Rapid Industrial Prototyping and Scheduling of 3G/4G SoC Architectures with HLS Methodology

Yuanbin Guo, *Member, IEEE,* Dennis McCain, *Member, IEEE,* J. R. Cavallaro, *Senior Member, IEEE*

***Abstract*— In this paper, we present a Catapult C/C++ based methodology that integrates key technologies for high-level VLSI modelling of 3G/4G wireless systems to enable extensive time/area tradeoff study. A Catapult C/C++ based architecture scheduler transfers the major workload to the algorithmic C/C++ fixed-point design. Prototyping experiences are presented to explore the VLSI design space extensively for various types of computational intensive algorithms in the HSDPA, MIMO-CDMA and MIMO-OFDM systems, such as synchronization, MIMO equalizer and the QRD-M detector. Extensive time/area tradeoff study is enabled with different architecture and resource constraints in a short design cycle. The industrial design experience demonstrates significant improvement in architecture efficiency and productivity, which enables truly rapid prototyping for the 3G and beyond wireless systems.**

***Index Terms*— SoC, 3G/4G, MIMO, HLS, prototyping.**

I. Introduction

THE radical growth in wireless communication is pushing both advanced algorithms and hardware technologies for much higher data rates than current systems. Recently, UMTS and CDMA2000 extensions optimized for data services lead to the High Speed Downlink Packet Access (HSDPA)[1] and EV-DO/DV standards. On the other hand, MIMO (Multiple Input Multiple Output) technology [2] [3] using multiple antennas at both the transmitter and receiver sides is leading to MIMO-CDMA [4], MIMO-OFDM [5] etc., 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 work on the VLSI architectures for the CDMA [6][7] and VBLAST MIMO receiver [8] [9] have been reported.

Much more complicated signal processing algorithms are required for better performance, e.g., the linear MMSE equalizer [4] [10] for CDMA systems and the QRD-M detector [5] for MIMO-OFDM systems. This gives tremendous challenges for real-time hardware implementation, especially when the gap between algorithm complexity and the silicon capacity keeps increasing significantly for 3G and beyond wireless systems as shown in [11]. Although System-On-Chip (SoC) architectures offer more parallelism than DSP processors, the conventional gap between the algorithm researchers and the hardware teams is resulting in many algorithms that are not realistic for real-time implementation. 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 [13][14][15]. On the other hand, there are many area/time/power 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 and allows designers to identify the critical performance bottlenecks in meeting real-time requirement. However, this type of SoC design space exploration is extremely time consuming because of the current trial-and-optimize approaches using handcoded VHDL/Verilog or Graphical schematic design tools [12] [16]. To meet the fast changing market requirements, a design methodology that can study different architecture tradeoffs efficiently is highly desirable.

In [17], the author analyzed the design challenges from algorithm to architecture. A good development environment for wireless systems should be able to model various DSP algorithms and architectures at the right level of abstraction, i.e., 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 modelling of sample sequences, the grouping of the sample sequence into frames and the concurrent operation of multiple rates inherent in modern communication systems. 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 trade-offs, optimize system partitioning and adapt to new technologies as they become available.

Raising the language level to high-level-synthesis (HLS) [18] [19] can accomplish these requirements. However, raising the design-abstraction level is not enough. 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.

In this paper, we propose an un-timed C/C++ level design and verification methodology that integrates key technologies for truly high-level VLSI modelling to keep pace with the explosive complexity of SoC designs in the MIMO mobile devices. Part of the work was presented in the conference

Y. Guo and D. McCain are with Nokia Research Center, Irving, TX, 75039. J. R. Cavallaro is with Department of Electrical and Computer Engineering, Rice University, Houston, Tx, 77005. Part of the paper was presented in IEEE RSP'03 and Asilomar'04 conference.

of Rapid System Prototyping [20]. A Catapult-C based architecture scheduler is applied to explore the VLSI design space extensively for various types of computationally intensive algorithms in HSDPA, MIMO-CDMA and MIMO-OFDM systems. The major workload is transferred to the algorithmic C/C++ fixed-point design and high-level architecture scheduling. Extensive time/area tradeoff study is enabled with different architecture and resource constraints in a short design cycle. Synthesizable RTL is generated directly from a C/C++ level design and imported to the graphical tools for module binding.

In the case study, we will give our industrial experience in using the methodology to design some core algorithms in both CDMA and OFDM receivers. We first use two simple examples to demonstrate the concept of the methodology. Then we will present our experience with the SoC architecture scheduling for an FFT-based MIMO-CDMA equalizer that avoids the Direct-Matrix-Inverse [4] and the efficient architectures of a QRD-M matrix symbol detector in the MIMO-OFDM system. The key factors for optimization of the area/speed in loop unrolling, pipelining and the resource multiplexing are identified. Multi-level pipelining/parallelism are explored extensively to search for the most efficient VLSI architecture. The real-time architectures of the CDMA and HSDPA systems are implemented in a multiple FPGA-based Nallatech^{TM} [13] real-time demonstration platform, which was successfully demonstrated in the CTIA'03 trade show. The QRD-M MIMO detector for OFDM systems is implemented in a Wildcard^{TM} hardware accelerator with compact form factor [14], which achieves up to $100\times$ speedup in the simulation time.

This methodology frees us from the traditional time-consuming design and verification process for computationally intensive algorithms in wireless systems. Architecture efficiency and much improved productivity are achieved by reducing the design cycle by $50\% - 70\%$, enabling truly rapid prototyping for the computational extensive signal processing algorithms. This significantly shortens the technology transition time from algorithm to reality and reduces the risk in product development for 3G/4G wireless systems.

II. Real-time System-On-Chip (SoC) Technologies

A. Hardware Architectures for DSP and FPGA

As implementation is concerned, high-level software solutions, such as general-purpose processors (GPP), or software programmable DSP processors are preferable if applicable. However, although these two technologies provide higher flexibility and programmability, they are not powerful enough in speed for the physical layer of 3G/4G MIMO systems, especially for a compact mobile device. Communication chipset design has been the core technology in the wireless communication industry. System-on-Chip (SoC) architectures are a major revolution taking place in the design of integrated circuits due to the unprecedented levels of integration possible and many advantages in the power consumption and compact size. This leads to a demand for new methodologies and tools to address design, verification and test problems in

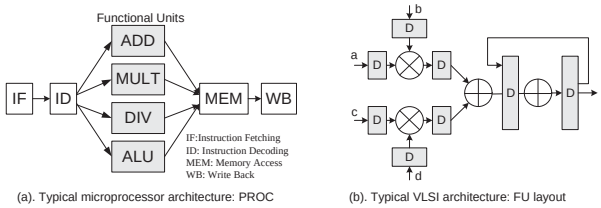

Fig. 1. Underlying architectures for DSP and VLSI: (a). DSP architecture based on ILP; (b). VLSI architecture based on FU layout.

this rapidly evolving area. "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 pre-designed behavioral or physical descriptions 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.

