Hardware Implementation of Wireless Communications Algorithms: A Practical Approach

Antonio F. Mondragon-Torres
Rochester Institute of Technology
USA

1. Introduction

Wireless communication algorithms are implemented using a wide spectrum of building blocks such as: source coding; channel coding; modulation; multiplexing in time, frequency and code domains; channel estimation; time and frequency domain synchronization and equalization; pre-distortion; transmit and receive diversity; combat and take advantage of fading and multi-path channels; intermediate frequency (IF) processing in software defined radio, etc.

Due to this breadth of different algorithms, the traditional approach has been to create a system model in a high level language such as Matlab (Mathworks, 2011), C/C++ and recently in SystemC (SystemC, 2011). Usually these models use floating point representations, are architecture agnostic, and are time independent, among others characteristics. After the system model is available, then based on the specifications it is manually converted into a fixed point model that will take care of the finite precision required to implement the algorithm and compare its performance against the "Golden" floating point model. The reason to perform this conversion is due to cost and performance. While it is possible to program the algorithm on a floating point Digital Signal Processor (DSP) or using floating point hardware on application specific integrated circuit (ASIC) technology, the resulting: complexity; signal throughput; silicon area and cost; and power consumption among others, usually prohibits its implementation in floating point arithmetic. This is one of the reasons most of the wireless communications algorithms are implemented using a finite precision fixed point number representation.

In the last decade several technologies have made the conversion from floating point to fixed point seamless to a certain point. These technologies rely either on either a high level language such as C or C++ or a set of hardware model libraries for a particular field programmable gate array (FPGA) or ASIC technologies. In addition to these, there are some other electronic system level (ESL) design tools that can take a floating point algorithm and even preserve the same floating point testbench and transform the algorithm into a fixed point representation, where different architectural trade-offs can be made based on the area/power/latency/throughput requirements are in the system specifications.

In this chapter we do not propose a one solution fits all applications methodology, rather we will navigate through the author's encounters with different technologies at different stages in his career and how different applications have been and are currently approached. This is a summary of the last ten years of working with different tools, methodologies and design flows. What has prevailed due the level of integration of current Systems on a Chip (SoC) has been for example: component and systems reusability; fast algorithm and architecture exploration; algorithm hardware emulation; and design levels of abstraction.

2. System level design

By system level design (SLD), we refer to the modeling of the wireless communications systems based solely on the specifications or target standard. At this stage, individual and collective block level performance can be evaluated and also interconnects with other components in the system can be specified. There are two major known approaches for system design, top-down and bottoms-up methodologies.

System level design calls for a top-down methodology. In sophisticated systems such as SoCs, their complexity can be very large and it is a common practice in system level design to create a set of high level specifications with a complete vision of the system including their complete set of interconnects. The next phase is to divide the system into functional blocks, specify all internal interconnects and design each block in the subsystem. This allows the complete system to be simulated using for example a system level language such as SystemC and then be able to replace each block with its Register Transfer Level (RTL) functional equivalent. These techniques are also being heavily used to speed up system verification in which it is not possible to perform in a reasonable amount of time a complete RTL or gate level simulation due to time to market (TTM) constraints or because it is not computationally feasible. SLD methodologies allow performing a complete system level simulation at a higher level of abstraction by just including the key blocks required at the gate level to test interconnectivity and performance.

A system level simulation is in the order of tenths to thousands times faster than gate level simulations, thus assuring that all individual blocks or combinations of blocks will work after being interconnected. In Figure 1 it is shown an ideal case where a system level model or commonly referred as the "running specification" is first generated and creates a "golden" model against all performance implementations will be compared against. Ideally we would like to keep the original testbench for all modeling, design, implementation, simulation and verifications tasks, but this is not always possible. The problem arises when manual or automatic translations could change the behavior of the original testbench. One of the most critical problems in SLD development is that once you descend in the level of abstraction, the system level testbench and models are no longer updated and maintained, then deviating from the original running specification reference.

2.1 System modelling

SLD has been traditionally been done using C language, therefore it is common to refer in industry to the "C-model" as the running specification or "golden" model. The advantage is that C language is particularly fast, runs on all platforms and can represent fixed point precision easily after taking care of the fixed point operations such as rounding, truncation, saturation, etc. One disadvantage of this methodology is that it is not very straight forward to couple C simulations with RTL simulations and then obtain the complete benefits of system level modeling.

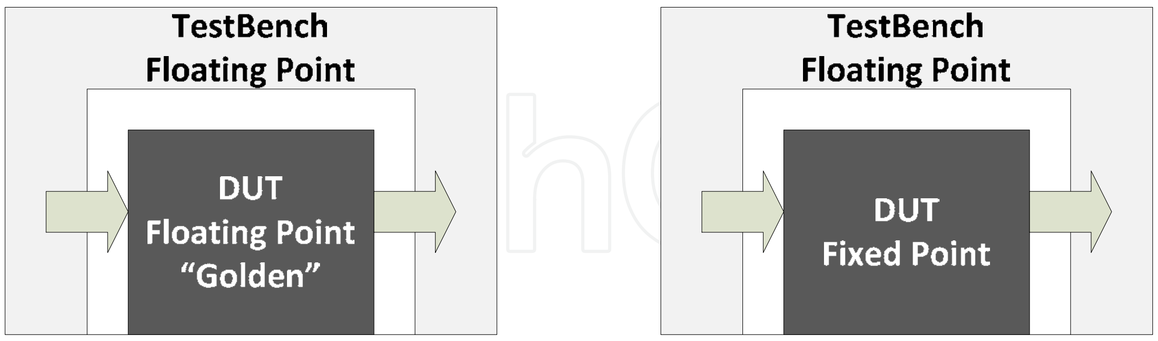


Fig. 1. System Level Modeling approach. Testbench should ideally be reused while verifying the Device Under Test (DUT) at different level of hierarchy. E.g. Behavioral, RTL, gate level netlist and parasitic extracted netlist.

More recently, SLD has been done using C++, since the level of abstraction can be taken one level further and the interfaces and testbenches can be encapsulated and reused. SystemC is a set of libraries that extend C++ to brings capabilities such as fixed point types, transaction level modeling (TLM), parallel event driven simulation compatibility, and testbench reutilization among several other features. Recently SystemC have been used to create complex reusable testbenches that interface directly with RTL code and can be executed using most of the high performance RTL event driven simulators.

A relatively new player in the SLD is System Verilog which in addition to have unique properties to perform verification and design tasks, it can also be used for system level design due to its enhancements comparable with SystemC features. The current belief is that System Verilog can be the "one size fits all" language due to its system and blocks level modeling, system and block level verification, synthesis constructs, and simulation capabilities. One company working in this space is Bluespec that provides high level system modeling, architecture exploration, verification and synthesis using a System Verilog (Bluespec, 2011).

So far, we have talked about languages that are capable of performing SLD, but the drawback of these languages is that they rely on the user knowing the architectural constraints of the design. There is also another very popular complete set of SLD languages that also allow to perform system level modeling at the same time that its users are closer to the algorithm development rather than the language options we just mentioned above. The primary SLD system language for modeling is Mathwork's Matlab and it's time-driven block-based tool Simulink. There are also other tools that also used for system level modeling such as Agilent's SystemVue (Agilent, 2011) and Synopsys' SPW (Synopsys, 2011a) to cite a few used previously by the author.

The author has been exposed to more SLD projects done in Matlab, and in some cases the complete running specification has been kept in Matlab m-code, even the fixed point implementation and test vector generation. Other projects, had Matlab as the main algorithm verification driver, followed by a C model implementation and then by an RTL implementation. Each tool/language translation can potentially introduce errors in the system level design and verification stages. In an ideal world, we should only deal with one system level language, one system level testbench and multiple implementations at different levels of abstraction. By having different models at different levels of abstraction we can have a different model to resolve efficiently different problems such as interconnection, timing, programming, functional verification, synthesizability, and feasibility of implementation.

2.2 Algorithmic focused system level design

The focus on this chapter will center around Matlab and especially on Simulink. The two major FPGA providers Xilinx (Xilinx, 2011b) and Altera (Altera, 2011a), make available libraries that allow efficient block level modeling of wireless communications algorithms and its automatic conversion to RTL. The code can be either downloaded to the FPGA for standalone algorithm implementation or used with hardware in the loop (HIL) functionality that allows a particular block of the system to be emulated using an FPGA device, this is with the purpose of performing hardware acceleration.

Nowadays the common first step taken by researchers is to test their ideas in Matlab's m-code. Matlab as a system level platform allows a very fast and efficient algorithm implementation of complete systems. Matlab does not include the conception of time; it is more comparable to high level programming languages; has a vast set of libraries or toolboxes in many disciplines; and it is not limited to math or engineering. Matlab has become an indispensable tool in modern electronic design and engineering in general.

If the designer would like to model the system including time as another design dimension, Simulink could be used to design complete dynamic systems that are time aware and also include a large number of libraries or toolboxes for a large number of disciplines.

2.3 System architecture

When evaluating an algorithm, the designer is mostly concerned on modeling a system. One of the problems is that the final implementation cannot be readily extracted from this system level modeling easily. There are different levels of system models, some models can be bit accurate and/or cycle accurate.

In a bit accurate model, the system traditionally has been modeled using floating point precision, and then the algorithm has been converted into fixed point precision for efficient implementation. At this stage the main concern is that the signal to quantization noise ratio (SQNR) will dictate the losses due by the effects of for example: quantization, rounding and saturation. This transformation stage can be performed in Matlab/Simulink, SystemC and C/C++. A bit accurate model will have a very close representation of the final implementation in terms of hardware cost and performance. One problem here is that the internal precision of the operation is difficult to model until the final architecture has been decided.

In a cycle accurate model, the systems are architected such that the generated hardware corresponds one to one to the behavioral model in terms of time execution. The advantage is that a true bit accurate and cycle accurate simulation can be obtained, but at much higher simulation speed to their RTL or gate level simulations. In the author's experience, this model has not been used much in the past, since it is tied up with a fixed architecture so the conversion to RTL is straightforward with no ambiguities.

After the fixed point precision has been proposed, it is traditionally coded either in a high level language or in a hardware description language. Of course at this stage the model can continue to be modeled in Simulink. Typically and architectural description is being pursued at this level and the model should closely represent the hardware to be implemented.

What is interesting is that at this stage, there are at least from two or more "system models." One very common error is to not update the higher level with architectural changes once high level modeling stage has "finished", this could lead to inaccuracies on the implementation since it is no longer compared with the "golden" model anymore. As we mentioned, the models can get out of synchronization due to lack of communication between the system's team and the implementation's team. It is of extreme importance throughout the life of the project to have all models updated to reflect the latest changes in both SLD and RTL since each one represents a running specification of the system at different levels of hierarchy.

2.4 System testbench

A testbench is created at the behavioral level, what this means is that the testbench is not to be synthesized, that is why the testbench can include language constructs that represent stimulus and analysis rather than processing and are not directly synthesizable. The testbench is designed to test a "black box" or commonly known as the Device Under Test (DUT), generate inputs, measure responses and compare with known "golden" vectors. One very useful feature in Verilog HDL is to be able from the testbench to descend into the design hierarchy and probe on internal signals that are not available at the interface level. VHDL 2008 includes hierarchical names for verification as well.

A rule of thumb says that when a design is "finished", it is just 30% complete and the validation and verification (V&V) stages will start to cover the remaining 70% effort to have a verified finished design. There are different methodologies to accomplish this and unfortunately Verilog HDL and VHDL have not been robust enough to allow complete and efficient design verification. Due to the later, several proprietary verification languages evolved and recently several methodologies such as Open Verification Methodology (OVM)(Cadence, 2011), Verification Methodology Manual (VMM)(Accellera, 2011) and Universal Verification Methodology (UVM)(Synopsys, 2011c) have been developed to fill the gap between HDL and proprietary verification languages including a common framework for verification. The common denominator in all these methodologies is the use of System Verilog as the driver of all three. System Verilog is evolving as the verification and design solution language since it contains the best of design, synthesis, simulation and verification features, the versatility of the HDLs, and it is designed for system level verification.

Talking about levels of design abstraction, another very common approach is to use the popular C and C++ languages to describe algorithms to be implemented in hardware. We have found that several Electronic System Level (ESL) design tools generate SystemC testbenches that could be used as standalone applications as well as integrated into event driven simulators that are the core when designing hardware implementations. Some examples are Pico Extreme from Synfora (acquired by Synopsys and now is SynphonyC compiler)(Synopsys, 2011b) and CatapultC from Mentor Graphics (CatapultC is more like C++rather than SystemC) (MentorGraphics, 2011).

We have talked about Matlab/Simulink being used at the system level design phase. In order to take full advantage of a common testbench, a hardware design could rely entirely on this platform for rapid prototyping by accomplishing transformations at the level of modeling hierarchy.

