FPGAs: RE-INVENTING THE SIGNAL PROCESSOR

Chris Dick
Xilinx Inc.

ABSTRACT

FPGAs are increasingly being employed for building real-time signal processing systems. They have been used extensively for implementing the PHY in software radio architectures. This paper provides a technology and market perspective on the use FPGAs for signal processing and demonstrates FPGA DSP using an adaptive channel equalizer case study.

KEY WORDS

Software radio, FPGAs, Adaptive Filter, Channel Equalizer, System-on-Chip, Simulink

INTRODUCTION

Software defined radios (SDR) are highly configurable hardware platforms that provide the technology for realizing the rapidly expanding third (and future) generation digital wireless communication infrastructure. Many sophisticated signal processing tasks are performed in a SDR, including advanced compression algorithms, power control, channel estimation, equalization, forward error control, adaptive antennas, rake processing in a WCDMA (wideband code division multiple access) system and protocol management. While there are a plethora of silicon alternatives available for implementing the various functions in a SDR, field programmable gate arrays (FPGAs) are an attractive option for many of these tasks for reasons of performance, power consumption and configurability. This paper provides a high-level perspective on the role of FPGAs in signal processing systems. A design example in the context of an adaptive channel equalizer is presented. The implementation is achieved using a system level development tool: *System Generator*® for DSP from Xilinx.

THE FPGA SIGNAL PROCESSOR

The traditional technology choice for real-time signal processing has been DSP microprocessors, ASSPs (application specific standard parts) or custom ASIC (application specific integrated circuit) solutions. First some observations. Semiconductor process technology continues to advance according to Moore's Law and device geometries continue to shrink - this is likely to be the situation for the next 15+ years. As highlighted by Bass and Christensen [1], this has produced the interesting situation whereby semiconductor fabrication facilities now offer circuit design teams more

transistors than they need. This so called design gap has been widening for some time. In fact, the National Technology Roadmap for Semiconductors noted it 5 years ago, observing that while the number of transistors that could be put on a die was increasing at a rate of about 60 percent a year, the number of transistors that circuit designers could design into new independent circuits was going up at only 20 percent a year [1]. This trend is observed in the DSP processor space where we note that even high-end DSP microprocessors do not push the transistor densities described by Moore's Law. Why is this the case? The author believes the answer to this question has its roots in the computing paradigm and computing architectures on which these devices are based – the von Neuman machine. When von Nueman and his colleagues were developing their early computers, the design optimization criterion was different and the base technology used to construct these machines was very different to that available today. In fact, the machines where constructed using heterogeneous materials for each major subsystem. The storage devices were wire and magnets (relays), or even mercury (acoustic wave based mercury delay lines). The actual computing components were constructed out of glass and electric fields guiding electron beams (vacuum tubes). The very diversity of the materials required the physical separation of the various subsystems.

Today we construct silicon based computing machines. From a materials perspective they are homogeneous systems in which the memory and computing components are constructed from the same material – silicon. Yet much of the time the architecture developed by von Neuman still persists, when in fact it need not. Given a large transistor budget it is difficult to effectively use his resource to evolve a DSP microprocessor along its traditional trajectory. How many multiply-accumulate (MAC) functional units can be incorporated into an instruction set architecture (ISA) computing machine without introducing significant issues with functional unit scheduling and compiler writing?

To bring value to a state-of-the-art semiconductor product the transistor budget must be used in a different way, and this is precisely what the FPGA does. As highlighted in [1], while price and performance are still key metrics valued in the market, there are signs that a seismic shift is occurring, giving way to a new era in which customization matters more. FPGAs are the ultimate in customization, signal processing systems can be constructed that are limited only by the imagination of the designer. FPGAs spend the transistor budget in a fundamentally different way than ISA machines. To see this consider the architecture of the Xilinx Virtex-II^{TM} [2] FPGA shown in Figure 1. The device is organized as an array of logic elements and programmable routing resources used to provide the connectivity between the logic elements, FPGA I/O pins and other resources such as on-chip memory, delay lock loops, digital clock managers, and embedded hardware multipliers. A non-trivial number of transistors is used to support the programmable routing resources. However, having invested part of our transistor budget in this manner we have a highly configurable part that can be used to construct customized, and yet configurable, datapaths to solve the problem at hand. The FPGA approach to computing is like having a miniature silicon foundry on at your disposal with a turn around time of hours instead of months or even years as it is for many complex ASICs. The device personality is held in static RAM (SRAM) so that modifications, extensions and bug fixes can be easily applied – even after the system has been deployed.

