System Level Tools for DSP in FPGAs

James Hwang, Brent Milne, Nabeel Shirazi, and Jeffrey D. Stroomer

Xilinx Inc, 2100 Logic Drive, San Jose, CA 95124, USA

Abstract. Visual data flow environments are ideally suited for modeling digital signal processing (DSP) systems, as many DSP algorithms are most naturally specified by signal flow graphs. Although several academic and commercial frameworks provide a high level of abstraction for modeling DSP systems, they have drawbacks as design tools for FPGAs. They do not provide efficient implementations, and their system behavior only approximates the hardware implementation. In this paper, we describe a software system that employs a visual data flow environment for system modeling and algorithm exploration. In this environment, the bit and cycle behavior of the FPGA implementation are manifest. By observing circuit behavior in the system environment, one obtains significant speed improvement over hardware simulation, while gaining substantial flexibility afforded by functional abstraction. In addition, the software automatically generates a faithful hardware implementation from the system model. Specific issues addressed include the mapping of system parameters into implementation (e.g., sample rates, enables), and implications of system modeling for testing (e.g., testbench generation).

1 Introduction

In recent years, field-programmable gate arrays (FPGAs) have become key components in implementing high performance digital signal processing (DSP) systems, especially in the areas of digital communications, networking, video, and imaging[1]. The logic fabric of today's FPGAs consists not only of look-up tables, registers, multiplexers, distributed and block memory, but also dedicated circuitry for fast adders, multipliers, and I/O processing (e.g., giga-bit I/O)[3]. The memory bandwidth of a modern FPGA far exceeds that of a microprocessor or DSP processor running at clock rates two to ten times that of the FPGA. Coupled with a capability for implementing highly parallel arithmetic architectures, this makes the FPGA ideally suited for creating high-performance custom data path processors for tasks such as digital filtering, fast Fourier transforms, and error correcting codes.

All major telecommunication providers have out of necessity adopted FPGAs for high-performance DSP. A third-generation (3G) wireless base station typically contains FPGAs and ASICs in addition to microprocessors and digital signal processors (DSPs). The processors and DSPs, even when running at GHz clock rates, are increasingly used for relatively low MIPs packet level processing, with the chip and symbol rate processing being implemented in the FPGAs and

ASICs [2]. The fluidity of emerging standards often makes FPGAs, which can be reprogrammed in the field, better suited than ASICs.

Despite these characteristics, broad acceptance of FPGAs in the DSP community has been hampered by several factors. First, there is a general lack of familiarity with hardware design and especially, FPGAs. DSP engineers conversant with programming in C or assembly language are often unfamiliar with digital design using hardware description languages (HDLs) such as VHDL or Verilog. Furthermore, although VHDL provides many high level abstractions and language constructs for simulation, its synthesizable subset is far too restrictive for system design.

Fundamentally, there is a lack of high-level tools and flows for DSP design in FPGAs. In this paper we describe a new software tool called System Generator for modeling and designing DSP systems in a visual data flow environment. In addition to providing a great deal of functional abstraction, the tool automatically maps the system model to a faithful hardware implementation. What is most significant is that the software provides these services without substantially compromising the quality of either the functional representation or the performance of the hardware implementation.

2 System Level Modeling

Two major trends have emerged in tools and techniques for system level design: the use of high-level languages and visual data flow. Language based approaches [4,5] have proven effective for system modeling, specification, and algorithm verification, but remain unsuitable for implementation in any target other than microprocessors. For targeting state-of-the-art DSPs, today's compilers often do not provide sufficiently good code for high-performance applications. Consequently, DSP programmers frequently resort to writing either in assembly or low level, highly stylized C [6].

Visual data flow environments[7,8,9,10] are ideally suited for modeling DSP systems, as many algorithms are most naturally specified by signal flow graphs. Data flow tools are similar to traditional schematic capture tools in that they provide libraries of functional blocks that can be composed graphically to model a system. In contrast to schematic tools, however, the library blocks and the simulation environment in a data flow tool provide a high level of functional abstraction, with polymorphic data types and operators to model arithmetic on integer, fixed-point, and floating point data. Time evolution is specified by sample rates rather than by wiring explicit clocks. Although previous environments support system modeling and in some cases derived implementations, their costs of abstraction include inefficient FPGA implementations, and inexact modeling of the hardware in the system simulation.