Once a design is transformed for example from Matlab m-code to Simulink, or perhaps the design was started in Simulink directly, there are a series of custom libraries that allow the designer to transform their design directly into hardware and keep the original Simulink testbench to feed the hardware design. The design could be verified by generating HDL RTL and by running event driven simulations side by side the Simulink engine and compare with the original Simulink model to verify that the RTL code generated matches the desired abstracted model. Not only a standalone simulation is conceivable, it is possible to download the application directly into an FPGA and generate excitation signals and receive the data back in Simulink. This allows to verify hardware performance at full speed or to accelerate algorithm execution that will take a long time on an event driven simulator. There are several products with similar capabilities such as National Instrument's LabView (NI, 2011) that also allows the option to have "Hardware In the Loop" (HIL) as a way to accelerate computing performance by implementing the algorithm directly in hardware.

The philosophy at this level is to try to reuse the testbench as much as possible to verify correctness of the design at a very high level of abstraction and to code a single testbench that could be used at the system level, while still being able to run the components at single levels of abstraction, namely behavioral, RTL and gate level.

3. Fixed point number representation

This section will cover the different formats used to represent a number using fixed point precision. In addition, the effects of truncation, rounding, and saturation will be covered. SystemC provides a standard set of fixed point types that have been also adopted and adapted by electronic system level (ESL) tools. We will talk about SystemC's fixed number representation. We will talk also about traditional RTL fixed point implementations and the required hardware, complexity and performance.

3.1 SystemC fixed point data types

SystemC includes the *sc_fixed* and *sc_ufixed* data types to represent fixed point signed and unsigned numbers the syntax to include these in a SystemC program is the following:

sc_fixed<*wl, iwl, q_mode, o_mode, n_bits*>
sc_ufixed<*wl, iwl, q_mode, o_mode, n_bits*>

where

wl: total word_length
iwl: integer word length
q_mode: quantization mode
o_mode: overflow mode
n_bits: number of saturated bits

Quantization modes: SC_RND, SC_RND_ZERO, SC_RND_INF, SC_RND_MIN_INF, SC_RND_CONV, SC_TRN, SC_TRN_ZERO

Overflow modes: SC_SAT, SC_SAT_ZERO, SC_SAT_SYM, SC_WRAP, SC_WRAP_SM

For example if we would like to declare a signed integer variable with 16 total bits of which 8 bits are integer, we declare:

sc_fixed<16,8> *number;*

As can be observed in Figure 2, the 16 bit number will contain 8 integer bits and 8 fractional bits. The maximum number that can be represented is $2^7 - 2^{-8} \approx 128$ and the minimum number will be -2^7 = -128 with a $2^{-8} \approx 3.9\text{x}10^{-3}$ resolution. By default, the number will have a quantization mode of *q_mode* = *SC_TRN* which means that the number precision will be truncated after each mathematical operation or assignment, and the number will have an overflow mode o_mode=SC_WRAP which means that the number will wrap from approximately 128 to -128. The different modes allow for flexibility in the rounding and saturation operations that are useful to limit the number of bits enhance the SQNR and also to allow infrequent numbers to be saturated and save on the total number of bits. Of course, the price is additional hardware and probably timing to perform these operations.

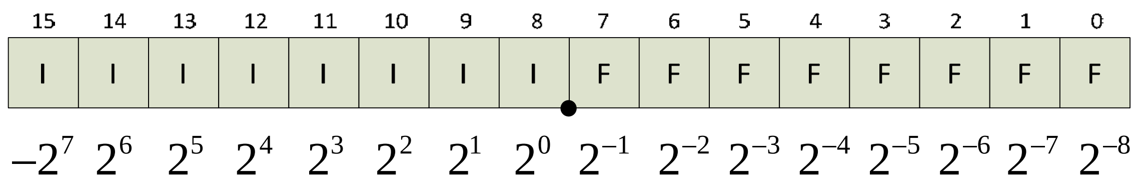


Fig. 2. *sc_fixed<16,8>* representation of a fixed point number.

There are too many ways to describe fixed point notations and representations, but we think that this represents a commonly used format in most of ESL tools that we have explored.

4. Floating to fixed point design considerations

A practical implementation of a wireless communication algorithm involves the conversion of a floating point representation into a fixed point representation. This process is related to the optimum number of bits to be used to represent the different quantities through the algorithm. This process is performed to save complexity, area, power, and timing closure. A fixed point implementation is the most efficient solution since it is customized to avoid waste of resources. The tradeoffs against a floating point implementation are noise, non-linearities and other effects introduced by the processes of: quantization, truncation, rounding, saturation and wrapping among the most important.

Both the floating point and the fixed point solutions have to be compared against each other and one of the most common measure of fixed point performance is the signal to quantization noise ratio (SQNR) (Rappaport, 2001).

Several tools are available to allow the evaluation of a fixed point implementation against a floating point implementation. One of the most important factors is the dynamic range of the signal in question. Floating point adapts to the signal dynamic range, but when the conversion is to be done, a good set of statistics has to be obtained in order to get the most out of the fixed point implementation. The probability density function of the signal will give insight on the range of values that occur as well as their frequencies of occurrence. It may be acceptable to saturate a signal if overshoots are infrequent. We need to carefully evaluate the penalty imposed by this saturation operation and the ripple effects that it could have. This process allows to use just the necessary number of bits to handle the signal most of the time, thus saving in terms of area, power and timing. In section 5.3, we talk about some of the little steps that have to be taken throughout the design in order to save in power consumption. As mentioned, power consumption savings start at the system level architecture throughout the ASIC and FPGA methodologies.

Sometimes the processed signal could be normalized in order to have a unique universal hardware to handle the algorithm. It is very important to take into consideration the places where the arithmetic operations involve a growth in the number of bits assigned at each operation. For example, for every addition of two operands, a growth of one bit has to be appended to account for the overflow of adding both signals. If four signals are added, only a growth of two bits is expected. On the other hand a multiplication creates larger precisions since the number of bits in the multiplication result is the addition of the number of bits of the operands and also it has to be taken into account if the numbers are signed or unsigned.

The fixed point resolution at every stage needs to be adapted and maintained by the operations themselves and specific processing needs to be done to generate a common format. These operations are the truncation, rounding, saturation and wrapping covered briefly for SystemC data type in section 3.

A nice framework of the use of fixed point data types that could be incorporated into C/C++ algorithm simulations are the SystemC fixed point types available in the IEEE 1666™-2005: Open SystemC Language Reference Manual (SystemC, 2011). There are some other alternatives to fixed point data types such as the Algorithmic C Datatypes (Mentor-Graphics, 2011) that claim to simulate much faster than the original SystemC types and used in the ESL tool CatapultC. The ESL tool Pico Extreme uses the SystemC fixed point data types as the input to the high level synthesis process.

Matlab/Simulink also has a very nice framework to explore floating to fixed point conversion. When hardware will be generated directly from Simulink, it is very natural to alternate between floating point and fixed point for system level design. Designs that are targeted for Xilinx or Altera FPGAs could naturally use this flow and reuse the floating point testbench to generate the excitation signals that could be used within the Matlab/Simulink environment in for example Hardware in the Loop (HIL) configuration or fed externally to the FPGA using an arbitrary waveform pattern generator.

Another very useful tool for creating executable specifications in C++ is to use IT++ (IT++, 2011) libraries available for simulation of communication systems.

Each EDA vendor has a different set of tools that allow designers to make the implementation of floating point to fixed point as seamless as possible. This conversion process is a required step that cannot be avoided and traditionally it has been done manually and by matching the results of the Golden model against HDL RTL simulation. Sometimes this comparison is bit accurate, but in some cases the comparison is just done at the SQNR level due to the difficulty to model all the internal operations and stages of a particular hardware implementation.

5. Register transfer level design

Once a system has been verified for performance and has been converted from a floating to a fixed point representation, the specifications are passed to the register transfer level (RTL) design engineer to come up with an architecture that will achieve the desired performance, while consuming minimum power at the right frequency of operation, using minimum area, sharing resources efficiently, reusing as much components as possible, and coming with an optimum tradeoff between hardware and software implementations. We can see that this is not usually an easy task to perform, even for experienced designers.

5.1 Architecture

In this section we will give an overview of the importance of the architecture in RTL design. Examples of different architectures for complex multipliers, finite impulse response (FIR) filters, fast Fourier transforms (FFT) and Turbo Codes will be given comparing their complexity, throughput, maximum frequency of operation and power consumption.

When an efficient architecture is sought, each gate, each register, each adder and each multiplier counts. Sometime it is a good approximation at the system level to count the number of arithmetic operations to get an initial estimate of the silicon area that will be used for the algorithm. While this is a crude approximation it is a very good start point to allocate resources on the System on a Chip (SoC). Many companies have spreadsheets that contain average values for different operations in a particular technology; based on hundredths of designs. The architecture task is to find the optimum implementation of a particular algorithm while accomplishing all the above referred design parameters.

When an algorithm is implemented, what will be the final underlying technology for implementation? ASIC or FPGA; or will it be driven by software and just primitive building blocks will be used as coprocessors or hardware accelerators. Whenever a product needs to be designed on an application area that continues to grow and generate new algorithms and implementation such as video processing, sometime an analytics engine must be architected that will provide co-processing or hardware acceleration by implementing the most common image processing algorithms. This idea could be applied to any communications system or signal processing system where a solution could include a common set of hardware accelerators or coprocessors that realize functions that are basic and will not easily change. One very good example is the TMS320TCI6482 Fixed-Point Digital Signal Processor (Texas-Instruments, 2011) that is used for third generation mobile wireless infrastructure applications and contains three important coprocessors: Rake Search Accelerator, Enhanced Viterbi Decoder Coprocessor and Enhanced Turbo Decoder Coprocessor.

So the question is: When implementing a particular algorithm, how can we architect it such that it is efficient in all senses (are, power, timing) as well as versatile? The answer depends on the application. That is why hardware/software partitioning is a very important stage that has to be developed very carefully by thinking ahead of possible application scenarios. In some cases there is no option, and the algorithm has to be implemented in hardware, otherwise the throughput and performance requirements may not be met. Let's explore briefly some practical examples of blocks used in wireless communication systems and just brainstorm on which architectures may be suitable.

Finite Impulse Response Filters

An FIR filter implementation can be thought as a trivial task, since it involves the addition of the weighted version of a series of delayed versions of an input signal. While it seems very simple, we have several tradeoffs when selecting the optimum architecture for implementation. For an FIR filter implementation we have for example the following textbook structures: Transversal, linear phase, fast convolution, frequency sample, and cascade (Ifeachor, 1993). When implementing on for example on FPGAs, then we found for example the following forms: Standard, transpose, systolic, systolic with pipelined multipliers(Ascent, 2010).

Most of the FPGA architectures are enhanced to make more efficient the implementation of particular DSP algorithms and the architecture selection may fit into the most efficient configuration for a particular FPGA vendor or family. If we are targeting ASIC, then the architecture will be different depending on the library provided by the technology vendor. When implementing an FIR or any other type of filter or signal processing algorithm, we need to evaluate the underlying implementation technology for tuning the structure for efficient and optimum operation.

Turbo Codes

One interesting example is on Turbo Codes, while the pseudo-random interleaver is supposed to be "random", there has been a pattern defined on how the data could be efficiently accessed. Some interleavers are contention free, while some others have contentions depending on the standard. For example, one of the major differences on the third generation wireless standards namely 3GPP(W-CDMA) and 3GPP-2 (CDMA2000) is on the type of interleaver generator used, this means that to a certain degree it would be possible to design a Turbo Coder/Decoder that could easily implement both standards.

The purpose of an efficient implementation of an interleaver hardware is to have different processing units accessing different memory banks in parallel, some examples on the search for common hardware that could potentially be used for different standards are shown in (Yang, Yuming, Goel, & Cavallaro, 2008), (Borrayo-Sandoval, Parra-Michel, Gonzalez-Perez, Printzen, & Feregrino-Uribe, 2009) and (Abdel-Hamid, Fahmy, Khairy, & Shalash, 2011). The architecture is a function of the standard and sometimes it is very difficult to find a "one architecture fits all" type of solution and in some case to make the interleaver compatible with multiple standards, on-the-fly generation is the best approach, but there can be irregularities or bubbles inserted into the overall computation. This is one of the challenges in mobile wireless that sometimes is easier to implement complete different subsystems performing efficiently one particular standard, rather than having an architecture that could perform all. This is the case in mobile cellular second generation GSM (Global System for Mobile Communications, originally Groupe Spécial Mobile) and third generation cellular W-CDMA (wideband code division multiple access) that minimum reusability could be achieved and to a certain extent there are two complete wireless modems implemented for each standard.