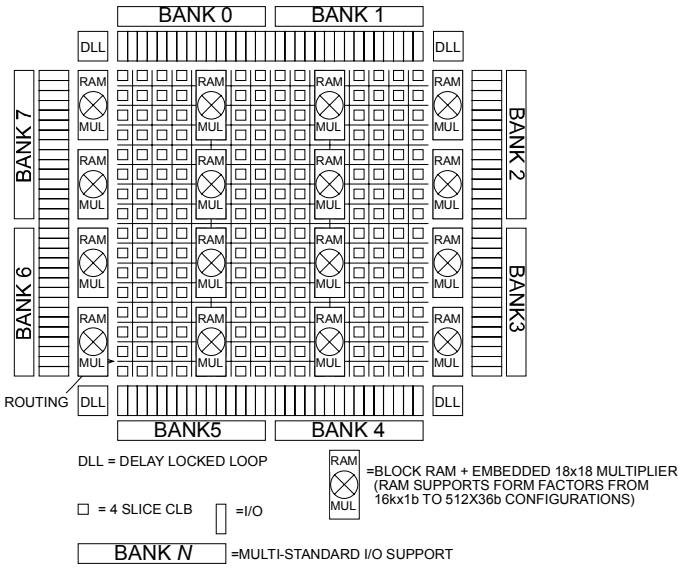

Figure 1: Virtex-II FPGA architecture. This FPGA family provides an array of 18x18-bit (168 in the XC2V8000) precision multipliers for addressing advanced signal processing applications.

Even though we have a large transistor budget at our disposal, in the spirit of efficiency, it is still worth considering how certain commonly used functions could be integrated into an FPGA. This is not an easy task and leads to what I like to refer as *The Integrator's Dilemma* (Figure 2). This figure is a representation of level-of-integration versus application space. Region 1 is illustrative of ASIC *system-on-a-chip* (SoC) based design. As more specialized functions are integrated into the device, the coverage in application space is reduced until we are at the apex of the region which states that this chip is only good for a single application. Region 2 describes the situation for FPGAs. While FPGAs will have the same trend in the figure as an ASIC, the gradient is much reduced, which leads to a broadening of the single application coordinate for the ASIC. Consider the Xilinx Virtex series of FPGAs as an example. Virtex-I devices consisted primarily of logic fabric, memory, clock management and I/O. Virtex-II, the evolution of Virtex-I, included the addition of custom 18bx18b precision multipliers. In the context of Figure 2, this is reflected as movement up the *level-of-integration* axis. The Virtex-II Pro family is the extension of Virtex-II technology. Virtex-II Pro extended Virtex-II technology by incorporating a Power PC 405 processor (up to 4 in the largest family member) and multi-gigabit transceivers (MGTs). This evolution is movement further along the *level-of-integration* axis. However, the configurable dimension of the technology has been preserved, and custom datapaths for implementing DSP or processing the data being presented to or received from the MGTs can be constructed in the logic fabric on an application-by-application basis.

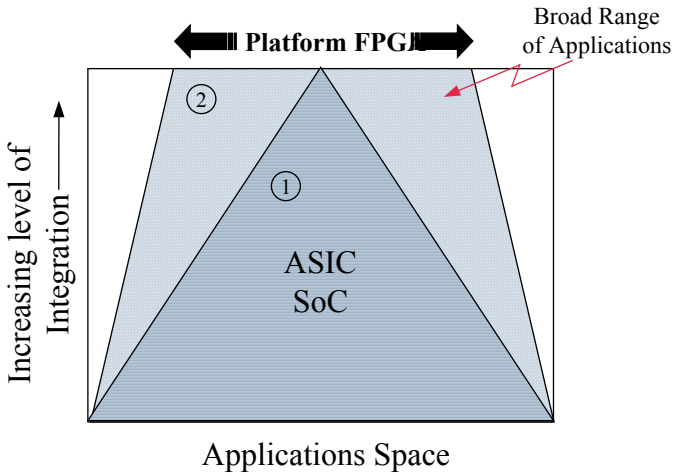

Figure 2: The integrator's dilemma.

