Modeling and Implementation of DSP FPGA Solutions

Robert D. Turney, Chris Dick[1], David B. Parlour[2], and James Hwang[3]

Xilinx Inc.
2100 Logic Dr.
San Jose, CA 95124, USA

ABSTRACT

Recent developments in the simulation capabilities of high level mathematical modeling tools have opened exciting new design flow possibilities. System level support for bit true modeling enables a designer to use a single environment to create floating and fixed-point models, and to make scaling and rounding decisions early in the design process. At the same time, FPGA vendors have expanded commercial IP offerings to incorporate higher level DSP functions. Together, these technologies enable a new flow for data path design that includes design iterations at the system level. In the past, DSP FPGA design required the combined efforts of a DSP engineer and a hardware engineer familiar with HDL or schematic based design. In this workshop we present a new method to derive an HDL netlist for a data path directly from a system level tool. The steps include construction of an ideal mathematical model, investigation of implementation effects, test-bench creation, and hardware netlist generation. Traditional HDL design methods are then used to complete the design implementation. These concepts are illustrated through examples of digital filter realizations, Discrete Wavelet Transforms (DWT), and digital communications applications.

[1] Robert D. Turney (robert.turney@xilinx.com) and Chris Dick (chris.dick@xilinx.com) are Senior Systems Engineers in the CORE Solutions Group, Xilinx Inc., San Jose CA, USA
[2] David B. Parlour (dave.parlour@xilinx.com) is a Principal Engineer in Xilinx Labs, Xilinx Inc.
[3] James Hwang (jim.hwang@xilinx.com) is a Staff Engineer at Xilinx Inc., San Jose CA, USA

1. INTRODUCTION

Field programmable gate arrays (FPGAs) now possess sufficient performance and logic capacity to implement a number of digital signal processing (DSP) algorithms effectively. In a tutorial article [1], Dick and Krikorian demonstrate that DSP algorithms can be implemented in an FPGA with levels of performance unattainable using a traditional single-chip processor. This is accomplished by exploiting parallelism as well as mapping techniques such as distributed arithmetic. For example, they demonstrate a tunable band-pass filter operating at a sample rate of 120MHz, which uses a single Xilinx XCV600 device to deliver 2.61×10^9 multiply-accumulate operations per second.

High performance FPGA circuits are often achieved using design techniques generally unfamiliar to the DSP design community, which traditionally relies on the processor to implement DSP systems. In reviewing the evolution of DSP processors, Ackland and D'Arcy [2] observe that "what is required is a combination of efficient DSP hardware (capable of exploiting much of the available parallelism in the application) together with smart compiler technology". If one finds shortcomings in tools for designing with DSP processors which take advantage of familiar design techniques developed for the general-purpose processor market, problems with tools that target FPGA logic will only be more acute.

The present research seeks to provide to the DSP system architect a design environment and flow that is familiar and easy to use, yet targets FPGA hardware directly. The key design goals for the project are to:

- Provide DSP system modeling capability at a high level of abstraction, with simple operators such as delay, addition and multiplication, as well as FIR and IIR filters, FFTs, and Discrete Wavelet Transforms.
- Ensure that the same model can be used throughout the design process, from performing initial theoretical design to quantizing coefficients or selecting data word lengths.
- Generate an FPGA implementation of the datapath from the system model automatically, without requiring the addition of device-specific information.
- Support the task of synthesizing control logic for the FPGA implementation.
- Automate the creation of test benches for performing logic simulation on the final FPGA implementation.

This work is enabled by the recent addition of bit-true simulation capabilities to commercially available system simulators ([3], [4]). A bit-true simulation is one where finite precision fixed-point operations are modeled precisely, obviating the need for an additional logic simulation of the final hardware implementation to determine actual system performance and to generate test vectors. Once the system architect has completed the design and simulation of a DSP system in a bit-true modeling environment, that model can be the ultimate specification for the behavior of the hardware.

The specific system simulator used in this investigation is Simulink[3], which runs within the Matlab programming environment[5]. Matlab is familiar to a large number of engineers and scientists, and as part of an open programming environment, provides an ideal platform for DSP system level tool development. The user can develop application specific functions with .m files (Matlab) or S-functions (Simulink). The Matlab environment allows the DSP designer the advantage of using familiar stimulus generation in addition to output data analysis tools.

The design flow targets Xilinx FPGAs[6], but the approach described here is equally applicable to other FPGAs, as well as custom logic including standard cells and mask-programmed gate arrays. In addition to the Simulink tool, the design flow makes use of logic synthesis libraries, a hardware macro generator, and place-and-route software known collectively as the Xilinx Foundation Series [7]. A key project goal is to enable system design at a high level of abstraction, so the choice of a particular target device and FPGA tool suite is not directly apparent to the user.

2. DESIGN FLOW

A block diagram of the design flow for DSP system modeling and implementation appears in Figure 1.

Working in Simulink, the DSP system designer creates a model of the hardware system as well as one or more test environments in which to simulate the model. The elements used in constructing the system model are all capable of operating on real (double precision, floating point) or integer (quantized, fixed point binary) data types. When the model is first entered, simulation is typically performed using floating data types to verify that its theoretical performance is as desired. The internal data types are then converted to the bit true representations that will be used in the hardware implementation, and the model is re-simulated to verify its performance with quantized coefficient values and limited data bit widths, which can lead to overflow, saturation and scaling problems. User

defined black boxes can also be incorporated in the modeling and elaboration process. When the model has been converted to a form realizable in the FPGA and its performance meets specification, the designer can invoke the netlister and the test bench generator.