Fast Fourier Transform

Many of the modern wireless communications algorithms migrated from the CDMA to the Orthogonal Frequency Division Multiple Access (OFDMA) technologies. One of the main reasons to transfer to a completely new technology might have been that the current state of the art on integrated circuit design allowed the efficient implementation of algorithm architectures that were not previously convenient to implement in hardware. This is the case of the Fast Fourier Transform (FFT) which is the core of Orthogonal Frequency Division multiplexing (OFDM) and its derivatives such as OFDMA (Yin & Alamouti, 2006).

OFDM and FFT techniques are not new, as a matter of fact they have been around longer that many of the current wireless technologies. What it is new, is the feasibility of the algorithms to be implemented on silicon. An efficient architecture implementation for a pipelined FFT (Shousheng & Torkelson, 1998) has been used as a benchmark for hardware implementation of the FFT algorithms, this technique allows all hardware units to be used at all times once the pipeline is full and is very convenient for FPGA or ASIC implementation.

We will just briefly talk about this on section 10, since it is one example that comes with the FPGA libraries and the purpose of this chapter is not to develop a new FFT form, but rather to see how it can be implemented.

5.2 Maximum operating frequency

While it could be easy to convert an algorithm from floating point to fixed point and to identify architectures for its implementation, the final underlying technology should be taken into account to determine the maximum operating frequency and in some cases the required level of parallelism and/or pipelining. It can be true that an algorithm designed for FPGA will run without major modifications on ASIC, but the reverse is not always true. FPGAs are used widely to perform ASIC emulation, but it does not make much sense to have two different versions of the algorithm running on either technology, since this could invalidate the overall algorithm validation. Sometimes the same code could be run, but in slow motion on FPGAs if real time constraints are not required. If real time is a factor, only some of the low throughput modes could be run on the FPGA platform and simulated for ASIC.

5.3 Power consumption

Power consumption in mobile devices is a crucial part of the algorithm selection and it is tightly coupled to architecture's implementation, frequency of operation, underlying technology, voltage supply, and gate level node toggle rates to give some examples. In this section we will cover some of the important features to be considered when designing power optimized algorithms implementations.

When designing digital systems we all know that a magic button exists that reduces power consumption to the minimum. Unfortunately this is not the case, the magic button does not exist and power savings start at the system level design, the architecture selection, the RTL implementation, the operating frequency, the integrated circuit technology chosen, the gate clocking methodology, use of multi-V_{dd} and multi-V_{th} technologies, and leakage among some of the most important factors. In reality power savings are being done in small steps starting from efficiency at the system and RTL level design. One power saving criteria is: if you do not have to toggle a signal, don't do it! Power consumption is a function of the frequency of operation, the load capacitance and the power supply voltage. On average, the gate level nodes switch at around 10% to 12%, while an RTL level simulation could have toggles close to 50% meaning that all units are being used all the time and there is no waste in terms of hardware resources.

When deciding the fixed point representation, every bit in the precision counts towards the total power consumption, the number of gate levels between registers the load capacitance of each node. If we decide to include saturation and/or rounding, there are additional gates required to perform these operations. The cost of additional hardware can be worth the gates if the bit precision is reduced from a system with a wide dynamic range that takes into account no overflow for signals that can have very large excursions but are very infrequent. So what could be the best tradeoff between complexity, fixed point precision, internal normalizations, and processing? There is not a single solution to the problem, the best will be to statistically characterize the signals being handled to find out their probability distributions and then based on these determine the dynamic range to be used and if saturation/wrapping and truncation/rounding could be used and within these which methods to apply as mentioned in section 3.

Power consumption depends on the circuit layout as well, while old technologies used to be characterized in terms of gate delays, input capacitance and output load driving capacitance, the end game has changed and modern technologies have to take into account the effects of interconnection delays due to distributed resistance, inductance and capacitance. The solution to the power consumption estimate is not final until the circuit has been placed and routed and transistors are sized. If an FPGA implementation is sought, a similar approach is taken but control is coarser due to the huge number of paths that the signals have to flow in order to be routed among all resources.

Another important factor are the power supply V_{dd} and the threshold voltage V_{th} of the transistors. These two factors control the voltage excursion of the signals and most important the operation region of the transistor. Most of the digital logic design rules assume that the transistors are operating in saturation, power is consumed while transitioning through the active region and this is the region where you want to get out as fast as possible. A transistor operating under saturation regime has a quadratic transconductance relation of the current I and the input gate voltage V_g. When a transistor is not in saturation, it could be in linear region or even in sub-threshold. A transistor in the latter does not have a quadratic, but an exponential transconductance relation. While this is the most power efficient operating regime, it is also the slowest. Many circuits that need very low power consumption can work in sub-threshold, but there is a huge variability and precision constraints. Most of these designs involve linear analog mode operations.

So what is the secret formula to design power efficient devices? The answer is discipline! Try to save as much as possible at each level in the design hierarchy. If it is in software, set the processor to sleep if there is nothing important to do. If it is hardware, do not toggle nodes that do not require to be toggled, gate the clocks so you can lower power consumption in blocks not used, reduce powers supply V_{dd} to the minimum allowed for efficient operation of the algorithm and design using just the right number of bits. More techniques for low-power CMOS design have been published and good overviews are given in (Chandrakasan & Brodersen, 1998) and (Sanchez-Sinencio & Andreou, 1999).

6. Electronic System Level Design

Electronic System Level Design (ESL) design has come from a promising technology to a reality. Companies such as Cadence, Mentor Graphics and Synopsys have their own ESL tools and have integrated these into their System on a Chip (SoC) design flows. In this section we will address some of the most important features of ESL which are architecture exploration, power consumption estimation, throughput, clock cycle budgets allocated, and the overall integrated verification framework from untimed C/C++ golden model, all the way to gate level synthesis.

One of the advantages of ESL tools is that the same testbench used to design a block could be reused at all levels of abstraction thus minimizing the probability of introducing errors at different levels of the implementation. While RTL design requires thinking very carefully on a target architecture, ESL allows exploring different architectures and taking tradeoffs using a high level description of the algorithm, and avoids the designer to go to the RTL level to verify block's performance. We will go through examples of an OFDM FFT implementation as well as MIMO signal processing. ESL niche applications are hardware accelerators that traditionally are hooked to a microcontroller platform such as an ARM processor and handle data processing intensive operations. This is a common practice in SoC design, several intellectual property (IP) vendors concentrate their products in offering very high performance blocks that interface with a common bus architecture such as AMBA.

7. FPGA implementation

For FPGA implementations we could always resort to the traditional RTL implementation of the algorithm. For this section we will resort to Mathwork's Matlab/Simulink implementations of particular algorithms by the automatic generation of RTL code to be either downloaded to the FPGA and to be tested standalone or to the Matlab/Simulink testbench that could be used to drive the simulation and the actual RTL code will be executed in the FPGA. The latter is referred as hardware in the loop (HIL).

We will give examples of: converting a chaotic modulator/demodulator from Matlab code to a Simulink model; to a Simulink model using Altera DSP builder blocks; and demonstrating the algorithm working on a development board after digital to analog and analog to digital conversions.

In FPGAs the pool of resources is fixed. Depending on the particular algorithm, it could be better placed in one of the different families of FPGAs available by different vendors. Datapath architectures can be very efficiently instantiated on FPGAs since most of building blocks included in these devices are designed for very high performance digital signal processing algorithms. We will talk about the tradeoffs when FPGA utilization is low and high and the effort to place and route (P&R) as well as timing closure.

8. ASIC implementation

Most of the wireless communication algorithms would have two versions: one for wireless infrastructure that needs high performance and power is important but not critical since it is always connected to an external power source, and another for mobile wireless devices in which performance is a requirement but power has to be optimized in order to make the device usable, power efficient and competitive. In this section we will explore these two types of implementation in applications specific integrated circuits (ASIC). We will give an example of a turbo code interleaver/de-interleaver that had been implemented and verified using simulation and an FPGA platform and the changes required to take it to an ASIC implementation.

9. Hardware acceleration

Sometimes it is not possible to evaluate an algorithm using regular simulation techniques due to the computing power that is required to perform these tasks. SoC designs are a good examples of these constraints, not all block could be implemented and verified at the gate level in simulation due to the fact that it will take from hours to weeks to perform these simulations. For these cases it is common to use FPGAs as hardware accelerators or ASIC emulators. ESL tools are very efficient in generating these type of blocks that can be either instantiated for FPGA or ASIC and the only real difference is on the characterized libraries used as well as the system clock frequency.

The basic requirements while designing custom datapath components is to create hardware accelerators that could work as standalone blocks. Normally these components will become part of a large SoC. Many of the current embedded products recently designed are composed of a microcontroller such as an ARM core, a standard bus such as AMBA, and a series of Intellectual Property (IP) blocks that realize specific functions that require high performance and low-power. This is mostly true on cellular mobile devices, while for base stations a dedicated Digital Signal Processor (DSP) could be used since throughput is a more important constraint than power consumption. It is worth mention that these designs could be done in the same technology geometry, but with different characteristics: base station would most likely use a high performance, higher threshold voltage and large leakage process while the mobile device will be constrained to medium performance, very low leakage process and low and probably variable threshold voltages.

Some examples of systems that are designed as hardware accelerators in cellular technologies are:

- Equalizers
- Viterbi, Turbo and LDPC decoders

- OFDM Modems
- Rake receivers
- Correlators
- Synchronizers
- Channel estimators
- Matched filters
- Rate matching filters
- Encryption/decryption
- Modulator/demodulator
- Antenna diversity and MIMO processing

The question is which functions will run on software and which functions will run on hardware. This lies in the gray area of hardware/software partitioning. There are different specifications that need to be considered before taking an educated decision. In theory, anything that could be done in hardware could be done in software and vice versa (of course having an infinitely fast processor with a humongous bus bandwidth and a large number of I/Os). We must carefully evaluate the hardware components to be implemented since no field upgradeability will be possible once an ASIC has been manufactured; we need to find the equilibrium where a firmware patch could potentially get rid of any anomaly not detected at verification and validation time.

In particular, the author worked for many years in teams concentrated on hardware accelerators, but all these components were part of a SoC where traditionally an ARM processor was used with a standard interconnect such as AMBA(ARM, 2011) or OCP (OCP, 2011) and the hardware accelerators were mapped as peripherals in the processor memory space. The ASIC design was first simulated, then emulated on a large FPGA platform at a constrained speed and then the ASIC could finally be developed.

In academia we are more involved with FPGA designs and in particular the platforms being used for teaching include the possibility of a soft core processor. For the author's particular case the platform is Altera and the soft core processor is the Nios II. It is interesting to find that a C to RTL application program exists that allows functions implemented in software could be converted into hardware accelerators. The application is C2H (Altera, 2011b) and even that the author has not been able to test it, it looks promising since it allows the exploration of different hardware/software partitions that could impact the total silicon area, performance, power and cost of a particular application (Frazer, 2088). In the case of FPGA design it could lead to be able to reduce costs or performance by moving back and forth different FPGA migration devices that are pin compatible, but vary in the number of logic elements available, the number of I/O pins available and cost. An equivalent tool exist from Xilinx called Auto-ESL (Xilinx, 2011a) that generates code from C/C++/SystemC.

10. Hardware implementation examples

10.1 MOC digital communications system implementation

In this design example, we will walk through the steps required to implement a mutually orthogonal chaotic (MOC) digital communications system (Glenn, 2009) algorithm architected in Simulink to run on FPGA hardware and the constraints imposed by these steps that were not considered in the original design, that affect the systems performance.

The MOC algorithm was coded first in m-code and later converted into a Simulink model. This is shown in Figure 3. The model allows following what the algorithm does without going deep into the details and the model is time dependent. The data rates at the input and output of each block are not shown and this is one of the most important features to consider in a datapath Simulink model.

After looking at the architecture presented for implementation, each of the blocks was substituted by the equivalent Altera DSP Builder available blocks. Some of the blocks have a direct equivalent while some others have to be converted into an equivalent hardware component. This is shown in Figure 4.

Since this block is originally excited by a binary signal, some digital components were used to group the bitstream into a fixed number of bits that will be used to select the modulation waveform. The original Simulink model does not have time restrictions and could potentially generate a waveform with a very large precision, but for practical reasons the implementation is restricted to a particular clock frequency and thus the number of samples to choose for the modulation waveform has an impact on the algorithm performance. A study of the optimum number of samples and the optimum number of bits to represent each modulation waveform had to be done. Each modulated waveforms also could change in sign and or magnitude, for Simulink the operation is just a simple multiplication, but for a hardware implementation it is more efficient to allocate ROM tables and access the correct magnitude and phase. This is similar to storing one quarter of the phase of a sine wave and generates sine and cosine waveforms out of this reduced table. The difference is that the basis functions for this algorithm are chaotic waveforms, then it is difficult to exploit any symmetry property.

