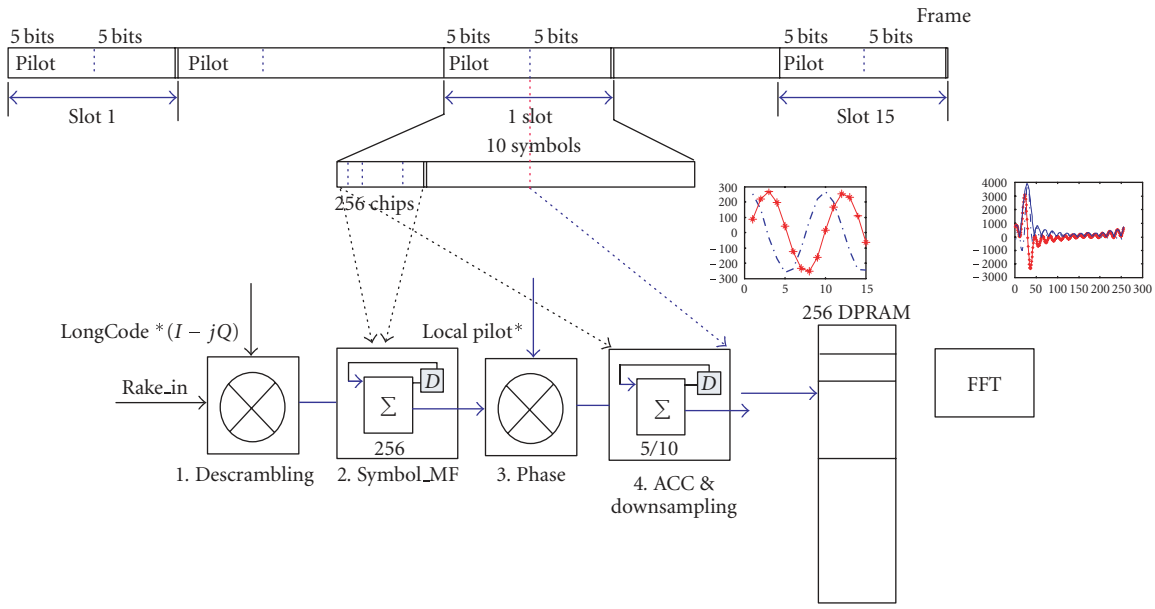

Figure 7: Spectrum-analysis-based automatic frequency control.

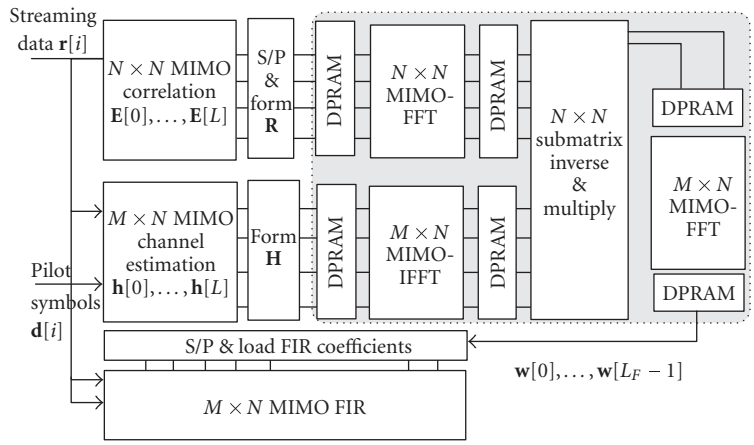

Figure 8: VLSI architecture blocks of the FFT-based MIMO equalizer.

by approximating the block Toeplitz structure of the correlation matrix with a block circulant matrix. With a timing and data-dependency analysis, the top-level VLSI design blocks for the MIMO equalizer are shown in Figure 8. In the front end, a correlation estimation block takes the multiple input samples for each chip to compute the correlation coefficients of the first column of $\mathbf{R}_{rr}$. Another parallel data path is for the channel estimation and the $(M \times N)$ dimensionwise FFTs on the channel coefficient vectors. A submatrix inverse and multiplication block take the FFT coefficients of both channels and correlations from DPRAMs and carry out the computation. Finally an $(M \times N)$ dimensionwise IFFT module generates the results for the equalizer taps $\hat{\mathbf{w}}_m^{\text{opt}}$ and sends them to the $(M \times N)$ MIMO FIR block for filtering. To reflect the correct timing, the correlation and channel estimation modules and MIMO FIR filtering at the front end will work in a throughput mode on the streaming input samples. The FFT-inverse-IFFT modules in the dotted-line block construct the postprocessing of the tap solver. They are suitable to work in a block mode using dual-port RAM blocks to communicate the data.

4. ADVANCED RECEIVER FOR 4G MIMO-OFDM

4.1. Reduced-complexity QRD-M detection

The complexity of the optimal maximum-likelihood detector in MIMO-OFDM systems increases exponentially with the number of antennas and symbol alphabet. This complexity is prohibitively high for practical implementation. In this section, we explore the real-time hardware architecture of a suboptimal QRD-M algorithm proposed in

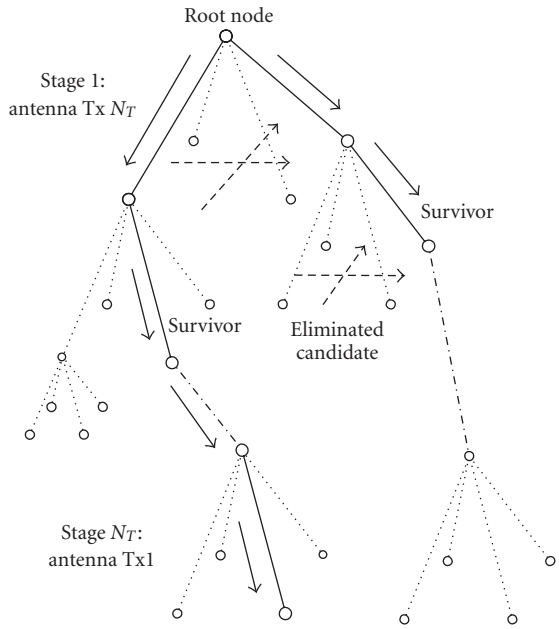

Figure 9: The limited-tree search in QRD-M algorithm.

[5] to approximate the maximum-likelihood detector. It is shown that the symbol detection is separable according to the subcarriers, that is, the components of the N_F subcarriers are independent. Thus, this leads to the subcarrier-independent maximum-likelihood symbol detection as $\mathbf{d}_{ML}^k = \arg\min_{\mathbf{d}^k \in \{\mathcal{S}\}^{N_T}} \|\mathbf{y}^k - \hat{\mathbf{H}}^k \mathbf{d}^k\|^2$, where $\mathbf{y}^k = [y_1^k, y_2^k, \ldots, y_{N_R}^k]^T$ is the kth subcarrier of all the receive antennas, $\mathbf{H}^k$ is the channel matrix of the kth subcarrier, $\mathbf{d}^k = [d_1^k, d_2^k, \ldots, d_{N_T}^k]^T$ is the transmitted symbol of the kth subcarrier for all the transmit antennas. The QR-decomposition [25] reduces the K effective channel matrices for N_T transmit and N_R receive antennas to upper-triangular matrices. The M-search algorithm limits the tree search to the M smallest branches in the metric computation. The complexity is significantly reduced compared with the full-tree search of the maximum-likelihood detector. The procedure is depicted in Figure 9 for an example with QPSK modulation and N_T transmit antennas where only the survival branches are kept in the tree search.

4.2. System-level hardware/software partitioning

As explained earlier, there is a new requirement for a pre-commercial functional verification and demonstration of the high-complexity 4G receiver algorithms. To reduce the high industrial investment of complete system prototyping before the standard is available, it makes more sense to focus on the core algorithms and demonstrate them by the hardware-in-the-loop (HITL) testing. Although the Nallatech system could also be applied for this purpose, we prefer an even more compact form factor. Thus, we propose to use Annapolis WildCard to meet both the HITL and simulation acceleration requirements. The WildCard is a single PCMCIA card which contains a Virtex II V4000 FPGA for laptops. The details of the hardware platform are found in [23].

To achieve simulation-emulation codesign, an efficient system-level partitioning of the MIMO-OFDM Matlab chain is very important. The simulation chain is depicted in Figure 10. In the simplified simulation model, the MIMO transmitter first generates random bits and maps them to constellation symbols. Then the symbols are modulated by IFFTs. A multipath channel model distorts the signal and adds AWGN noises. The receiver part is contained in the function *Hard_qrdm_fpga*, which consists of the major subfunctions such as demodulator using FFT, sorting, QR decomposition, the M-search algorithm in a C-MEX file, the demapping, and the BER calculator.

In the implementation of the QRD-M algorithm, the channel estimates from all the transmit antennas are first sorted using the estimated powers to make $\hat{P}_2^{(n_1)} \leq \hat{P}_2^{(n_2)} \leq \cdots \leq \hat{P}_2^{(n_T)}$. The data vector $\mathbf{d}^k$ is also reordered accordingly. Then the QR decomposition algorithm is applied to the estimated channel matrix for each subcarrier as $\mathbf{Q}_k^H \hat{\mathbf{H}}^k = \mathbf{R}_k$, where $\mathbf{Q}_k$ is the unitary matrix and $\mathbf{R}_k$ is an upper-triangular matrix. The FFT output $\mathbf{y}^k$ is premultiplied by $\mathbf{Q}_k^H$ to form a new receive signal as $\mathbf{Y}_k = \mathbf{Q}_k^H \mathbf{y}^k = \mathbf{R}_k \mathbf{d}^k + \mathbf{w}_k$, where $\mathbf{w}_k = \mathbf{Q}_k^H \mathbf{z}_k$ is the new noise vector. The ML detector is equivalent to a tree search beginning at level (1) and ending at level (N_T), which has a prohibitive complexity at the final stage as $\mathcal{O}(|\mathcal{S}|^{N_T})$. The M-algorithm only retains the paths through the tree with the M smallest aggregate metrics. This forms a limited tree search which consists of both the metric update and the sorting procedure. The readers are referred to [5] for details of the operations.