Although an ASIC is compact and less expensive when the product volume is large, it is not easy to configure to change in the design specifications at the prototyping stage. 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. It can achieve the concept of SoC with different hardware configurations. When the design is mature, the register-transfer-level (RTL) of FPGA design can act as reference design or be converted to ASIC for mass production.

In principle, these technologies reflect different hardware architectures as in Fig. 1. Sub-figure (a) is a processor architecture (PROC) based on instruction sets in GPP and DSP. It usually has some common functional units (FU) such as adders, multipliers etc, that are reused for each instruction. There are several steps for the execution of the instruction: Instruction Fetching (IF), Decoding (ID), Execution (EXE), Memory access (MEM) and Write back (WB) to registers. This forms an Instruction-Level-Pipelining (ILP) [21]. Fig. 1 (b) is a typical VLSI layout architecture in FPGA or ASIC. The layout architecture usually has fewer and simpler control circuits and more FUs than a PROC architecture. In the FU layout architecture, we can map many FUs in parallel to achieve high pipeline performance. Although the instruction scheduler and multiple FUs are used in some advanced processor architectures, the processor architecture still achieves only instruction-level pipelining while the layout architecture achieves FU-based pipelining through explicit design layout which significantly improves real-time performance.

The SoC realization of a complicated end-to-end communication system, such as MIMO-CDMA, highly depends on the task partitioning based on the real-time requirement and system 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

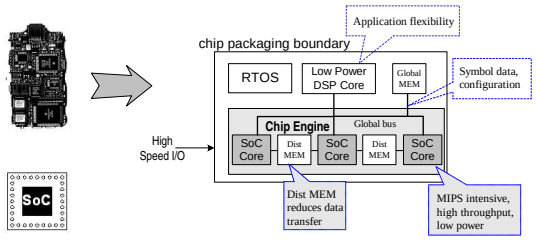

Fig. 2. SoC partitioning for computational efficiency, configurability, MOPS/μW and flexibility/scalability.

and Turbo decoders have been implemented as ASIC co-processors. 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 co-processor for the wireless link. Unlike a processor type of architecture, high efficiency and performance will be the major target specifications of the SoC design. The architecture partitioning strategy is shown in Fig. 2. The architectures should be efficiently parallelized and/or pipelined and functionally synthesizable in hardware.

B. Classical 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 [16], based on dataflow, structural or behavioral models. Graphical schematic design tools such as Hardware Design System (HDS) from Cadence or HDL Designer from Mentor Graphics are more intuitive. However, the design process is still manual and the intrinsic architecture tradeoffs need to be studied off-line. It is not easy to change a design dramatically once the hardware architecture is laid out.

The High-Level-Synthesis (HLS) methodology [18] provides a bridge by offering rapid system prototyping to the SoC design. The success of a HLS design tool highly depends on both the efficiency in the synthesized architectures and the improved productivity from the convenience in using the tool. Some C/C++ level RTL tools such as System-C [22] and Handel-C [23] attempt to combine a high-level of abstraction with the ability to generate synthesizable RTL. However, these design flows requires detailed knowledge of hardware implementation such as clocking, control logic, resource allocation etc. Moreover, detailed knowledge of hardware components is still required because all of the hardware components need to be synthesizable for a hardware implementation. They are still not intuitive to system engineers to understand and the detailed hardware specification in the language requires the designer to manually decide the architectural parallelism and pipelining. Manual optimization makes the tradeoff study in terms of time and area of the design difficult to evaluate, especially when the re-timing is critical for high-speed designs.

III. Integrated Catapult-C SoC Methodology

Scheduling and allocation are among the most important and difficult tasks in HLS of DSP systems. Scheduling involves assigning every node of the Data-Flow-Graph (DFG) to control

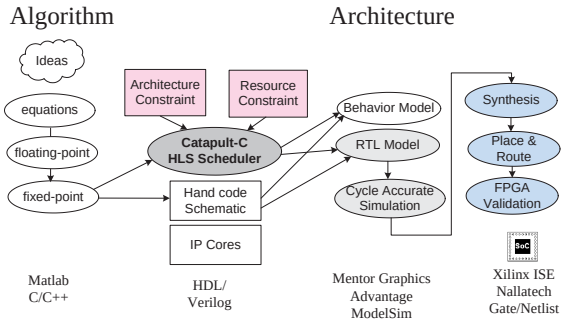

Fig. 3. Catapult-C based High-Level-Synthesis design methodology.

time steps. Control time steps are the fundamental sequencing units in synchronous systems and correspond to clock cycles. 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, memory elements, multiplexes, and communication data paths. In this section, we present the integrated Catapult-C [24] design flow which achieves both architectural efficiency and productivity for modelling, partitioning and synthesis of the complete system.

A. Catapult-C based High-Level Synthesis Methodology

Catapult-C synthesizer is a new RTL design tool optimized for hardware design from Mentor Graphics. It is an un-timed C/C++ level architecture scheduler without requiring timing specification in the C source code. We were one of the first Beta users of the tool and one of the first in the industry to integrate Catapult-C in a complete rapid prototyping methodology for advanced wireless communication systems. In the beta stage in 2002, Catapult-C was called Tsunami HLS designer. It was then renamed as Precision-C in the 2003 production release. The current name was officially released in the ACM Design-Automation-Conference (DAC) 2004 in San Diego CA, where the first author was one of the speakers in an expert panel.