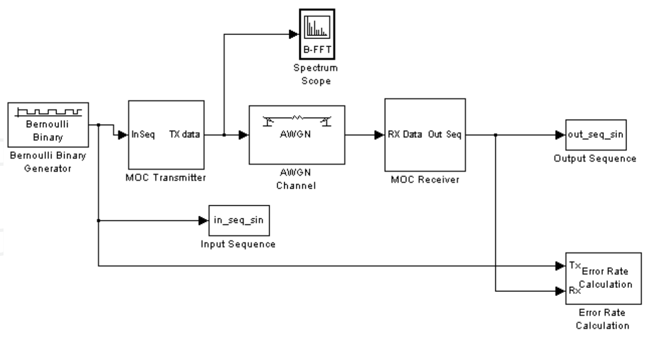
In Simulink it is very convenient to add very high level functions such as the modulators and demodulators observed in Figure 3b and Figure 3c, while this may not be required for a baseband algorithm like the one that are implemented on FPGAs. For implementation and testing we decided to work just at the baseband level.

After the model was converted, we compared the values generated by the Simulink blocks simulation against the one generated by using the Altera DSP Builder blocks. The signals were matched and SQNR was computed to validate the approach as well as rate matching was performed to match the samples. The bit sequence and the modulated waveforms are shown in Figure 4d.

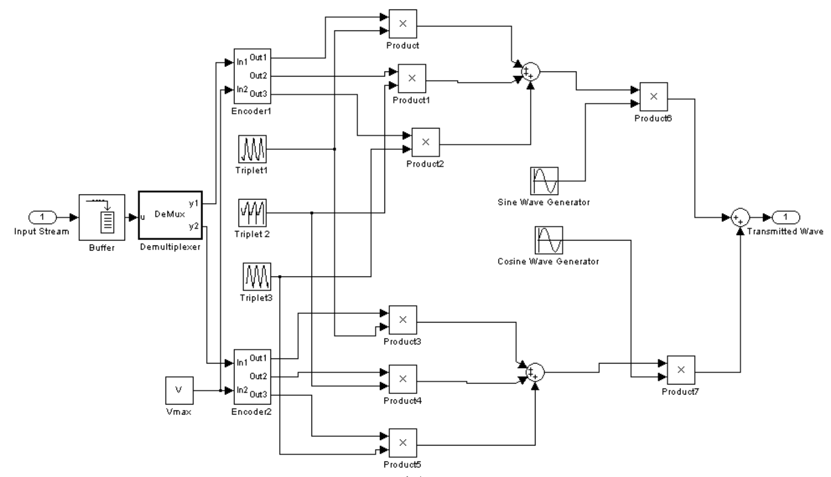
The next step is to generate HDL RTL out of the Altera DSP Builder blocks. This is shown in Figure 5a where RTL code is generated, a Simulink simulation is run, followed by a Modelsim RTL simulation and both simulations are compared and the differences are noted. The generated HDL RTL now can be synthesized and programmed into the FPGA for further development. Since for this particular system the excitation is being generated in the test bench by using a Bernoulli random number generator, we decided to use a pseudo random noise (PRN) sequence generator to embed into the FPGA for standalone testing.

The results for the transmitter are shown in Figure 6, where a) is the Altera Cyclone II FPGA testing board with two 14-bit resolution and data rate up to 65 MSPS analog to digital converters and two 14-bit resolution and data rate up to 125 MSPS digital to analog converters. This configuration is suited for testing communication transceiver applications, digital signal processing algorithms and as a platform for various modulation techniques such as the presented in this implementation example.

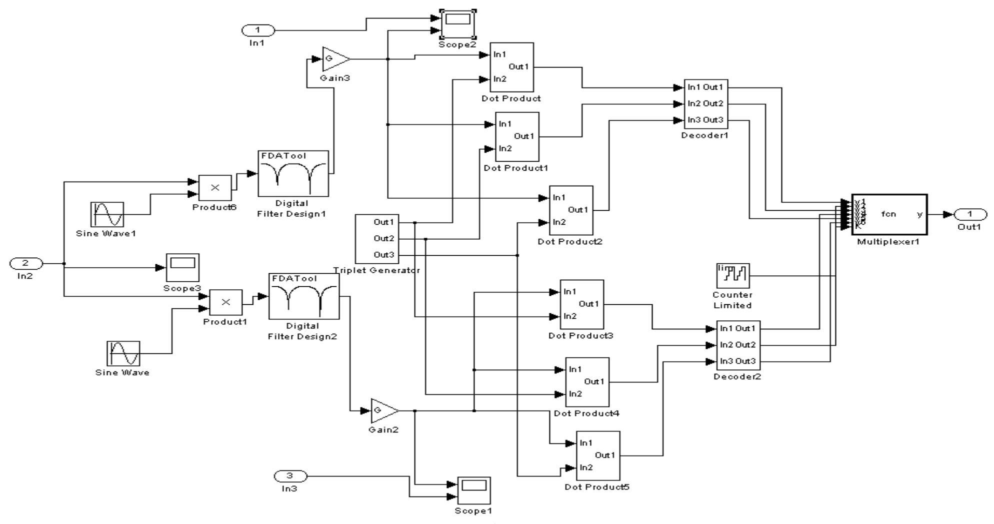
Figure 6b shows the modulation operation when an all zero pattern is generated. Figure 6c shows the PRN sequence excitation modulation waveforms and Figure 6d shows a screen capture of the MOC modulated waveforms.



a)

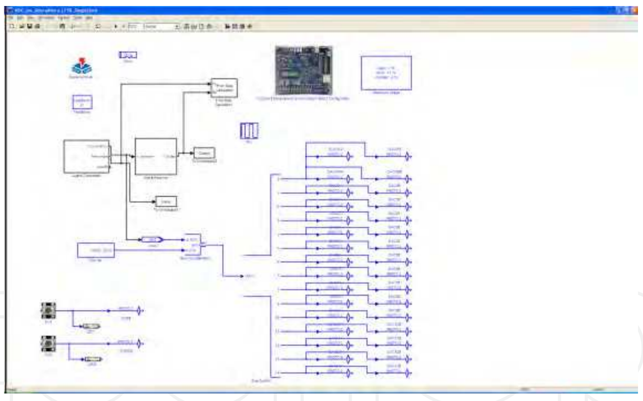


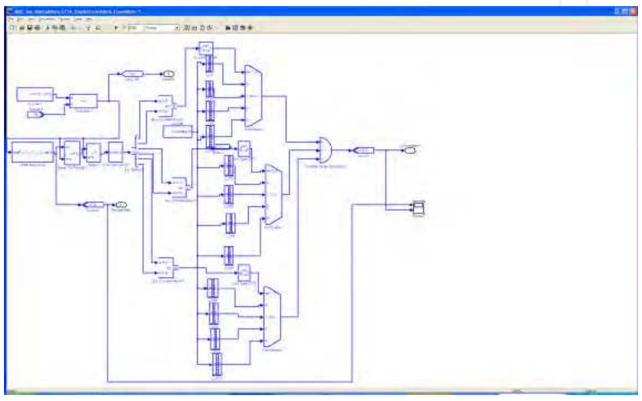
b)

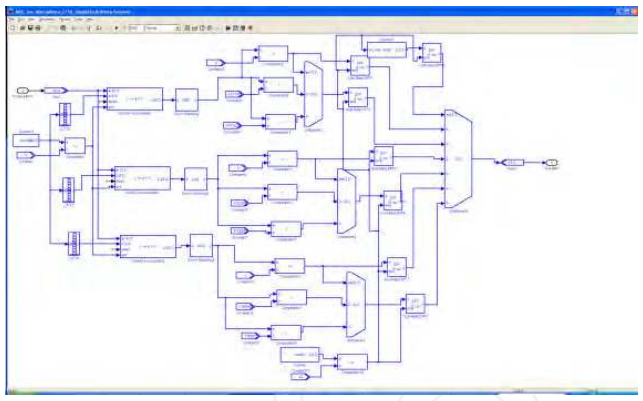


c)

Fig. 3. MOC algorithm architecture implemented as Simulink models.
a) Complete MOC communications system block diagram including channel modeling.
b) MOC transmitter block diagram. c) MOC receiver block diagram.


a)


b)


c)

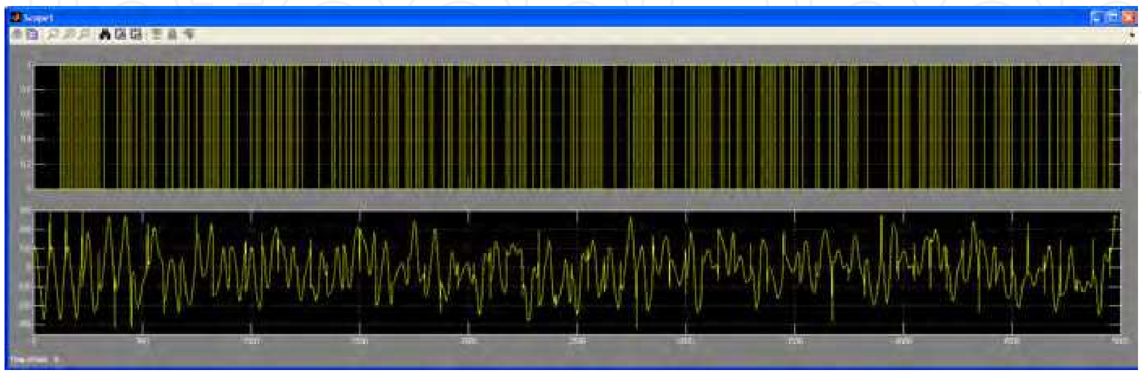
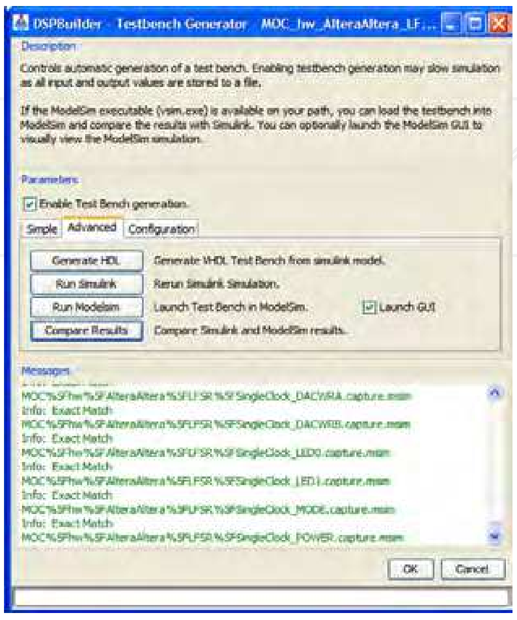

d)

Fig. 4. MOC algorithm transformed to use Altera DSP Builder blocks to automatically generate HDL for FPGA implementation. a) Testbench and interface signals to FPGA. b) Transmitter sub-system. c) Receiver subsystem. d) Simulink simulation waveform.



Controls automatic generation of a test bench. Enabling testbench generation may slow simulation as all input and output values are stored to a file.

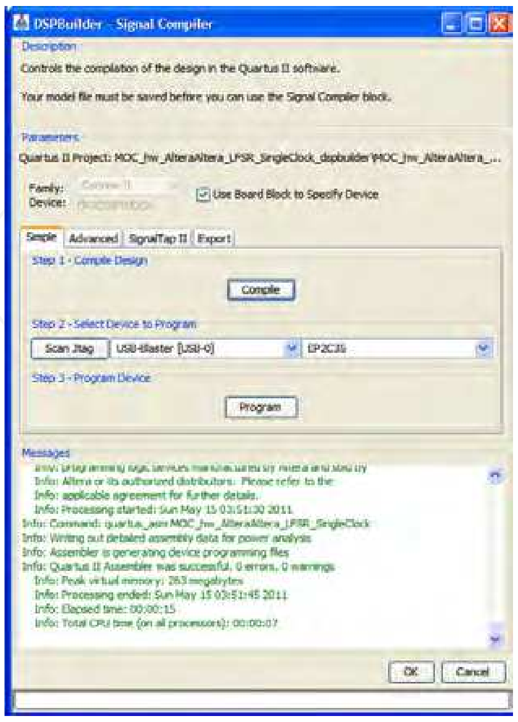
If the ModelSim executable (vsim.exe) is available on your path, you can load the testbench into ModelSim and compare the results with Simulink. You can optionally launch the ModelSim GUI to visually view the ModelSim simulation.

[✓] Enable Test Bench generation.

