

Integrating MATLAB Algorithms into FPGA Designs

The integration of AccelChip DSP Synthesis and Xilinx System Generator for DSP provides a seamless path from MATLAB/Simulink to verified FPGA-based DSP systems.

by Eric Cigan
Product Marketing Manager
AccelChip Inc.
eric.cigan@accelchip.com

Narinder Lall
Senior DSP Marketing Manager
Xilinx, Inc.
narinder.lall@xilinx.com

There are two kinds of people in electronic design: those who think in terms of words and those who think in terms of pictures. This dichotomy is most apparent in the world of DSP. Some designers prefer to develop algorithms in language form, while others choose to draw out block diagrams showing data flows. Until now, different sets of tools were required for each method – but why should you have to choose?

Xilinx® System Generator for DSP is well established as a productive tool for creating DSP designs using graphical methods. With a visual programming environment that leverages The MathWorks Simulink tool and its predefined Xilinx block set of DSP functions, System Generator meets the needs of both system architects (to integrate design components) and hardware designers (to optimize implementations). (For more details, see “Implementing

DSP Algorithms in FPGAs” in the Winter 2004 issue of the *Xcell Journal*.)

Many DSP algorithm developers have found that the MATLAB language best meets their preferred development style. With more than 1,000 built-in functions, as well as toolbox extensions for signal processing, communications, and wavelet processing, MATLAB offers a rich environment for algorithm development and debugging.

In addition to the IP functions provided in MATLAB, the MATLAB lan-

guage is uniquely adept with vector- and array-based waveform data at the core of algorithms in applications such as wireless communications, radar/infrared tracking, and image processing.

The AccelChip DSP Synthesis tool was developed specifically for algorithm developers and DSP architects who have embraced a language-based flow. With AccelChip, you begin with MATLAB M-files to perform stimulus creation, algorithm evaluation, and post-processing. You can load the M-files into the AccelChip tool; they become the “golden source” for a design flow that ultimately produces optimized implementations in Xilinx FPGAs.

Unifying Words and Pictures

In the past, design teams looked on System Generator and AccelChip DSP Synthesis as mutually exclusive design tool options, but wished for access to the best aspects of each tool. In response, AccelChip and the DSP division at Xilinx collaborated in an effort to combine AccelChip’s tools with System Generator. The result allows you to mix language-based algorithm design in MATLAB and graphical block-oriented design in Simulink in a novel unified electronic-system-level (ESL) design environment (Figure 1).

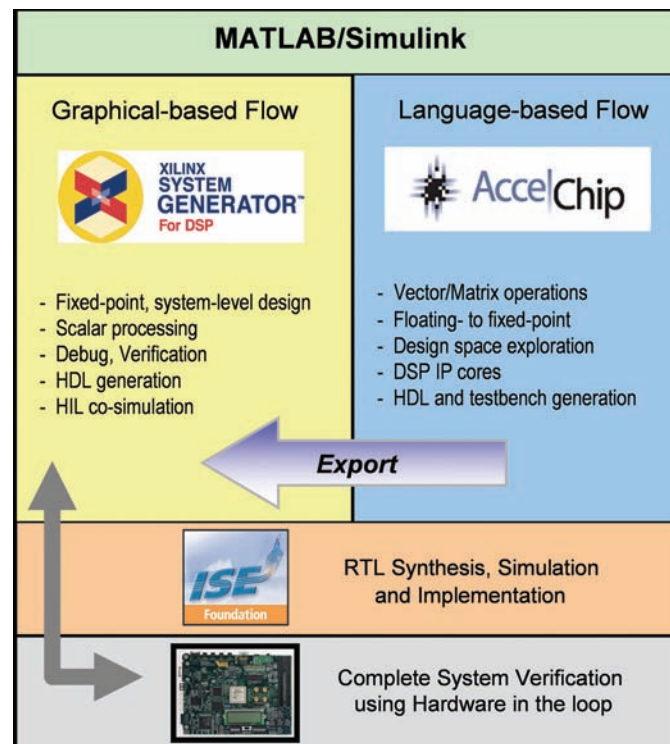


Figure 1 – System Generator/AccelChip interface

AccelChip's DSP Synthesis tool augments System Generator by providing a seamless integration path for algorithm developers, enabling the rapid creation of IP blocks, directly from M-files, that enhance the Xilinx block set in System Generator. In addition, AccelChip has optional AccelWare toolkits that complement System Generator with additional IP cores optimized for Xilinx cores. AccelWare toolkits include mathematical building blocks, signal processing, communications, and advanced math to implement linear algebra functions.

Kalman Filter Design Example

To illustrate this approach, let's take an advanced algorithm written in MATLAB, use AccelChip to synthesize the design, and then integrate it into a System Generator model. Our example is a Kalman filter – a recursive, adaptive filter well-suited to combining multiple noisy signals into a clearer signal (for details on the topic, see Arthur Gelb's book, "Applied Optimal Estimation").

