WP241 (v1.0) April 19, 2006

Using MATLAB to Create IP for System Generator for DSP

By: Thomas Hill

Custom DSP algorithms are best modeled mathematically using MATLAB®, while complete systems are best modeled *cycle-accurately* using Simulink. The marriage of these two modeling domains provides an efficient means to design DSP systems into FPGAs. DSP systems are often best described using a combination of graphical and language-based methods. The MathWorks, the industry leader in DSP modeling software, caters to this dichotomy by providing a cycle-accurate graphical design environment called Simulink® and a mathematical modeling language called MATLAB.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Overview

Simulink is well suited for the *system* aspects of DSP design, including control and synchronization of data flow to and from interfaces and memories. Simulink also provides a rich set of pre-defined DSP algorithms in the form of *blocksets* from which DSP systems can be constructed. Simulink is not, however, the most effective environment for modeling proprietary algorithms. It unnecessarily burdens the designer with cycle-accurate considerations and forces low-level arithmetic operations and array accesses to be constructed from graphical blocksets rather then concise, textual expressions.

DSP algorithm developers have found that the MATLAB language best meets their preferred style of development. With more than 1000 built-in functions, as well as toolbox extensions for signal processing, communications, wavelet processing, etc., MATLAB offers a rich and easy-to-use environment for the development and debugging of sophisticated algorithms. Simulink 6.0 unifies these two modeling environments with the Embedded MATLAB Block that allows MATLAB models to simulate within Simulink and compile into C-code through Real-Time Workshop® for processor-based DSP hardware implementations.

Xilinx System Generator for DSP is well established as a productive tool for creating DSP designs in FPGAs. With a graphical environment based on Simulink and a predefined Blockset of Xilinx DSP cores, System Generator for DSP meets the needs of system architects to integrate the components of a complete design and for hardware designers to optimize implementations. System Generator for DSP, however, lacks support for a design flow based on MATLAB.

The AccelDSP™ Synthesis Tool was developed specifically for algorithm developers and DSP architects who have embraced language-based modeling of DSP algorithms. With the AccelDSP Synthesis Tool, the algorithm developer begins with his or her floating-point MATLAB M-files to perform stimulus creation, evaluation of the algorithm, and post processing of the results. These M-files can be loaded into the AccelDSP tool and become the golden source for a design flow that ultimately produces optimized implementations in Xilinx FPGAs.

DSP Hardware Systems

System Generator for DSP is well suited for modeling the DSP *system*, which consists not only of the core DSP algorithm, but synchronous interfaces to external buses, memory read/write accesses, system data synchronization, and overall system control.

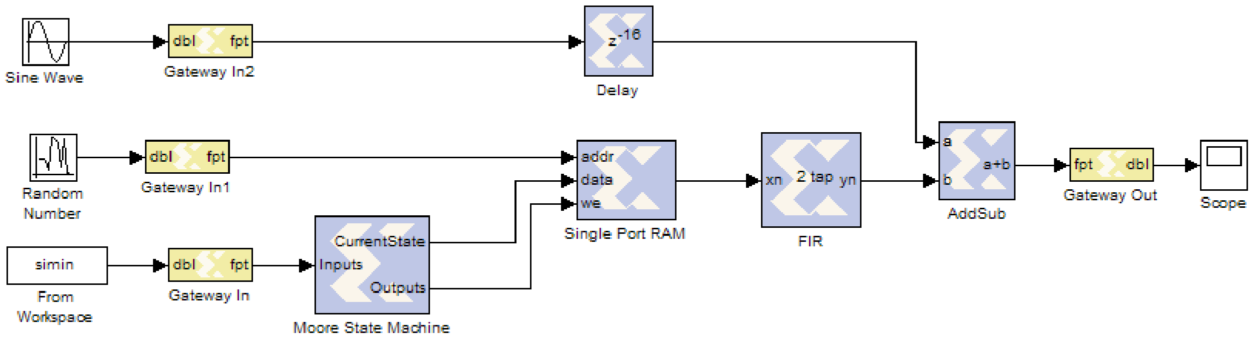

Figure 1: **Xilinx System Generator Diagram showing System Control and Synchronization Logic**

Custom DSP Algorithms

The heart of any DSP system is the algorithm. Algorithms distinguish themselves from systems in that a resulting output is a function of a given set of inputs without a sense of clock or hardware as described by the simple equation:

$$y = f(x) \qquad \text{Equation 1}$$

An algorithm defined in MATLAB can be executed on an FPGA, DSP processor, or software, each of which has a different sense of *cycle-accuracy*.