2.1 Related Work

The Ptolemy Project at U.C. Berkeley has developed a powerful visual environment for system modeling, with an emphasis on embedded system design [8].

It is an academic framework that does not support FPGA design. The Signal Processing Worksystem (SPW) from Cadence Design Systems is widely used for ASIC design in digital communications[7]. For the most part, SPW requires the user to provide an HDL implementation for the FPGA portion of a system. Although SPW provides libraries with high level functional abstractions (timed and untimed), it does not readily support bit and cycle accurate modeling of user and third-party intellectual property (IP) blocks except through HDL co-simulation. SPW is a powerful system, but its cost is many times that of a full suite of commercial FPGA implementation tools.

Another commercial tool of particular interest is Simulink, which runs within MATLAB, a popular mathematical modeling environment from The MathWorks, Inc. Simulink supports simulation of continuous-time and space as well as discrete-time and space dynamical systems. The latter makes it a suitable platform to model the evolution of hardware over time. Coupled with numerous libraries for modeling DSP and communications systems, as well as MATLAB's capabilities for data analysis and visualization, Simulink is an excellent platform upon which a system design tool for FPGAs can be built.

2.2 System Generator

System Generator is a new system level tool built on top of Simulink. It facilitates DSP design for FPGAs. Simulink provides a convenient high level modeling environment for DSP systems, and consequently is widely used for algorithm development and verification. System Generator maintains an abstraction level very much in keeping with the traditional Simulink blocksets, but at the same time automatically translates designs into hardware implementations that are faithful, synthesizable, and efficient.

The implementation is faithful in that the system model and hardware implementation are bit-identical and cycle-identical at sample times defined in Simulink. The implementation is made efficient through the use of Intellectual Property (IP) cores that provide a range of functionality from arithmetic operations to complex DSP algorithms. These cores have been carefully designed to run at high speed and be area efficient. In System Generator, the capabilities of IP cores are transparently extended to fit gracefully into a system level framework. For example, although most underlying IP cores operate on unsigned integers, System Generator allows signed fixed point numbers to be used as well, including saturation arithmetic and rounding. User-defined IP blocks can also be incorporated into a System Generator model as black boxes.

3 System Level Design with System Generator

The creation of a DSP design begins with a mathematical description of the operations needed and concludes with a hardware realization of the algorithm. The hardware implementation is rarely faithful to the original functional description;

instead it is "faithful enough". The challenge is to make the hardware area and speed efficient while still producing acceptable results.

In a typical design flow, also supported by System Generator, the designer is faced with the following steps:

1. Describe the algorithm in mathematical terms;
2. Realize the algorithm in the design environment using double precision;
3. Trim double precision arithmetic down to fixed point;
4. Translate the design into efficient hardware.

Step 4 is time consuming and error prone because it can be difficult to guarantee the hardware implements the design faithfully. System Generator eliminates this concern by automatically generating a faithful hardware implementation.

Step 3 is error prone because an efficient hardware implementation uses just enough fixed point precision to give correct results. System Generator does not automate this step, which typically involves subtle trade off analysis, but it does provide tools to make the process tractable. The reader might wonder why it is not possible to eliminate Step 3 and simply use floating point operations in hardware. The answer is that most operations have a sufficiently small dynamic range that a fixed point representation is acceptable, and the hardware realization of fixed point is considerably smaller and faster.

In the remainder of this section, we describe the capabilities of System Generator that simply the design process and contrast these capabilities with those of other tools.

3.1 Arithmetic Data Types

System Generator provides the three arithmetic data types that are of greatest use in DSP: double precision floating point, and signed and unsigned fixed point numbers. It does not provide mechanisms to convert floating point algorithms into hardware, but it does support them for simulation and modeling.

The set of arbitrary precision, fixed-point numbers has nice mathematical properties, with several advantages over familiar floating point representations. Operations on floating point numbers entail implicit rounding on the result, and consequently, desirable algebraic characteristics such as associativity and distributivity are lost. Both are retained for arbitrary precision, fixed-point numbers.