Generate HDL: Generate VHDL Test Bench from simulink model.
Run Simulink: Rerun Simulink Simulation.
Run ModelSim: Launch Test Bench in ModelSim. [✓] Launch GUI
Compare Results: Compare Simulink and ModelSim results.

```
MOC%5Fhw%5FAlteraAltera%5FLFSR%5FSingleClock_DACWRA.capture.msm
Info:  Exact Match
MOC%5Fhw%5FAlteraAltera%5FLFSR%5FSingleClock_DACWRB.capture.msm
Info:  Exact Match
MOC%5Fhw%5FAlteraAltera%5FLFSR%5FSingleClock_LED0.capture.msm
Info:  Exact Match
MOC%5Fhw%5FAlteraAltera%5FLFSR%5FSingleClock_LED1.capture.msm
Info:  Exact Match
MOC%5Fhw%5FAlteraAltera%5FLFSR%5FSingleClock_MODE.capture.msm
Info:  Exact Match
MOC%5Fhw%5FAlteraAltera%5FLFSR%5FSingleClock_POWER.capture.msm
```

a)



Controls the compilation of the design in the Quartus II software.

Your model file must be saved before you can use the Signal Compiler block.

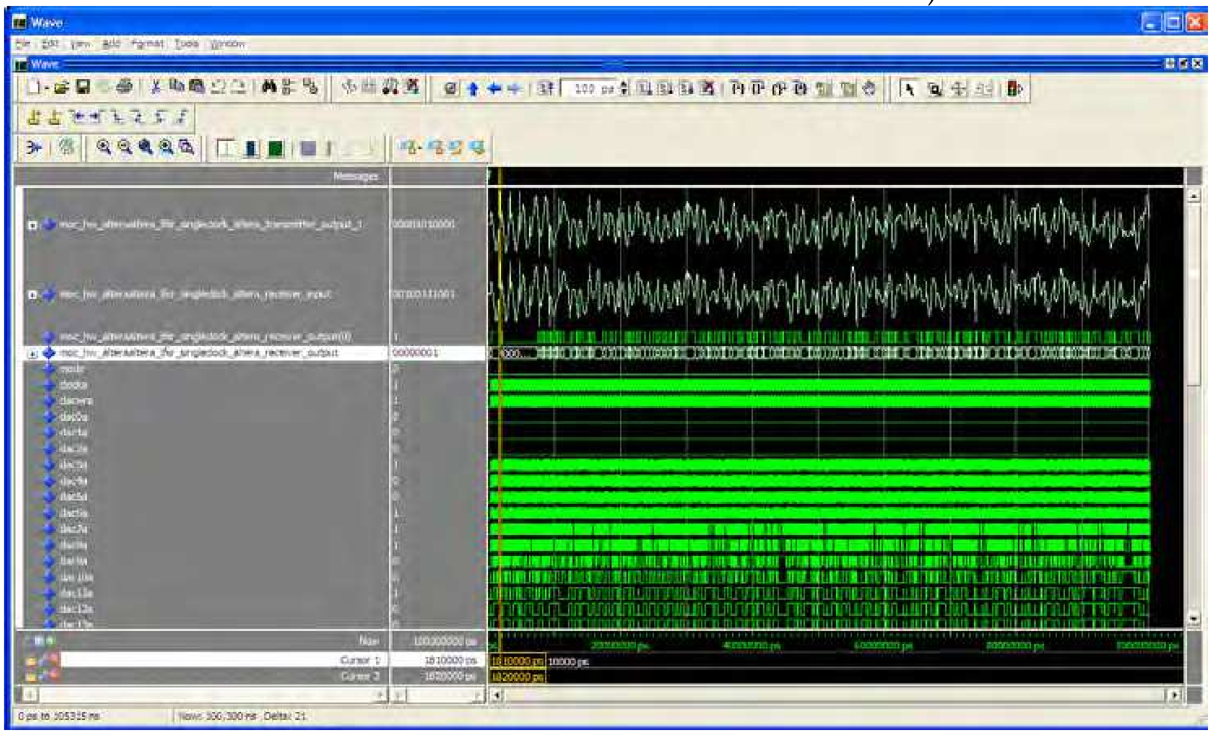
Quartus II Project: MOC_hw_AlteraAltera_LFSR_SingleClock_dspbuilder\MOC_hw_AlteraAltera_...

Family:
Device:
[✓] Use Board Block to Specify Device

USB-Blaster [USB-0]
EP2C35

```
  Info: programming logic devices manufactured by Altera and sold by
  Info: Altera or its authorized distributors.  Please refer to the
  Info: applicable agreement for further details.
  Info: Processing started: Sun May 15 03:51:30 2011
Info: Command: quartus_asm MOC_hw_AlteraAltera_LFSR_SingleClock
Info: Writing out detailed assembly data for power analysis
Info: Assembler is generating device programming files
Info: Quartus II Assembler was successful. 0 errors, 0 warnings
  Info: Peak virtual memory: 263 megabytes
  Info: Processing ended: Sun May 15 03:51:45 2011
  Info: Elapsed time: 00:00:15
  Info: Total CPU time (on all processors): 00:00:07
```

b)



c)

Fig. 5. In addition to a system level simulation within Simulink, it can also control an RTL simulation of the generated HDL code and compare against the system level simulation. a) Test bench generator for RTL simulation. b) RTL HDL simulation of the code generated by DSP Builder. c) Signal compiler for synthesis, place and route, and FPGA programming.

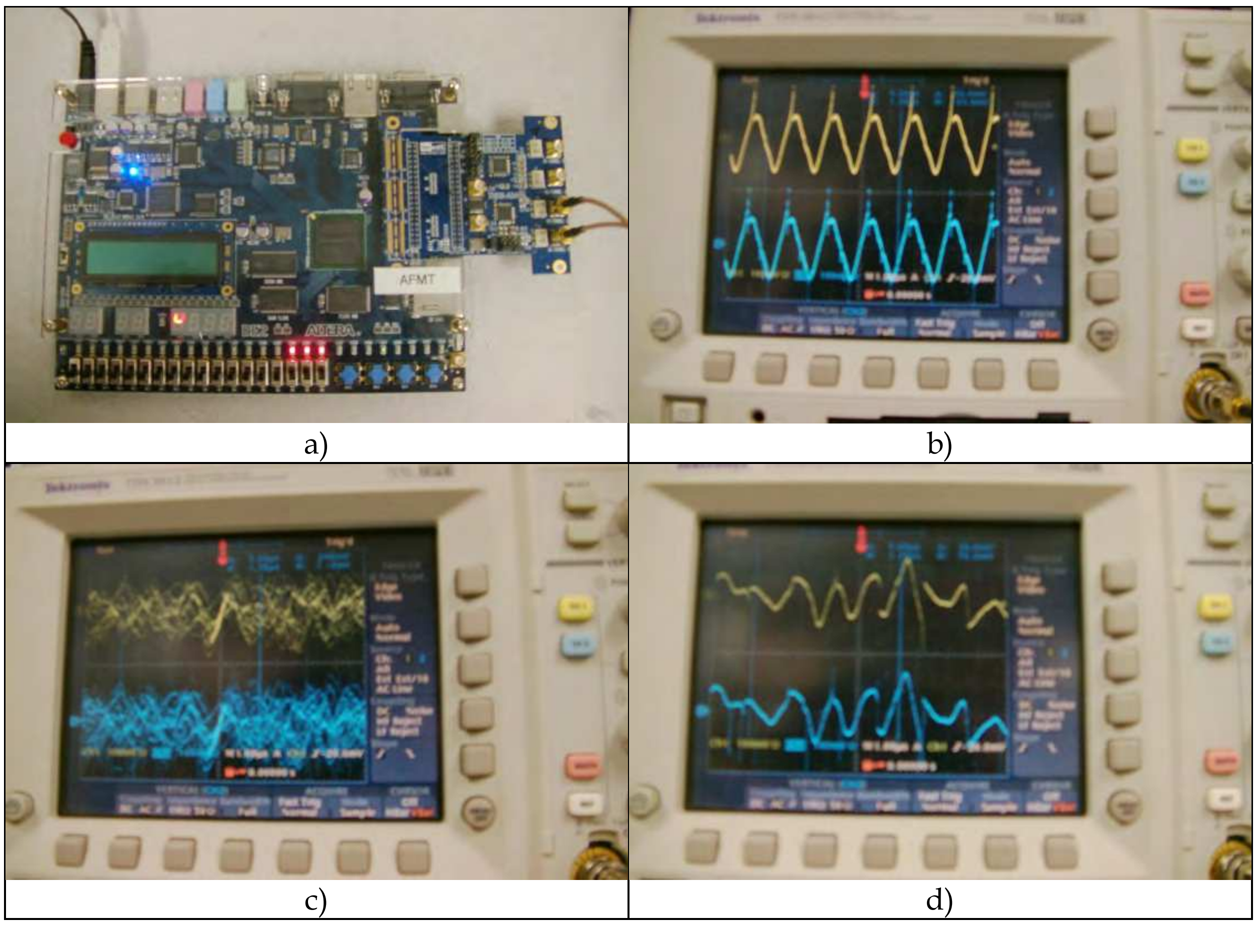


Fig. 6. MOC hardware implementation on an Altera Cyclone II FPGA.
a) Altera DE-2 with daughtercard dual AD channels with 14-bit resolution and data rate up to 65 MSPS and dual DA channels with 14-bit resolution and data rate up to 125 MSPS.
b) MOC modulation output when the input is a stream of constant zeros.
c) MOC modulation output when the input is driven by a PRBN sequence generator.
d) MOC modulation output snapshot when the input is driven by a PRBN sequence generator.

10.2 Improving the performance of DSP systems for MIMO processing

In the paper "Improving the performance of DSP systems for MIMO processing" (Horner, Kwasinski, & Mondragon, 2011), we explored the efficient implementation of select Multiple Input Multiple Output (MIMO) communications algorithms. Two implementation approaches were considered: adding new instructions to the DSP instruction set and adding a hardware accelerator to the DSP system. Of the two approaches, the second was concluded to be best, as it resulted in notable processing speedups and a more efficient use of the computational resources.

While the research into MIMO algorithms have reached levels of development that show important wireless systems performance improvements, the development of DSP systems to implement them has limited the realization of these algorithms to the simplest and least performing ones. This example addresses this technological gap by studying how to design DSP systems to better handle the increased complexity arising from the particular operations typical of MIMO processing algorithms.

Two hardware co-processors were designed, as shown in Figure 8 one for a Householder decomposition algorithm and one for a Greville pseudo inverse algorithm. These hardware co-processors resulted in a simulated speedup of 2.7 for the Greville algorithm and between 4 and 4.7 for the Householder algorithm.

For the design of the hardware accelerator, Synfora's Pico Extreme (acquired recently by Synopsys) ESL tool was used. The author had previous experience with the tool and the task performed for this work was limited to architecture exploration and to find which ASIC implementation would result in the best compromise between throughput, area, power, and easy of interfacing. The algorithms were written in floating point C code and then converted to fixed point C code by evaluating the impact in performance due to the hardware implementation.

Pico Extreme is a very versatile tool since it is structured as a series of logical steps from running an untimed sequential ANSI C program, to single-to-multi-threaded transformations; to hierarchical block-level resource sharing & scheduling; to automatic retiming and pipelining; to performance and throughput analysis; to rapid exploration of performance impacts of loop unrolling, scheduling, and other optimizations; and to RTL verification among others. The flow methodology is shown in Figure 7.

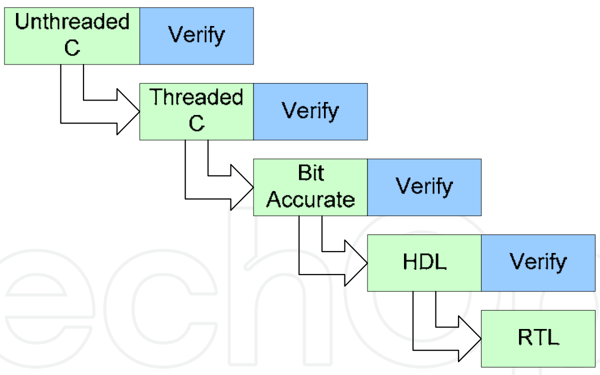


Fig. 7. PICO Extreme design flow.

While this seems to be a dream in which the system designer can implement his design by exploring architectures and trade-offs, then pushing a button and get verified RTL as an output, the reality is that the learning curve of these tools is quite steep and it is not as straight forward as it looks. Even that a very thorough architecture exploration can be performed, the designer still needs to think in terms of hardware when writing the C code to have the same effect as writing in HDL RTL. The C code has to be written in terms of functional units, pipeline stages, memory implementations, operator sharing and general hardware efficiency.

There are two basic methods to specify the design (Synfora, 2009). The number of clock cycles between iteration starts is called II (Initiation Interval) and the number of clock cycles to start all iterations is called MITI (Maximum Inter Task Interval). For this example, MITI can be as small as N*II (where N is the number of loop iterations).

The user is able to provide a target maximum number of clock cycles taken per stage MITI and the tool will select from the library of high-speed components the optimum to achieve higher levels of parallelism at the same time of sharing resources and achieving performance.