The unique characteristics of an algorithm offer two distinct advantages. First, the algorithm developer is completely unencumbered by hardware implementation details and is free to focus solely on functional behavior. This is exactly why an estimated 90 percent of the algorithms used today in DSP originate as MATLAB models, even when a design flow dictates they are re-implemented at a later time as Simulink or System Generator for DSP diagrams. Calculating the FFT of a 4 x 1024 matrix of data can be done with a simple MATLAB statement, without concern for radix, scaling, buffering, or synchronization of valid signals, as shown below:

$$y = fft(data,1024) \qquad \text{Equation 2}$$

Second, when modeling an algorithm, a given set of outputs corresponds to a given set of inputs; therefore, synchronization issues do not need to be addressed in the generated hardware. This makes an algorithm inherently *schedulable* through a DSP synthesis tool, such as the AccelDSP Synthesis Tool. Hardware requirements might dictate the need to use multiple clock cycles to compute an output, as the in case of a resource-shared MAC FIR filter, but this operation lends itself nicely to the AccelDSP automated flow. The introduction of a simple hardware handshaking interface enables easy integration into the overall system as shown in Figure 2.

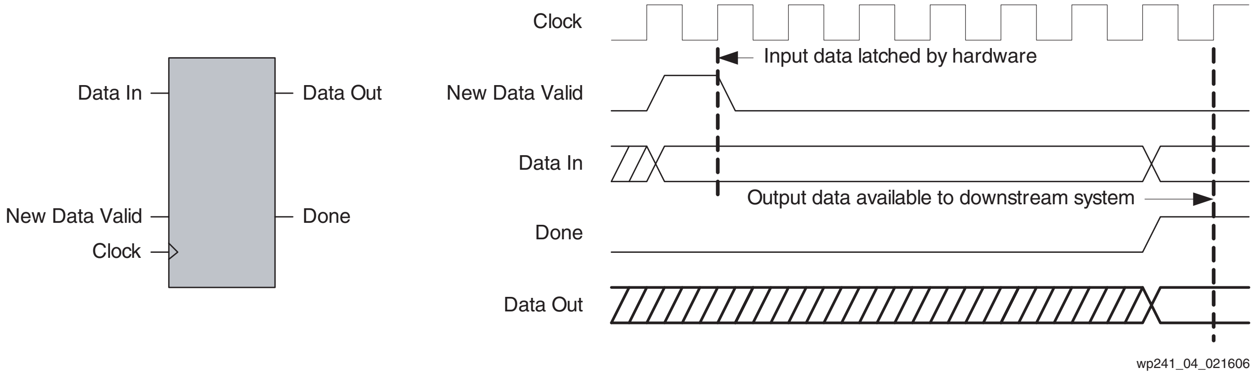

Figure 2: **AccelDSP Synthesis Tool Handshaking Interface**

AccelDSP Synthesis Tool Export to System Generator Option

The AccelDSP Synthesis Tool enables System Generator for DSP to support both DSP system and algorithm modeling methods by exporting System Generator intellectual property (IP) based on floating-point MATLAB models. This results in a design flow for FPGAs that is similar to the functionality obtained through the use of the Embedded MATLAB block. The system aspects of the design can be captured using the Xilinx DSP Blockset, while the algorithm can be captured using floating-point MATLAB. The System Generator IP blocks created by the AccelDSP Synthesis Tool are fixed-point, cycle-accurate.