The silicon system that has evolved enables complex systems to be constructed, frequently using a single device. This is the *platform based* approach to system realization.

The question of economic viability is frequently raised in the context of deploying FPGAs in end products. One use model that is referenced is the application of FPGAs for prototyping and initial product deployment, followed by migration to an ASIC for high volume production. Let us examine this model in the context of current market and technology trends. First, as highlighted by the noted Harvard economist Clayton Christensen in [1], there are fundamental changes occurring in the market place. Once performance requirements have been satisfied in a particular application domain, in order to remain viable and competitive, the product emphasis must evolve. What we are seeing now is the market favoring customized SoC solutions that retain flexibility. Another market related trend is the increasing emphasis on time-to-market. On the technology front we are still experiencing growth according to Moore's Law. For the ASIC designer this is an issue. One component of this is deciding where to spend the transistor budget, the second is related to the engineering effort required to bring a state-of-the-art ASIC to market. The design methodologies required to design, implement and verify a state-of-the-art ASIC are not tracking Moore's Law. This situation is reflected in the lengthy development cycles for bleeding edge designs, which is of course in direct contradiction to the ever increasing focus and importance on time-to-market and being first to market. Figure 3 is an attempt to capture some of these observations. There are two important components to the curves: an initial offset and a gradient. As process geometries shrink the non-recurring engineering (NRE) costs for an ASIC development increase. For example, the NRE on a 0.15 micron process is much greater than that for a 0.25 micron development. The NRE costs to an FPGA user are much smaller, consisting essentially of a modest investment in software. Hence, the FPGA offset on the vertical axis in the figure are well below that of an ASIC, independent of the process. The gradient of the cumulative NRE plus unit cost as a function of volume curve for the ASIC is shallower than the equivalent curve for the FPGA. However, as semiconductor manufacturing

process improves, the intersection of the ASIC and FPGA curves moves to the right, indicating that over time, the FPGA solution only becomes more effective. This later trend is only emphasized when the cost of being late to market is factored into the calculation.

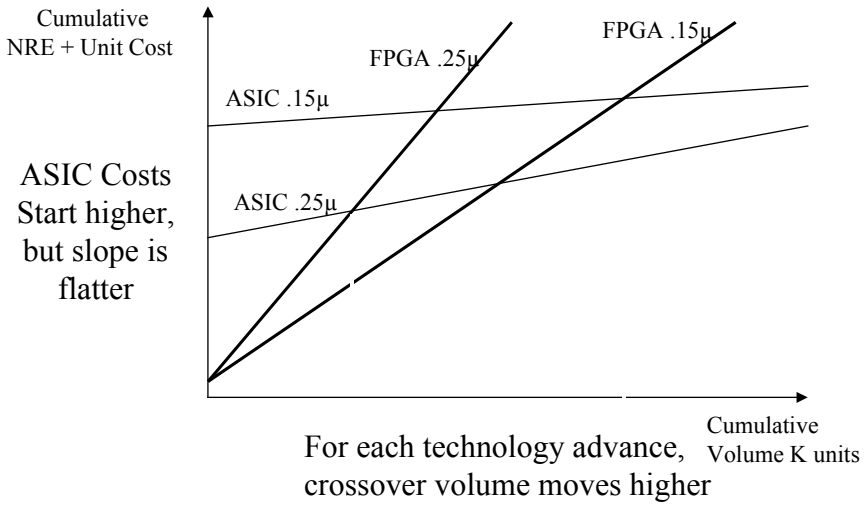

Figure 3: Economic trends over time for ASICs and FPGAs.