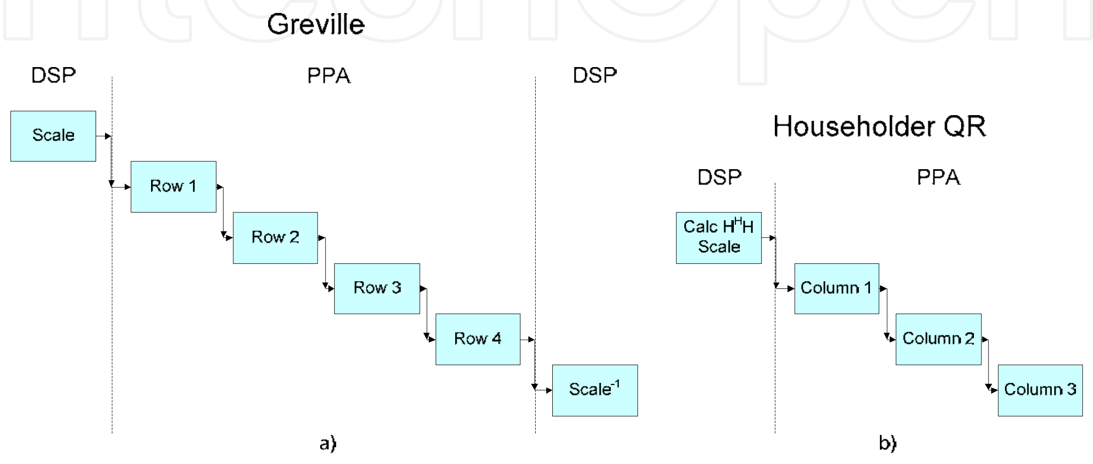


Fig. 8. Processing pipeline for Greville and Householder decomposition methods.

To provide a tradeoff between complexity and speedup, different implementations with different target MITIs were generated. It was noted that as timing constraints tightened, hardware multipliers were switched from two-cycle to one-cycle and the number of multipliers increased to be able to complete complex multiplications (requiring three multiplies) in a single cycle.

MITI timing constraints were used to determine the lowest complexity implementation for each algorithm. The constraints within these ranges of target clock cycles were then used to produce a tradeoff between complexity and resulting speedup. Resulting ranges of targeted number of clock cycles were 230 to 330 for the Householder implementation and 130 to 210 for the Greville implementation.

The resulting speedup was calculated as the ratio of cycles on the DSP-only implementation to the cycles of the DSP-PPA implementation. The resulting silicon area was calculated based on the estimated number of gates given by Pico Extreme and using a characterized CMOS 65nm technology library with an estimate of 854,000 gates per mm^2. This technology was selected, given that is the one in which the DSP was manufactured and can provide an estimate of the growth of the silicon area for the DSP to enable MIMO processing. A plot of speedup vs. complexity for both clocks and both simulators is shown in Figure 9.

The resulting maximum speedups were close to 2.75 for the Greville algorithm and between 4 and 4.7 for the Householder QR decomposition algorithm. This speedup would result in a large reduction (129 µs for the Greville implementation and 521 µs for the Householder implementation) in the amount of time required to compute the channel equalization matrices for an entire OFDM channel in MIMO communication. There is an upper limit to the speedup, however. Because the DSP is still required for some pre-processing operations, there is an asymptotic limit on the actual speedup achieved. Once the PPA unit is able to compute one stage of the processing pipeline in the same amount of time as the software pre-process, there is little added benefit to faster clock or higher complexity. There is also not a major advantage in the 1 GHz clock over the 500 MHz. While the slower clock would require the more complex implementations to compute faster than the DSP software, the savings on power consumption could outweigh the cost of higher complexity.

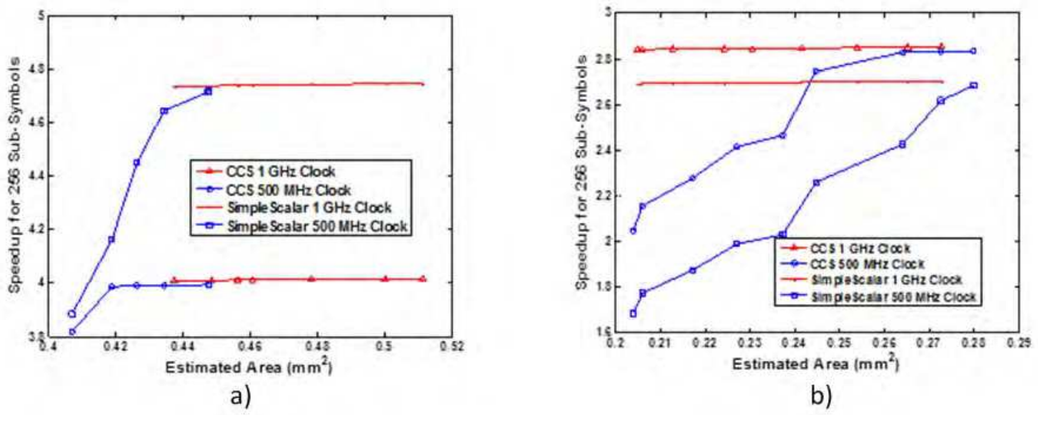


Fig. 9. a) Speedup vs. Complexity for Householder implementation b) Speedup vs. Complexity for Greville implementation.

10.3 OFDM – FFT example

In (Mondragon-Torres, Kommi, & Bhattacharya, 2011), the author proposes the development of an OFDM educational platform that will make use of all the methodologies and tools presented in this chapter with the objective of creating a single system that will allow students to explore different levels of abstraction on hardware design as well as to quantify the effects of the decisions taken on the fixed point precisions as well as all the intermediate signal processing and conditioning through the datapath.

The heart of the OFDM modulation technique lies in the use of the Fast Fourier Transform (FFT), which is a very structured algorithm to convert a time domain signal into the frequency domain and by taking the inverse FFT (IFFT) can be transformed back into the time domain. In Figure 10, a complete digital communication system that employs OFDM modulation is shown (Cho, 2010).

The approach in OFDM systems is to have digital information encoded by traditional phase modulation techniques such as Quadrature Amplitude Modulation (QAM). This modulation technique maps a series of bits into QAM modulated symbols. The number of symbols used for each OFDM frame is traditionally a power of two. Then the IFFT of a block is performed on the frame to convert it back into a time domain representation that can be further processed and sent through the transmitter chain and through the antenna. On the receiver side the process is reversed after frame synchronization by taking the FFT of the received block and obtaining an estimate of the QAM symbols which are mapped back into a series of bits. This sounds pretty straightforward but there are many subtle details that could be investigated in terms of the effects of: quantization, distortion, channel noise, multipath propagation, fading, Doppler shift, synchronization, etc.

A very simple implementation of a 256 point FFT is presented in this section as shown in Figure 12. No architectural decisions were performed and a regular textbook implementation is used just to demonstrate some of the capabilities of CatapultC. In Figure 11, technology parameters and some common definitions are shown as reference for the reader. Based on the above definitions, we started to change the system parameters to get a feel of their implications.

In Figure 13 it is shown how by unrolling and pipelining the input and output operations we can drastically reduce the latency. What is the price for this? Answer: Memory bandwidth. We can observe that the area has been maintained constant and this is due to the fact that no memories have been considered in these solutions.

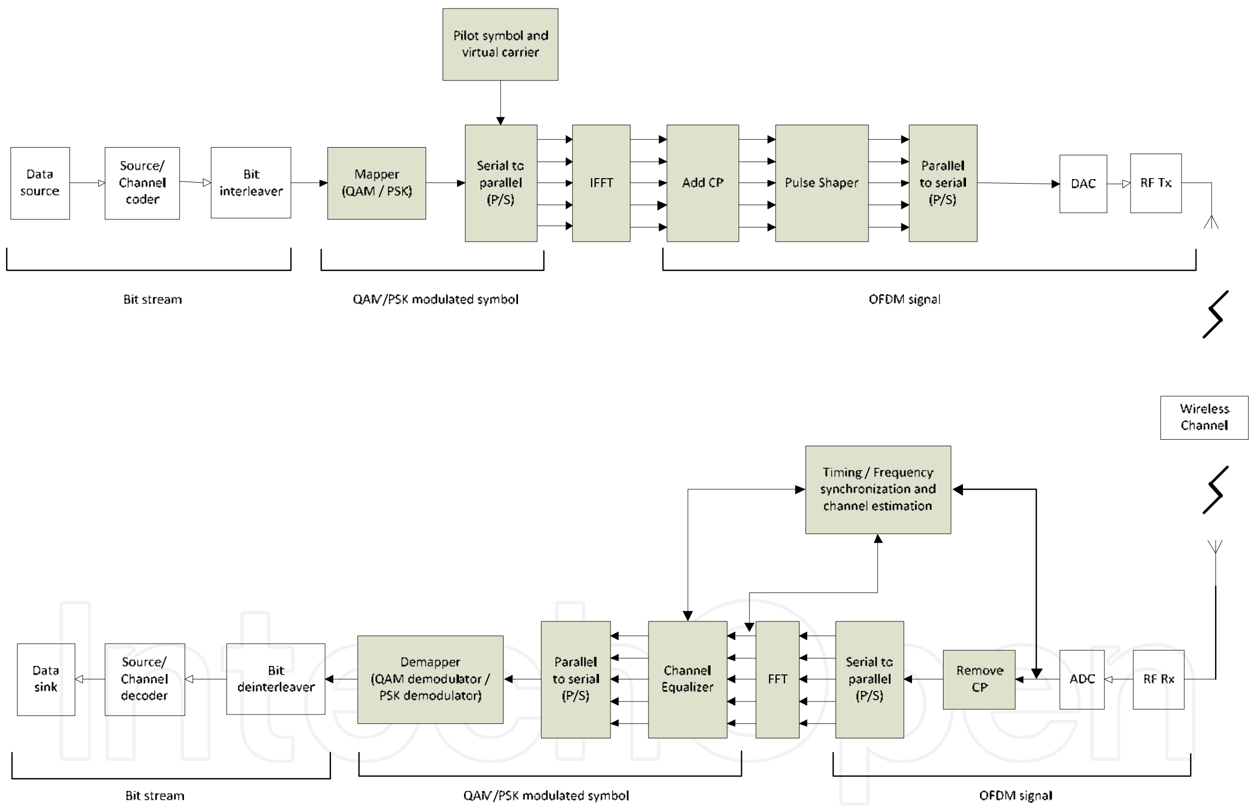


Fig. 10. Digital communications system using OFDM modulation.

Figure 14 and 15 shows the complexity of the solution and we can observe that most of the area is being used in multiplexers to route the signals. On the other hand, more memory will be required for unrolling *printing* and pipelining *reading*. So far we have not touched a single line of code and just by modifying the outer input and output loops we have been able to reduce the latency by 2x at the cost of 2x memory. This is a simple illustration of using the same code to tradeoff performance vs. complexity.

Technology used: Generic CMOS ASIC 90 nm, 200MHz

Definitions

Loop unrolling: Loop unrolling can be used to compute multiple loop iterations in parallel.