The top five most time-consuming functions in the simulation chain are shown in Figure 11 for the original C-MEX design for 64-QAM. The run time is obtained by the Matlab "*profile*" function. Function "*fhardqrdm_*" is the receiver function including all "*m_mex_orig*," "*channel*," "*qr*," and "*mapping*" subfunctions, where the QR-decomposition calls the Matlab built-in function. It is shown that for the original floating-point C-MEX implementation, the C-MEX implementation of the M-search function "*m_mex_orig*" dominates more than 90% of the simulation time. Moreover, all the other functions consume negligible time compared with the M-search function.

The M-search algorithm in the C-MEX file is thus implemented in the FPGA hardware accelerator. APIs talk with the CardBus controller in the card board. The controller then communicates with the processing element (PE) FPGA through the local address data (LAD) bus standard interface, which is part of the PE design. The data is stored in the input buffer and a hardware "start" signal is asserted by writing to the in-chip register. The actual PE component contains the core FPGA design to utilize both the multistage pipelining in the MIMO antenna processing and the parallelism in the subcarrier. After the output buffer is filled with detected symbols, the interrupt generator asserts a hardware interrupt signal, which is captured by the interrupt wait API in the C-MEX file. Then the data is read out from either DMA channel or status register files by the LAD output multiplexer.

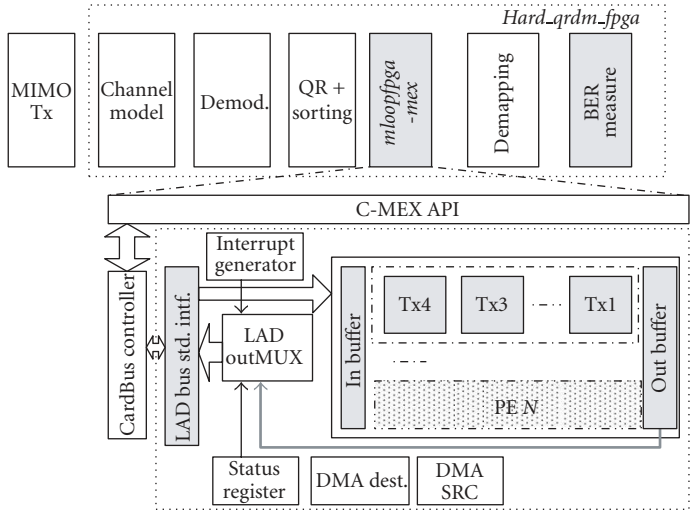

Figure 10: The system partitioning of the MIMO-OFDM simu/emulation codesign and PE architecture of the M-algorithm.

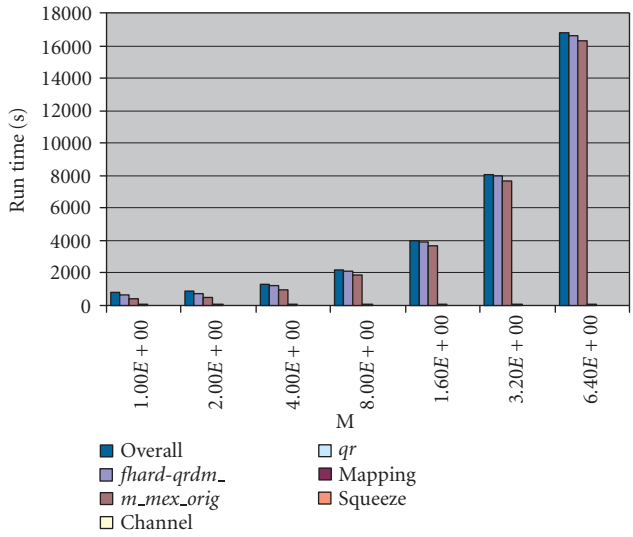

Figure 11: Measured run-time profile original C-MEX: 4×4, 64-QAM.

To achieve the bidirectional data transfer, both the source and destination DMA buffers are needed.

The architecture is designed in multistage processing elements with shared DPRAM for communication between stages. Each stage processes the detection of one Tx antenna. The symbol detection of each antenna includes three major tasks: the metric computation, sorting, and symbol detection as shown in Figure 12. An example for the antenna $nT4$ is shown in Figure 13. All the central antennas have the same operations with much higher complexity than the first and last antennas.

4.3. Partial limited tree search

Although the number of complex multiplications is an important complexity indicator because it determines the number of multipliers in a VLSI design, the real-time latency bottleneck is the sorting function. This is because the metric computation can be pipelined in the VLSI architecture with a regular structure, but the sorting function involves extensive memory access, conditional branching, element swapping, and so forth depending on the ordering feature of the input sequence.

Theoretically, the fastest sort function has the complexity at the order of $\mathcal{O}(MC * \log_2(MC))$. However, the complexity of the full sorting is too high. For example, for 64-QAM with $M = 64$, the sequence length is 4096. Then there are at least 40152 operations. If the sequence needs to be stored in block memory, this means at least these many cycles in hardware latency without counting the swapping, branching overheads. This results in 500 microseconds for a single subcarrier and one antenna assuming 100 MHz clock rate, which is very challenging to meet the real-time requirement.

However, we note that because we only retain the M smallest survivor branches, we do not care about the order of the other sequences above the M smallest metric. So only the M smallest metrics from the MC metric sequence need to be sorted. Using this observation, we modified the standard "quick-sort" procedure to the so-called "partial quick-sort" architecture.

For the partial quick-sort architecture, the metric sequence is computed separately and stored in the tmpMetric shared DPRAM blocks. Moreover, the Qsort index DPRAM contains the initial value of the sequence indices. A "*i*stack" RAM block acts as the stack to store the temporary boundary of the partitioned potential subsequences *il*, *ir*. A partial Qsort Core loads/writes data from and to the DPRAM blocks according to a finite-state machine (FSM) according to the logic flow of the partial quick-sort procedure. If the partitioned and exchanged subsequence reaches a short length, the short subsequence is sorted using the insert sort.

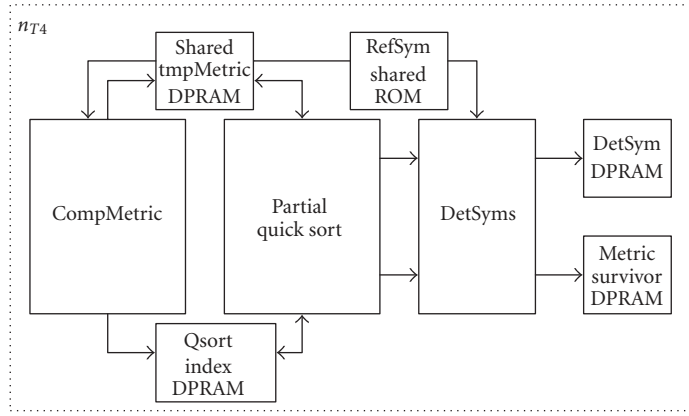

Figure 12: The block diagram of one antenna processing with quick sort.

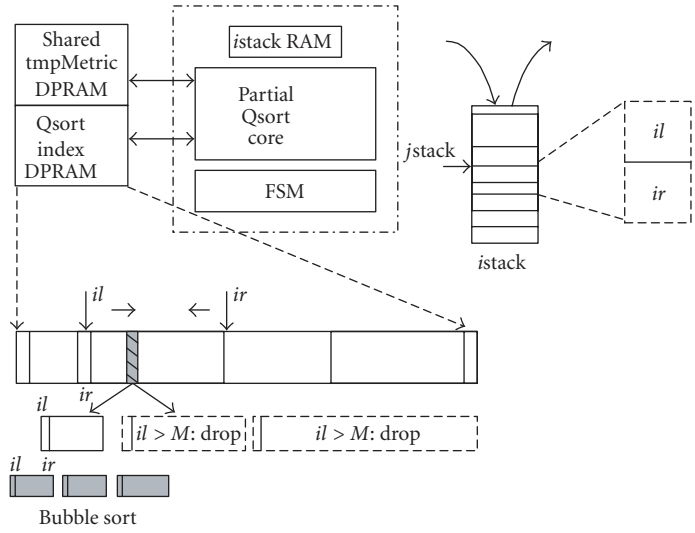

Figure 13: The block diagram of the stack-based partial quick sort.

5. CATAPULT C HLS DESIGN METHODOLOGY

5.1. Classical hardware implementation technologies

The most fundamental method of creating hardware design for an FPGA or ASIC is by using industry-standard hardware description language (HDL), such as VHDL or Verilog [13], based on data flow, structural or behavioral models. The design is specified in a register-transfer level (RTL) where the cycle-by-cycle behavior is fully specified. The details of the microarchitecture are explicitly coded. When writing the RTL description, the designer has to specify what operations are executed in what cycles, what registers are going to store the results of the operations, how and when memories are going to be accessed, and so forth. The RTL design process is manual and the intrinsic architecture tradeoffs need to be studied offline. After the architecture is crafted, the RTL code is written and validated using simulation by comparing the behavior of the RTL against the behavior of the original algorithm. After a few iterations of simulating, debugging, and code fixing, the RTL design is ready to be synthesized into the target technology (ASIC or FPGA). The results of synthesis may reveal that the design will not run at the specified frequency due to delays that were not fully accounted for when crafting the architecture such as delays from routing, multiplexing, or control logic. The results of synthesis may also reveal that the design exceeds the allocated budget for either area or power. However, it is not easy to change a design dramatically once the hardware architecture is laid out.

5.2. Raising the level of abstraction

The fundamental weakness of the RTL design methodology is that it forces designers to mix both algorithmic functionality (what the design computes) with detailed cycle timing of how that functionality is implemented. This means that the RTL code is committed to a given performance and interface requirements in conjunction to the target ASIC/FPGA technology. The low level of abstraction makes the RTL code complex and highly dependent on the crafted architecture.