The netlister extracts a hierarchical representation of the model's structure annotated with all the element parameters and signal data types. A mapper then analyzes the elements in the hierarchy and creates a VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language) description of the design. Where possible, the mapper uses the Xilinx CORE Generator™ to make hardware macros for specific design elements. When an element or its parameter values imply functionality unavailable in CORE Generator, the mapper instantiates a reference to a parameterized, synthesizable entity in a synthesis library or user supplied model. The actual hardware entities used have additional inputs and outputs for control signals that are not evident at the level of abstraction used in Simulink. The mapper inserts the necessary control ports and connects them up to control logic blocks. Multi-rate clocking can be supported through time step information provided during simulation. Each control logic block is given a default synthesizable behavior which may require alteration by a logic designer to achieve a working implementation. This manual intervention in the back end is shown in the flow as the block labeled Control Design.

The test bench generator is an interactive tool that runs in the Matlab environment, in which the designer captures the input stimuli and system outputs of selected simulation runs for conversion to test vectors. The generator converts the captured simulation data into VHDL code that will exercise the implemented model and test its outputs against the expected results.

The Xilinx Foundation Series tools are used to synthesize the control logic and those elements for which no hardware macros exist and combine all the pieces into a single fully-realized netlist, and place and route the design in an FPGA. The outputs of this back-end process are a bit-stream (FPGA programming file) and an EDIF (Electronic Design Interchange Format) structural netlist of the hardware annotated with timing information. This netlist can be simulated with the test vectors produced previously from system simulations to verify the performance of the completed FPGA hardware realization.

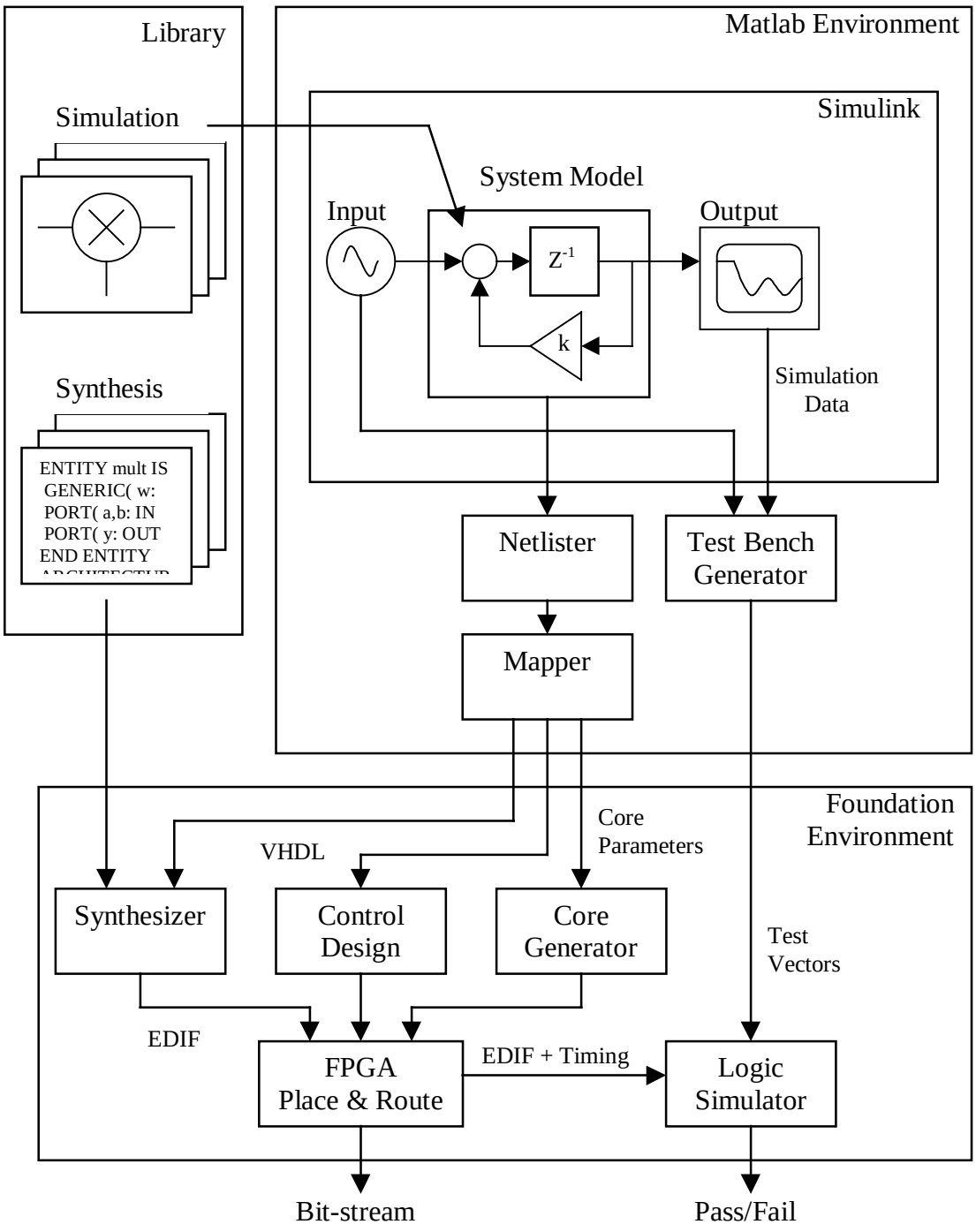

Figure 1: Design flow for DSP system modeling and implementation.