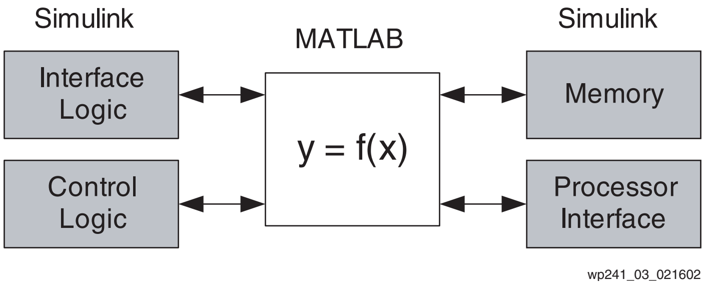

Figure 3: **DSP System Block Diagram**

Developing a Kalman Filter Example

The following Kalman filter example shows how to take an advanced algorithm based on MATLAB, use the AccelDSP Synthesis Tool to synthesize the design, and then integrate into a System Generator for DSP model. A Kalman filter is a special class of recursive, adaptive filters that is well suited to combining multiple noisy signals into a clearer signal (for details on the topic, see the book *Applied Optimal Estimation* by Arthur Gelb). Kalman filters embed a mathematical model of an object—for instance, a commercial aircraft being tracked by a ground-based radar—using the model to *predict* future behavior, and then use measured signals (such as the signature of the aircraft returned to the radar receiver) to periodically *correct* the prediction.

The MATLAB M-file describing the Kalman filter is shown below. The algorithm defines matrices R and I that describe the statistics of the measured signal and the predicted behavior. The last nine lines of the algorithm are the code that predicts forward and the code that corrects itself.

```
function [S] = simple_kalman(A)

DIM = size(A,2);
persistent p P_cap
if isempty(P_cap)
     P_cap = [8 0 0; 0 8 0; 0 0 8];
     p = ones(DIM,1)/2;
end;

I = eye(DIM);
R = [128 0 0;0 128 0; 0 0 128];

% estimate step:
P_cap_est = P_cap+I;

% correction step:
K  = P_cap_est * inv(P_cap_est+R);
p = p + K * ( A' - p );
P_cap = (I - K)*P_cap_est;
S = p';
```

This algorithm illustrates the flexibility and conciseness of the MATLAB language. Common operators like addition and subtraction operate on arrays like A or P_cap without the need to write loops as would be required in languages like C. Two-dimensional arrays are automatically multiplied as matrices without any special annotation. MATLAB operators such as matrix transposition allow the MATLAB code to be compact and easily readable. And, complex operations like matrix inversion are done using MATLAB's extensive linear algebra capabilities. While such an algorithm can be constructed as a block diagram, doing so will obscure the algorithm structure so readily apparent in MATLAB.

With the AccelDSP Synthesis Tool, complex MATLAB toolbox and built-in functions, such as the matrix inverse used in the Kalman filter example, can be taken directly to hardware using the AccelWare™ DSP IP Toolkits. These toolkits offer numerous matrix inverse methods. Core selection will depend on the size, structure, and values of the matrix. In this case, the most suitable approach is to use the AccelWare QR matrix inverse core. AccelWare cores are generated based on MATLAB syntax and are available in a variety of implementations that lets users optimize designs for speed, area, power, or noise.   A small code edit would be required to use the AccelWare function as shown below.

```
% K   = P_cap_est * inv(P_cap_est+R);
K   = P_cap_est * qr_inverse_001(P_cap_est+R);
```

Synthesizing RTL with the AccelDSP Synthesis Tool

With the MATLAB M-file now loaded into the AccelDSP Synthesis Tool, a floating-point simulation can be performed to establish a baseline. The design is then converted into a fixed-point representation. An array of features helps trim bits from the design and verifies the fixed-point design effects like saturation and rounding. Bit growth is automatically propagated through the design in a manner that allows for user-controlled overrides. This *algorithmic design exploration* process helps obtain the ideal quantization that minimizes bit widths while managing overflows/underflows, allowing early trade-offs of silicon area versus performance metrics.

After suitable quantizations have been determined, the next step with AccelDSP is to generate RTL for the target Xilinx device. Hardware intent can be dictated through the use of *synthesis directives* shown in Table 1.

Table 1: **DSP Synthesis Directives**

DSP Synthesis Directive	Effect on Results
Rolling/unrolling of For loops	Improves input sampling rate by reducing throughput.
Expansion of vector and matrix additions and multiplications	Improves input sampling rate by reducing throughput.
RAM/ROM memory mapping of 1D and 2D arrays	Improves FPGA utilization by mapping 1D and 2D arrays into dedicated Xilinx Block RAM resources.
Pipeline insertion	Improves input sampling rate by improving clock frequency performance.
Shift register mapping.	Improves FPGA utilization by mapping shift register logic into SRL16s.