System Generator allows the quantization of the design to be addressed as an issue separate from the implementation of the mathematical algorithm. The transition from double precision to fixed point can be done selectively. In practice this means the designer gets the design working using double precision, then converts to fixed point incrementally. At all times, these three representations can be freely intermingled without any changes to the signal flow graph. This mixing is possible because library building blocks change their internal behavior based on the types of their inputs.

There is another benefit from this scheme in which quantization events are broken out as separate design parameters. At every point of the design, the designer can specify how both overflow and rounding are to be addressed. For overflow, the designer can choose whether saturation should be applied and do so in consideration of the hardware cost versus the benefit to the system design. Saturation is a more faithful reflection of the underlying mathematics, but more expensive in hardware; wrapping is inexpensive but less faithful. It is also possible to trap overflow events in the system level simulation. This is useful when debugging a subsystem that should never overflow.

Likewise, when quantizing at the least significant bit, the designer can choose whether the value should be truncated (with no hardware cost) or rounded under some particular rule (possibly improving the system design but with added cost in hardware).

In System Generator, many operators support "full precision" outputs, which means that the output precision is always sufficient to carry out the operation without loss of information. Combined with the data type propagation rules supported in Simulink, this can reduce the clerical burden in algorithm design.

The designer specifies the translation to fixed precision at key points in the model, namely, at gateways from non-System Generator blocks and in feedback loops. (Clearly, any operator whose output width exceeds that of its input cannot feed back on itself with full precision.) System Generator then propagates signal types and precisions as appropriate. The automatically chosen type is the least expensive that preserves full precision. Translations from signed to unsigned and vice versa are automatic as well .

3.2 Hardware Handshaking

In Simulink, time evolution is defined by sample rates for each block in the system. Sample rates propogate along signals and through blocks automatically, so in most cases it is not necessary for the designer to assign explicit rates to blocks. This is extremely flexible, but has implications for modeling hardware. A bit and cycle true simulation must provide mechanisms for defining and controlling clocked behavior in the system model. System Generator attempts to provide control mechanisms that do not compromise the abstract view afforded by Simulink.

In one such mechanism, every signal carries an implicit boolean "valid bit" that can be used to achieve hardware handshaking between blocks. For example, upon startup a pipeline may define its output "invalid" until it has flushed its pipe. By inspecting the valid bits of its inputs, a block can determine how to process its input data.

3.3 Multirate Systems

Multirate systems can be implemented in System Generator by using sample rate conversion blocks for up-sampling and down-sampling. The necessary control logic is automatically generated when the design is netlisted. Before netlisting, the sample rates in the system are normalized to integer values; in hardware, the system clock period corresponds to the greatest commond divisor of the integer sample periods. Clock enables are used to activate the hardware blocks at the appropriate moment in time relative to the system clock.

Consider for example, the multirate system model shown in Fig. 1. This system consists of I/O registers, an up-sampler, an anti-aliasing filter, and a down-sampler. The input signal is up-sampled by a factor of two, and subsequently down-sampled by a factor of three, giving an overall sample rate conversion by a factor of $\frac{2}{3}$. The `ST` blocks in the system model extract sample periods from Simulink signals. In the example, the input sample period is one. In the generated hardware implementation, as shown in the same figure, each element is driven by the system clock, with its respective clock enable driven according to its sample period in the original system model.

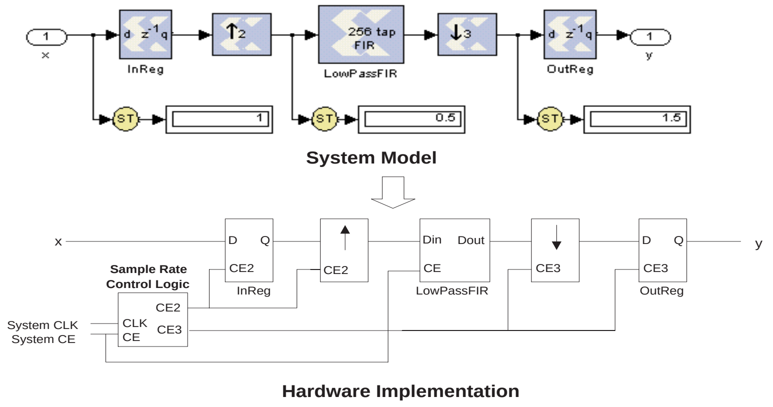


Fig. 1 Sample rate conversion by a factor of $\frac{2}{3}$.