The support for more abstract modelling provides predictive analysis and verification. Synergistic integration of all these technologies in a unified platform to create higher automation will treat the traditional Register-Transfer-Level (RTL) as an assembler language for system-level languages. To explore the VLSI design space, the system level VLSI design is partitioned into several subsystem blocks (SB) 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 Fig. 3 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 etc, 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 can specify both timing and area constraints in the tool and Catapult-C will schedule efficient architecture solutions according to the specified constraints in Catapult-C. 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 Finite State Machines (FSM) for accessing these resources are generated. In this way, we can study several architecture solutions efficiently and achieve the flexibility and productivity of a DSP with the performance of an FPGA.
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 intellectual property (IP) cores might be integrated, such as the efficient cores from Xilinx CoreGen library (RAM/ROM blocks, CORDIC, FIFO, pipelined divider etc) and HDL Module ware components for the test bench.
4) Gate level hardware validation: Leonardo Spectrum or Precision-RTL is invoked for gate-level synthesis. Place & 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 by comparing the logic analyzer or ChipScope^{TM} probes with the ModelSim simulation.

B. Architecture Scheduling and Resource Allocation

In general, more parallel hardware FUs means 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 in OFDM based systems for high data throughput rate, 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 etc. The hardware architecture scheduling is to generate efficient architectures for different resource/timing requirements.

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. For example, the use of array will be mapped to memory while the use of variables is mapped to a register file. Unlike SystemC [22], Catapult-C does not require very detailed knowledge of the hardware components. Here we only highlight some important features of the Catapult-C architecture. Catapult-C will schedule architectures in two basic modes according to the behavior of the real-time system: the throughput mode or the block computation mode.

- **Throughput mode**: It assumes that there is a top-level main loop. In each computation period, the data is input into the function sample by sample. The function will process for 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.
- **Block mode**: In block mode, the function processes once after a block of data is ready. The input data are either arrays or vectors in C code. The hardware interface will use RAM blocks to pass the data. Catapult-C will generate FSMs for the write enable, MEM address/data bus and control logic. Typical block computations are FFT, turbo decoder etc. Usually the throughput mode will be used for the front-end pre-processing blocks because of high-speed real-time requirement while the block mode is used for lower speed post-processing modules.

C. Layered Pipelining and Parallelism

In Catapult-C, first we can 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 Fig. 4. Then we can specify the building blocks in the design by choosing different technique libraries, e.g. 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 "/".

We will schedule architectures in the two basic modes according to the behavior of the real-time system. The keys for optimization of the area/speed are loop unrolling, pipelining and resource multiplexing. Loop unrolling is a procedure to repeat the loop body by trading higher speed for increased area. By unrolling, we may have multiple copies of FUs. But these FUs can be used in parallel if there is no dependency between the computations. Pipelining is basically a computational assembly line where multiple operations are overlapped in execution [21]. The use of memory can affect the performance dramatically.

In a C level design, the arrays are usually mapped to memory blocks. We can also map the internal or external RAM/ROM blocks used in the algorithm. 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 multiplexing 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 max number of cycles in resource constraints. We can analyze the Bill-Of-Material (BOM) used

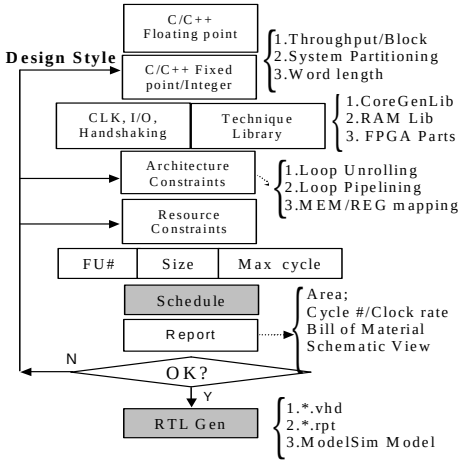

Fig. 4. Procedure for Catapult-C architecture scheduling.

in the design and identify the large size FUs. We can limit the number of these FUs and achieve a very efficient multiplexing. In the scheduling result, we can study the computational dependency. Usually the logic and the multiplexing in the design will determine the clock rate and the cycle number. 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.

In the following, we give case study for several major different types of computational intensive algorithms in standard HSDPA, MIMO-CDMA and MIMO-OFDM systems.

IV. Case Study I: CDMA Receiver Synchronization

HSDPA is the evolutionary mode of WCDMA [1]. High data rates up to 10 Mbps for the cellular downlink mobile system can be achieved to support wireless multi-media services in the future. The system diagram for the HSDPA prototype system is depicted in Fig. 5. In the transmitter, the host computer running the network layer protocols and applications interfaces with the DSP, which hosts the 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 CRC code. After the turbo encoder, rate matching and interleaver, a QPSK/QAM mapper modulates the data according to the HARQ control signals. With the CPICH and SCH information inserted, it is spread and scrambled with PN long code and then ported to the RF transmitter. At the receiver, the searcher will find the synchronization point. Clock tracking and AFC are applied for fine synchronization. After the matched filter receiver, received symbols are demodulated and de-interleaved before the rate de-matching. Then after a HARQ buffer, a turbo decoder decodes the soft decisions to a bit stream, which is sent to upper layer applications. In the figure, we also depict other key advanced algorithms including channel estimation, chip-level equalizer and multi-stage interference cancellation to eliminate the distortions caused by the wireless multi-path 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

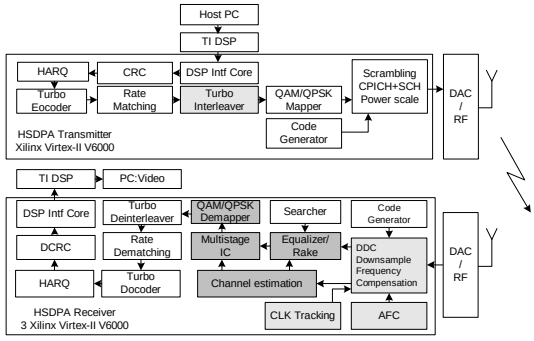

Fig. 5. System blocks for the HSDPA demonstrator.

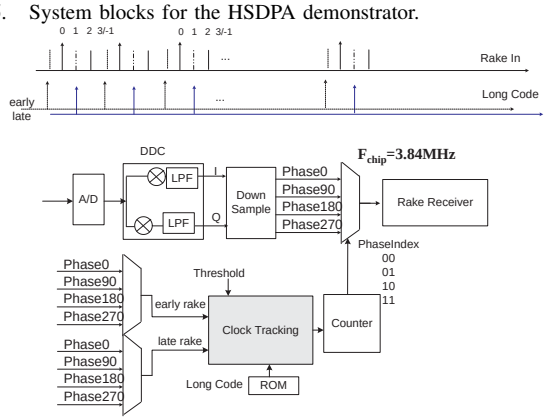

Fig. 6. The principle of clock tracking in CDMA systems.

design methodology. The darkly shaded blocks in the MIMO scenario will be the focus for case study in next section.