Use of these directives constitutes *hardware-based design exploration* that allows the design team to further improve quality of results. In synthesizing the RTL, AccelDSP evaluates the entire design and schedules the entire algorithm, performing boundary optimization in the process.

Throughout this flow, AccelDSP maintains a uniform verification environment through the use of a self-checking testbench: the input/output vectors that were generated when verifying the fixed-point MATLAB design are used to verify the generated RTL. This step also gives AccelDSP the information necessary to compute the throughput and latency of the Kalman filter, which is essential to gauge whether the design meets specifications and to produce a cycle-accurate System Generator for DSP model.

Exporting from AccelDSP Synthesis Tool to System Generator

Upon successful completion of RTL verification, the design is now ready for export to System Generator for DSP by going to the Export pulldown menu in the AccelDSP GUI and selecting **System Generator** (Figure 4). The AccelDSP Synthesis Tool generates a System Generator IP block that supports both simulation and RTL code generation.

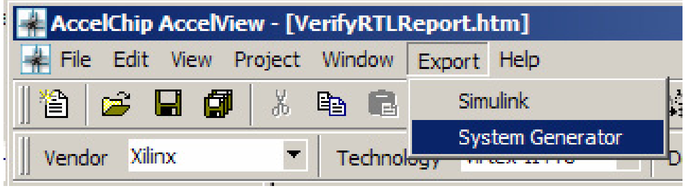

Figure 4: **AccelDSP Export System Generator Command**

At this point, the design flow transitions to System Generator for DSP, where the new block for the Kalman filter is available in the Simulink Library Browser. The user need only select the Kalman filter block and drag it into the destination model to incorporate the AccelDSP -generated Kalman filter into a System Generator design as shown in Figure 5.

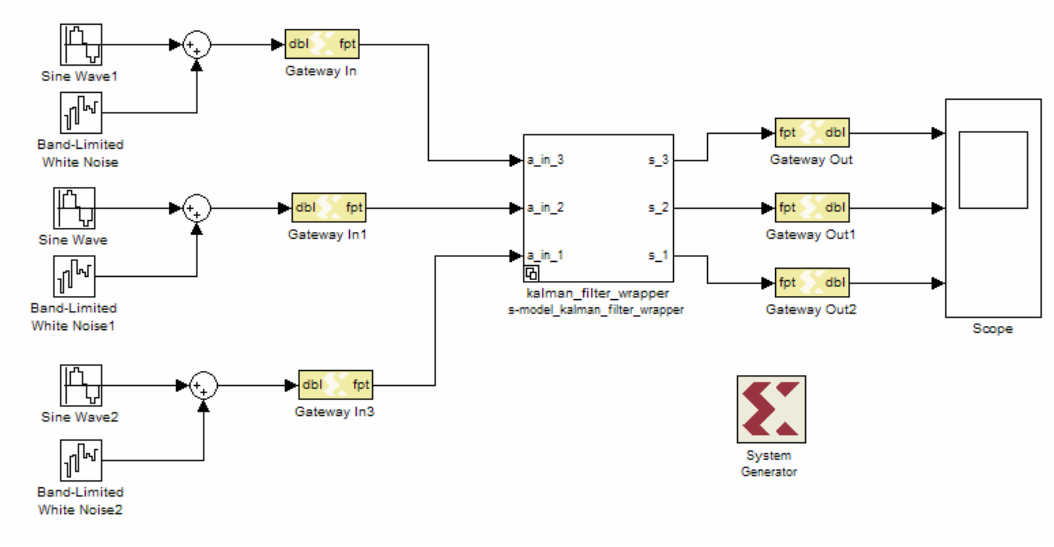

Figure 5: **Kalman Filter Exported to System Generator**

Conclusion

Custom DSP algorithms are best modeled mathematically using MATLAB while complete systems are best modeled cycle-accurate using Simulink. The marriage of these two modeling domains provides an efficient means to design DSP systems into FPGAs by allowing the use of the rich MATLAB language, including the built-in and toolbox functions, to create System Generator IP blocks of complex DSP algorithms. By using System Generator for DSP and AccelDSP Synthesis Tool in combination, design teams can employ the most productive means to model hardware for implementation, fully involve algorithm developers in the FPGA design process, and, as a result, complete higher quality designs in less time.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/19/06	1.0	Initial Xilinx release.