3. LIBRARIES AND MODELING

A key design decision in constructing this type of flow is whether to develop an FPGA-specific library of simulation elements or simply to use the libraries provided with the system tool. System level tool libraries offer the designer a wide range of modeling elements, but the mapping problem is made significantly more difficult, or even impossible.

The present design system includes a custom library of simulation elements that are guaranteed to have a mapping to FPGA logic, and which interconnect with their own signal type. The use of a special signal type ensures that blocks destined for FPGA implementation cannot be mixed with other Simulink blocks. The signal type can represent floating point numbers as well as signed and unsigned integers. The Simulink Library Browser window shown in figure 2 lists the elements supported in the custom library. The parameter entry pop-up window for the library element 'Sum' is overlaid to illustrate the level of abstraction a library element presents to the system designer.

To simplify design entry and preserve a high level of abstraction in the system block diagram, all of the library elements are designed to adapt automatically to the data representation of the incoming signal. By convention, the data representation of the output signal must be specified for each block, to avoid circular dependencies. The parameters for output data representation are illustrated in figure 2 for the Sum block, an operator that can act either as a two input adder or subtracter. Note that when a fixed point representation is chosen (signed 2's complement or unsigned), the designer must specify the number of bits used and a scaling parameter called the binary point shift. This is a weighting factor expressed as a power of two. For example, an unsigned integer with eight bits and a binary point shift of –7, denoted uns(8,-7), can represent numbers in the range 0 to 255×2^{-7}, while a signed integer with the same values, denoted sgn(8,-7), can represent the range -128×2^{-7} to $+127 \times 2^{-7}$.

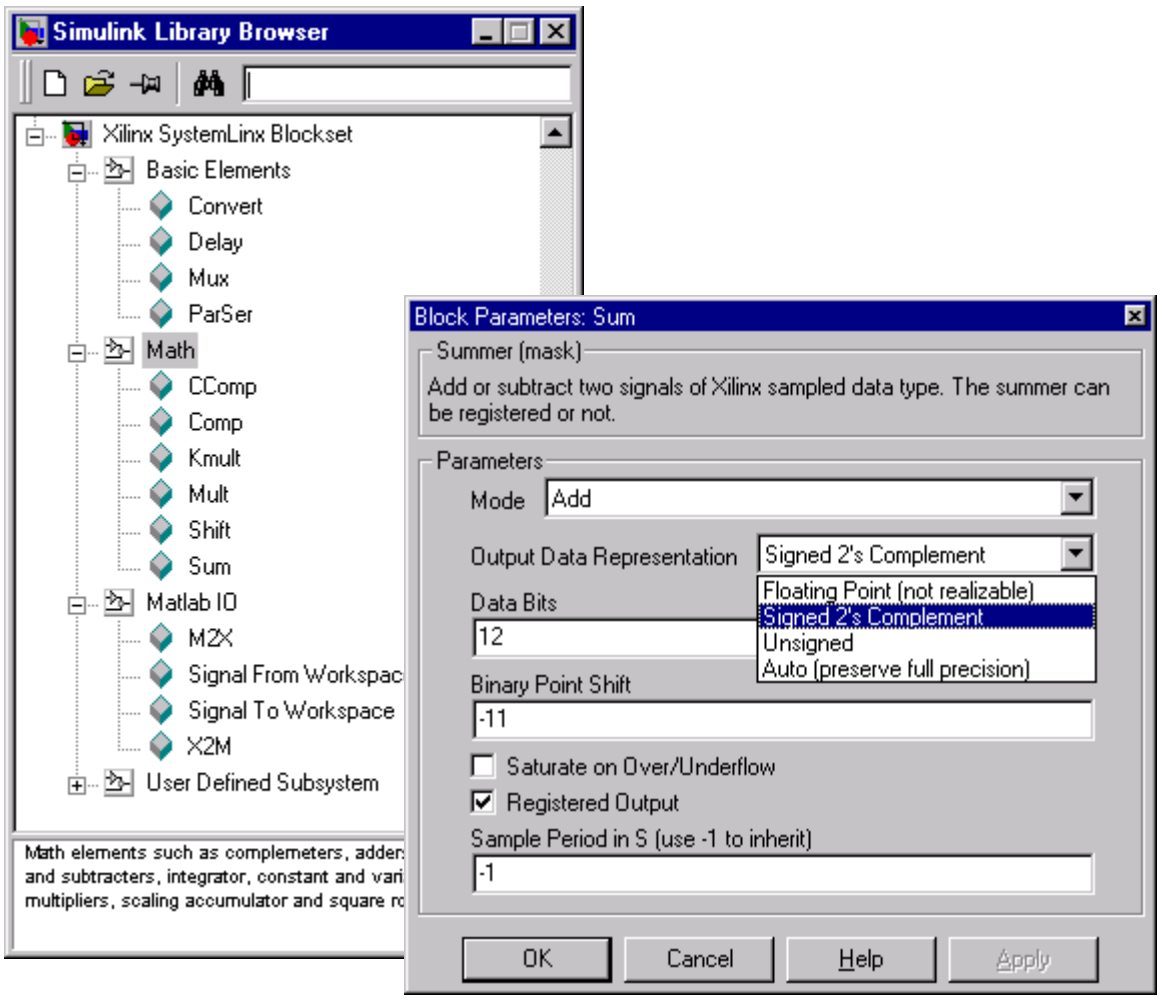

Figure 2: Library browser listing and block parameter entry GUI for Sum.