FPGAs provide high-performance signal processing using parallel processing – typically data parallel processing. A key feature of the FPGA DSP engine is that the user can define the amount of parallelism employed in the implementation. That is, the FPGA affords the opportunity to *right-size* the datapath – allocate only enough silicon to satisfy the computation requirements. Figure 4 tries to capture this concept, by showing a three-dimensional representation of system design space. Generally, increasing signal processing complexity will require an increase in the computation power (MIPs) of a DSP processor. Once the processor is 100% loaded, there is no further opportunity to increase signal processing complexity. In a 3G or 4G wireless system it is relatively easy to reach the computation boundary of a state-of-the-art processor. In an FPGA design, the systems engineer has a new dimension to allocate to the problem – *space* or FPGA area. More logic resources can be utilized to meet the performance requirements.
Similarly, when we move along the performance axis, and this might equate to bit-rate as measured in millions-of-bits-per-second in many systems, the computational demands will increase. With an FPGA-based design, logic resources can be added to meet these demands. In short, the signal processing engineer utilizing FPGA technology essentially has a design surface on which to work, rather than a single line in the *Performance Vs Signal Processing Complexity* plane of Figure 4.

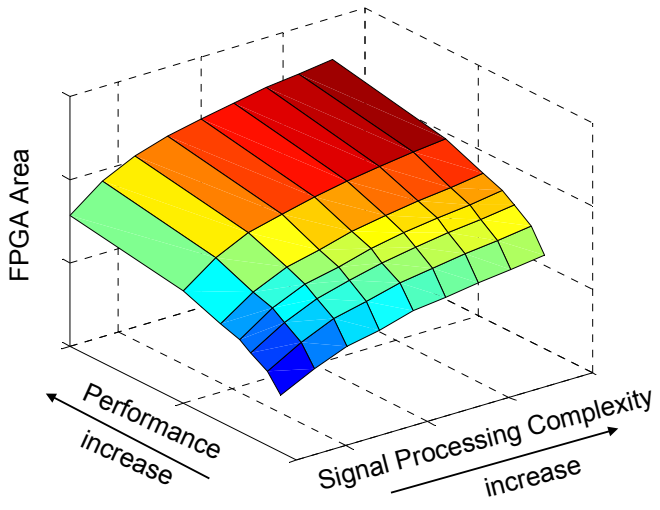

Figure 4: DSP design 3-space showing the dimensions of algorithm complexity, performance and silicon (FPGA resource) area.

DESIGN EXAMPLE: ADAPTIVE EQUALIZER

This case study considers the FPGA implementation of an adaptive equalizer for a 50 Mbps 16-QAM demodulator. The example demonstrates the FPGA implementation of one component on a soft radio system. Software and intellectual property (IP) is becoming *harder* than the supporting hardware platforms. In this example we emphasize the use of a new design methodology for FPGA DSP systems called *System Generator for DSP*^{TM} [4].

Adaptive equalizers operate in a receiver to minimize intersymbol interference (ISI) due to channel-induced distortion of the received signal. The equalizer operates in cascade with a matched filter (MF), synchronous sampler, and decision device (slicer) operating at the symbol rate. A gradient descent process such as the least-mean square (LMS) algorithm [3] adjusts the equalizer weights to minimize the difference between the input and output of the decision device. In modern receivers the sampling process precedes the matched filter, and in order to satisfy the Nyquist criterion for the matched filter, the sample rate is greater than the symbol rate by a ratio of small integers p-to-q such as 3-to-2 or 4-to-3 and often is 2-to-1 to simplify the subsequent task of down sampling prior to the slicer. If the down sampling occurs prior to the equalizer, the equalizer operates at 1-sample per symbol and it is termed a symbol-rate equalizer, and if the down sampling occurs after the equalizer, the equalizer operates on p/q-samples per symbol and it is termed a fractionally-spaced equalizer (FSE).

In the case of an adaptive decision directed (DD) channel equalizer the desired signal is produced by utilizing the known structure of the system alphabet. In this design example a fractionally spaced equalizer operating at two samples per symbol will be considered.