Raising the level of the abstraction was recognized by researchers as a necessary step to address the issues with RTL outlined above. The most important tasks in HLS are scheduling and allocation tasks that determine the latency/throughput as well as the area of the design. Scheduling involves assigning every operation (node in the CDFG) into control steps (c-steps). Resource allocation is the process of assigning operations to hardware with the goal of minimizing the amount of hardware required to implement the desired behavior. The hardware resources consist primarily of functional units, storage elements (registers/memory), and multiplexes. Once the operations in a CDFG have been scheduled into c-steps, an implementation consisting of an FSM and a data path can be derived. Depending on the delay of the operations (dependent on target technology), the clock frequency constraint, and performance or resource constraints, a variety of designs can be produced from the same specification. Parallelism between operations in the same basic block (data-flow graph) is analyzed and exploited according to what hardware resource is allocated. Parallelism across control boundaries is exploited using loop unrolling, loop pipelining, and by analyzing data dependencies across conditionals. The research studied ways to optimize the hardware by means of how functional resources are allocated, how operations are scheduled and mapped to the available resources, and how variables are mapped to registers or to memory.

The first commercial encarnalizations of HLS took an incremental approach to HLS and most HLS synthesis tools have, to this date, followed that trend. The goal was to improve productivity by partially raising the abstraction of RTL and applying HLS techniques to synthesize such specifications. The specification style is a mixture of functionality and I/O timing expressed in languages such as VHDL, Verilog, SystemC [17], Handel-C [18], or System Verilog. One of the main reasons for the desire of keeping I/O timing in the specification is to explicitly code interface timing into the specification. Interface exploration and synthesis are not built in as an intrinsic part of such methodologies. While the behavioral coding style appears more algorithmic (e.g., use of loops), the mixture of such behavior with I/O cycle timing specification provides an awkward way to specify cycle timing that often overconstrains the design.

5.3. Catapult C-based high-level synthesis methodology

Catapult C synthesis is the first HLS approach that raises the level of abstraction by clearly separating algorithmic function from the actual architecture to implement it in hardware (interface cycle timing, etc.). The inputs to the Catapult C are (a) the algorithmic specification expressed in sequential, ANSI-standard C/C++ and (b) a set of directives which define the hardware architecture. The clear separation of function and architecture allows the input source to remain independent of interface and performance requirements and independent of the ASIC/FPGA target technology. This separation provides important benefits.

```
#pragma design top
void fir (int 8 x, int 8 * y) {
  static int 8 taps [12];
  ⋮
}
```

Algorithm 1

(i) The source is concise, the easiest to write, maintain, and debug. Because of its high-level of abstraction, its behavior can be simulated at much higher speeds (×10 000 faster) than RTL, cycle accurate, or traditional behavioral-level specifications.

(ii) The source can be leveraged as algorithmic intellectual property (IP) that may be targeted for various applications and ASIC/FPGA technologies.

(iii) Obtaining a new architecture is a matter of changing architectural constraints during synthesis. This reduces the risk of prolonged manual recoding of the RTL to address last-minute changes in requirements or to address timing closure or to satisfy power and area budgets.

(iv) By avoiding manual coding of the architecture in the source, functional bugs that are common when coding RTL are also avoided. It is estimated that 60% of all bugs are introduced when writing RTL. The importance of avoiding such bugs cannot be overstated.

5.3.1. Algorithmic specification

The algorithmic specification is expressed in ANSI C/C++ where the function to be synthesized is specified either at the source (with a #pragma design top) or during synthesis. The interface of the function determines what data goes in and out of the function, though it does not specify how the data is transferred over time (that is determined during synthesis). For instance, the specification for an FIR filter may look as in Algorithm 1.

In this case, the FIR function is called with an input x and returns the output value y. Past values of x are stored in the local array *taps*. The array is declared *static* so that it preserves its value across invokations of the function. There are virtually no restrictions on the type of arguments: arrays, structs, classes are all supported. Currently, the only unsupported types (at any point in the source) are unions and wchar. The size of the array needs to be known at compile time, so it is important to specify its size when arrays are used at the interface: int $x[800]$ rather than just int $*x$.

It is important to use bit-accurate data types at the interface as the generated RTL will be dependent on the their bit widths. For instance in the case of the FIR filter, both x and y were specified to be 8-bit signed integers. Variables that are not at the interface may often be left unconstrained (using a type with more than the required numerical word length). Numerical analysis that is done during synthesis will minimize bit widths in a way that still preserves the bit-accurate

```
class Cplx {
    public: int r, i;
    Cplx (int r, int i) : r(r), i(i) {}
    Cplx operator + (const Cplx & other){
      return Cplx (r + other · r, i + other · i);};
    Cplx operator* (const Cplx & other){
      return Cplx (r* other · r − i* other · i,
                   r* other · i + i* other · r);};
    Cplx conj(){return Cplx (r, −i); };
    int pow(){ return r*r + i*i;}
}
```

Algorithm 2: Catapult C code for complex class definition.

behavior at the interface. Catapult C provides feedback to determine if a variable was optimized and it is possible to further numerically refine the algorithm to improve performance, area, and power. All numerical refinement should be done at the source level as the generated RTL by Catapult C should faithfully reflect the bit-accurate behavior of the source. Floating-point arithmetics at the precision of the C floating-point types are not practical for hardware implementation so they are not fully supported for synthesis. They can be mapped to fixed-point types, but that flow is mainly used for estimation as the bit accuracy of the source is in general not preserved in the generated RTL.

While it is possible to use the native C integers in conjunction with shifts and bit masking to model bit-accurate fixed-point and integer arithmetics, it makes the source less readable and makes coding of the algorithm more error prone. By using bit-accurate fixed-point and integer data types, it is possible to cleanly numerically refine an algorithm that originally used floating-point types. The data types are simply classes that encapsulate the data and provide the relevant operators to perform arithmetic, access bits or bit ranges, rounding and saturation, and so forth. Catapult C supports the integer and fixed-point data-type library provided by SystemC as well as the *Algorithmic C* data-type library. The *Algorithmic C* library provides arbitrary-length bit-accurate fixed-point and integer types that offer fast simulation speed with uniform and well-defined synthesis semantics. For either data-type libraries, the precision, quantization, and overflow modes are specified as template parameters. For example, the int 8 used in the FIR example is a typedef to the *Algorithmic C* type *ac_*int <8, *true*>, where the first parameter specifies its bit width and the second parameter specifies whether it is unsigned or signed. Moreover, it is possible to define new data types by using the class or structure to simplify the C level code. Such an example shown in Algorithm 2 will be used later in this paper to show the code style.

There are no restrictions on what functions are called with the only exception that recursive functions are not supported (though recursion based on template parameters is supported). Functions that are not marked as designs are inlined during synthesis. Memory allocation and deallocation are not supported. Pointers are supported provided that the object that they point to is statically determinable. This condition is not overly restrictive as function inlining helps to resolve pointers.

5.3.2. Architectural synthesis

The design is crafted during synthesis by interactively applying architectural constraints, analyzing the results, and further refining the architectural constraints if necessary. There are two constraints that the user must provide: the target technology and the clock frequency. If no other constraint is specified, default constraints are used to define the interface and required performance. A number of analysis tools provide feedback about the schedule, allocation, variable-to-register mappings, latency/throughput, area, timing reports, schematics with cross-probing links among them, and to the input source.

Architectural constraints define the interface, how internal arrays are mapped to storage (memory, registers, register file), and the level of interloop or intraloop paralellism that is exposed to synthesis. Interface synthesis provides a way to instruct synthesis to transfer data with a specific protocol. The generation of all the appropriate signals and their scheduling constraints are automatically captured by synthesis. For example, the interface could be a register/register file, memory, a FIFO, a bus, and so forth. The granularity of the data transfer is also defined during interface synthesis. For example in many cases, the even and odd array elements are transferred concurrently to increase the transfer bandwidth. Internal arrays may be mapped to any defined storage elements including memories or register files supporting different numbers of concurrent reads and writes.

Parallelism between operations across iterations of a loop (intraloop parallelism) can be exploited using either loop unrolling or loop pipelining. Loop unrolling (whether full or partial) unfolds one or more iterations of the loop by creating one or more copies of the body. For instance, unrolling a loop by 2 reduces the iteration count in half. If there is intraloop parallelism (and resource constrains do not interfere), the new loop body will have less than twice the latency of the original loop body. The net effect is that the latency of the loop is reduced. Loop pipelining on the other hand is a structural approach that overlaps execution of subsequent loop iterations. If there is intraloop parallelism (and resource constraints do not interfere), then the interval between iterations of a loop will be shorter than the latency of the loop body. The net effect is that the overall latency of the loop is reduced or that the throughput of the loop is increased. Interloop parallelism is mainly accomplished by loop merging. Loop merging cannot only improve the latency of the design by merging two or more loops in one, but often reduces storage requirements as data, that otherwise would need to be saved, is immediately consumed in the merged loop.

5.3.3. Integrated Catapult C verification methodology

Catapult C provides a flow to verify that the functionality of the generated RTL is consistent with the original C source.

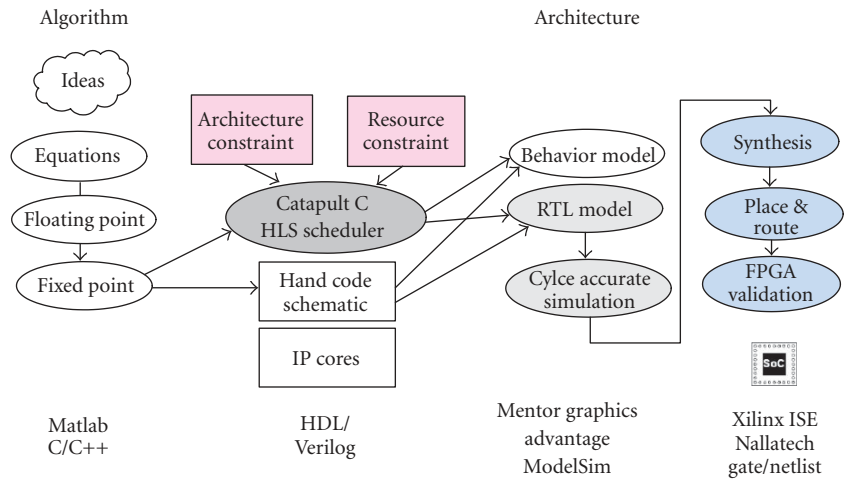

Figure 14: Catapult C-based high-level-synthesis design methodology.