In adapting to the representations of incoming signals, all blocks apply the same rule: the block's internal operations are performed with enough precision to produce any possible result without overflow or truncation unless the scaling is part of the fixed point algorithm (e.g. Scaled FFT). The result of the block's calculation is then converted to the specified output representation, possibly with overflow or truncation. For example, the addition of two integers with representations uns(8,-7) and sgn(6,-10) will require an internal result with precision of sgn(13,-10), since the unsigned input must be converted to a signed integer (adding a sign bit) and aligned with the signed input (adding 3 padding bits in the least significant position). An additional bit is then required to accommodate the result of the addition. One of the options shown for output data representation in figure 2 is 'Auto', which causes the block to use the internally computed precision at the output. This representation must be used with care, since it can lead to significant growth

in word-length through a design, and cannot be used for signals that are fed back, either directly or through registers, making it impossible to determine an appropriate representation.

The library includes two special purpose conversion blocks, called M2X (Mathworks to Xilinx) and X2M (Xilinx to Mathworks), that are used to connect the FPGA model to any of the signal sources and sinks in the libraries supplied with Simulink. These conversion blocks do not represent actual hardware, so they should not be included in the model of the FPGA hardware.

In addition to simulation models, the element library must contain logic models that can be used for behavioral simulation in the hardware domain and for the synthesis of logic for all elements and parameter ranges for which hardware macros do not exist. In this case, the logic models are implemented in parameterized VHDL code.

4. NETLISTING AND MAPPING

The netlister derives a hierarchical data structure from a Simulink model and annotates it with parameter values and signal type information. The mapper then examines this structure and determines which elements can be implemented using pre-defined hardware macros, known as cores in the Xilinx Foundation environment. Cores are implementations of specific logic functions, ranging in complexity from adders and shifters to FIR filters and FFTs. These implementations are fully realized structures that include placement directives for the back-end FPGA tools. The use of cores results in guaranteed performance during place and route of the system.

Whether the data path elements of the system model are implemented as cores or as synthesizable logic, their hardware realizations will have additional control ports such as clocks, enables and resets, which are not apparent in the system block diagram. The mapper adds these ports to the netlist and wires up the control signals at each level of the hierarchy to a separate control logic block. A set of master controls, for signals such as system clock and system reset, are distributed tree-fashion from the top level down through the hierarchy.

The netlister and mapper include a mechanism that permits the user to identify any Simulink subsystem as a black box for which a VHDL entity will be provided explicitly, by tagging it with the following parameters:

- VHDLEntity
  Specifies a VHDL entity which defines the logic for this subsystem

- VHDLPortList
  Defines the port interface of the VHDL entity and maps those ports to the ports of the Simulink subsystem. The syntax also permits the declaration of control ports so that they can be included in the automatic control logic generation process.
- VHDLGenericList
  Defines the parameters of the VHDL entity and provides a mechanism for feeding the parameter values chosen within Simulink into the final VHDL netlist.

In the Simulink environment any model can be implemented within this black box Subsystem. It is the user's responsibility to ensure that the logical behavior of the named VHDL entity is equivalent to the simulation behavior of the referring subsystem.

The output of the mapping process includes a VHDL netlist of the entire system, and module generator directives for each instantiated core. Design elaboration is achieved by invoking a logic synthesis tool to produce FPGA logic for the behavioral portions of the design, and the Xilinx Core Generator™ to create the core files. User-supplied VHDL entities can be either behavioral or structural, and synthesis directives determine whether or not this code is elaborated by the synthesizer. When all portions of the design have been reduced to FPGA logic and combined into a netlist that is purely structural, the design can be placed and routed in an FPGA.

5. TEST BENCH GENERATOR

The test bench generator is used to convert Simulink simulation results into a form that can be used to verify the behavior of the FPGA hardware. In an interactive process, the user specifies which model inputs and outputs should be monitored and then initiates a simulation of the system. The test bench generator captures the simulation results and generates a VHDL test bench for each simulation run. This test bench includes an instance of the system being modeled driven by the same inputs as the Simulink model, and the behavioral code to test that its output values are identical to those of the Simulink simulation. With these automatically generated test benches, a logic designer can detect problems with the mapping of the Simulink model to HDL and errors introduced by the manually designed control logic.

6. DESIGN FLOW EXAMPLES

Example 1 - *Wavelet De-Noising*

To illustrate the modeling and implementation of a simple DSP system we show a wavelet filter example. The system comprises a Discrete Wavelet Transform (DWT), a

de-noising algorithm, and an Inverse DWT. Our goal in this example is to process data sets of 1024 frame length in approximately 10 μS. This defines a real time system where dedicated silicon area needs to be allocated to the DWT function. FPGA technology enables us to alter the de-noising algorithm through field programmability while giving the performance of dedicated silicon to the DWT and IDWT functions.

To begin modeling and elaboration of the wavelet system, the DWT and IDWT functions need to be designed. In design of a DWT, the choice of scaling function and mother wavelet can be made by consulting references for the particular application by using the Wavelet Toolbox from Mathworks as a design tool[8]. This will define the length of the filters in the wavelet structure. The next choice of tree structure involves the number of levels and Discrete Wavelet Packet or Discrete Wavelet decomposition. Floating point simulation of the algorithm can then be performed. To perform at the requested transform time, fixed point arithmetic is chosen by quantizing the input and filter coefficients. Fixed point simulation with a Bit True DWT model is performed to ensure the desired transform output. Verification with the actual DWT Core is critical at this stage of development with special attention to rounding and scaling between the levels.