Kalman filters embed a mathematical model of an object – such as a commercial aircraft being tracked by ground-based radar – and use the model to predict future behavior. Kalman filters then use measured signals (such as the signature of the aircraft returned to the radar receiver) to periodically correct the prediction.

```
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_est = p;
    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';
```

Figure 2 – Kalman filter example M-file

Figure 2 shows the MATLAB M-file describing the Kalman filter. 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 and corrects the estimate.

This algorithm illustrates the flexibility and conciseness of the MATLAB language. Common operators such as addition and subtraction operate on variables like the two-dimensional arrays **A** or **P_cap** without having to write loops, as you would in languages like C. Multiplication of two-dimensional arrays is automatically performed as matrix multiplication 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 completed using MATLAB's extensive linear algebra capabilities. Although such an algorithm could be constructed as a block diagram, doing so would obscure the algorithm structure so readily apparent in MATLAB.

With AccelChip, a first step in synthesizing a complete algorithm is to generate any major cores that are referenced – in this case, the matrix inverse indicated by the function call `inv(P_cap_est+R)`. But you can implement a matrix inverse in many ways; the choice of which method to use depends on the size, structure, and values of the matrix.

Using the matrix inverse IP core from the AccelWare toolkit, you can choose from micro-architectures designed for different applications. These micro-architectures can be optimized for speed, area, power, or noise. In this case, the most suitable approach is to use the AccelWare QR matrix inverse core.

Synthesizing RTL with AccelChip

With the MATLAB M-file loaded into AccelChip, the next step is to simulate the floating-point design to establish a baseline. You would then use AccelChip to convert the design to fixed-point math, verifying it in MATLAB as shown in Figure 3. AccelChip offers an array of tools to help you trim bits from the design and verify

fixed-point design effects like saturation and rounding. AccelChip aids in this process by propagating bit growth throughout the design and letting you use directives to set constraints on bit width. This algorithmic design exploration allows you to attain the ideal quantization that minimizes bit widths while managing overflows or underflows, allowing early trade-offs of silicon area versus performance metrics.

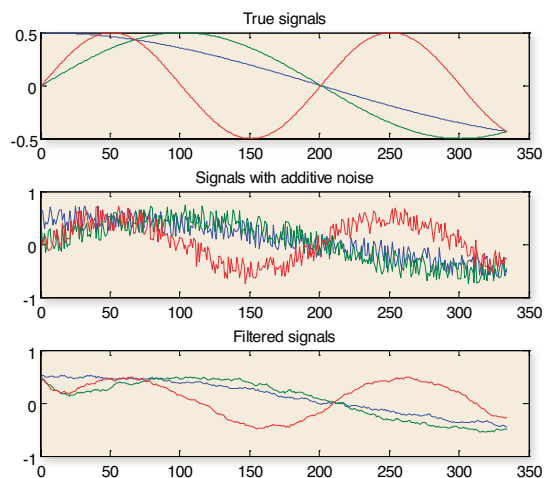


Figure 3 – The Kalman filter constructs estimates of the three signals based on input signals corrupted by noise.

Once you have attained suitable quantization, the next step is to generate RTL for your target Xilinx device. At this point, you can use the AccelChip GUI to set constraints on the design using the following design directives to achieve further optimizations:

- Rolling/unrolling of FOR loops
- Expansion of vector and matrix additions and multiplications
- RAM/ROM memory mapping of vectors and two-dimensional arrays
- Pipeline insertion
- Shift-register mapping

Using these directives constitutes hardware-based design exploration, allowing the design team to further improve quality of results. In synthesizing the RTL, AccelChip evaluates the entire design and schedules the entire algorithm, performing necessary boundary optimization in the process.