3.4 Bit-True and Cycle-True Modeling

System Generator produces a hardware implementation that is bit and cycle true to the system level simulation. We define the term “bit and cycle true” at the boundaries of the design. The boundaries of a design in System Generator are specified by `Gateway In` and `Gateway Out` blocks. These form interfaces between data representations within System Generator and data types standard to the Simulink environment. When hardware is generated, `Gateway In` blocks become input ports and `Gateway Out` blocks become output ports.

In the Simulink simulation, gateway blocks have data samples flowing in or out at regular sample periods. The values flowing in provide the stimuli, and those flowing out represent the response. In the generated hardware, if an identical stimulus sequence is presented at the input ports (at clock events corresponding to the input sample periods), then identical output sequences will be observed (at clock events corresponding to Simulink output events). The values presented to the hardware input ports and produced by the output ports are bit vectors interpreted as representing the fixed point values of the Simulink simulation. This correspondence between Simulink and hardware results is guaranteed to hold regardless of the particular input stimulus to the design or the positioning or number of `Gateway Out` blocks.

3.5 Automatic Testbench Generation

For a black box instantiation, the designer must provide both a Simulink model and an implementation. System Generator cannot automatically provide the verification that the two representations of the black box match. To assist the designer in verifying that the system model simulated in Simulink mirrors the generated hardware circuit, a VHDL testbench is automatically created during HDL code generation.

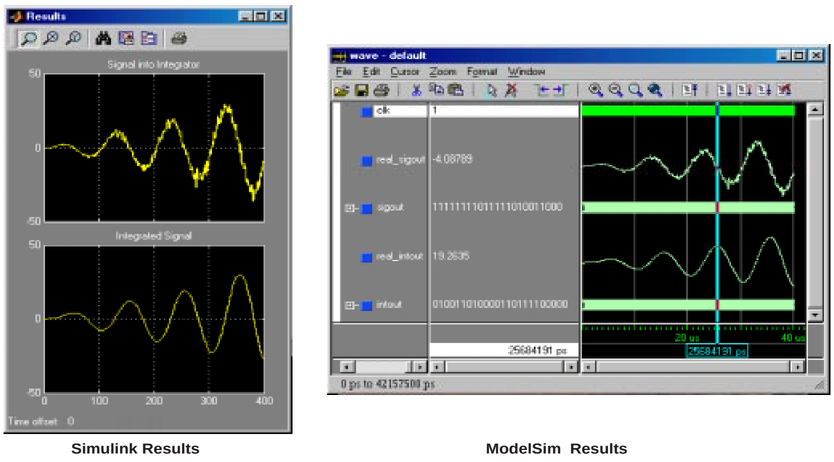


Fig. 2 Simulation results from Simulink and ModelSim VHDL simulator.

Testbench input stimuli are recorded by `Gateway In` blocks during Simulink simulation. These blocks quantize double precision input data into a fixed point representation. The fixed point values are saved to a file and then used as input stimuli during VHDL simulation.

During HDL code generation, each `Gateway In` block is translated to a VHDL component which reads the input stimuli. `Gateway Out` blocks are translated to components that compare the VHDL results to the expected results. The comparisons are performed at the block sample rates. Only values which are tagged as valid by the valid bit are compared. The fixed point data type in Simulink is represented using a `std_logic_vector` in VHDL. The position of the binary point, size of the container, and treatment of sign are supplied to the VHDL as generic parameters. To ease the interpretation of fixed point types in VHDL, the gateway blocks convert the `std_logic_vector` into a VHDL real number by using the generic parameter information. A sequence of real numbers can be viewed as an analog waveform in an HDL simulator such as ModelSim from Model Technology Inc. This enables the user to view data in the same manner as a Simulink Scope. An example of this shown in Fig. 2.

3.6 Simulation Results

The automatically generated testbench was used to compare simulation times in Simulink to a behavioral VHDL simulation as well as a simulation of a back-annotated netlist. Table 1 shows the simulation results for a 1024-point FFT, a fully serial implementation of a distributed arithmetic FIR filter, and a fully parallel implementation of the same filter. Simulations were performed using ModelSim SE 5.5a on a 650 MHz Pentium III with 500 MBytes of RAM.