A. Clock Tracking

Algorithm: The mismatch of the transmitter and receiver crystal will cause a phase shift between the received signal and the long scrambling code. The "Clock-Tracking" algorithm [25] 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 down-sampled to be 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 de-scrambling 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 Fig. 6.

Architectures: The system interface for Clock Tracking is also depicted in Fig. 6. At the down sampling after the DDC (Digital Down-Converter Xilinx core), the in-phase, early, late phase 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 code tracker

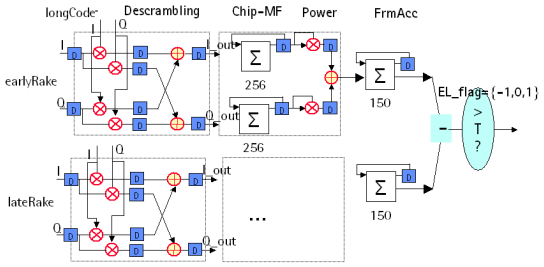

Fig. 7. A typical manual layout architecture for clock tracking.

TABLE I

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**

is integrated with IP cores and the other HDL Designer blocks (down-sampling, MUX etc).

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 Fig. 7 for rapid prototyping. 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/late paths. This requires 16 multipliers and 12 adders. This architecture is optimal for fully pipelined computation where a sample will be input in each cycle. However, in our system, since we use a 38.4 MHz clock rate, only one sample will be input at the chip-rate for each 10 cycles. The pipeline is idle for the other 9 cycles and the resources are wasted.

This computationally intensive algorithm is also suitable for Catapult-C scheduling. The C level function will get both early and late phase as input. With Catapult-C, we can schedule several solutions by setting different constraints as in Table I. In these designs, the 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 cycles latency. The size is huge with 5600 FPGA Look-Up-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, i.e., solution 4, for implementation while still meeting the timing constraint.

Fig. 8 and 9 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. Fig. 8 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

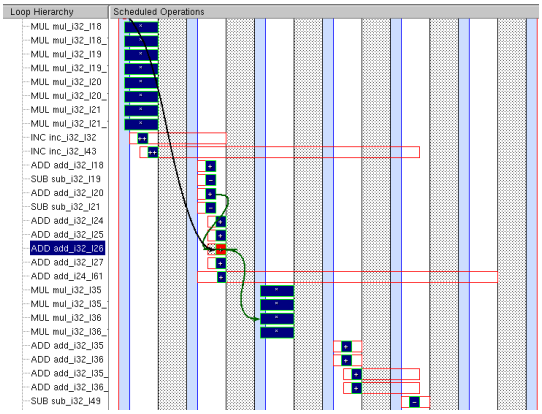

Fig. 8. Gantt graph for speed constrained architecture for clock tracking from Catapult-C scheduling: 8 multipliers, 6 adders, 4 subtractor, 7 cycles latency.

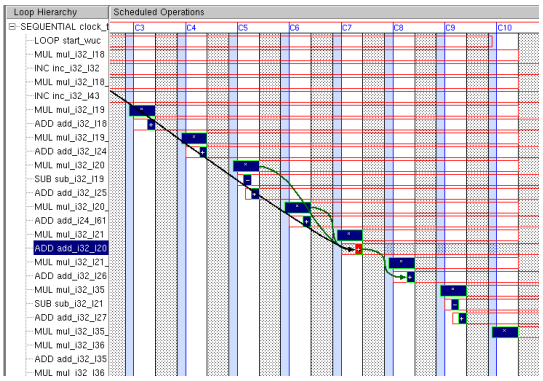

Fig. 9. Gantt graph for area constrained architecture for clock tracking: 1 adder, 1 subtractor, 10 cycles latency.

are not used any more for the rest of the computation period. However, as shown in solution 4 in Fig. 9, 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 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.

B. Automatic Frequency Control

The frequency offset is caused by the Doppler shift and frequency offset between the transmitter and receiver oscillators. This make the received constellations rotate in addition to the fixed channel phases, thus dramatically degrading performance. Automatic Frequency Control (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 Fig. 10. 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

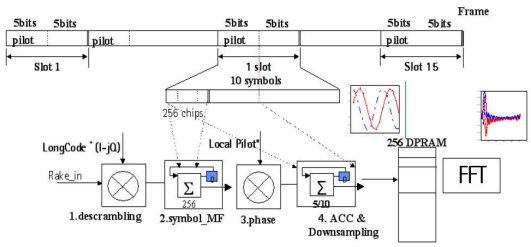

Fig. 10. Principle of the spectrum analysis based AFC.

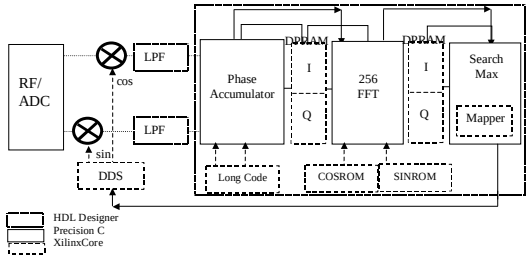

Fig. 11. HDL Designer integration of the Catapult-C-based AFC.

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 de-scrambling and matched-filter as well as accumulation, we almost achieve a very stable sinusoid waveform for the frequency offset signal as shown in the figure.

The C source code has a very similar style to the standard ANSI C syntax. The following shows a short example Catapult C code for the well-known radix-2 FFT/IFFT module. There is only minimal effort to modify this ANSI-C code for Catapult C synthesis. The modifications are shown in bold ***Italics***. First, we need to include the *mc_bitvector.h* to declare some Catapult C specific bit vector types such as the int16, uint2 etc. 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 *cosv* and *sinv*. 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. In the core algorithm, there are different levels of loop structure, the stage level, the butterfly-unit level and the actual implementation of the butterfly units. 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. 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.