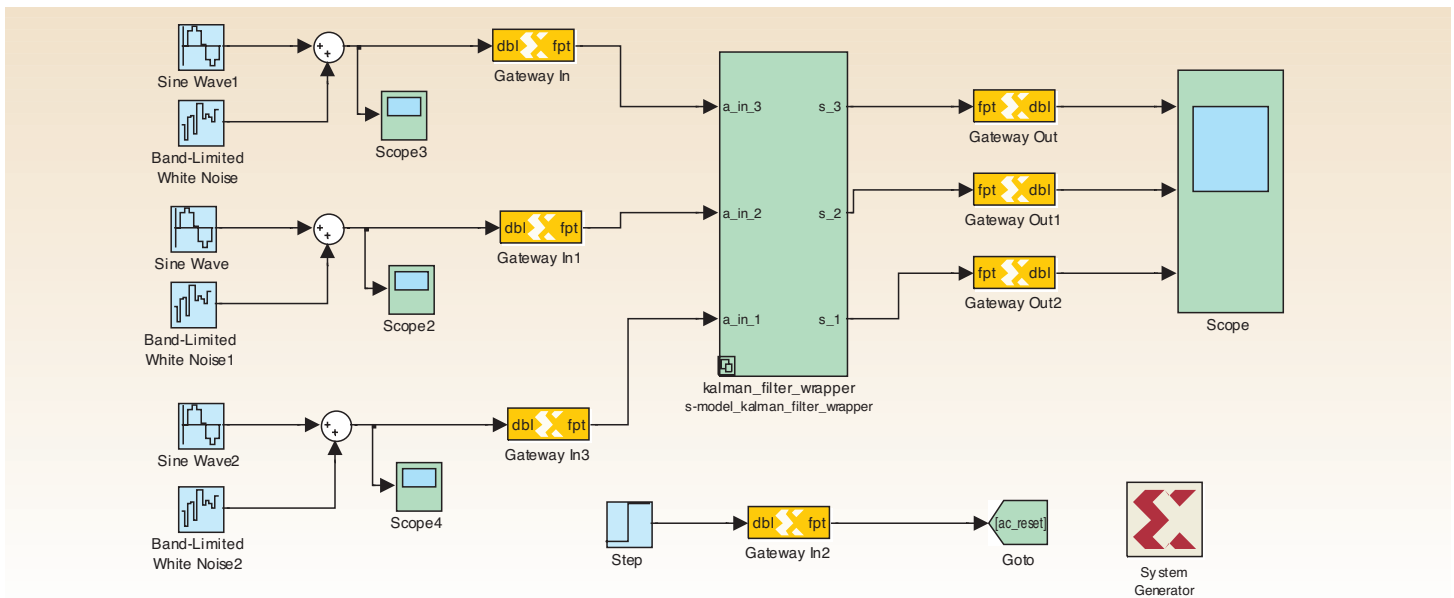


Figure 4 – AccelChip exports a cycle-accurate System Generator block that supports both simulation and RTL code generation.

Throughout this flow, AccelChip maintains a uniform verification environment through a self-checking test bench; the input/output vectors that were generated when verifying the fixed-point MATLAB design are used to verify the generated RTL. The RTL verification step also gives AccelChip the information necessary to compute the throughput and latency of the Kalman filter. This is essential information to assess whether the design meets specifications and is critical for achieving cycle-accurate simulation.

Exporting from AccelChip to System Generator

Although RTL verification is a key step in the design flow, designers want to see algorithms running in hardware. System Generator’s hardware-in-the-loop co-simulation interfaces make this a push-button flow, allowing you to bring the full power of MATLAB and Simulink analysis functions to hardware verification.

Now that you have run RTL verification in AccelChip, you are ready to export the AccelChip design to System Generator by going to the “Export” pull-down menu in the AccelChip GUI and selecting “System Generator.” AccelChip then generates a cycle-accurate System Generator block that supports both simulation and RTL code generation.

At this point, you transition the design flow to System Generator, where a new

block for the Kalman filter is available in the Simulink library browser. You need only select the Kalman filter block and drag it into the destination model to incorporate the AccelChip-generated Kalman filter into a System Generator design.

Figure 4 shows the resulting diagram for the Kalman filter. In the center of the System Generator diagram is the Kalman filter, and around it are the gateway blocks needed for a System Generator design.

Once the AccelChip-generated block is included in the System Generator design, you can perform a complete, system-level simulation of cycle-accurate, bit-true models to verify that the system meets specifications. You can use AccelChip-generated blocks for System Generator in conjunction with the Xilinx block set. Once this system-level verification step is completed, the next step in the System Generator flow is to move on to design implementation. The “Generate” step in System Generator compiles the design into hardware.

Verification Options

All design files generated by AccelChip, including exported System Generator files, are verified back to the original “golden” source MATLAB M-file. AccelChip’s verification approach is based on the generation of a test bench from the MATLAB source – this test bench is applied at the RTL level within AccelChip and can be

applied in System Generator to verify the correctness of the design.

Once verified in the System Generator environment, you can verify the AccelChip-generated block using System Generator’s supported methods – including HDL co-simulation and hardware-in-the-loop – to accelerate hardware-level simulation 10 to 100 times.

Conclusion

The integration of AccelChip with Xilinx System Generator is the first solution to unite MATLAB-based algorithmic synthesis favored by algorithm developers with the graphical design flow used by system architects and hardware designers. It uses the rich MATLAB language and its companion toolboxes to create System Generator IP blocks of complex DSP algorithms.

By using these tools together, design teams can employ the most productive means of modeling hardware for implementation, fully involving algorithm developers in the FPGA design process and completing higher quality designs more quickly.

For more information on AccelChip and its interface to Xilinx System Generator, visit www.accelchip.com. For more information on System Generator, visit www.xilinx.com/products/design_resources/dsp_centralgrouping/index.htm.