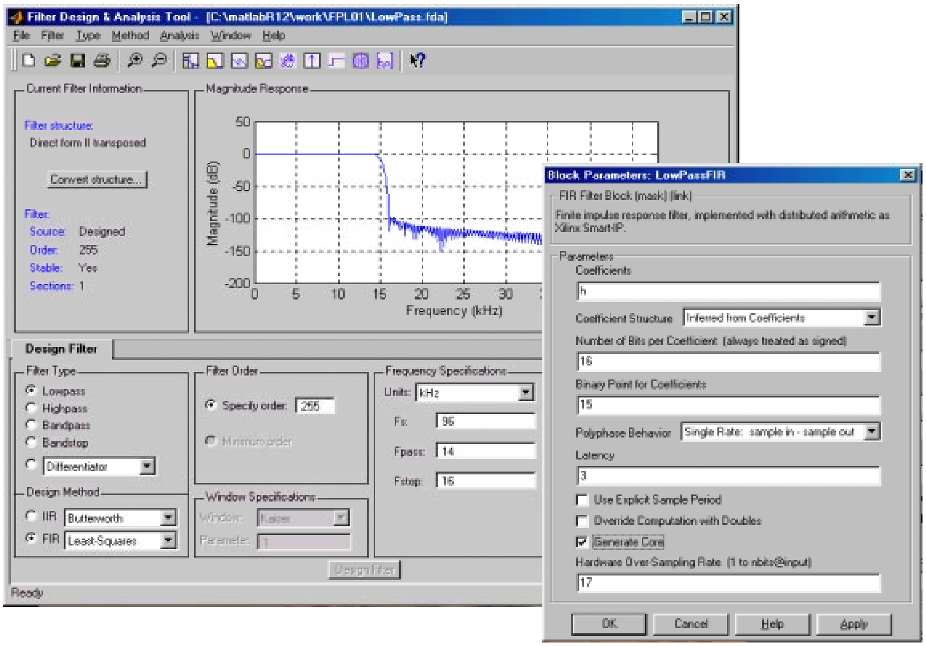
It can be seen that system level modeling not only increases design flexibility, it also provides much faster simulation than traditional HDL simulators. At the same time, the bit and cycle true simulation in Simulink gives an accurate idea of detailed system behavior.

Table 1 Simulation Times for DSP Designs.

Simulation Phase	1024 point FFT (15 transforms)	256 tap Parallel FIR (2K samples)	256 tap Serial FIR (2K samples)
Simulink	30 sec.	16 sec.	16 sec.
Behavioral VHDL	2.5 min.	2 min.	2 min.
Back-annotated VHDL w/o Timing Info.	36 min.	21 min.	24 min.
Back-annotated VHDL w/ Timing Info.	59 min.	40 min.	46 min.

4 Design Example

To demonstrate some of the significant features discussed in the previous section, we revisit the system model for the audio application depicted in Fig. 1. This system converts a single channel audio signal from a digital audio tape format, sampled at 48kHz, to a format for digital audio broadcast, sampled at 32 kHz [11]. This is accomplished by up-sampling by a factor of two and down-sampling by a factor of three. The Nyquist sampling rate of the digital output is 16kHz, and the up-sampler has an output rate of 96kHz, so we require an anti-aliasing filter with a stop band having a normalized frequency of $\pi/3$.



Filter structure:
Direct form II transposed

Filter:
Source: Designed
Order: 255
Stable: Yes
Sections: 1

Filter Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator

Design Method: IIR Butterworth [✓ FIR Least-Squares]

Filter Order: [✓ Specify order: 255] Minimum order

Window Specifications: Window: Kaiser Parameter: 1

Frequency Specifications: Units: kHz

Fs: 96

Fpass: 14

Fstop: 16

FIR Filter Block (mask) (link)

Finite impulse response filter, implemented with distributed arithmetic as Xilinx Smart-IP.

Coefficients: h

Coefficient Structure: Inferred from Coefficients

Number of Bits per Coefficient (always treated as signed): 16

Binary Point for Coefficients: 15

Polyphase Behavior: Single Rate: sample in - sample out

Latency: 3

☐ Use Explicit Sample Period

☐ Override Computation with Doubles

☑ Generate Core

Hardware Over-Sampling Rate (1 to nbits@input): 17

Fig. 3 Using FDATool and System Generator FIR parameterization GUI used to design a 256-tap anti-aliasing FIR filter. Coefficient vector h in the parameterization GUI is a vector exported from FDATool.