Partial unrolling: Computes 'n' copies in parallel

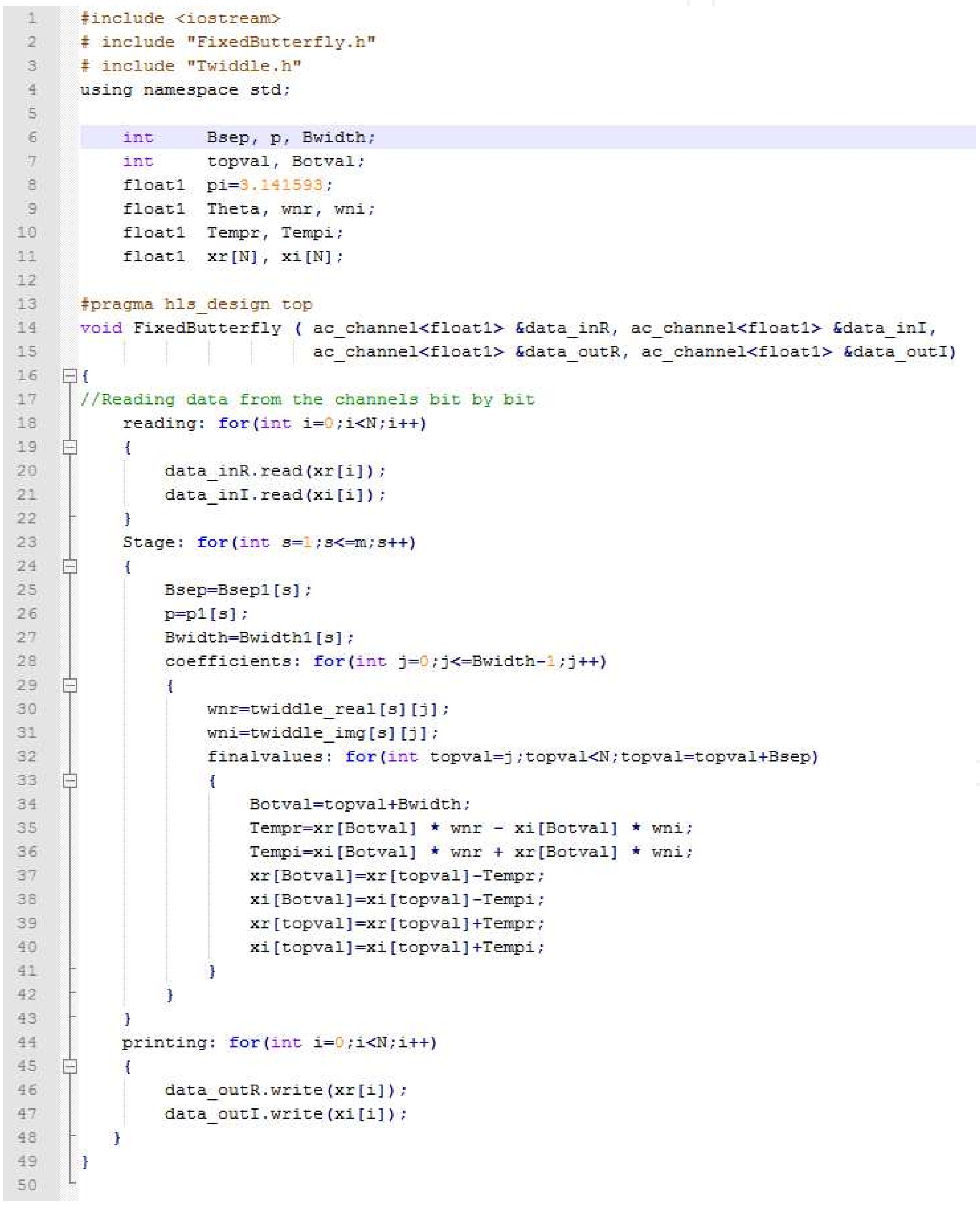
Pipelining: Starts the next loop iteration before the current iteration of the data path contained in the loop has completed

Initial Interval: indicated how often to start a new loop iteration

Latency: Latency refers to the time, in clock cycles, from the first input to the first output

Throughput: Throughput, not to be confused with IO throughput, refers to how often, in clock cycles, a function call can complete.

Fig. 11. Technology used and some common definitions.



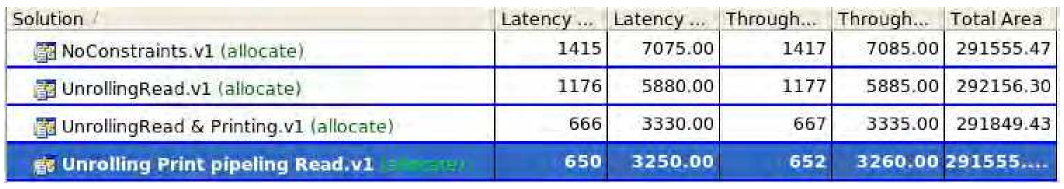
```
#include <iostream>
# include "FixedButterfly.h"
# include "Twiddle.h"
using namespace std;

    int     Bsep, p, Bwidth;
    int     topval, Botval;
    float1  pi=3.141593;
    float1  Theta, wnr, wni;
    float1  Tempr, Tempi;
    float1  xr[N], xi[N];

#pragma hls_design top
void FixedButterfly ( ac_channel<float1> &data_inR, ac_channel<float1> &data_inI,
                      ac_channel<float1> &data_outR, ac_channel<float1> &data_outI)
{
//Reading data from the channels bit by bit
    reading: for(int i=0;i<N;i++)
    {
        data_inR.read(xr[i]);
        data_inI.read(xi[i]);
    }
    Stage: for(int s=1;s<=m;s++)
    {
        Bsep=Bsep1[s];
        p=p1[s];
        Bwidth=Bwidth1[s];
        coefficients: for(int j=0;j<=Bwidth-1;j++)
        {
            wnr=twiddle_real[s][j];
            wni=twiddle_img[s][j];
            finalvalues: for(int topval=j;topval<N;topval=topval+Bsep)
            {
                Botval=topval+Bwidth;
                Tempr=xr[Botval] * wnr - xi[Botval] * wni;
                Tempi=xi[Botval] * wnr + xr[Botval] * wni;
                xr[Botval]=xr[topval]-Tempr;
                xi[Botval]=xi[topval]-Tempi;
                xr[topval]=xr[topval]+Tempr;
                xi[topval]=xi[topval]+Tempi;
            }
        }
    }
    printing: for(int i=0;i<N;i++)
    {
        data_outR.write(xr[i]);
        data_outI.write(xi[i]);
    }
}
```

Fig. 12. Program to compute 256 point FFT.

The FFT algorithm itself has not been optimized due to the data dependency among inner and outer loops. Additional pipe stages will need to be implemented in order to break the loop dependency implicit in the direct implementation of the FFT. This probes the point that there the designer has to guide the tool by writing the C code in such a way that the hardware can be inferred.

Another simple tradeoff was executed by increasing the frequency of operation from 100 MHz to 500 MHz as shown in Figure 16. We can observe that the area remained almost constant, while the latency cycles increased by 3% with respect to the 200 MHz implementation baseline, the latency cycles increased by 19%. We can interpret these numbers as the logic required to implement the FFT had a larger critical path, but since the clock was increased 2.5x, the latency time was reduced by 2.0x demonstrating that there is not a linear relationship between the parameters and depends on the implementation given by the particular constraints.

Talking about power, increasing the frequency by 2.5x will have an impact on the power, but at the same time if it is 2.0x faster, we can think for example on reusing the FFT for some other part of the OFDM processor such as computing the IFFT and FFT using the same hardware and sharing it on the time domain rather than have two cores to perform both operations independently.



Solution /	Latency ...	Latency ...	Through...	Through...	Total Area
NoConstraints.v1 (allocate)	1415	7075.00	1417	7085.00	291555.47
UnrollingRead.v1 (allocate)	1176	5880.00	1177	5885.00	292156.30
UnrollingRead & Printing.v1 (allocate)	666	3330.00	667	3335.00	291849.43
Unrolling Print pipeling Read.v1 (allocate)	**650**	**3250.00**	**652**	**3260.00**	**291555....**

Fig. 13. Different solutions by selecting different architectural constraints.

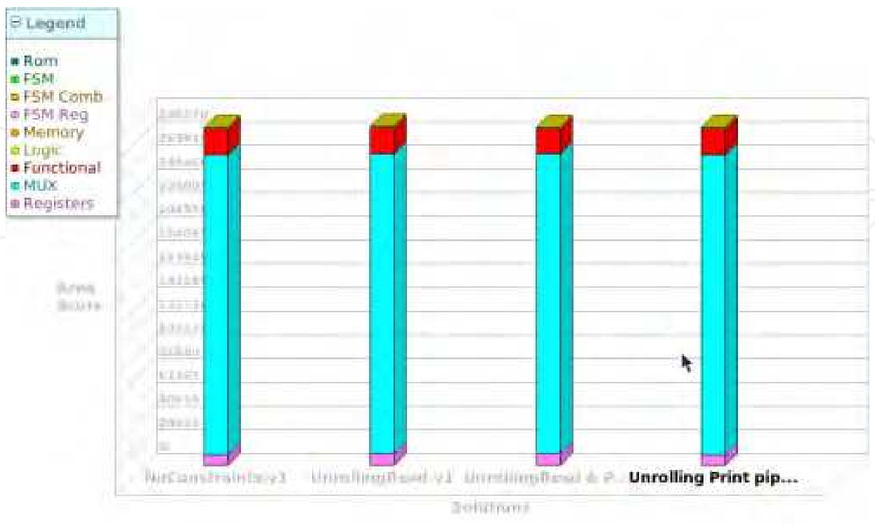


Fig. 14. Graphical view plotting Area.

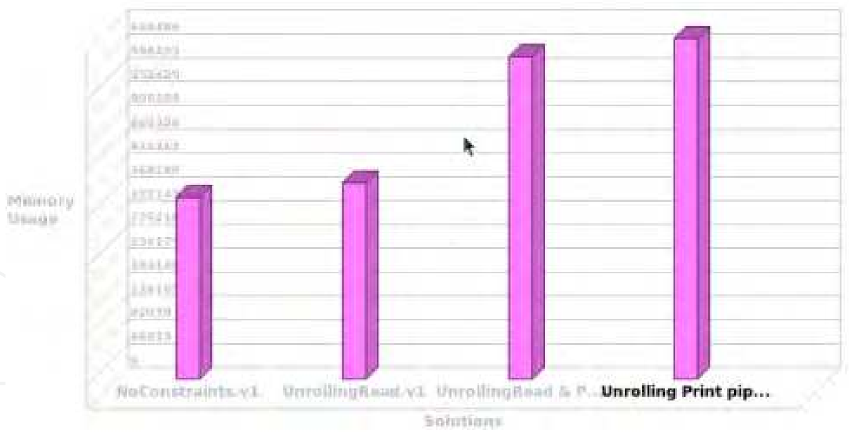
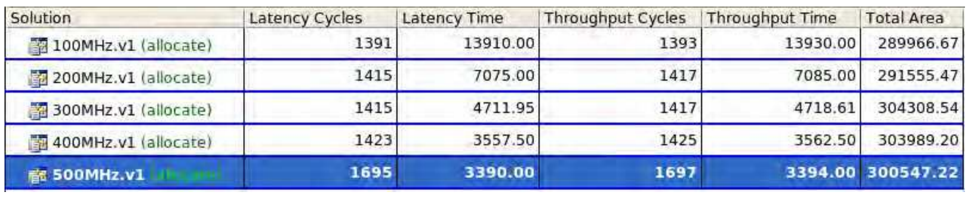


Fig. 15. Graphical View plotting memory usage.



Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Total Area
100MHz.v1 (allocate)	1391	13910.00	1393	13930.00	289966.67
200MHz.v1 (allocate)	1415	7075.00	1417	7085.00	291555.47
300MHz.v1 (allocate)	1415	4711.95	1417	4718.61	304308.54
400MHz.v1 (allocate)	1423	3557.50	1425	3562.50	303989.20
500MHz.v1 (allocate)	1695	3390.00	1697	3394.00	300547.22

Fig. 16. Change in performance with change in frequency.

10.4 Hardware In the Loop (HIL)

Hardware in the loop has become a buzz word when designers want to run their algorithm at full speed or at least hundredths or thousands times faster than an RTL or gate level simulation. In SoCs, simulation can take days, weeks and sometimes months, and that depends on the level of detail that is included in the top level simulation. That is why it is important to be able to replace each block by its behavioural, RTL and gate level models in order to refine the level of simulation control and granularity.

Rather than talking about ASIC emulators that are not traditionally available for small companies or universities, we will take a poor's man approach and show how we can integrate hardware in our computations to able to speed up the testing and processing of algorithms.

Let's take a closer look at the first level of implementation which is generating automatic HDL code from a Simulink model. Each block or a set of few blocks of the entire communication system can be implemented on hardware this was demonstrated in Section 10.1. So far, we have used an Altera Stratix III FPGA to do system level hardware testing of the Fast Fourier Transform block in the OFDM communication model. For this purpose we have used Hardware in Loop (HIL) block provided by the DSP builder Altera library. This block acts as a link between Simulink and the actual hardware we want to configure.

In modern digital communication systems, the current trend is to implement a pipelined FFT to generate orthogonal sub-carriers. A pipelined FFT generate an output every clock cycle which helps in real-time applications like digital communication systems where data is being continuously fed. We have designed Simulink models to implement FFT using butterfly diagrams which use simple Simulink blocks as well as pipelined FFT which use the advanced block set from DSP Builder. In this section we are going to talk more about the pipelined FFT for the above mentioned reasons. For more information on the architecture of the pipelined FFT implemented refer to (Shousheng & Torkelson, 1998).

The hardware implementation was done using the Altera's Quartus II version 10.1 and DSP Builder version 10.1. Care must be taken to properly design a Simulink model which would involve block sets from both advanced and standard block sets of DSP Builder. We created this model in layers. The lower level consists of the device block which has the information about the FPGA available in the hardware platform (Stratix III) and the functional blocks that essentially form the FFT. However, on the top level we could only use the signal and control blocks from the advanced block set and other blocks have to be at the lowest level in the design hierarchy.

We make use of the signal compiler and testbench from the standard block set on the top level. The signal compiler is used for creating a Quartus II project, start synthesis, to launch place and route after generating the HDL code. The testbench is used to compare the block level simulations in Simulink and the HDL simulations using Modelsim. Input and output blocks are inserted before and after the subsystem that contains the advanced block set. These blocks have external type parameters to convert from floating or other format handled by Simulink to fixed point as FPGA implementations can only be configured for fixed point. These blocks act as boundaries to the advanced and basic block sets. The procedure to convert the FFT model to HDL, configure the FPGA with the HDL code, and running it from Simulink is detailed below.

