

# Using MATLAB to create an Executable Specification for Hardware

## *Part 1 – Describing Hardware Systems using MATLAB*

*Robert Anderson, DSP Tools Marketing, Xilinx, Inc.*

### **Introduction**

This article is the first in a three part series describing techniques and methods for using MATLAB as an executable specification for hardware development. MATLAB, the defacto industry standard for DSP algorithm development, includes abstractions such as complex numbers, matrix operations, built-in and toolbox libraries of DSP functions and waveform analysis that make this language highly useful for DSP algorithm development. These abstractions, however, create a significant gap between algorithmic description and the system hardware.

So why use MATLAB as an executable specification for hardware when other options exist such as Simulink, C or RTL? The answer is verification. When the application is signal processing MATLAB offers a vastly superior environment for input vector generation and output waveform analysis. Using MATLAB allows the algorithm developer to create a model that more accurately describes the characteristics of the final hardware implementation. This could involve incorporating the finite precision effects of fixed-point arithmetic or modeling the effects of specific numerical methods used such as a CORDIC implementation of a trigonometric function. This flow enables the algorithm developer to reduce design iterations, shorten development cycles and assert more control and influence over the hardware development process.

### **MATLAB as an Executable Specification**

If MATLAB is to be used as an executable specification for hardware then does the code need to be modified to avoid the use of abstract constructs such as complex numbers matrix operations, or built-in functions? The answer depends on how much hardware detail needs to be revealed. For example, if an algorithm is sensitive to the reduced precision effects of fixed-point numbers, then a hardware accurate fixed-point model needs to be created that accounts for word lengths of all variables in the design, and the details of the hardware architecture. For example, the fixed-point effects of a CORDIC based divider are very different from the fixed-point effects of a lookup table based approach yet the MATLAB language only provides the backslash “/” operator that does not accurately represent either of these numerical approaches.

The remainder of this paper will discuss coding styles that facilitate the use of MATLAB as an effective executable specification language. Also discussed are co-simulation methods, and rapid prototyping flows that are useful for hardware feasibility analysis, verification acceleration, proof of concept and rapid production.

### **Partitioning MATLAB code between Hardware and the Test-Bench**

When creating a MATLAB executable specification the hardware must be clearly distinguished from the testbench. Typically, MATLAB algorithms are not written with

this distinction but rather intersperse algorithmic calculations with analysis code. This distinction can easily be made by inserting a few simple comments like “hardware starts here” and “hardware ends here” A better method is to use a function. This coding style serves to clearly identify the portion of the MATLAB code being targeted to hardware and will facilitate the use of an automated MATLAB to FPGA design flow if so desired. The remainder of the MATLAB code becomes the test-harness and serves to pass data into the hardware function and analyze the results.

Following is an example of the coding style described above:

```
f = imread(<image file>);
F = my_fft_function(double(f)); % Design function for hardware
F2 = log(abs(F));
imshow(F2,[-1 5], 'InitialMagnification','fit');
colormap(jet);
colorbar;
```

## Defining Hardware Accurate Dataflow

Consideration should also be given to restructuring the MATLAB model to reflect the actual hardware dataflow. MATLAB algorithms are typically written to operate on arrays or vectors of data. Often this yields the best simulation run time performance and fewer lines of code but this is generally inconsistent with the behavior of the hardware which tends to process data in a streaming fashion. To account for this behavior the design function, described in the previous paragraph can be placed into a loop that “streams” the data to the function inputs and records the data from the function outputs on a cycle by cycle basis. This is referred to as a “streaming loop” with an example provided below.

```
A=rand(1,100)*2^(1)*0+1;
B=rand(1,100)*2^(10);

% Define streaming loop %
for n = 1:100
    [outputC_All(1:1,n)] = rdivide_df(A(1:1,n),B(1:1,n));
end

% Create reference signal to compare results against %
referenceIdeal = A./B;
inf_loc = find(B==0);
```

The advantages of using a streaming loop are twofold. First the MATLAB testbench becomes an element of the hardware validation process. Test vectors can be manually generated and saved in a hardware accurate format that can then be used in Simulink, C/C++ or RTL simulations. Second, hardware verification can be automated by using one of a small number of 3<sup>rd</sup> party co-simulation flows that connect directly to the MATLAB simulation. These options will be discussed in greater detail later in this paper.