This verification flow consists of building all the necessary SystemC wrappers and transactors to use the original C testbench to test the generated RTL. This is an important piece of the C-based methodology as manual generation of testbenches is time consuming, error prone, and architecture dependent. The verification flow also provides a convenient way to gather toggle information that is useful for power estimation.

6. APPLYING CATAPULT C METHODOLOGY FOR 3G/4G: DESIGN FLOW AND EXPERIMENTAL RESULTS

In this section, we describe how we apply Catapult C [21] in an integrated design flow for our high-complexity 3G/4G core algorithms design. We also show our experimental results and demonstrate how to achieve both architectural efficiency and productivity for modeling, partitioning, and synthesis of the complete system.

The first author at Nokia Research Center became a Beta user of Mentor's HLS technology in 2002 in order to develop a complete rapid prototyping methodology for advanced wireless communication systems. The computationally intensive nature of wireless algorithms made it a perfect match for HLS synthesis. Catapult C was officially released in the ACM Design Automation Conference (DAC) 2004 in San Diego, Calif, where the first author was one of the speakers in an expert panel.

To explore the VLSI design space, the system-level VLSI design is partitioned into several subsystem blocks (SBs) according to the functionality and timing relationship. The intermediate tasks will include high-level optimizations, scheduling and resource allocation, module binding, and control circuit generation. The proposed procedure for implementing an algorithm to the SoC hardware includes the following stages as shown in Figure 14 and is described as follows.

(1) Algorithm verification in Matlab and ANSI C/C++: in the algorithmic-level design, we first use Matlab to verify the floating-point algorithm based on communication theory. The matrix-level computations must be converted to plain C/C++ code. All internal Matlab functions such as FFT, SVD, eigenvalue calculation, complex operations, and so forth need to be translated with efficient arithmetic algorithms to C/C++.

(2) Catapult C HLS: RTL output can be generated from C/C++ level algorithm by following some C/C++ design styles. Many FUs can be reused in the computational cycles by studying the parallelism in the algorithm. We specify both timing and area constraints in the tool and let Catapult C schedule efficient architecture solutions according to the scheduling directives. The number of FUs is assigned according to the time/area constraints. Software resources such as registers and arrays are mapped to hardware components, and FSMs for accessing these resources are generated. In this way, we can study several architecture solutions efficiently and achieve the flexibility in architectural exploration.

(3) RTL integration and module binding: in the next step of the design flow, we use HDL designer to import the RTL output generated by Catapult C. A test bench is built in HDL designer corresponding to the C++ test bench and simulated using ModelSim. At this point, several IP cores might be integrated, such as the efficient cores from Xilinx CoreGen library (RAM/ROM blocks, CORDIC, FIFO, pipelined divider, etc.) and HDL ModuleWare components for the test bench.

(4) Gate-level hardware validation: Leonardo spectrum or precision-RTL is invoked for gate-level synthesis. Place and route tools such as Xilinx ISE are used to generate gate-level bit-stream file. For hardware verification and validation, a configurable Nallatech hardware platform is used. The hardware is tested and verified

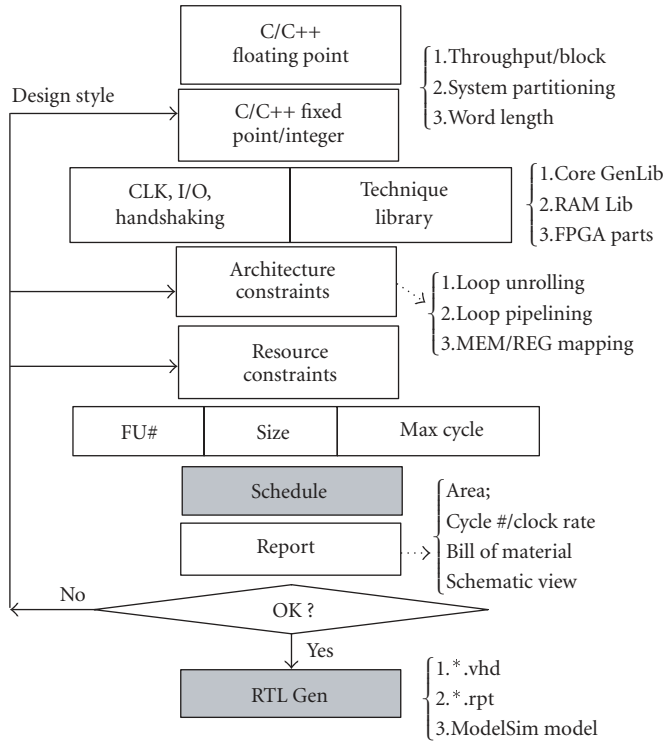

Figure 15: Procedure for Catapult C architecture scheduling.

by comparing the logic analyzer or ChipScope probes with the ModelSim simulation.

6.1. Architecture scheduling and resource allocation

In general, more parallel hardware FUs mean faster design at the cost of area, while resource sharing means smaller area by trading off execution speed. Even for the same algorithm, different applications may have different real-time requirements. For example, FFT needs to be very fast for OFDM modulation to achieve high data throughput, while it can be much slower for other applications such as in a spectrum analyzer. The best solution would be the smallest design meeting the real-time requirements, in terms of clock rate, throughput rate, latency, and so forth. The hardware architecture scheduling is to generate efficient architectures for different resource/timing requirements.

In Catapult C, first we specify the general requirements on the CLK rate, standard I/O, and handshaking signals such as RESET, START/READY, DONE signals for a system. The detailed procedure within Catapult C is shown in Figure 15. Then we can specify the building blocks in the design by choosing different technique libraries, for example, RAM library and CoreGen library. This will map the basic components to efficient library components such as divider or pipelined divider from the C/C++ language operator "/."

In a C-level design, the arrays are usually mapped to memory blocks. In some cycles, some FUs might be in IDLE state. These FUs could be reused by other similar computations that occur later in the algorithm. Thus, there will be many possible resource multiplexings in an algorithm. Multiplexing FUs manually is extremely difficult when the algorithm is complicated, especially when hundreds or even thousands of operations use the same FUs. Therefore, multiple FUs must be applied even for those independent computations in many cases. The size can be several times larger with the same throughput as in Catapult C solution. In Catapult C, we specify the maximum number of cycles in resource constraints. We can analyze the bill of material (BOM) used in the design and identify the large-size FUs. We can limit the number of these FUs and achieve a very efficient multiplexing. With the detailed reports on many statistics such as the cycle constraints and timing analysis, we can easily study the alternative high-level architectures for the algorithm and rapidly get the smallest design by meeting the timing as much as possible.

The programming style is essential to specify the hardware architectures in the C/C++ program. Several high-level conventions are defined to specify different architectures to be used. We use Catapult C to design architectures in two basic modes according to the behavior of the real-time system: the throughput mode for front-end processing and the block mode for postprocessing modules.

6.2. Throughput-mode front-end processing architectures

Throughput mode assumes that there is a top-level main loop. In each computation period, the data is inputted into the function sample by sample. The function will process for

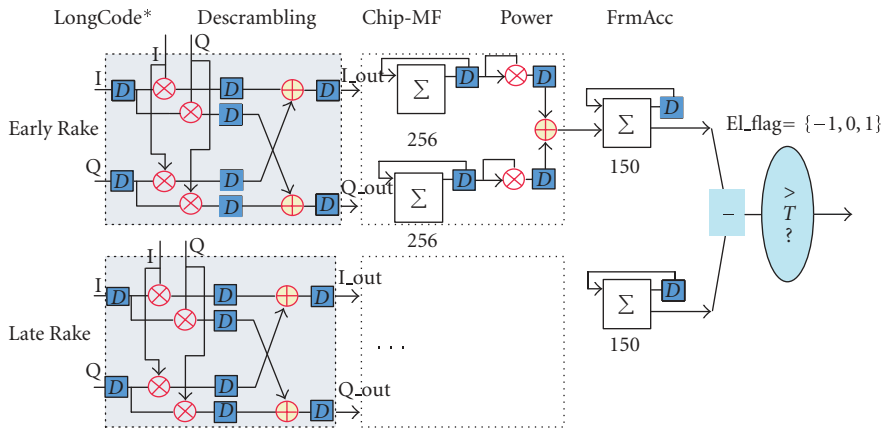

Figure 16: A typical manual layout architecture for clock tracking.

each sample input. Usually, no handshaking signals are required. The temporary values are kept by using static variables. The throughput is determined by the latency of the processing for each sample. Therefore, it is more suitable for the sample-based signal processing algorithms. Typical computations for this mode are filtering and accumulation-type computations in wireless systems.

6.2.1. Clock tracking

The clock-tracking algorithm could be designed with a conventional manual layout architecture in HDL designer. We would most likely build a parallel architecture with duplicate FUs as in Figure 16. First, we will have a descrambling procedure that is a complex multiplication with the long code. Then we will have a chip-matched filter that is basically mapped to an accumulator. Then after each symbol, we need to compute power and accumulate for each frame. We finally have a comparator to make a decision. Altogether, we will have copies for both early and late paths. This requires 16 multipliers and 12 adders. This architecture is optimal for fully pipelined computation where a sample will be inputted in each cycle. However, in our system, since we use a 38.4 MHz clock rate, only one sample will be inputted at the chip rate for each 10 cycles. The pipeline is idle for the other 9 cycles and the resources are wasted.

We now use Catapult C flow to design and schedule the architecture of this algorithm. The throughput-mode code is shown in Algorithm 3. We use the complex class to declare the interface variables "*eRake*," "*LRake*," which denote early- and late-phase Rake outputs, respectively. We also obtain the "*LongCode*" for each chip to do the descrambling. Note that we use *static* to declare the accumulation variables "*aEarly*" and "*aLate*" as well as the chip counter "*i*" and frame counter "*j*." All the I/O variables are not arrays. Thus, this module is called for each chip duration based on the streaming input chip samples. With Catapult C, we scheduled several solutions for the same source code by setting different constraint directives as in Table 1. In these designs, FUs are multiplexed within the timing constraints. Because of the computation dependency, there will be a necessary latency for the first computation result to come out even if we use many FUs. For example, in solution 1, although we use 8 multipliers and 6 adders, the best we can achieve is 7-cycle latency. The size is huge with 5600 FPGA lookup tables (LUTs). By setting the number of constraints and the maximal acceptable number of cycles (10 cycles), we will have different solutions with sizes ranging from 2000 to 1300 LUTs. We can choose the smallest design, that is, solution 4, for implementation while still meeting the timing constraint.