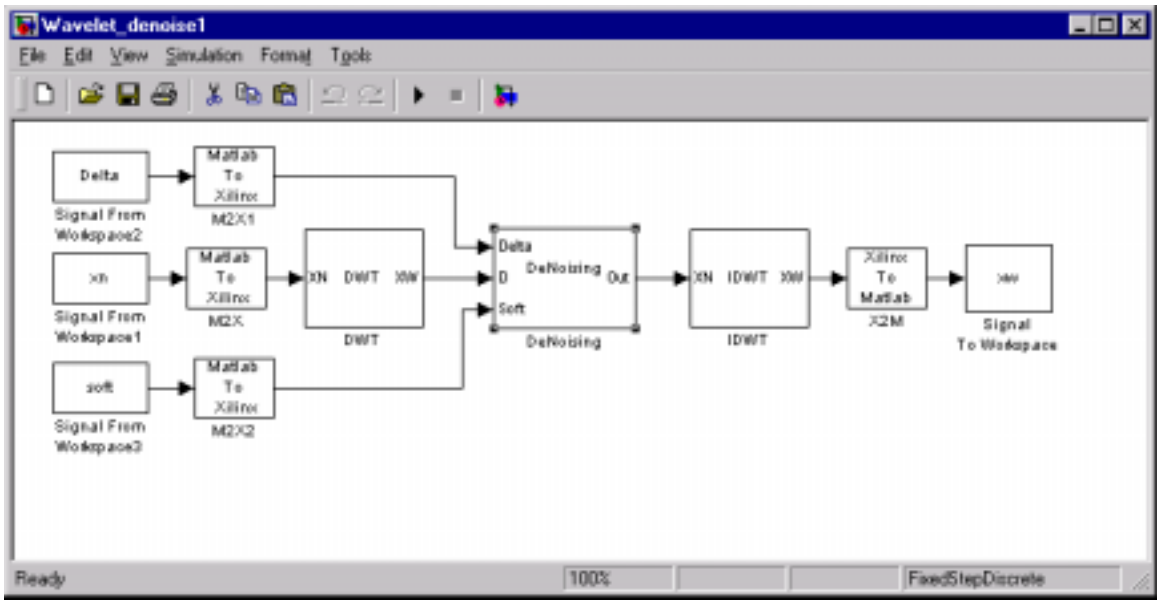

Figure 3: Simulink model for wavelet de-noising.

The design parameters relating to the DWT are entered in the Simulink block and passed to the HDL generic map and port map in the elaboration process. In this flow a parameterized DWT HDL design has been completed and only needs to be instantiated by the elaborator. In a similar flow the IDWT design process can take place.

In this example the de-noising block represents an algorithm specific to the application and thus not readily available as a predefined core. For the modeling of this block, basic level elements are used to realize the algorithm. Once the de-noising model has been simulated, it can be included into the system model for system level simulation. The elaboration of this block can be made by either generating the lower level netlist if the Xilinx Blockset has been used or by setting the HDL parameters on the de-noising symbol thus informing the HDL netlister to stop at that level. The digital designer would then have to provide the implementation for the de-noising function.

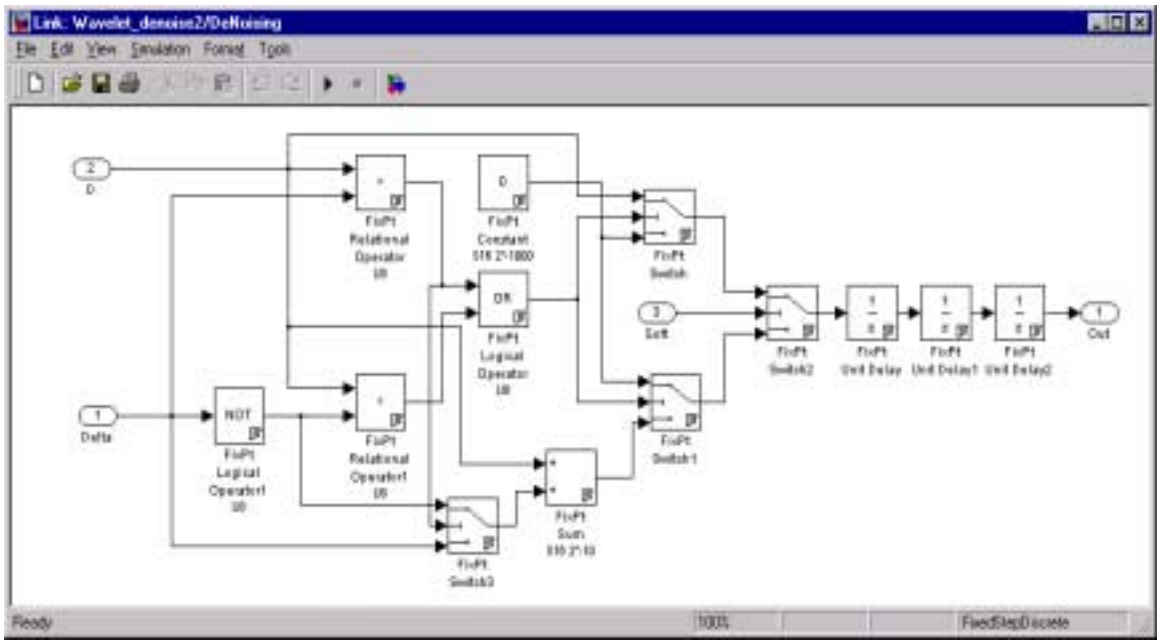

Figure 4: Simulink model for de-noising.