## Arrays, Matrices and Complex Numbers

The use of vector, matrix, and imaginary numbers are prevalent throughout a range of signal processing applications and are widely used in communications, military, medical, imaging, and other application spaces. The availability of these powerful abstractions is a key reason why MATLAB is deemed superior to languages like C or C++ for these

modeling tasks. But can these abstractions be used in an executable specification for hardware?

There are limited hardware architecture options available for these types of arithmetic operations and therefore it is not necessary to include this level of detail in an executable spec for the purpose of asserting control over implementation. That being said, however, if an accurate fixed-point representation is necessary to account for the quantization effects then it will be necessary for all of the arithmetic constructs to be bit accurate. In that case the abstract coding styles intrinsic to floating point MATLAB must be replaced with their fixed-point equivalents in the executable specification.

### **Functions and toolboxes**

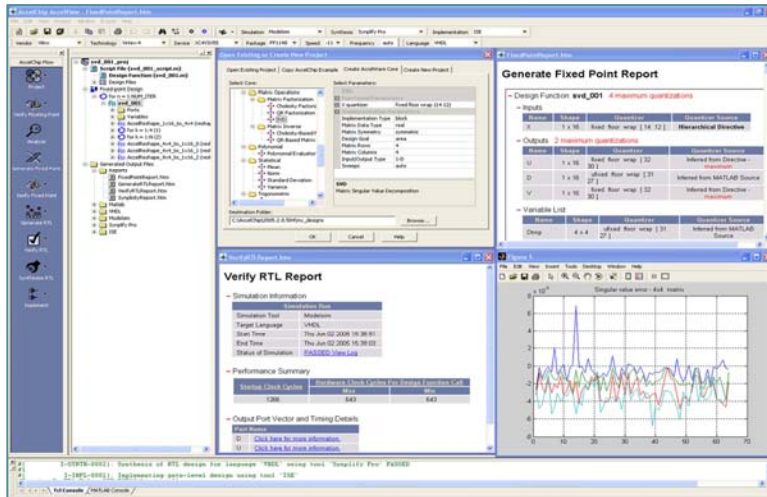
MATLAB's rich set of built-in and toolbox functions extend the abstraction of the language beyond the basic operations described above. These functions range from the elementary functions such as division, to trigonometric functions such as `sin()`, `cos()` and `tan()`, and extend to higher level application specific functions available through various MATLAB toolbox options. In general, these built-in and toolbox functions do not provide an accurate representation of any corresponding hardware implementation. These functions are developed to be numerically exact and are not designed to exhibit trade-offs that reflect the realities of a hardware implementation. At some point in the design process, they will need to be replaced with hand coded versions that accurately describe the hardware architecture of the implementation. For example the use of a `sin()` function would have to be replaced with a user created function call to something like `“sin_cordic()”` or `“sin_lut()”`.

### **Considering the use of Automatic Fixed-Point Model Generation**

The need for an accurate fixed-point model to creates a dilemma for the MATLAB programmer. If each use of a matrix, vector or complex operation or a function call must be removed then the value of using MATLAB as a golden hardware executable specification seems significantly diminished. If this is the case then suddenly the idea of using an automated floating-point MATLAB to fixed-point model generation tool becomes compelling.

Although the original floating point MATLAB description is indispensable, it is the fixed-point representation that is needed for hardware system verification. For this flow to be effective and useful it must support the use of matrix and vector operations, complex number operations, and at least a basic level of support for built-in and toolbox functions. A developer should not have to perform extensive research on numerical methods to implement division, or complex vector arithmetic but instead should have the freedom just to use the appropriate abstractions (such as the backslash) in their code.