The equalizer result is of course generated at the baud rate *T*. The equalizer is a multirate structure that is most efficiently implemented using a polyphase decimator architecture.

An 8-tap equalizer is to be employed in our system. Each polyphase segment will comprise a 4-tap filter. Each symbol in the 16-QAM alphabet carries 4 bits of information. To achieve the required 50 Mbps data rate the symbol rate must be 12.5 Mbaud. Each polyphase segment in the equalizer will operate at the low output symbol rate in contrast to the higher input sample rate. That is, the equalizer must generate a new output at a sample rate of 12.5 MHz. We will assume that the coefficients need to be updated at the symbol rate. The equalizer architecture must now be defined. There are many options. For example, a fully parallel design consisting of 8 FIR processing elements (PE) and 8 LMS processors could be employed. In this case the system would only be required to support a clock frequency of 12.5 MHz. Since this is a very modest operating point for current generation FPGAs, a folded design is considered that runs at a higher clock rate and uses only a small number of functional units to service all of the operations. This in turn minimizes the FPGA device utilization. The performance objective can be achieved using a single FIR and LMS PE in each of the two polyphase arms. This only requires a clock frequency of 50 MHz. In each polyphase segment, the four filter operations will be scheduled onto a single complex multiply-accumulate (MAC) engine. Similarly, the 4 coefficient updates will be folded onto a single LMS PE.
The System Generator implementation of the equalizer is shown in Figure 5.

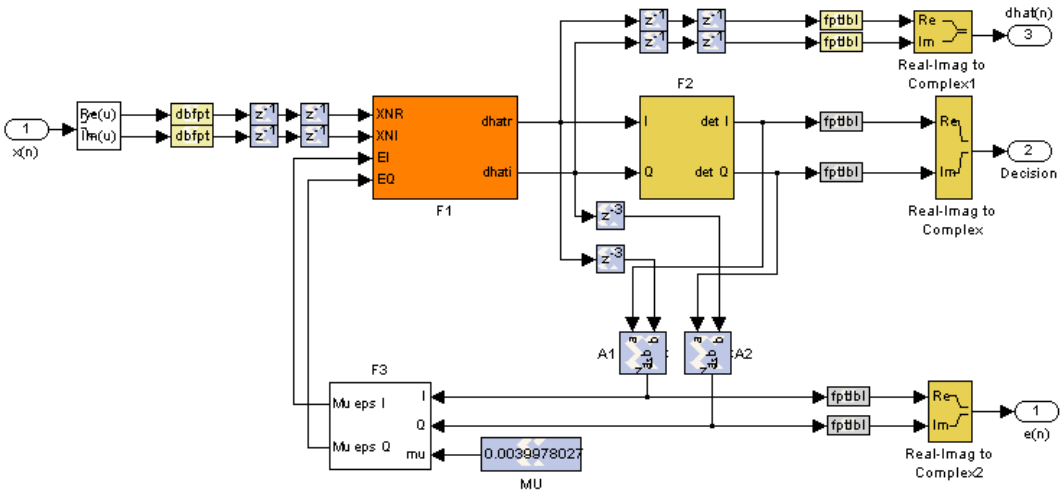

Figure 5: System Generator implementation of the adaptive equalizer. Top level design.

Subsystem *F*1 contains the polyphase filter (Figure 6) and LMS processors.

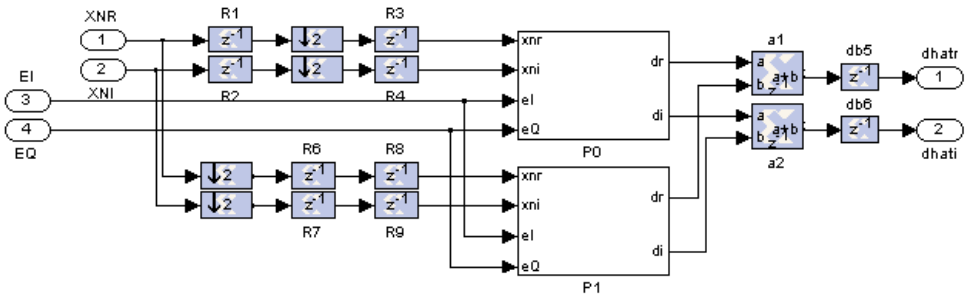

Figure 6: System Generator implementation of the equalizer polyphase filter. Input samples are presented at a rate of 2 samples/symbol while the output samples are generated at the baud rate *T*.