This modeling and implementation example illustrates how, for a given performance criteria, a system can be designed and realized in an FPGA. As often happens, cost is traded for performance in many DSP systems. In the silicon world, cost typically reveals itself in silicon area. To illustrate this point, consider our wavelet system is 1.5 times over our cost target. The designer would like to relax the requirement of 10 μS to 20μS and alter the design accordingly to save silicon area. To achieve this goal a new model is generated (Figure 5) which uses a block that can switch between DWT and IDWT on a frame basis. The design cost of the architectural change is increased system level control to manage the use of the DWT/IDWT engine. It would be preferable to leave the model as in Figure 3 and incorporate some level of decision making in the elaboration tools but at present this is a topic for research.

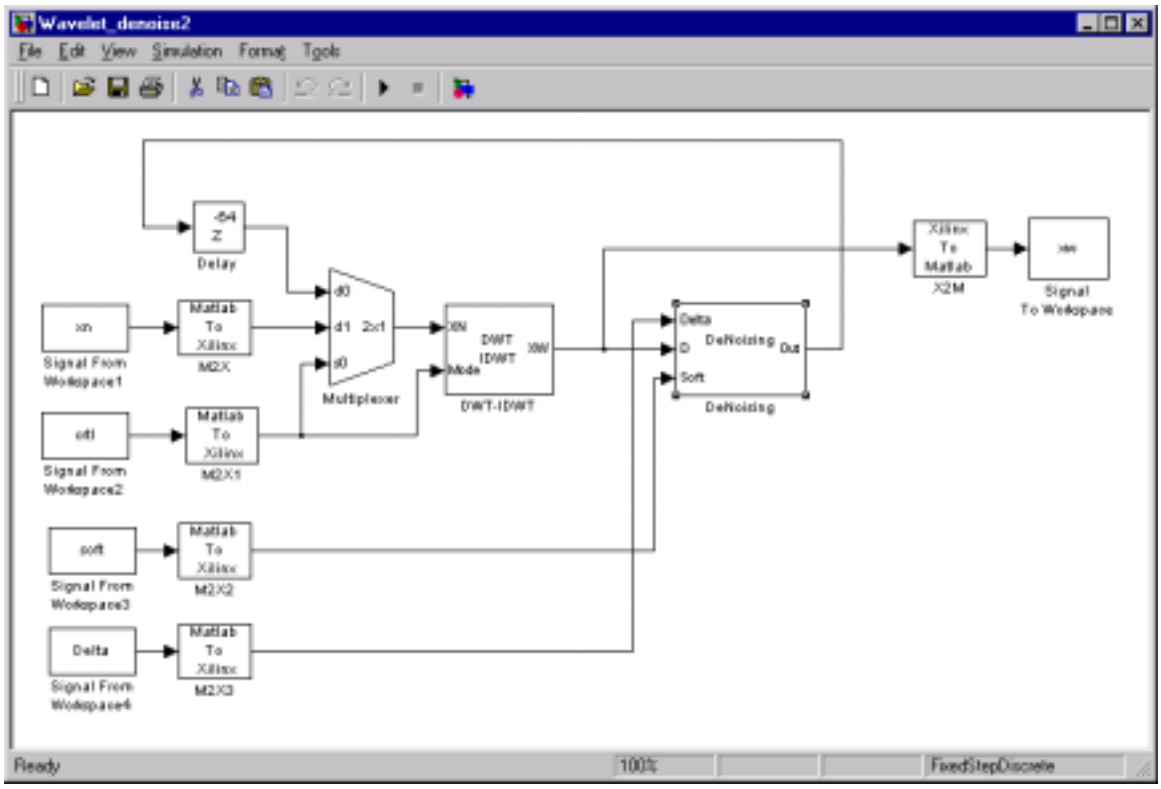

Figure 5: Simulink model for reduced performance wavelet de-noising.

This example demonstrates the use of the Matlab/Simulink system to model the wavelet system performing floating and fixed point simulation. Since we now have fixed point data, elaboration to an FPGA circuit can be performed utilizing IP blocks from an FPGA vendor, an IP house, or application-specific IP. Control functions and memory interfaces can now be added to the design using traditional EDA tool flows. Thus we have achieved a goal of designing the data path in the environment best suited for simulation and (through netlisting) allow for integration into EDA tools for logic and control functions best suited to the EDA environment.

Example 2 - ***A Variable Bandwidth Spectrum Analyzer***

The second design example is the variable bandwidth spectrum analyzer shown in Figure 6. The input heterodyne frequency translates a desired section of bandwidth to baseband. The baseband signal is processed by a cascade of halfband filters. Each filter successively reduces the bandwidth by a factor of two. By employing polyphase decimators, the signal data rate is simultaneously reduced to match the new output bandwidth at each stage. A multiplexer selects which signal is presented to the FFT processor. When the complex baseband data that is directly available at the output of the

heterodyne is processed, the analysis system channel spacing is f_s / N where f_s is the input sample rate and N is the transform length. As the signal is accessed at successive stages of the polyphase filter cascade, the spectrum analyzer channel spacing is commensurately reduced. A total of four channel spacings are available: $f_s / N, f_s / 2N, f_s / 4N$ and $f_s / 8N$.