#include <mc_bitvector.h>

TABLE II
SPECIFICATIONS COMPARISON FOR DIFFERENT SOLUTIONS OF FFT

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

```
#define NFFT 32
#define LogN 5
const int16 cosv[LogN]={-1024,0,724,946,1004};
const int16 sinv[LogN]={0,-1024,-724,-392,-200};
#pragma design top
void fft32int16(int16 ar0[NFFT],int16 ai0[NFFT],const int16
cosv[LogN],const int16 sinv[LogN], uint1 flag)
{  short i,j,k,l,le,le1;
   int16 rtmp0,itmp0,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=cosv[l-1]; //rw=cos(PI/le1);
     if(flag==0)       { //forward fft
       iw=sinv[l-1];       } //iw=-sin(PI/le1);
     else       { //backward ifft
       iw=-sinv[l-1];       }
     for(j=0;j <le1;j++)    {
      for(i=j;i<NFFT;i+=le)   {// BFU level
       k=i+le1;
       rtmp0=(ar0[k]*ru-ai0[k]*iu)>>10;
       itmp0=(ai0[k]*ru+ar0[k]*iu)>>10;
       ar0[k]=ar0[i]-rtmp0; ai0[k]=ai0[i]-itmp0;
       ar0[i]+=rtmp0; ai0[i]+=itmp0;  }
     r=(ru*rw-iu*iw)>>10;  iu=(ru*iw+iu*rw)>>10;
     ru=r;   }
   }
}
```

In this design, we have several tradeoffs to study. The phase accumulator has a similar architecture as the Clock-Tracking algorithm. We will focus on the architecture tradeoff for the FFT. 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, we do not need the FFT to be very fast, so we can relax the timing constraint to get a very compact design. The complete AFC algorithm only needs to be updated once in each frame length, which is $10ms$. With Catapult-C scheduling, we can have 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 $3\times$ in the number of LUTs over the Xilinx Core as shown in Table. II.

Fig. 11 shows the integration in HDL Designer. 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

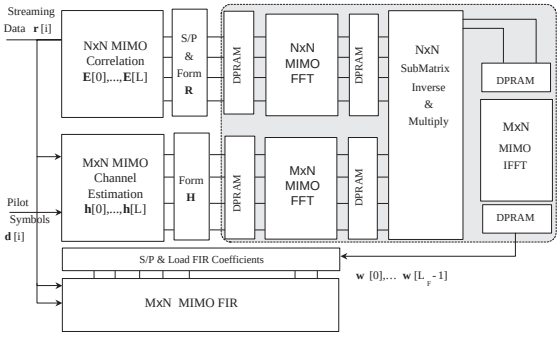

Fig. 12. VLSI architecture blocks of the FFT-based MIMO equalizer.

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 needs to process for each input sample and will work in a throughput mode. The FFT only processes once for each complete frame block. The Search is invoked by the FFT once the FFT is finished. So these two blocks will work in block mode. The processes will use dual-port RAMs for communication. All the IP cores are integrated in HDL Designer with additional glue logic.

V. Case Study II: MIMO-CDMA Downlink Receiver

A. VLSI System Architecture for FFT-based Equalizer

Linear-Minimum-Mean-Square-Error (LMMSE)-based chip equalizer is promising to suppress both the Inter-Symbol-Interference (ISI) and Multiple-Access-Interference (MAI) [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 this section, we apply the Catapult-C methodology to explore the design space of an FFT-based equalizer, whose detail is given in [4].

In our previous paper [4] [28], the direct matrix inverse in the chip equalizer is avoided 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 design blocks for the MIMO equalizer are shown in Fig. 12. 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 $(M \times N)$ dimension-wise FFTs on the channel coefficient vectors. A sub-matrix inverse and multiplication block takes the FFT coefficients of both channels and correlations from DPRAMs and carries out the computation. Finally a $(M \times N)$ dimension-wise IFFT module generates the results for the equalizer taps $\hat{\mathbf{w}}_m^{opt}$ 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 post-processing of the tap solver. They are suitable to work in a block mode using dual-point RAM blocks to communicate the data.

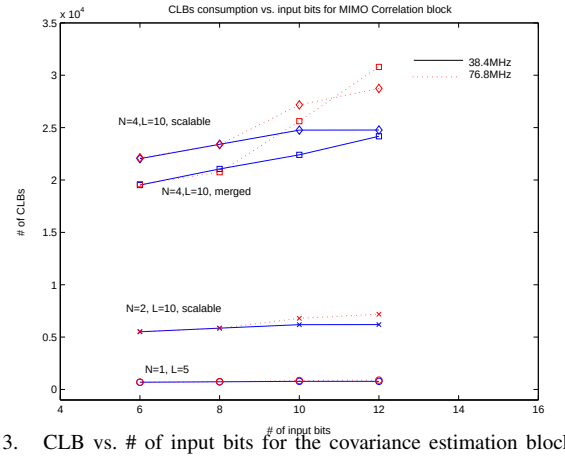

Fig. 13. CLB vs. # of input bits for the covariance estimation block with different architectures.

B. Scalable Architecture for MIMO Covariance Estimation

For designs with a different numbers of input bits, we only need to change the word length in C level for only a few variables at the interface. Catapult C can figure out the optimal word length for the internal variables. For the same C source, we can generate dramatically different RTL with different latency and resource utilization by changing the architectural/resource constraints within the Catapult C environment. If we are not satisfied with some of the design specification, 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 different number of antennas. Thus, the same design is configurable to different number 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.

Fig. 13 shows the CLB consumption with different number of input bits for the covariance estimation module with different architectures. Fig. 14 summarizes the usage of the dedicated ASIC multipliers versus the number of input bits for different architectures. The details of the architectures are not explained in this paper. To achieve such an extensive study and verify in the real-time environment in a short time is virtually impossible with the conventional design methodology. However, this demonstrates that we can explore the design specifications with much less cost with the Catapult C based methodology.

C. Design Space Exploration of MIMO-FFT Modules

For the multiple FFTs in the tap solver, the keys for optimization of the area/speed are loop unrolling, pipelining and resource multiplexing. Although Xilinx provides IP cores for FFTs, it is not easy to apply the commonality by using the IP core for the MIMO FFTs. To achieve the best area/time tradeoff in different situations, we apply Catapult-C to schedule customized FFT/IFFT modules. We design the

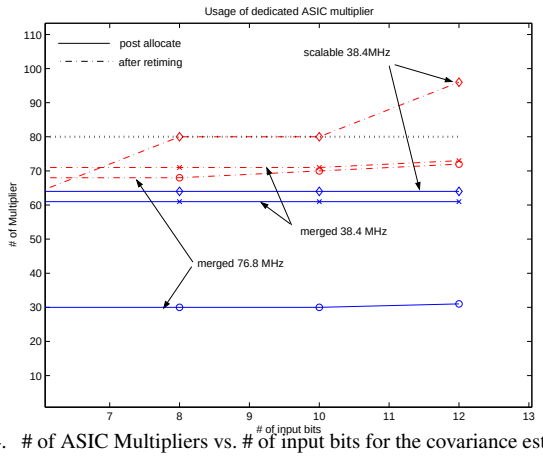

Fig. 14. # of ASIC Multipliers vs. # of input bits for the covariance estimation block with different architectures.