Figures 17 and 18 show the computation procedures of two typical solutions of clock tracking in Gantt graphs. The horizontal axis is the cycle for one period, and the vertical axis shows the mapped FUs for each computation. The long bars denote multiplications and short bars denote either "+" or "−" operations. The connections lines show the data dependency between operations. Figure 17 shows the fully parallel speed-constrained solution 1 with 8 multipliers. All 8 multipliers are used in parallel in cycle 1. Then 4 MULTs are used again in cycle 3. But in several other cycles, they are not used any more for the rest of the computation period. However, as shown in solution 4 in Figure 18, one single multiplier is reused in each cycle, by avoiding the dependency. After each multiplication, an addition follows and for the cycles 2–9, multiplications and additions are done in parallel. Moreover, we still meet the 10-cycle timing constraint easily. In solution 4, the hardware is used most efficiently. This is almost the minimal possible size that could be achieved theoretically for this particular algorithm. The savings in hardware can also reduce the power consumption that is a critical specification for mobile systems.

6.2.2. MIMO covariance estimation design space exploration

There are two major front-end modules for covariance estimation and channel estimation for the MIMO chip equalizer in Figure 8. These modules essentially are similar to the clock-tracking algorithm. While the clock-tracking algorithm computes the cross-correlation between the chip

```
#include "mc_bitvector.h"
#pragma design top
uint2 ctrk (Cplx eRake, Cplx LRake, Cplx LongCode, short threshold)
{
    static Cplx aEarly(0,0), Cplx aLate(0,0);
    static unsigned pAE = 0, pAL = 0; static uint9 i = 0, j = 0;
    int 2 flag = 0; Cplx tEarly, tLate;
    //descramble
    tEarly = eRake* LongCode. conj(); tLate = LRake* LongCode.conj();
    //accumulate through symbol
    aEarly = aEarly + tEarly; aLate = aLate + tLate;
    + + i;
    if (i = 256) {//accumulate through frame
        pAE+ = aEarly · Pow(); pAL+ = aLate · Pow();
        aEarly = Cplx(0, 0); aLate = Cplx(0, 0); i = 0; + + j;
        if (j = 150) { j = 0; //compare early/late gate with threshold
            if ((pAE − threshold) > pAL) flag = −1;
            else if ((pAL − threshold) > pAE) flag = 1;
            else flag = 0;
            pAE = 0; pAL = 0;
        }
    }
    return(flag);
}
```

Algorithm 3: Catapult C code for throughput-mode-based clock tracking.

Table 1: Catapult C-scheduled architectures for clock tracking.

Solution	LUTs	Cycle	MULT #	ADD #	MUX (LUT)
1	5628	7	8	6	1221
2	2004	10	2	2	1152
3	1426	16	1	1	623
4	1361	10	1	2	616

samples and the long scrambling codes, the covariance estimation computes the autocorrelation of the chip samples and the channel estimation computes the cross-correlation between the pilot symbols and the chip samples. Thus, these two modules are also suitable for the similar throughput mode Catapult C modeling, which will also generate similar architectures except that the number of functional units will be much higher than the clock tracking because the computation complexities of these two modules are much higher than the clock tracking. For example, both estimation modules need to compute the correlation with window length L, where L is corresponding to the channel length and could be up to 10 for an outdoor environment.

However, using the Catapult C HLS design methodology, we easily explored the design space for different specifications either by simply changing the synthesis directives in Catapult C or quickly making a change to the C-level code. This is demonstrated by the CLB consumption exploration for different architectures and different number of input bits in Figure 19. For designs with a different number of input bits, we only need to change the word length in C level for variables at the interface. Catapult C will automatically figure out the optimal word length for the internal variables. For the same C source, we can generate dramatically different RTLs with different latencies and resource utilizations by changing the architectural/resource constraints within the Catapult C environment. If we are not satisfied with some of the design specifications, we can easily change the source code to reflect a different partitioning for the purpose of scalability. For the MIMO scenario, we can scale the covariance estimation module for a different number of antennas. Thus, the same design is configurable to different numbers of antennas in the system. This scalability provides an approach for shutting down some idle modules so as to save the power consumption in the design, which is essential to mobile devices. An experiment shows that more than 60 dedicated ASIC multipliers are needed in a 4×4 MIMO system with $L = 10$. To achieve such an extensive study for such big designs and verify in the real-time environment in a short time is virtually impossible with the conventional design methodology. However, we demonstrated that we can explore the design specifications with much less effort using the Catapult C-based methodology.

6.3. Block-mode postprocessing architectures and integration

6.3.1. AFC

Corresponding to the AFC algorithm shown in Figure 7, we designed the hardware architecture as shown in Figure 20. IP cores from different sources are integrated in HDL designer.

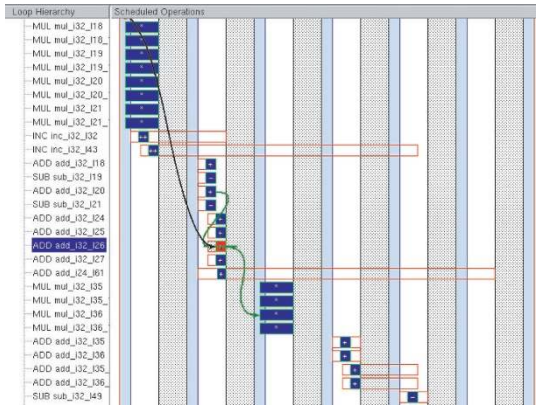

Figure 17: Gantt graph for speed-constrained architecture for clock tracking from Catapult C scheduling: 8 multipliers, 6 adders, 4 subtractors, 7-cycle latency.

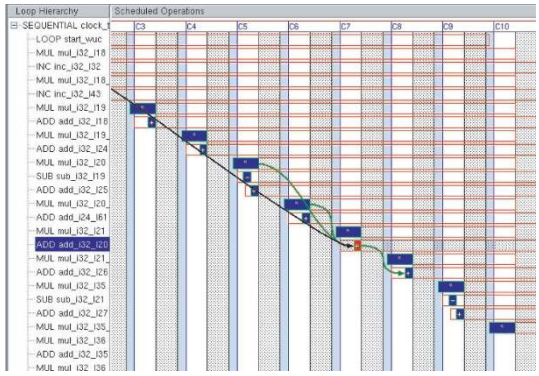

Figure 18: Gantt graph for area-constrained architecture for clock tracking: 1 adder, 1 subtractor, 10-cycle latency.

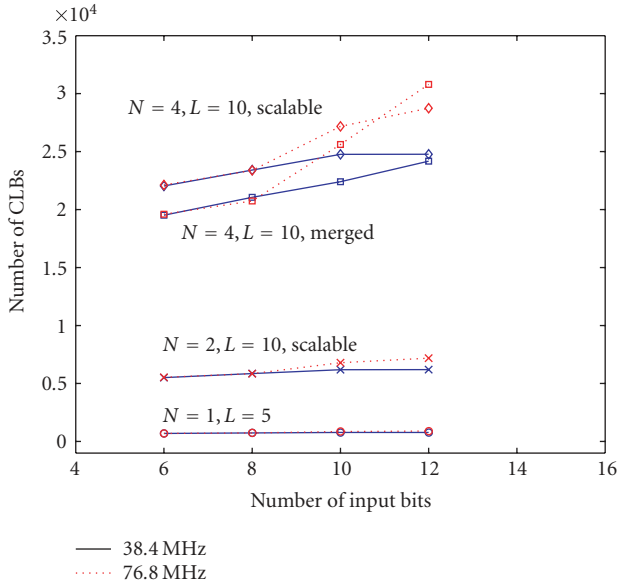

Figure 19: CLB versus number of input bits for the covariance estimation block (MIMO correlation block) with different architectures.

A Xilinx core direct digital synthesis (DDS) block controlled by the AFC module generates the local frequency to demodulate the RF front-end received signal. Some ROM cores are used to store the long codes and pilot symbols as well as the phase coefficients for the FFT. Three separate Catapult C blocks are pipelined: the AFC accumulation block, the 256-point FFT block, and a SearchMax block. The accumulator and descrambler need to process for each input sample and will work in a throughput mode. The search is invoked by the FFT once the FFT is finished. The processes will use dual-port RAMs for communication. All the IP cores are integrated in HDL designer with additional glue logic.

Although the Xilinx core library also provides a variety of FFT IP cores, they are usually for high-throughput applications, and they usually have considerably large sizes. But in our algorithm, the FFT only processes once for each complete frame block, so we can relax the timing constraint to get a very compact design with minimum resource usage. In block mode, the function processes once after a block of data is ready. Example Catapult C code style is shown in Algorithm 4 for a 32-point radix-2 FFT/IFFT module. The minimum resource code for 256-point FFT is essentially the same as that of a 32-point FFT except for the cosine/sine coefficients and the number of loops. First, we need to include the *mc_bitvector.h* to declare some Catapult C-specific bit vector types such as the int16, uint2, and so forth. We first convert the cosine/sine phase coefficients to integer numbers and store them in two vectors that will be mapped to ROM hardware as $\cos v$ and $\sin v$. If we consider the FFT module as the top level of the partitioned Catapult C module, we need to declare the *#pragma design top*. The input and output arrays ar0[], ai0[] could be mapped to dual-port RAM blocks in hardware. The flag is a signal to configure whether it is an FFT or IFFT module. It can be seen that the Catapult C style is almost the same as the ANSI-C style. There is no need to specify the timing information in the source code.