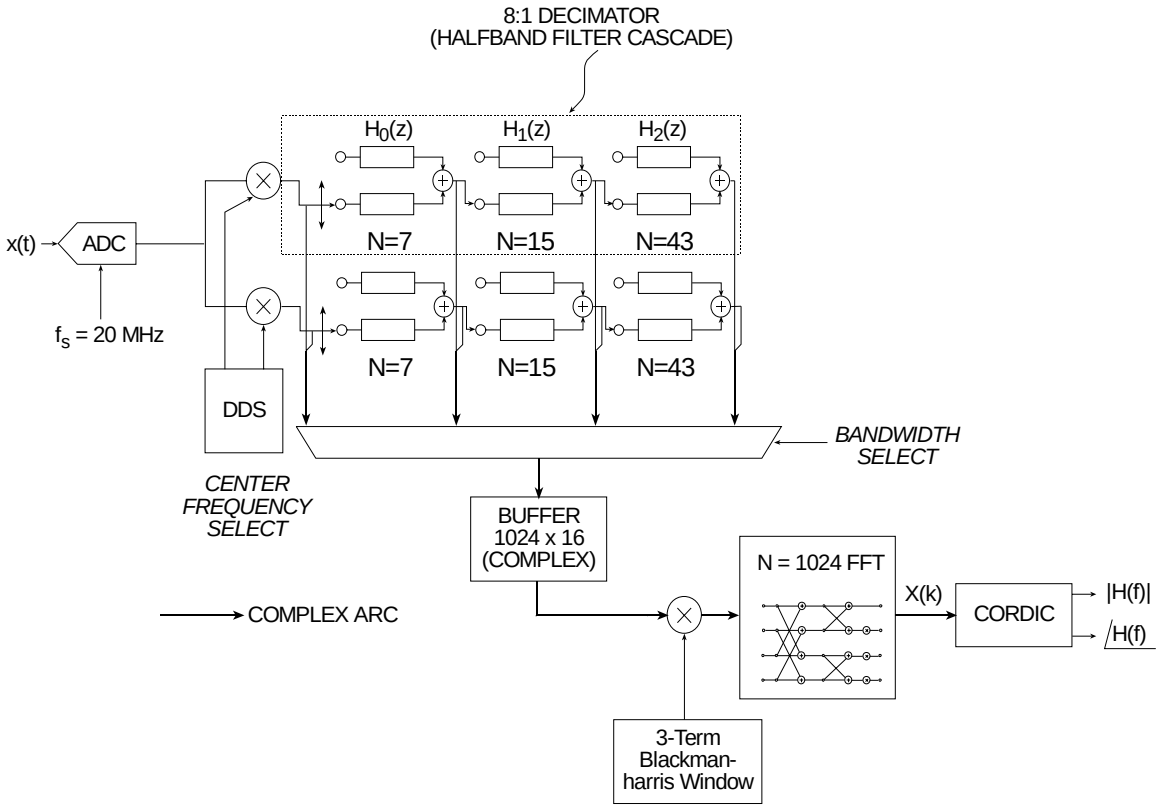

Figure 6: Variable bandwidth spectrum analyzer.

The Simulink model of the system is shown in Figures 7 and 8. This design and development approach allows a top-down design methodology to be employed and reduces design cycle time by enabling optimization and understanding of the system problems before committing to an implementation. Typically, a datapath of this complexity might be developed, simulated, verified and implemented using a combination of tools. For example, the algorithm might first be developed and simulated using a floating-point realization. Next, a fixed-point version of the system would be developed and verified using a C model. Finally, the design would be coded in VHDL as part of the hardware development and implementation process. Using the Simulink visual dataflow approach enables the design team to efficiently implement a system and avoid wasting energy by repeating design cycles. In addition, since the one unified environment

is used for both simulation and hardware generation, changes to the system parameters can be pushed back to later stages of the development schedule.

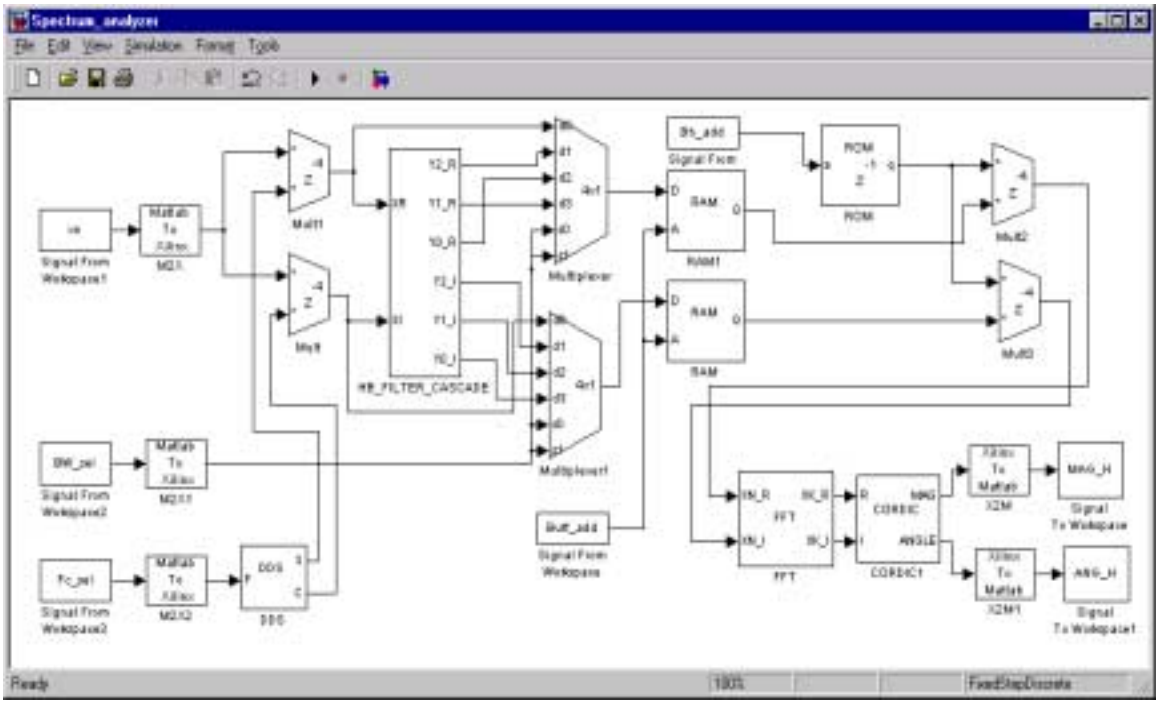

Figure 7: Simulink model of Variable bandwidth spectrum analyzer.