TABLE III

ARCHITECTURE EFFICIENCY COMPARISON.

Architecture	mult	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 IV

DESIGN SPACE EXPLORATION FOR 4 MERGED 32-POINT FFT.

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

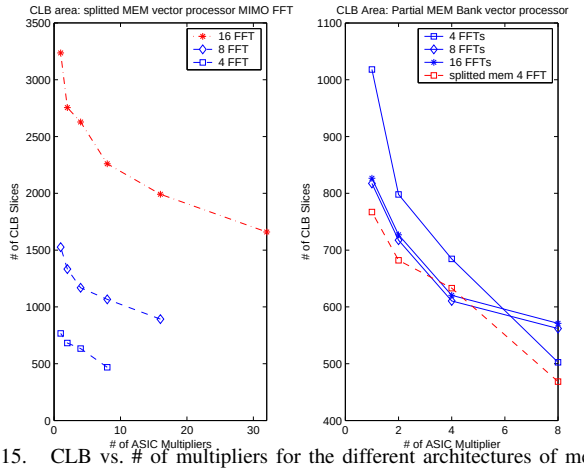

Fig. 15. CLB vs. # of multipliers for the different architectures of merged MIMO-FFT module.

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. III for a single FFT. Catapult-C design demonstrates much smaller size for different solutions, e.g. 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. 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 $4\times$ 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 $4\times$ 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. IV 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 Fig. 15 and Fig. 16. Fig. 15 shows the CLB consumption for different architectures versus the different number of multipliers. Fig. 16 shows the latency versus the number of multipliers for the merged MIMO-FFT modules. For the input and output arrays, two different type of memory mapping schemes are explored. One scheme applies split sub-block 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. The details of the implementation are omitted here. However, this demonstrates the design space exploration capability enabled by the Catapult C methodology. We also designed the merged submatrix inverse and multiplication module also in block mode hardware mapping following the described VLSI architectures. The details of the VLSI architecture can be found in [28].

VI. CASE STUDY III: 4G MIMO-OFDM SYSTEMS

A. 4G MIMO-OFDM Architecture Using QRD-M Detector

MIMO-OFDM converts the multipath frequency-selective fading channel into flat fading channel and simplify the channel estimation by inserting cyclic prefix to eliminate the Inter-Symbol Interference (ISI). The complexity of the optimal maximum likelihood detector 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. In this section, we explore the hardware architecture of the algorithm.

The MIMO-OFDM system model with N_T transmit and N_R receive antennas is shown in Fig. 17. At the p^{th} 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, a N_F-point FFT is operated on the received signal

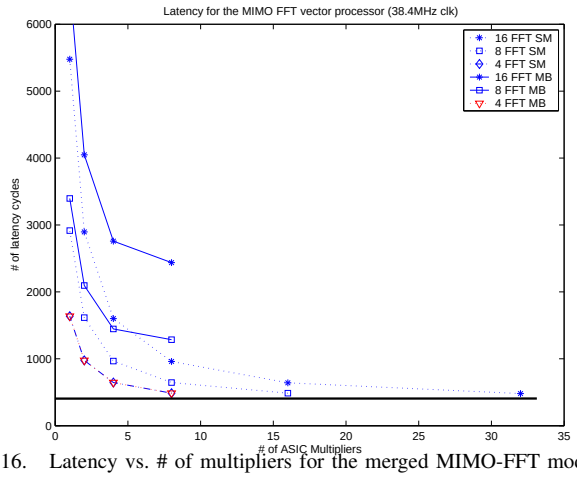

Fig. 16. Latency vs. # of multipliers for the merged MIMO-FFT module.

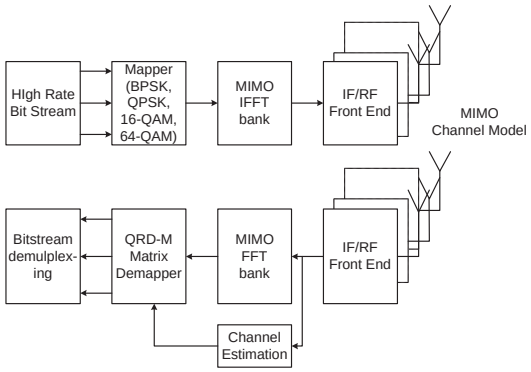

Fig. 17. System Model of the MIMO-OFDM Using Spatial Multiplexing.

at each of the q^{th} receive antenna to demodulate the frequency domain symbols.

The goal of the receiver is to detect the symbols effectively from the received signal and estimated channel coefficients. It is shown that the symbol detection is separable according to the subcarriers, i.e., the components of the N_F subcarriers are independent. Thus, this leads to the subcarrier-independent Maximum Likelihood symbol detection as $\mathbf{d}^k_{ML} = \arg\min_{\mathbf{d}^k \in \{\mathcal{S}\}^{N_T}} ||\mathbf{y}^k - \hat{\mathbf{H}}^k \mathbf{d}^k||^2$, where $\mathbf{y}^k = [y^k_1, y^k_2, \cdots, y^k_{N_R}]^T$ is the k^{th} subcarrier of all the receive antennas. $\mathbf{H}^k$ is the channel matrix of the k^{th} subcarrier. $\mathbf{d}_k = [d^k_1, d^k_2, \cdots, d^k_{N_T}]^T$ is the transmitted symbol of the k^{th} subcarrier for all the transmit antennas. The QR-decomposition [27] 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 Fig. 18 for an example with QPSK modulation and N_T transmit antennas.