In the core algorithm, there are different levels of loop structures, that is, the stage level, the butterfly-unit level, and the implementation of the butterfly units. Based on the loop structure and the storage hardware mapping, we can specify the different architecture constraints within the Catapult C GUI interface to generate the desired RTL architecture. The hardware interface used RAM blocks to pass the data. Catapult C will generate FSMs for the write enabling, MEM address/data bus, and control logic. The complete AFC algorithm only needs to be updated once in each frame length, which is 10 milliseconds. We designed several solutions with only 1 multiplier and 1 adder reused for each MULT and ADD operation. The latency is larger than the Xilinx core, but the area is smaller. Finally, for all three blocks and different-point FFT, we achieve the same minimal size around 1000 LUTs, saving about ×3 in the number of LUTs over the Xilinx Core as shown in Table 2. Although in general a design with more functional units could work faster than a design with less functional units, it still depends on the design of parallelism and pipelining. This extreme example that using more functional units does not necessarily promise a fast design is also demonstrated in the

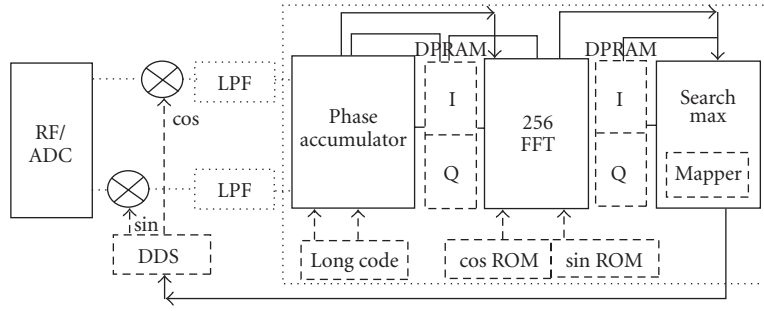

Figure 20: HDL designer integration of the Catapult C-based AFC: solid line: Catapult C module; dotted line: HDL designer module; dashed line: Xilinx core.

"256 Catapult C (2)" row in the table. The scheduling of an architecture with resource constraints in Catapult C is not always straightforward but needs some insightful architectural constraints. In the architecture with only one multiplier, the butterfly unit is utilized sequentially and only one multiplier is multiplexed for the butterfly unit.

For comparison purpose, we also show an example code of the high-throughput streaming style FFT in part two of Algorithm 4. In this eight-point FFT, we designed a template for one-stage butterfly unit. We also designed a FIFO for streaming data in one-stage butterfly unit. All the stages are laid out in a pipelined mode, by using output of stage 5 as the input of stage 4. Thus, very high throughput can be achieved which will be suitable for the OFDM modulation application.

6.3.2. Design space exploration of MIMO-FFT

The MIMO-FFT/IFFT design in the MIMO chip equalizer architecture shown in Figure 8 is another example of using Catapult C to search for efficient architecture with minimum-resource block mode design. For the multiple FFTs in the tap solver, the keys for optimization of the area/speed are loop unrolling, pipelining, and resource multiplexing. It is not easy to apply the commonality by using the Xilinx IP core for the MIMO-FFTs. To achieve the best area/time tradeoff in different situations, we apply Catapult C to design customized FFT/IFFT modules. We design the merged MIMO-FFT modules to utilize the commonality in control logic and phase coefficient loading. By using merged butterfly unit for MIMO-FFT, we utilize the commonality and achieve much more efficient resource utilization while still meeting the speed requirement. The Catapult C-scheduled RTLs for 32-point FFTs with 16 bits are compared with Xilinx v32FFT Core in Table 3 for a single FFT. Catapult C design demonstrates much smaller size for different solutions, for example, from solution 1 with 8 multipliers and 535 slices to solution 3 with only one multiplier and 551 slices. Overall, solution 3 represents the smallest design with slower but acceptable speed for a single FFT.

For the MIMO-FFT/IFFT modules, we can design a fully parallel and pipelined architecture with parallel butterfly units and complex multipliers laid out in a fully pipelined butterfly tree at one extreme; or we can just reuse one FFT module in serial computation at another extreme. In a parallel layout for an example of 4 FFTs, all the computations are localized and the latency is the same as one single FFT. However, the resource is $\times 4$ of a single FFT module. For a reused module, extra control logic needs to be designed for the multiplexing. The time is equal to or larger than that $\times 4$ of the single FFT computation. However, we can reuse the control logic inside the FFT module and schedule the number of FUs more efficiently in the merged mode. The specifications for 4 merged FFTs are listed in Table 4 with different numbers of multipliers. Compared to 4 parallel FFT blocks (each with 1 MULT) at 2204 slices and 810 cycles or 4 serial FFT at 3240 cycles, the resource utilization is much more efficient, where FU utilization is defined as $\#\ Multipliers/(\#\ Cycles\ *\ \#\ Multiplications)$.

The design space for different numbers of merged FFT modules is shown in Figure 21. Figure 21 shows the CLB consumption for different architectures versus the different number of multipliers. For the input and output arrays, two different types of memory mapping schemes are explored. One scheme applies split subblock memories for each input array labelled as SM. This option requires more memory I/O ports but increases the data bandwidth. Another option is a merged memory bank to reduce the data bus. However, the data access bandwidth is limited because of the merged memory block. This demonstrates the design space exploration capability enabled by the Catapult C methodology.

6.4. Scheduling control-dominated architectures

The QRD-M algorithm is a control-dominated architecture because it contains many conditional branches which are extremely difficult with the conventional manual design. The sorting procedure also leads to unpredictable latency depending on the input sequence. Catapult C can synthesize the complex FSM automatically for these types of complex logics. Moreover, it is easy to verify different pipelining tradeoffs. We studied three major different algorithms for the sorting function, each with many partitioning and storage mapping options. The partial sorting C design is described by a flow diagram as shown in Figure 22. To support the high modulation order and large M, the updated metrics are first stored

```
(1) Minimum resource code style for radix-2 FFT
#include <mc_bitvector.h>
#define NFFT 32
#define LogN 5
const int 16 cos v[LogN] = {−1024, 0, 724, 946, 1004};
const int 16 sin v[LogN] = {0, −1024, −724, −392, −200};
#pragma design top
void fft32 int 16 (int 16 ar0[NFFT], int 16 ai0[NFFT],
    const int 16 cos v[LogN], const int 16 sin v[LogN], uint1 flag)
{   short i, j, k, l, le, le1;
    int 16 rt0, it0, ru, iu, r, rw, iw; le = 1;
    for (l = 1; l ≤ LogN; l + +) { //stage level
      le1 = le; le = le*2; ru = 1024; iu = 0; rw = cos v[l − 1]; //rw = cos(PI/le1);
      if (flag = 0){ iw = sin v[l − 1]; } //forward fft, iw = − sin(PI/le1);
      else { iw = − sin v[l − 1]; } //backward ifft
      for (j = 0; j < le1; j + +) {
        for (i = j; i < NFFT; i+ = le){// BFU level
          k = i + le1;
          rt0 = (ar0[k]*ru − ai0[k]*iu) ≫ 10; it0 = (ai0[k]*ru + ar0[k]*iu) ≫ 10;
          ar0[k] = ar0[i] − rt0; ai0[k] = ai0[i] − it0;
          ar0[i]+ = rt0; ai0[i]+ = it0; }
        r = (ru*rw − iu*iw) ≫ 10; iu = (ru*iw + iu*rw) ≫ 10; ru = r;
        }
      }
}
(2) High throughput streaming code style for FFT
void Bfly_HT(Cplx a, Cplx b, Cplx *x, Cplx * y, Cplx W)
{ *x = a + b; * y = (a − b)* W; }
//One FFT stage: Template N represents the stage number
template< int N > struct stages
{
    unsigned int cnt; bool doit; uint10 Indx = 1 ≫ N;
    Cplx FIFOx[Indx], FIFOy[Indx], W;
    stages (){ cnt = 0; doit = false; };
    void newStage(cplx *x, cplx * y)
    { cplx xt = *x, yt;
     if (! doit) {FIFOx[cnt] = xt; yt = FIFOy[cnt];}
     else { W = Cplx(cos[Indx],sin[Indx]);
       Bfly_HT(FIFOx[cnt], xt, &yt, &FIFOy[cnt], W); }
     * y = yt; cnt + +; if (cnt ≧ (Indx)) {cnt = 0; doit = !doit;}
    };
};
#pragma design top
void fft_HT(cplx *x, cplx * y) {
    static stages< 5 > s5; static stages< 4 > s4;     static stages< 3 > s3;
    static stages< 2 > s2; static stages< 1 > s1; static stages< 0 > s0;
    Cplx t5, t4, t3, t2, t1;
    s5.newStage (x, &t5); s4.newStage (&t5, &t4); // Stage 5, 4;
    s3.newStage (&t4, &t3); s2.newStage (&t3, &t2); // Stage 3, 2;
    s1.newStage (&t2, &t1); s0.newStage (&t1, y); // Stage 1,0;
}
```

Algorithm 4: Catapult C code for FFT architectures: minimum resource versus high throughput.

in an input buffer memory block. The following inputs and parameters are defined: $S[N]$ is the input sequence stored in memory blocks; $I[N]$ is the index sequence initialized to be $I(i) = i$; M is the partial factor; $N = MC$ is the sequence length; *i*stack [NSTACK] is the stack to store the left and right pointers.