*F*2 is the symbol de-mapping circuit consisting primarily of several comparators and a small look-up table. The error signal is generated using the complex subtractor comprising *A*0 and *A*1. *F*3 weights the error signal by the adaptation rate constant. The heart of both the FIR and LMS PE is a complex MAC. This was implemented using 4 multipliers and 2 additions. The multipliers are realized using the embedded multipliers in the Virtex-II (and Virtex-II Pro) FPGA family.

What may not be obvious from the previous description and the associated figures is that the FIR and LMS PEs each have their own local copy of the complex channel data. Why has this architectural choice been made when the filter computation and the LMS coefficient update process each use the same data [3]? The reason is hardware performance related, and in particular associated with the desire to pipeline the design to achieve a suitable clock frequency. The memory systems in the FIR and LMS processors are skewed in time with respect to each other. This effectively decouples the processes and permits the filter and LMS engines to be heavily pipelined. In looking ahead to the integration of the equalizer and carrier recovery loop, the symbol decisions required to drive the coefficient adjustment will no longer be derived from the detector within the equalizer, but from the carrier recovery loop. The relative timing between the decisions and the regressor vector samples needs to be handled carefully, and the decoupled memory architecture meets this objective.

The equalizer was tested using an exponentially weighted multipath channel model. A 16-QAM alphabet was generated, waveform encoded and upsampled using an interpolation factor of 4 before being applied to the channel. The channel output samples were re-timed, done manually in this case since timing recovery is not in the design under consideration, decimated by 2 and presented at a rate of 2 samples/symbol to the equalizer.

The next question that arises is related to quantizing the design, or defining suitable bit-precisions for the various nodes in the circuit. Because System Generator employs the Simulink® kernel and has access to all of the flexibility offered by Matlab, it is straightforward to establish a test and verification procedure that permits rapid design exploration. In the System Generator design, the data format of each functional unit is specified symbolically. The references are resolved by executing a Matlab script that allocates workspace variables. Each equalizer customization is defined by unique script.

When the simulation is run, various signals are logged to the workspace and used in post-simulation analysis. Figure 7 shows the received signal with ISI, the instantaneous equalization error, the constellation diagram in the initial acquisition phase, and then the constellation again with the startup transient removed for 12- (subplots (a) to (d)), 18- (subplots (e) to (h))and 24-bit (subplots (i) to (l)) precision arithmetic. Plots (c), (g) and (k) for each precision presents symbols 1 to 4000 of the simulation while plots (d), (h) and (l) in each case shows symbols 4000 to 5000. For all cases the adaptation rate constant is 0.004.