Designing a 256-tap FIR filter with a -100dB response over the stop band is straightforward using `FDATool`, a tool which runs in MATLAB (see Fig. 3). The filter coefficients were exported to the MATLAB workspace as the variable h. The System Generator GUI for the filter block, shown in Fig. 3, allows the user to choose from several implementation options.

System Generator detects that the filter coefficients are symmetric, and exploits this in the elaboration of the IP core to reduce the area required. The IP core employs distributed arithmetic [12] to compute the dot product. The number of input bits at a time used to index into the distributed memory is a user-selectable design parameter. For example, using a 16-bit input width yields a fully parallel (PDA) implementation, whereas using a single bit yields the most compact serial distributed arithmetic (SDA) implementation. In hardware, the IP core is over-clocked accordingly to ensure the system sample rate is seen at the filter output.

The SDA FIR filter can run well in excess of 102MHz in a Xilinx XCV50E-8 device, which for 16-bit data implies a sample rate of $102/17 = 6$MHz. Consequently, a single filter can in principle service $\lfloor \frac{6 \times 10^6}{9.6 \times 10^4} \rfloor = 62$ channels.

5 Conclusion

Modern FPGAs are powerful DSP devices, and with the availability of system level tools like System Generator, are poised to gain broad acceptance in the DSP community. System Generator supports bit and cycle true modeling, and automatically generates an FPGA implementation from a system model. In addition, it provides access to auxiliary tools for filter design, data analysis, visualization, and testbench creation. System Generator targets an increasingly broad IP library of DSP functions to take advantage of dedicated features in the FPGA to achieve high system performance. The combination of tools and IP libraries helps the system designer manage design complexity, provides a flexible modeling framework, and facilitates migration from algorithms into silicon.

Acknowledgements

The authors would like to thank their colleagues at Xilinx, Inc. and The MathWorks, Inc. for many fruitful discussions. In particular, they would like to acknowledge the contributions of D. Parlour, R. Turney, and C. Dick

References

1. C. H. Dick and f. j. harris,“Configurable Logic for Digital Communications: Some Signal Processing Perspectives”, *IEEE Comm. Magazine*, vol. 2, Aug. 1999, pp. 107–111.
2. C. H. Dick and H. M. Pedersen, “Design and Implementation of High-Performance FPGA Signal Processing Datapaths for Software Defined Radios”,*Embedded Systems Conference* Apr. 2001.
3. *Virtex-II Platform FPGA Handbook*, Xilinx, Inc., 2001.
4. http://www.systemc.org, SystemC.
5. M. Gokhale, J. Stone, and J. Arnold, “Stream-Oriented FPGA Computing in the Streams-C High Level Language”, in *Proc. FCCM00*, B. Hutchings (ed.), IEEE Computer Society Press, 2000, pp. 49–56.
6. J. Eyre, “The Digital Signal Processor Derby”, *IEEE Spectrum*, June 2001, pp. 62–68.
7. M. R. Sturgill et. al., “Design and Verification of Third Generation Wireless Communication Systems”, White Paper, Cadence Design Systems, Inc.
8. E. Lee, “What’s Ahead for Embedded Software”, *IEEE Computer*, vol. 33, no. 9, September 2000, pp. 18–26.
9. B. Levine et. al., “Mapping of an Automated Target Recognition Application from a Graphical Software Environment to FPGA-based Reconfigurable Hardware”, in *Proc. FCCM99*, K.L. Pocek and J.M. Arnold (eds.), IEEE Computer Society Press, 1999, pp. 292–293.
10. M. Schiff, “Baseband Simulation of Communications System”, Application Note AN133, Elanix, Inc., April 2000.
11. R.Lagadec, D.Pellooni, and D. Weiss, “A 2-channel, 16-bit Digital Sampling Rate Converter for Professional Digital Audio”, *Proc. IEEE Intl. Conf. ASSP*, April 1982, pp. 93–96.
12. S. A. White, “Applications of Distributed Arithmetic to Digital Signal Processing”, *IEEE ASSP Magazine*, Vol. 6(3), July 1989, pp. 4-19.