The AccelDSP Synthesis tool from Xilinx includes the capability to automatically generate a fixed-point C/C++ model from a floating-point MATLAB model for the design function that works with the original MATLAB testbench. MATLAB language support includes vector and matrix operations, complex numbers and a rich set of built-in and toolbox functions commonly used in DSP algorithms.



**Figure 1 – Xilinx AccelDSP Synthesis Tool**

To aid the developer in this effort, AccelDSP includes a graphically assisted floating to fixed point conversion capability that propagates bit growth throughout the model. Word-lengths can be adjusted as necessary by the user such as trimming fractional bits after a multiplication.

The initial fixed-point model obtained through auto-quantization provides a good starting point but refinements to this model are generally necessary. This process is iterative and ultimately depends on the algorithm developer to analyze the fixed point performance of the model, and adjust the quantization accordingly to address any fixed point performance issues uncovered during fixed point simulation.

AccelDSP provides a window onto the fixed-point design through an interactive fixed-point report that lists the shape and quantization for each variable in the model. The fixed point report gives the user graphical access to a properties editor that allows the word-length for each variable in the model to be controlled.

Variable List

Name	Shape	Quantizer	Quantizer Source
LMS_SIZE	1 x 1	ufixed floor wrap [ 3 0 ]	Auto Inferred
MU	1 x 1	ufixed floor wrap [ 7 7 ]	Auto Inferred
dhat_del	1 x 1	fixed floor wrap [ 16 8 ]	<b>Directive</b> (delete)
hp0	1 x 4	fixed floor wrap [ 13 11 ]	<b>Directive</b> (delete)
hp1	1 x 4	fixed floor wrap [ 14 12 ]	<b>Directive</b> (delete)
rxp0	1 x 5	fixed floor wrap [ 14 12 ]	<b>Directive</b> (delete)
rxp1	1 x 5	fixed floor wrap [ 14 11 ]	<b>Directive</b> (delete)

**Figure 2 – AccelDSP Interactive Fixed-Point Report**

### Using the MATLAB Executable Specification for Hardware Validation

Assuming a fixed-point executable specification of the model exists that is a bit accurate representation of the hardware what is the best way to validate the hardware? The most

common method is to generate test vector files from MATLAB and pass these to the FPGA designer. The designer subsequently develops the RTL code for implementation on the FPGA and then passes back a set of additional test vectors for analysis in the MATLAB environment. This is a tried and true flow but other options now exist that realize significant time savings and even provide an automated path to hardware.

One option is to verify the RTL directly against the MATLAB testbench using an option to MATLAB called “EDA Simulator Link™ available from The Mathworks. This option allows an RTL model to co-simulate directly with the MATLAB testbench using a variety of commercially available RTL simulators such as ModelSim® from Mentor Graphics. Although functionally accurate this verification method would have marginal simulation performance and would not be practical for large test vector sets.

Another option would be to use MATLAB Hardware-in-the-loop Co-simulation provided as part of the Xilinx System Generator for DSP development environment. An interface is provided to allow MATLAB generated vectors to be passed from the testbench to a Xilinx device running on a supported hardware platform then back out again. This interface is accessed by adding a few lines of MATLAB function calls from a Xilinx provided library to the testbench code. This verification flow can improve simulation performance for computationally intensive designs up to 1000x.

### **MATLAB to FPGA Rapid Prototyping Flow**

The flow described above requires that the executable spec be turned into an equivalent hardware model that is suitable for implementation. This can be done by recreating the model using RTL or Simulink through a tool such as Xilinx System Generator or HDL Coder from The Mathworks. The Xilinx AccelDSP Synthesis tool provides a direct path from MATLAB to hardware. This rapid path to hardware can be used by algorithm developers with limited FPGA design backgrounds for feasibility analysis, proof of concept, hardware demonstration or early production.

### **Summary**

MATLAB is can be used effectively to create an executable specification for hardware development. By taking hardware into consideration the MATLAB model can more closely reflect the actual response of the hardware. Doing so closes the gap between algorithm and hardware development to reduce design iterations and development times. Tools such as AccelDSP can help automate this flow and facilitate the use of rapid prototyping and hardware verification.