B. Hardware Acceleration Prototyping Requirement

Despite of the significantly reduced complexity, 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 that in a matlab MIMO-OFDM simulation chain, the M-algorithm can occupy more

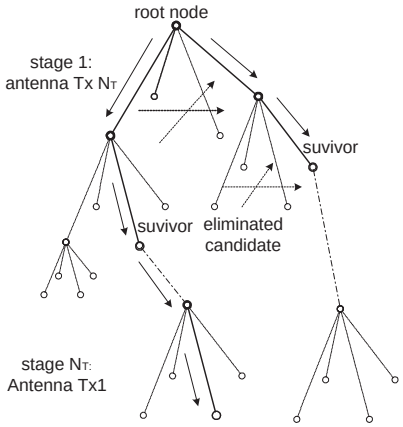

Fig. 18. The limited-tree search in QRD-M algorithm.

than 99% simulation time. 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.

To shorten the simulation time and facilitates the commercialization, a hardware accelerator platform with compact form factor was proposed in [14] based on Xilinx FPGAs. Such a platform is intended to achieve functional verification of the fixed-point hardware design and rapidly prototype the VLSI architectures for proof of concept in the future real-time $4G$ prototyping system. The limited hardware resource in the compact PCMCIA card and much lower clock rate than PC demands very efficient VLSI architecture to meet the real-time goal. However, the tree search structure is not quite suitable for VLSI implementation because of intensive memory operations with variable latency, especially for long sequence. Extensive algorithmic optimizations are required for efficient hardware architecture. The efficient VLSI hardware mapping to the QRD-M algorithm requires wide range configurability and scalability to accelerate the simulation time in matlab. This requires an efficient design methodology that can explore the design space efficiently. Catapult-C provides strong capability to meet these requirements by high level abstraction.

C. System Level Partitioning

To achieve simulation-emulation co-design, an efficient system-level partitioning of the MIMO-OFDM matlab chain is very important. The simulation chain is depicted in Fig. 19. Because the goal is for simulation time acceleration, we only need to implement the core algorithm with dominant complexity in FPGA hardware. In the simplified simulation model, the MIMO transmitter first generates random bits and map 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 *fhardqrdm_fpga_*, which consists the major subfunctions as demodulator using FFT, sorting, QR decomposition, the M-search algorithm in a C-MEX file, the de-mapping and the BER calculator. Because the M-search C-MEX file dominates more than 90% of the simulation time, the C-MEX file is redesigned in the FPGA hardware accelerator. The C APIs talk

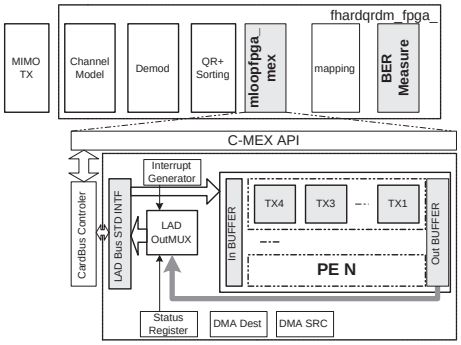

Fig. 19. The system partitioning of the MIMO-OFDM simu/emulation co-design and PE architecture of the M-algorithm.

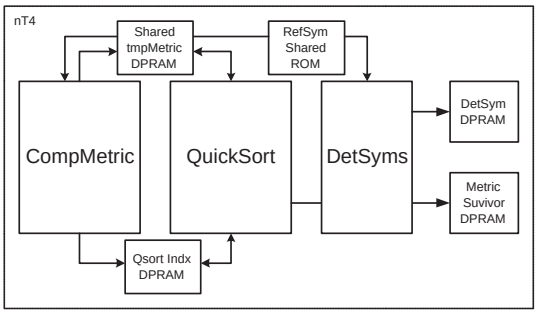

Fig. 20. The block diagram of one antenna processing with quick sort.

with the CardBus controller in the card board. The controller then communicates with the PE FPGA through the 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 multi-stage 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. To achieve the bi-directional data transfer, both the source and destination DMA buffers are needed. Because the focus of this paper is not the VLSI architecture of the M-algorithm, the architecture detail is omitted here.

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 re-ordered 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 $\mathbf{y}_k$ are pre-multiplied by $\mathbf{Q}_k^H$ to form a new receive signal as $\mathbf{\Upsilon}_k = \mathbf{Q}_k^H \mathbf{y}_k = \mathbf{R}_k \mathbf{d}_k + \mathbf{w}_k$, 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 architecture is designed in multi-stage 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. An example for the antenna $nT4$ is shown in Fig. 20. All the central antennas have same operations with much higher complexity than the first and last antennas. The sorting function becomes the bottleneck of processing since it involves many data dependencies in the sequence. Extensive memory read/write and swapping operations are required. The sorting procedure also leads to un-predictable latency depending on the input sequence. This type of computation contains a lot of "if" statement and "do-while" branches which are extremely difficult with the conventional manual design. Catapult-C can synthesize the complex FSM automatically for these types of complex logics. Moreover, it is easy to verify different pipelining tradeoffs. In this case study, we studied three major different algorithms for the sorting function, each with many partitioning and storage mapping options. It could take at least three month to design one architecture correctly using the conventional design method. However, we spent only one and half month to explore the architecture tradeoffs after the reference algorithm study is complete. From the extensive exploration, we can easily identify the most efficient architecture for a given constraint.

VII. Productivity and Efficiency

Table V 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 as baseline source code. 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 the given case study in the 3G and beyond MIMO-CDMA equalizer is significant compared with the conventional design methodology.

For the QRD-M in the MIMO-OFDM system, the run-time comparison of the original and FPGA implementation for the 4×4 MIMO configuration and 64-QAM modulation is shown in Fig. 21. We implemented 2 PEs in the V3000 FPGA in this case. For 64-QAM and $M = 64$, speedup of $100\times$ is observed with 33 MHz FPGA clock rate competing 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 & R result can be up to 90 MHz.

TABLE V

PRODUCTIVITY IMPROVEMENT FROM THE UN-TIMED 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 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 weeks tradeoff study
FIR Filtering	3 weeks/sol	1 weeks tradeoff study

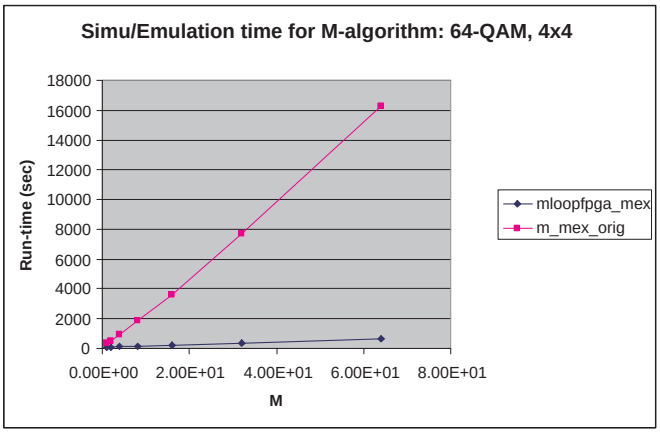

Fig. 21. Measured simulation speedup for the M-algorithm: 4×4, 64-QAM.