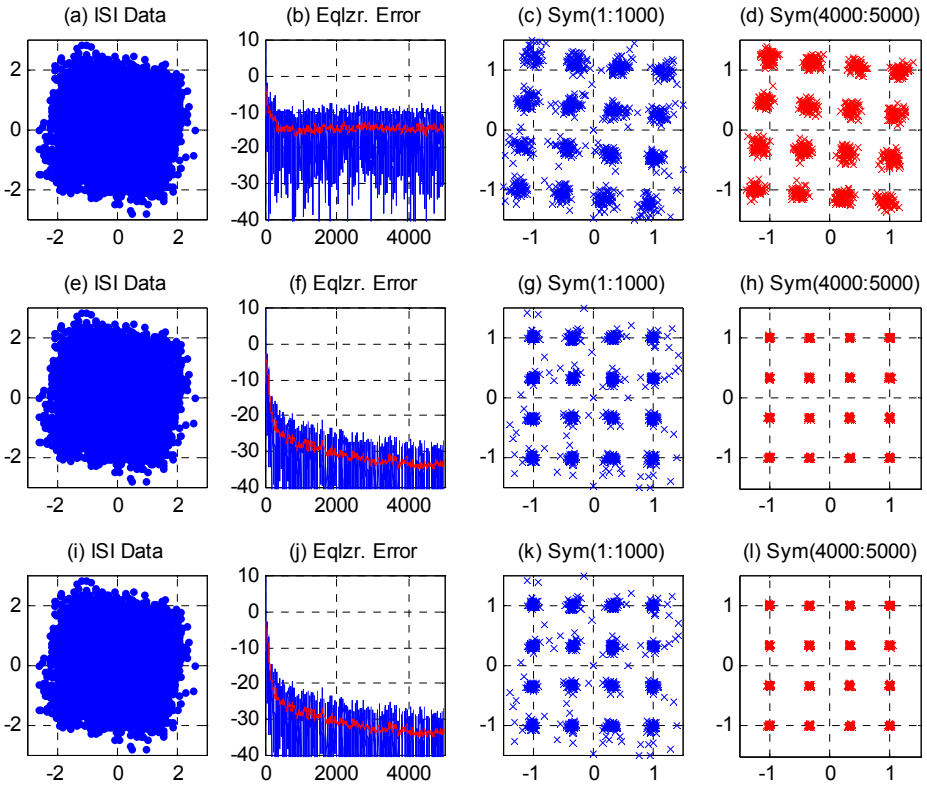

Figure 7: Adaptive equalizer simulation. (a),(b),(c),(d) 12-bit precision datapath. (e),(f),(g),(h) 18-bit precision datapath. (i),(j),(k),(l) 24-bit precision datapath.

From Figure 7 we note that a 12-bit precision design does not provide the required performance. The equalized constellation has a residual phase rotation due to the low-precision calculations and the error vector magnitude will be large as is evident from the significant dispersion of the constellation points. The 18-bit datapath provides adequate performance. As shown in Figure 7(j), (k) and (l), increasing the datapath precision to 24-bits provides little improvement. An additional point to note from the plots is the comparatively large value of the equalizer error for the low-precision design, compared to the 18b and 24b designs that exhibit a 20dB improvement after convergence.

The design was implemented using version 4.2i (speed-file version 1.96) of the Xilinx tool suite. The 50 MHz clock frequency was achieved using a -5 speed grade Virtex-II device. The utilization data is: 1808 logic slices and 18 embedded multipliers. Place-and-route was run with the switches *par -ol 5 -xe 2*.

CONCLUSION

As is the nature of the human condition, systems will continue to increase in complexity. It will become less useful to produce devices that solve only one part of the problem. Increasingly, sales of stand-alone digital signal processing and embedded chips will give way to SoCs that incorporate DSP or other functions [1]. For example, in a soft radio system, the PHY is dominated by sophisticated and arithmetically demanding DSP. But what about the other parts of the technology that are required to make a soft radio a reality? As we progress up the hierarchy of a soft radio we encounter problems that are best addressed using a RISC (reduced instruction set architecture) processor, for example the TCP/IP stack. We will also have a need to communicate with a broader network like the internet. Integrated components like the Power-PC 405 and the multi-gigabit transceivers in the Virtex-II Pro FPGA address these considerations by providing a platform based approach to system implementation.

Continually decreasing market windows and product life cycles mean that it is more difficult to achieve the high volume sales that we have seen in the past, making the ASIC proposition less attractive. Each year the transistor budget dramatically increases, while design methodologies and software systems for integrated circuit development continue to lag. The Platform FPGA approach to system development and product deployment is one solution to this complex set of market and technology dynamics. Platform FPGAs utilize the transistor budget in ways that are being increasingly rewarded by the market, which include timeliness, flexibility and customization.

REFERENCES

[1] Bass , M. J. and Christensen, C. M., "The Future of the Microprocessor Business", IEEE Spectrum, April 2002, pp. 34-39.

[2] Virtex-II Platform FPGA Handbook, Xilinx Inc, San Jose, CA, 95124.

[3] Haykin S. Adaptive Filter Theory, Prentice Hall, New Jersey, 1996.

[4] http://www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=System_Generator