In this design, the system architect has to make decisions about the order of the halfband filters, the type of window to employ in the spectrum analysis phase, as well as selecting suitable word precisions for the various nodes in the design. FPGA technology provides the signal processing architect with many different options for implementing DSP functionality. For example there are many ways to implement inner-product calculations. A conventional scheduled multiply-accumulate (MAC) is one option, while distributed arithmetic (DA) [7] is another. Different options are suitable for different parts of the space-time plane. Using the Xilinx Simulink library effectively provides the designer with an expert system to help, and if required automatically, make the best silicon area/performance tradeoffs. The important point to observe, is that the designer no longer needs to have intimate knowledge of the FPGA device architecture or FPGA DSP design techniques. He/she can focus on the system, rather than the implementation details. In addition, high-performance core technology can be easily accessed. In our spectrum analyzer example, high-performance area efficient cores for the multipliers, the direct digital synthesizer (DDS), FIR filters and FFT would be used in the FPGA implementation.

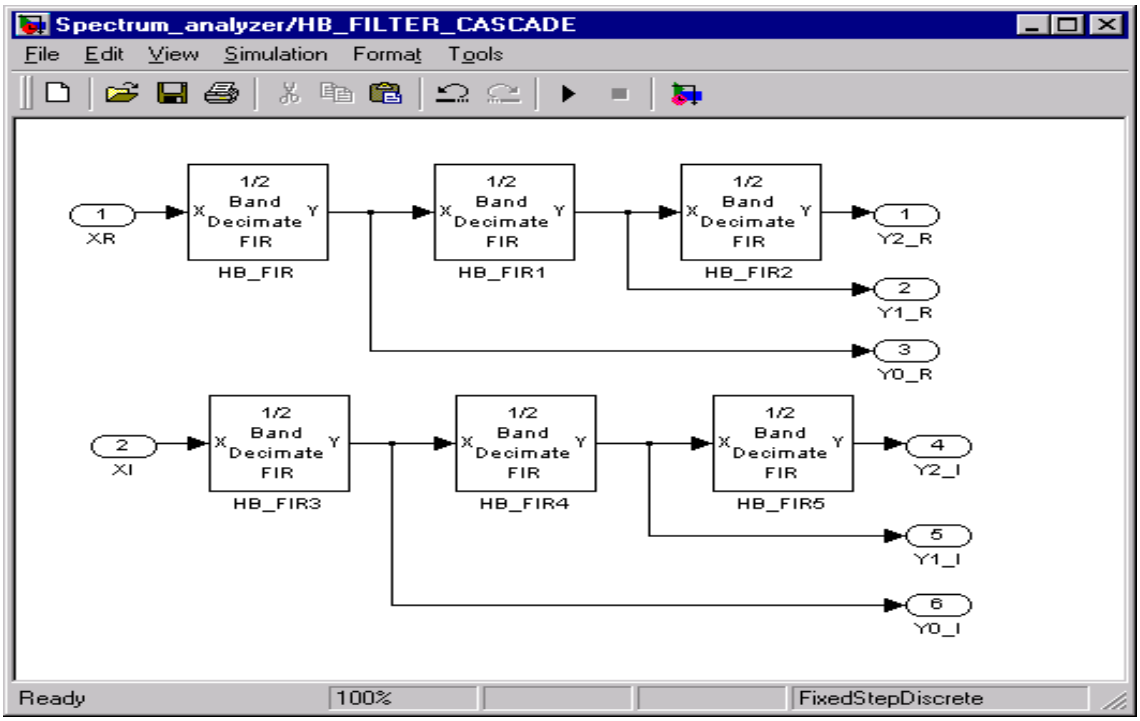

Figure 8: Simulink model of 8:1 Decimator Half Band Filter Cascade.

7. CONCLUSION

This work has demonstrated a design flow that makes the FPGA a viable platform for DSP system engineers to use in implementing DSP algorithms. Future work will provide extensions and refinements to improve quality of result and ease of use, including the following:

- Mapper improvements
- Control logic generation
- Test bench generation
- Library and toolbox development.
- Support for additional system level tools

REFERENCES

[1] Dick, C. and Krikorian, Y., "A System-Level Design Approach for FPGA-Based DSP Implementations", DSP World, Spring 1999.

[2] Ackland, B. and D'Arcy, P., "A New Generation of DSP Architectures", *IEEE Custom Integrated Circuits Conference*, May 1999.

[3] Mathworks Inc., "Simulink 3.0", http://www.mathworks.com/products/simulink/.

[4] Elanix Inc., "Bit-true DSP Library", http://www.elanix.com/dsp.htm.

[5] The Mathworks, Inc., "Matlab 5.3", http://www.mathworks.com/products/matlab/.

[6] Xilinx, Inc., Xilinx 1999 Data Book, http://www.xilinx.com/partinfo/db96.htm.

[7] Xilinx Inc., "Xilinx Foundation Series", http://www.xilinx.com/products/found.htm.

[8] Orofino, D., "Wavelets in Real-Time Applications", Matlab News and Notes, Summer 1999

[9] S. A. White, "Applications of Distributed Arithmetic to Digital Signal Processing", *IEEE ASSP Magazine*, Vol. 6(3), pp. 4-19, July 1989.