VIII. CONCLUSION

In this paper, we presented a rapid prototyping methodology integrating Catapult-C and other key technologies and our industrial experiences for the 3G/4G wireless systems. The standard clocking tracking and AFC blocks for CDMA systems are used as case studies to demonstrate the concept and capability of the proposed design methodology. We efficiently studied FPGA architecture tradeoffs and found the most efficient solution for a specific architecture/resource constraint. We then applied Catapult-C to explore the design space of different types of advanced core algorithms in both the MIMO-CDMA and MIMO-OFDM systems and integrated them within HDL Designer. The productivity was improved significantly, enabling extensive architectural research.

ACKNOWLEDGMENT

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.

REFERENCES

[1] A. Wiesel, L. Garca, J. Vidal, A. Pags and Javier R. Fonollosa, *Turbo linear dispersion space time coding for MIMO HSDPA systems*, 12^{th} IST Summit on Mobile and Wireless Communications, Aveiro, Portugal, June 15-18, 2003.

[2] G. D. Golden, J. G. Foschini, R. A. Valenzuela and P. W. Wolniansky, *Detection algorithm and initial laboratory results using V-BLAST space-time communication architecture*, Electron. Lett., Vol. 35, pp.14-15, Jan. 1999.

[3] G. J. Foschini, *Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas*, Bell Labs Tech. J., 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*, to appear in IEEE GlobeCom 2004.

[5] J. Yue, K. J. Kim, J. D. Gibson and R. A. Iltis, *Channel estimation and data detection for MIMO-OFDM systems*, IEEE Globecom, vol. 22, no. 1, pp. 581 - 585, Dec 2003.

[6] Y. Lee, V. K. Jain, *VLSI architecture for an advanced DS/CDMA wireless communication receiver*, Proc. of IEEE International Conference on Innovative Systems in Silicon, pp. 237-247, Oct. 1997.

[7] Y. Guo, J. Zhang, D. McCain and J. R. Cavallaro, *Scalable FPGA architectures for LMMSE-based SIMO chip equalizer in HSDPA downlink*, 37th IEEE Asilomar Conference, Monterey, CA, 2003.

[8] A. Adjoudani, E. C. Beck, ..., M. Rupp, et al, *Prototype experience for MIMO BLAST over third-generation wireless system*, IEEE JSAC, Vol. 21, No. 3, Apr. 2003.

[9] Z. Guo and P. Nilsson, *An ASIC implementation for V-BLAST detection in 0.35 μm CMOS*, accepted in IEEE International Symposium on Signal Processing and Information Technology (ISSPIT), Rome, Italy, Dec. 2004.

[10] K. Hooli, M. Juntti, M. J. Heikkila, P. Komulainen, M. Latvaaho and J. Lilleberg, *Chip-level channel equalization in WCDMA downlink*, EURASIP Journal on Applied Signal Processing, pp. 757-770, Aug.2002.

[11] J. M. Rabaey, *Low-power silicon architectures for wireless communications*, Design Automation Conference, Proceedings of the ASP-DAC 2000, Asia and South Pacific Meeting, pp. 377-380, Yokohama, Japan, 2000.

[12] A. Evans, A. Siburt, G. Vrchoknik, T. Brown, M. Dufresne, G. Hall, T. Ho and Y. Liu, *Functional verification of large ASICS*, ACM/IEEE Design Automation Conference, San Francisco, CA, pp. 650 - 655, June 1998.

[13] U. Knippin, *Early design evaluation in hardware and system prototyping for concurrent hardware/software validation in one environment*, Aptix Inc. IEEE RSP'2002, July 1-3, 2002, Darmstadt, Germany.

[14] Y. Guo and D. McCain, *Compact FPGA Hardware Accelerator for Functional Verification and Rapid Prototyping of 4G Wireless Systems*, to appear in Asilomar conference proceeding, Monterey, CA, Nov. 2004.

[15] Nokia HSDPA Demonstrator webpage: http://www.nokia.com/nokia/0,,53713,00.html

[16] J. Bhasker, *VHDL Primer: third edition*, Prentice-Hall, 1999.

[17] Y. Guo, *Advanced MIMO-CDMA Receiver for Interference Suppression: Algorithms, System-on-Chip Architectures and Design Methodology*, PhD dissertation, Rice University, Houston, May, 2005.

[18] G. De Micheli and D. C. Ku, *HERCULES - A System for High-Level Synthesis*, the 25th ACM Design Automation Conference, Anaheim, CA, June 1988.

[19] C. Y. Wang and K. K. Parhi, *High-level synthesis using concurrent transformations, scheduling, and allocation*, IEEE Trans. On Computer-Aided Design, vol. 14, no. 3, March, 1995.

[20] Y. Guo, G. Xu, D. McCain, J. R. Cavallaro, *Rapid scheduling of efficient VLSI architectures for next-generation HSDPA wire-less system using Precision-C synthesizer*, Proc. IEEE Intl. Workshop on Rapid System Prototyping'03, San Diego, CA, pp. 179-185, June 2003.

[21] Hennessy and Patterson, *Computer Architecture: a quantitative approach*, Morgan Kaufmann publishers Inc., 1996.

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

[23] http://www.celoxica.com/methodology/handelc.asp.

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

[25] H. Steendam, M. Moeneclaey, *The effect of clock frequency offsets on downlink MC-DS-CDMA* , IEEE,Internal Symposium on Spread Spectrum Techniques and Applications,Vol. 1, pp. 113-117, 2002.

[26] K. W. Yip, T. S. Ng, *Effects of carrier frequency accuracy on quasi-synchronous, multicarrier DS-CDMA communications using optimized sequences*, IEEE JSAC,Vol.17, pp. 1915-1923, Nov.1999.

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

[28] Y. Guo, J. Zhang, D. McCain, J. R. Cavallaro, *An Efficient Circulant MIMO Equalizer for CDMA Downlink: Algorithm and VLSI Architecture*, to appear in EURASIP JSIP, December 2005.