Once the metric update process is ready, it sends a "start" signal to the *PQSort* procedure. The partial quick sort has the same concept as the conventional quick sort based on "partition-exchange" method. First, a "partitioning element" a is selected from the subsequence. A pair of pointers il and ir are defined to set the boundary in terms of "left"

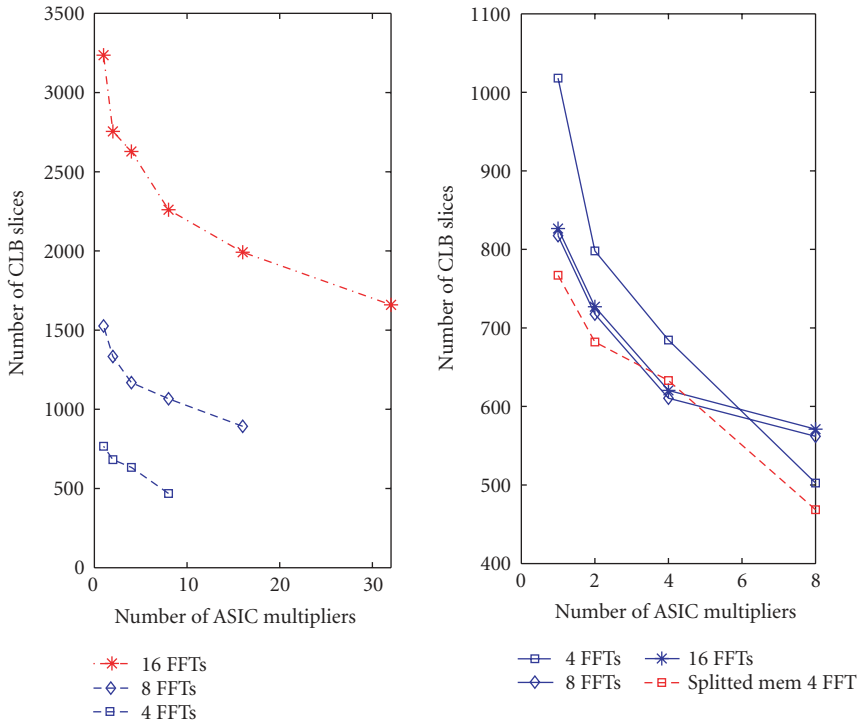

Figure 21: CLB versus number of multipliers for the different architectures of merged MIMO-FFT module: (a) splitted MEM vector processor MIMO-FFT; (b) MEM bank vector proceesor.

Table 2: Specifications comparison for different solutions of FFT.

Solution	BOM	Area (LUT)	Latency (cycles)
256 Core	12 mult	3286	768
1024 Core	12 mult	3858	4096
256 Catapult (1)	1 m + 1 a + 1 s	827	2076
256 Catapult (2)	4 m + 2 a + 2 s	1940	2387
1024 Catapult	1 m + 1 a + 1 s	1135	9381

Table 3: Architecture efficiency comparison.

Architecture	MULT	Latency (cycles)	Slices
Xilinx Core	12	128	2066
Catapult C Sol1	8	570	535
Catapult C Sol2	2	625	543
Catapult C Sol3	1	810	551

Table 4: Design space exploration for 4 merged 32-point FFTs.

MULT	Cycles	Slices	Util.	f_{clk} (MHz)
16	970	570	1/7	60
4	820	810	16/40	60
2	720	1135	16/28	60
1	680	1785	16/22	60

and "right" sides of the subsequence. A stack *istack* with the length of MSTACK is allocated to store the intermediate *il* and *ir* pointers of the pending subsequences. For the first stage, the subsequence is only the full sequence. So $il = 0$, $ir = N - 1$, and the top pointer of the stack $jstack = 0$. For a subsequence, when the length is shorter than some size LQS, it is faster to use the straight insert sort. To avoid the "worst-case" running time in the quick sort, which usually happens when the input sequence is already in order, the median of the first, middle, and last elements of the current subsequence is used as the "partitioning element." The partitioning process is carried out in the "do-while" loops for indices i and j. For the selected partitioning element a, we first scan the i pointer up until we find an element $> a$. Since we do not care about the order of the subsequences larger than M, we only need to push the "left" and "right" pointers lower than M to the stack. These pointers popped out from the stack only when the subsequence is short enough for the final "insert-sort" procedure. This not only reduces the size of the stack, but also reduces the latency in processing the subsequences higher than M.

For the QRD-M in the MIMO-OFDM system, the runtime comparison of the original and FPGA implementation for the 4×4 MIMO configuration and 64-QAM modulation is shown in Figure 23. We implemented 2 PEs in the V3000 FPGA in this case. For 64-QAM and $M = 64$, speedup of $\times 100$ is observed with 33 MHz FPGA clock rate competing

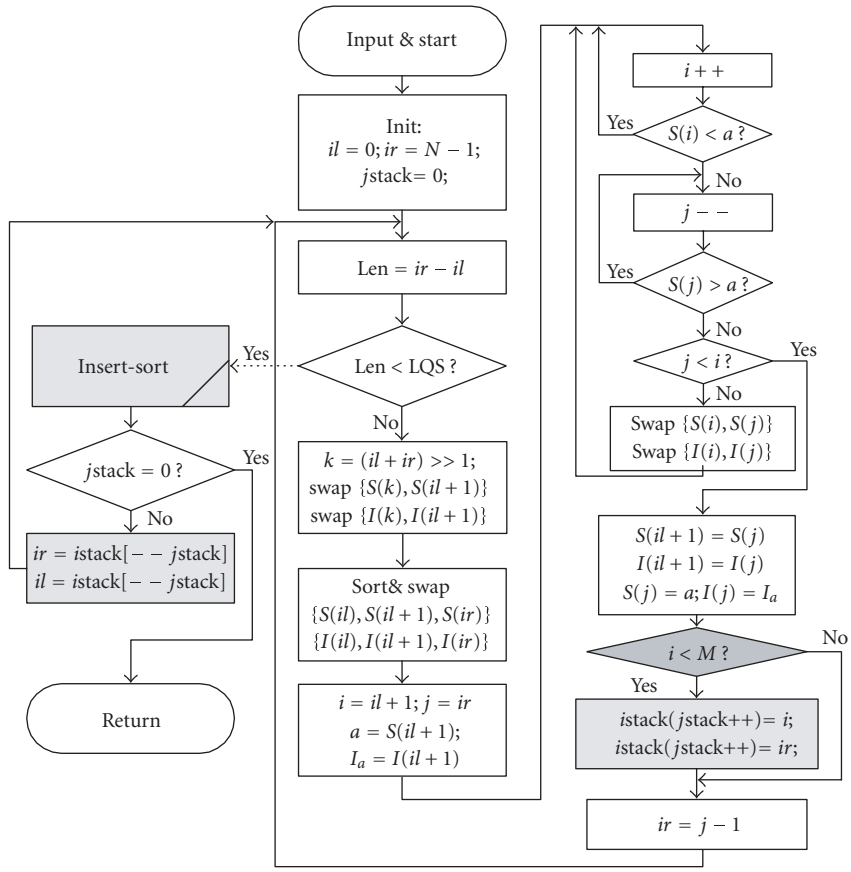

Figure 22: The logic flow of the partial quick-sort procedure.

with the 1.5 GHz Pentium 4 clock rate. Faster acceleration is achievable using more processing elements with the scalable VLSI architecture and clock rate from P and R result can be up to 90 MHz.

6.5. Strength and limitations

As we have shown, the Catapult C HLS methodology demonstrates significant advantages in rapid architecture scheduling and capability to allow extensive design space exploration for customized IP core designs of high-complexity signal processing algorithms. Table 5 compares the productivity of the conventional HDL-based manual design method and the Catapult C-based design space exploration methodology. For the manual design method, we assume that the algorithmic specification is ready and there is some reference design either in Matlab or C code as a baseline source code. The workloads are estimated based on the authors' extensive experience in using the conventional HDL design. Such estimate is commonly used for project planning by assuming average proficiency in the programming languages. For the Catapult C design, we assume that the fixed-point C/C++ code has been tested in a C test bench using test vectors. The work load does not include the integration stage either within HDL designer or writing some high-level wrapper in VHDL. For the Catapult C design flow, there are possibly many rounds of edit in the C source code to reflect different architecture specifications. It is shown that with the manual VHDL design, it may take much longer design cycle to generate one working architecture than the extensive tradeoff exploration using Catapult C. The improvement in productivity for our prototyping experience of 3G and beyond systems is obvious compared with the conventional HDL-based design methodology.

However, we also notice that there are still some aspects that the Catapult C methodology can improve. Firstly, in terms of the architecture scheduling capability, the efficiency for reusing multiplexers for a large design can be improved from our experience. Moreover, it is still difficult to predict the hardware architectures for control-dominated signal processing algorithms. Of course, this is also partly due to the fact that the control-dominated algorithms usually do not have structures which facilitate architecture pipelining. Secondly, Catapult C still has limitations in system-level simulation with fixed-point analysis. The fixed-point conversion still requires much manual work. Some Matlab-level synthesis tools such as the AccelChip might offer better integrated environment for extensive fixed-point analysis. Thirdly, there are many standard well-defined modules such as the FIR, FFT, besides the many advanced proprietary algorithms for

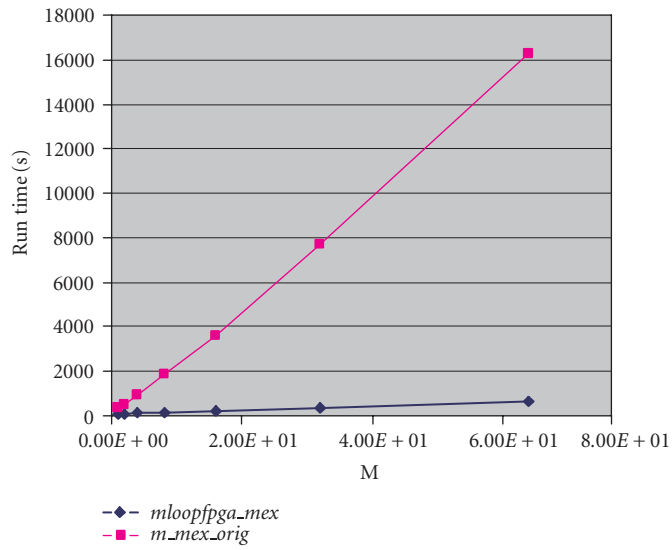

Figure 23: Measured simulation speedup (simu/emulation time) for the M-algorithm: 4×4, 64-QAM.

Table 5: Productivity improvement from the untimed C-based design space exploration.

Task	VHDL	Catapult C
Clock tracking	3 weeks	1 week
FFT	5 weeks	2 weeks
AFC	6 weeks	2 weeks
Turbo interleaver	2.5 months	3 weeks
Covariance estimation	3 weeks/sol	1-week tradeoff study
Channel estimation	3 weeks/sol	1-week tradeoff study
MIMO-FFT	5 weeks/sol	2-week tradeoff study
FIR filtering	3 weeks/sol	1-week tradeoff study

an prototyping project. It would be helpful if Catapult C could provide an extensive library for these standard DSP modules. System generator and AccelChip have stronger positions in this IP library feature. However, this IP integration feature should be considered independently with the HLS scheduling capability and the discussion is out of the scope of this paper.