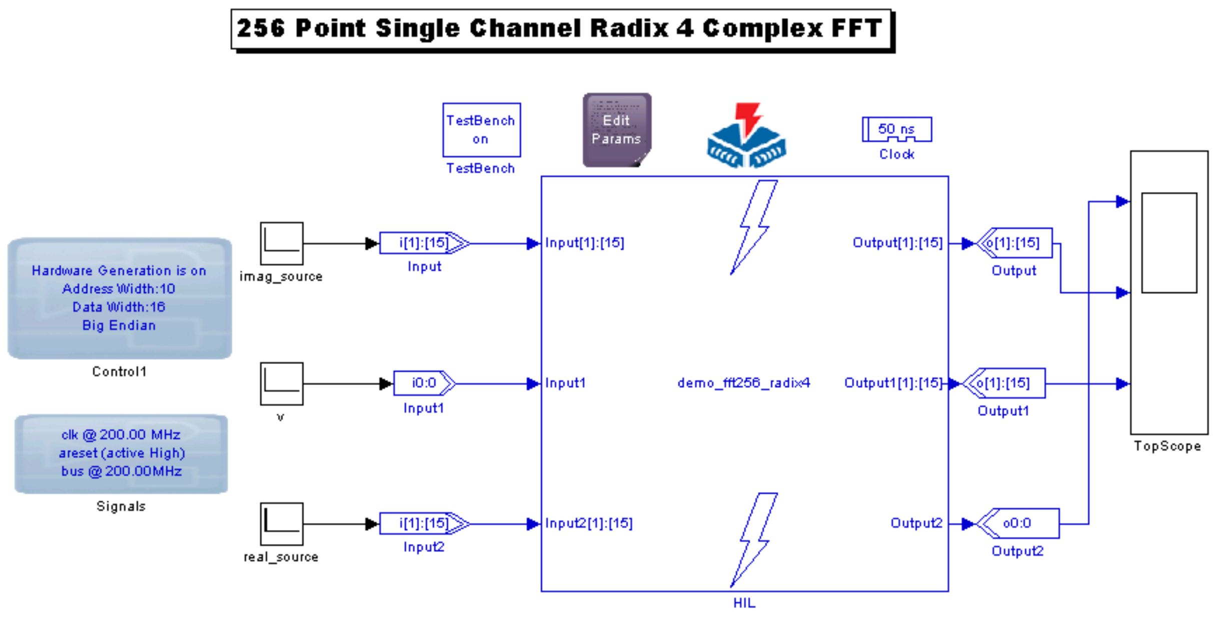


Fig. 17. Hardware In the Loop (HIL) Simulink simulation, actual code runs on the FPGA.

We first run the signal compiler block on the top level to generate HDL code and create a Quartus II project. Then compile the design with Quartus II using the compile option in the signal compiler block. We have now created a Quartus II project for the model and synthesized the HDL code for the same. Now save a copy of this model and instantiate a HIL block on the top layer of the new model from the Altera DSP Builder library found in the standard block set. Open the HIL block and copy the Quartus II project that was earlier created into the file path. This would generate proper ports for the HIL block. Connect these ports to the appropriate signals. Configure the simulation in burst mode to observe high speed of simulation. In the next menu entry of the HIL block, compile the Quartus II project again, scan JTAG in order to recognize the FPGA device and program it. If we simulate this model it runs at a remarkable speed when compared with the native Simulink simulation. Figure 17 above shows the model which has the advanced block set replaced with a HIL block. This example was modified from the one supplied by Altera to run the FFT on the FPGA platform and to be controlled by the Simulink simulation. We are in the process of converting some other algorithms into hardware following the same methodology to be able to create custom hardware acceleration blocks (Altera, 2007).

11. Conclusions

In this chapter we summarized a few of the methodologies, technologies, tools and approaches that can be taken to convert a wireless communications algorithm into a feasible hardware implementation.

While this chapter is far from being a single methodology to be followed when designing for hardware implementation of wireless communication circuits, we explored many of the practical aspects on how to achieve quick results and also to have a baseline where the final design may compare with.

Push button methodologies are still far from being a reality and even that ESL tools can achieve impressive results and can verify all the way from system level down to gate level against a golden model, there is still some reluctance from the backend teams to rely on automatic tools to do the job. While this approach has been done in automatic place and route in digital systems, ESL has been pushed the level of abstraction one level above RTL design.

What are the advantages of ESL system level design? The most valuable for the author is the ability to explore different architectures and the possibility of generating very complex datapath designs easily with simple constraints and with high hardware reusability.

Can a good RTL designer do it better? The answer is yes if he has all the time to select the best architecture for implementation. SoC design methodologoes rely on IP reutilization and to spend the valuable design time just on those blocks that will make the product differentiation.

Due to time to market constraints, design teams cannot spend much time trying to find the best and optimal architecture to implement, sometimes the task are reduced to get the job done on time. One important aspect to remember that most of the products, when the designer announces that the module is ready, it is still no more than 30% of the complete SoC design. Integration, verification & validation, design for testability, design for manufacturability, synthesis, automatic place and route will consume more than 70% of the SoC development time.

Another very important aspect is to be able to run an algorithm on hardware to take advantages of computational speed that for example could be obtained on an FPGA. This is a step required to prove if an algorithm is robust enough. ASIC technologies cannot be verified using FPGAs, but at least system level emulation can be performed to verify interconnectivity and overall signal flow.

12. Acknowledgements

There are many people that contributed directly and indirectly to the contents of this chapter with their algorithms, ideas for implementation, hard work and enthusiasm. I would like to recognize the following individuals and organizations that contributed in the following areas:

Name	**Project**
Dr. Chance Glenn Sr.	MOC Digital communications System Implementation
Padma Ragam	MOC Digital communications System Implementation
Nathaniel Horner	Improving the performance of DSP systems for MIMO processing
Dr. Andres Kwasinski	Improving the performance of DSP systems for MIMO processing
Mahesh Nandan Kommi	OFDM – FFT Hardware in the Loop (HIL)
Department of Electrical, Computer and Telecommunications Engineering Technology	Publishing funds.

13. References

Abdel-Hamid, E. M., Fahmy, H. A. H., Khairy, M. M., & Shalash, A. F. (2011, 15-18 May 2011). *Memory conflict analysis for a multi-standard, reconfigurable turbo decoder.* Paper presented at the Circuits and Systems (ISCAS), 2011 IEEE International Symposium on.

Accellera. (2011). UVM World: Universal Verification Methodology, 2011, from http://uvmworld.org/

Agilent. (2011). SystemVue ESL Software | Agilent, 2011, from http://www.home.agilent.com/agilent/product.jspx?cc=US&lc=eng&ckey=1297131&nid=-34264.0.00&id=1297131

Altera. (2007). An OFDM FFT Kernel for Wireless Applications (Vol. AN-452).

Altera. (2011a). Digital Signal Processing, 2011, from http://www.altera.com/products/software/products/dsp/dsp-builder.html

Altera. (2011b). Nios II C-to-Hardware Acceleration Compiler, 2011, from http://www.altera.com/devices/processor/nios2/tools/c2h/ni2-c2h.html

ARM. (2011). CoreLink System IP & Design Tools for AMBA - ARM, 2011, from http://www.arm.com/products/system-ip/amba/index.php

Ascent, S. (Ed.). (2010). *FPGAs for DSP and Communications Course Notes, UCLA Extension,* January 24-27, 2011 *Course Notes*.

Bluespec. (2011). Bluespec, Inc., 2011, from http://www.bluespec.com/

Borrayo-Sandoval, H., Parra-Michel, R., Gonzalez-Perez, L. F., Printzen, F. L., & Feregrino-Uribe, C. (2009, 9-11 Dec. 2009). *Design and Implementation of a Configurable Interleaver/Deinterleaver for Turbo Codes in 3GPP Standard.* Paper presented at the

Reconfigurable Computing and FPGAs, 2009. ReConFig '09. International Conference on.

Cadence. (2011). OVM-based verification flow 2011, from http://www.cadence.com/products/fv/pages/ovm_flow.aspx

Chandrakasan, A., & Brodersen, R. (1998). *Low power CMOS design / edited by Anantha Chandrakasan, Robert Brodersen*: Piscataway, NJ IEEE Press, 1998.

Cho, Y. S. (2010). *MIMO-OFDM wireless communications with MATLAB*: Singapore ; Hoboken, NJ : IEEE Press : J. Wiley & Sons (Asia), c2010.

Frazer, R. (2088). Reducing Power in Embedded Systems by Adding Hardware Accelerators, 2011, from http://www.eetimes.com/design/embedded/4007550/Reducing-Power-in-Embedded-Systems-by-Adding-Hardware-Accelerators

Glenn, C. M. (2009). MOC Technical brief (ECTET, Trans.). Rochester, NY: Rochester Institute of Technology.

Horner, N., Kwasinski, A., & Mondragon, A. (2011, 22-27 May 2011). *Improving the performance of DSP systems for MIMO processing.* Paper presented at the Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on.

Ifeachor, E. C. (1993). *Digital signal processing : a practical approach*. Wokingham, England ; Reading, Mass. :: Addison-Wesley.

IT++. (2011). Welcome to IT++! , 2011, from http://itpp.sourceforge.net/devel/index.html

Mathworks. (2011). MathWorks - MATLAB and Simulink for Technical Computing, 2011, from http://www.mathworks.com/

Mentor-Graphics. (2011). Algorithmic C Datatypes - Mentor Graphics, 2011, from http://www.mentor.com/esl/catapult/algorithmic

MentorGraphics. (2011). Catapult C Synthesis Overview - Mentor Graphics, 2011, from http://www.mentor.com/esl/catapult/overview

Mondragon-Torres, A. F., Kommi, M. N., & Bhattacharya, T. (2011). *Orthogonal Frequency Division Multiplexing (OFDM) Development and Teaching Platform*. Paper presented at the 2011 Annual Conference & Exposition, Vancouver, BC, CANADA. http://www.asee.org/search/proceedings?fields%5B%5D=title&fields%5B%5D=author&fields%5B%5D=session_title&fields%5B%5D=conference&fields%5B%5D=year&search=mondragon-torres&commit=Search

NI. (2011). NI LabVIEW FPGA - National Instruments, 2011, from http://www.ni.com/fpga/

OCP. (2011). OCP-IP : Home Page, from http://www.ocpip.org/

Rappaport, T. S. (2001). *Wireless communications : principles and practice* (2nd ed ed.). Upper Saddle River, N.J. : London :: Prentice Hall PTR.

Sanchez-Sinencio, E., & Andreou, A. (1999). *Low-Voltage/Low-Power Integrated Circuits and Systems: Low-Voltage Mixed-Signal Circuits, Wiley-IEEE Press,* January 1999

Shousheng, H., & Torkelson, M. (1998, 11-14 May 1998). *Design and implementation of a 1024-point pipeline FFT processor.* Paper presented at the Custom Integrated Circuits Conference, 1998. Proceedings of the IEEE 1998.

Synfora. (2009). *PICO USER MANUAL - Writing C Applications: Developer's Guide.* (PE-ASIC-UM-WCADG-VER 09.03-6). Mountain View, CA.

Synopsys. (2011a). Signal-Processing, 2011, from http://www.synopsys.com/systems/blockdesign/digitalsignalprocessing/pages/signal-processing.aspx

Synopsys. (2011b). Synphony C Compiler, 2011, from http://www.synopsys.com/Systems/BlockDesign/HLS/Pages/SynphonyC-Compiler.aspx

Synopsys. (2011c). Verification Methodology Manual for SystemVerilog, 2011, from http://vmm-sv.org/

SystemC. (2011). Home - Open SystemC Initiative (OSCI), 2011, from http://www.systemc.org/home/

Texas-Instruments. (2011). TMS320TCI6482 Fixed Point Digital Signal Processor, 2011, from http://www.ti.com/product/tms320tci6482

Xilinx. (2011a). AutoESL High-Level Synthesis Tool, 2011, from http://www.xilinx.com/tools/autoesl.htm

Xilinx. (2011b). System Generator for DSP, 2011, from http://www.xilinx.com/tools/sysgen.htm

Yang, S., Yuming, Z., Goel, M., & Cavallaro, J. R. (2008, 2-4 July 2008). *Configurable and scalable high throughput turbo decoder architecture for multiple 4G wireless standards.* Paper presented at the Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008. International Conference on.

Yin, H., & Alamouti, S. (2006, 27-28 March 2006). *OFDMA: A Broadband Wireless Access Technology*. Paper presented at the Sarnoff Symposium, 2006 IEEE.