7. CONCLUSION

In this paper, we present our industrial experiences for the 3G/4G wireless systems using a Catapult C HLS rapid prototyping methodology. We discuss core system design issues and propose reduced-complexity algorithms and architectures for the high-complexity receiver algorithms in 3G/4G wireless systems, namely MIMO-CDMA and MIMO-OFDM systems. We also demonstrate how Catapult C enables architecture scheduling and SoC design space exploration of these different classes of receiver algorithms. Different code design styles are demonstrated for different application scenarios. By efficiently studying FPGA architecture tradeoffs, extensive architectural research for high-complexity signal processing algorithms of 3G/4G wireless systems is enabled with significantly improved productivity.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Behnaam Aazhang and Gang Xu for their support in this work. J. R. Cavallaro was supported in part by NSF under Grants ANI-9979465, EIA-0224458, and EIA-0321266. Part of the paper was presented in IEEE RSP'03 and Asilomar'04 conferences.

REFERENCES

[1] A. Wiesel, L. García, J. Vidal, A. Pagès, and J. R. Fonollosa, "Turbo linear dispersion space time coding for MIMO HSDPA systems," in *Proceedings of 12th IST Summit on Mobile and Wireless Communications*, Aveiro, Portugal, June 2003.

[2] G. D. Golden, C. J. Foschini, R. A. Valenzuela, and P. W. Wolniansky, "Detection algorithm and initial laboratory results using V-BLAST space-time communication architecture," *Electronics Letters*, vol. 35, no. 1, pp. 14–16, 1999.

[3] G. J. Foschini, "Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas," *Bell Labs Technical Journal*, vol. 1, no. 2, pp. 41–59, 1996.

[4] Y. Guo, J. Zhang, D. McCain, and J. R. Cavallaro, "Efficient MIMO equalization for downlink multi-code CDMA: complexity optimization and comparative study," in *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM '04)*, vol. 4, pp. 2513–2519, Dallas, Tex, USA, November-December 2004.

[5] J. Yue, K. J. Kim, J. D. Gibson, and R. A. Iltis, "Channel estimation and data detection for MIMO-OFDM systems," in *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM '03)*, vol. 2, pp. 581–585, San Francisco, Calif, USA, December 2003.

[6] Y. Lee and V. K. Jain, "VLSI architecture for an advance DS/CDMA wireless communication receiver," in *Proceedings of the Annual IEEE International Conference on Innovative Systems in Silicon*, pp. 237–247, Austin, Tex, USA, October 1997.

[7] Z. Guo and P. Nilsson, "An ASIC implementation for V-BLAST detection in 0.35 μm CMOS," in *Proceedings of the 4th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT '04)*, pp. 95–98, Rome, Italy, December 2004.

[8] A. Adjoudani, E. C. Beck, A. P. Burg, et al., "Prototype experience for MIMO BLAST over third-generation wireless system," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 3, pp. 440–451, 2003.

[9] B. Razavi, *RF Microelectronics*, Prentice Hall Communications Engineering and Emerging Technologies Series, Prentice-Hall, Upper Saddle River, NJ, USA, 1998.

[10] K. Hooli, M. Juntti, M. J. Heikkilä, P. Komulainen, M. Latva-Aho, and J. Lilleberg, "Chip-level channel equalization in WCDMA downlink," *EURASIP Journal on Applied Signal Processing*, vol. 2002, no. 8, pp. 757–770, 2002.

[11] J. M. Rabaey, "Low-power silicon architecture for wireless communications: embedded tutorial," in *Proceedings of ASP-DAC 2000, Asia and South Pacific Design Automation Conference*, pp. 377–380, Yokohama, Japan, January 2000.

[12] A. Evans, A. Silburt, G. Vrckovnik, et al., "Functional verification of large ASICs," in *Proceedings of 35th ACM/IEEE Design Automation Conference (DAC '98)*, pp. 650–655, San Francisco, Calif, USA, June 1998.

[13] J. Bhasker, *A VHDL Primer*, Prentice-Hall, Upper Saddle River, NJ, USA, 3rd edition, 1999.

[14] R. Camposano and W. Wolf, *Trends in High-Level Synthesis*, Kluwer, Boston, Mass, USA, 1991.

[15] G. De Micheli and D. C. Ku, "HERCULES - a system for high-level synthesis," in *Proceedings of the 25th ACM/IEEE Conference on Design Automation (DAC '88)*, pp. 483–488, Anaheim, Calif, USA, June 1988.

[16] C.-Y. Wang and K. K. Parhi, "High-level DSP synthesis using concurrent transformations, scheduling, and allocation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 3, pp. 274–295, 1995.

[17] http://www.systemc.org/.

[18] http://www..doc.ic.ac.uk/~akf/handel-c/cgi-bin/forum.cgi.

[19] D. Knapp, T. Ly, D. MacMillen, and R. Miller, "Behavioral synthesis methodology for HDL-based specification and validation," in *Proceedings of 32nd ACM Design Automation Conference (DAC '95)*, pp. 286–291, San Francisco, Calif, USA, June 1995.

[20] J.-W. Weijers, V. Derudder, S. Janssens, F. Petré, and A. Bourdoux, "From MIMO-OFDM algorithms to a real-time wireless prototype: a systematic matlab-to-hardware design flow," *EURASIP Journal on Applied Signal Processing*, vol. 2006, Article ID 39297, 12 pages, 2006.

[21] *Catapult C Manual and C/C++ style guide*, Mentor Graphics, 2004.

[22] U. Knippin, "Early design evaluation in hardware and system prototyping for concurrent hardware/software validation in one environment," in *Proceedings of 13th IEEE International Workshop on Rapid System Prototyping (RSP '02)*, Aptix, Darmstadt, Germany, July 2002.

[23] Y. Guo and D. McCain, "Compact hardware accelerator for functional verification and rapid prototyping of 4G wireless communication systems," in *Proceedings of 38th IEEE Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 767–771, Pacific Grove, Calif, USA, November 2004.

[24] T. M. Aulin, "Breadth-first maximum likelihood sequence detection: basics," *IEEE Transactions on Communications*, vol. 47, no. 2, pp. 208–216, 1999.

[25] G. H. Golub and C. F. V. Loan, *Matrix Computations*, The Jones Hopkins University Press, Baltimore, Md, USA, 1996.

[26] M. Rupp, A. Burg, and E. Beck, "Rapid prototyping for wireless designs: the five-ones approach," *Signal Processing*, vol. 83, no. 7, pp. 1427–1444, 2003.

[27] H. Steendam and M. Moeneclaey, "The effect of clock frequency offsets on downlink MC-DS-CDMA," in *Proceedings of IEEE International Symposium on Spread Spectrum Techniques and Applications (ISSSTA '02)*, vol. 1, pp. 113–117, Prague, Czech Republic, September 2002.

Yuanbin Guo received the B.S. degree (E.E.) from Peking University, and the M.S. (E.E.) degree from Beijing University of Posts and Telecommunications, Beijing, China, in 1996 and 1999, respectively, and the Ph.D. degree from Rice University, Houston, Tex, in May 2005, in electrical and computer engineering. He was a winner of the Presidential Fellowship in Rice University in 2000. From 1999 to 2000, he was with Lucent Bell Laboratories, Beijing, where he conducted R&D in the Intelligent Network Department. He joined Nokia Research Center, Irving, Tex, in 2002. He is now a Senior Research Engineer and Research Specialist in the Signal Processing Architecture Group of Nokia Networks Strategy and Technology. His current research interests include equalization and detection for multiple-antenna systems, VLSI design and prototyping, and DSP and VLSI architectures for wireless systems, 3GPP long-term evolution (LTE), OFDM, WiMax. He is a Member of IEEE. He has 6 patents pending in wireless communications field.

Dennis McCain received his B.S. degree in electrical engineering from Lousiana State University in 1990 and his M.S. degree in electrical engineering from Texas A&M University in 1992. From 1992 to 1996, he served in the US Army as a Signal Officer responsible for deploying communication networks in tactical environments. From 1996 to 1998, he worked at Texas Instruments and Raytheon Systems as a Digital Design Engineer. In 1999, he joined Nokia Research Center in Dallas, Tex, where he developed prototype wireless communication systems. He is currently a Technology Manager in Nokia Networks Strategy and Technology leading a team responsible for implementing novel physical layer algorithms for next-generation cellular wireless systems. His interests are in the areas of hardware architecture research, digital baseband design, and rapid prototype design flows.

Joseph R. Cavallaro received the B.S. degree from the University of Pennsylvania, Philadelphia, Pa, in 1981, the M.S. degree from Princeton University, Princeton, NJ, in 1982, and the Ph.D. degree from Cornell University, Ithaca, NY, in 1988, all in electrical engineering. From 1981 to 1983, he was with AT&T Bell Laboratories, Holmdel, NJ. In 1988, he joined the faculty of Rice

University, Houston, Tex, where he is currently a Professor of electrical and computer engineering. His research interests include computer arithmetic, VLSI design and microlithography, and DSP and VLSI architectures for applications in wireless communications. During the 1996–1997 academic year, he served at the US National Science Foundation as Director of the Prototyping Tools and Methodology Program. During 2005, he was a Nokia Foundation Fellow and a Visiting Professor at the University of Oulu, Finland. He is currently the Associate Director of the Center for Multimedia Communication at Rice University. He is a Senior Member of the IEEE. He was the Cochair of the 2004 Signal Processing for Communications Symposium at the IEEE Global Communications Conference and general Cochair of the 2004 IEEE 15th International Conference on Application-Specific Systems, Architectures, and Processors (ASAP).

Andres Takach is a Chief Scientist, C-Based Design at Mentor Graphics. He joined Mentor Graphics in 1997, where he has worked on all aspects of high-level synthesis. His fields of interest are in high-level synthesis, synthesis for low-power, embedded system design, and hardware/software codesign. From 1993 to 1997, he was a faculty member at Illinois Institute of Technology, where he conducted research in high-level synthesis and hardware/software codesign. Andres Takach received his Ph.D. degree from Princeton University in 1993 and his B.S. and M.S. degrees in electrical and computer engineering from the University of Wisconsin-Madison in 1986 and 1988, respectively.