Integrating FinSimMath in an existing Design Flow

1. Introduction

1.1) Existing Design Flow

Let us assume that the existing design flow consists of:

a) Develop mathematical algorithm using Mathematica, m-language, mathCAD, Simplex, or any other similar high level mathematically oriented environment.

b) Convert computations expressed in High Level Math into gate-level Verilog

This document will describe how FinSimMath supports the various tasks necessary when converting the high level mathematical algorithm into gate-level Verilog and will provide as an example, the implementation of an FIR Filter.

1.2) Using FinSimMath

FinSimMath provides the following benefits:

a) Helps find the optimal representation (fixed point or floating point), as well as the sizes of fields for each operand. A simple example exploring the design space for the best format and size of operands is provided in www.fintronic.com/buterworth.html.

b) Helps transform the high level description of the computation into a network of resources (i.e. adders, multipliers, memory, etc) in small steps, by supporting mixed mathematical and Verilog expressions that compare the results produced by the high level mathematical description with the low level Verilog implementation.

c) Helps perform fast simulations, because all the interfaces of modules are described at the Verilog bit level and therefore modules such as multipliers can be described high level yet maintaining bit-accuracy.

d) Helps generate the input stimulus necessary to test both the high level description and the low level implementation

e) Helps write the test bench by connecting the input generation with the module under test and by providing the mathematical capabilities to process and display the results. e.g. fft, distance between arrays, graphical representations, etc.

1.3 Name of files referenced in this document

The FinSimMath descriptions in this document could have been written manually, but they were generated automatically each from a high level specification (a .cf file) and from a resource specification (a .res file) by FinFilter, an FIR filter generator. FinFilter produces a .v file containing the test bench and the generated filter, a .rep file indicating the costs of the design, and three graphs: amplitude response, ionput-output plots and input output spectrum.

The resource file is the same in all cases, Ex1.res and is described in www.fintronic.com/Ex1_res.pdf.

The first example is run by the script rmlt which is available in www.fintronic.com/rmlt.pdf . It uses the specification file f_mlt.cf which is available in www.fintronic.com/f_mlt_cf.html . This example produces a report presented in www.fintronic.com/firGTL_mlt_rep.pdf , and the three graphs presented in www.fintronic.com/mlt_Gain_c.html , www.fintronic.com/mlt_Input_Output_c.html , and www.fintronic.com/mlt_ioSpectrum_c.html .

The second example is run by the script radd which is available in www.fintronic.com/radd.pdf . It uses the specification file f_mlt.cf which is available in www.fintronic.com/f_add_cf.html . This example produces a report presented in www.fintronic.com/firGTL_add_rep.pdf , and the three graphs presented in www.fintronic.com/add_Gain_c.html, www.fintronic.com/add_Input_Output_c.html, and www.fintronic.com/add_ioSpectrum_c.html .

The third example is run by the script radd_decimate_interpolate which is available in www.fintronic.com/radd_decimate_interpolate.pdf. It uses the specification file f_add_decimate_interpolate.cf which is available in www.fintronic.com/f_add_decimate_interpolate_cf.html . This example produces a report presented in www.fintronic.com/firGTL_add_decimate_interpolate_rep.pdf, and the three graphs presented in www.fintronic.com/add_decimate_interpolate_Gain_c.html, www.fintronic.com/add_decimate_interpolate_Input_Output_c.html, and www.fintronic.com/add_decimate_interpolate_ioSpectrum_c.html .

Each of the examples represents a transformation step in exploring the design space.

2. High Level Math to FinSimMath

2.1) High Level Description

After having identified the stimulus to be used one must translate the high level model into FinSimMath. The general considerations regarding the usage of FinSimMath in order to convert a high level mathematical algorithm into gate-level Verilog will be exemplified using the implementation of an FIR Filter.

An FIR Filter performs at each clock cycle the scalar product of an array of N coefficients with the N point history of the sampled input. The scalar product is the sum of the N products involving one coefficient and one sampled input in a given order.

Therefore the high level description of the algorithm under consideration is:

```
for (i=0; i < NrSamples; i++) {
  output(i) = 0;
  for (j=0; j <  N; j++ {
    if (i >= j) {
      output(i) =  output(i) + a(j) * Input(i-j);
    }
```

```
    }
  }
```

First, one must determine the coefficients a(0) through a(N) which will attenuate the desired frequencies while keeping unchanged (with a ripple under a certain limit) other frequencies. In this example which was produced automatically by FinFilter, the coefficients are produced by FinFilter, as well. Alternatively, the coefficients can be obtained by other means.

2.2 High Level FinSimMath Sequential Algorithmic Description

The first version of the translation should be a construct to construct translation from a high level description of the algorithm to FinSimMath. This is easy to write using as data type the type real, which is 64 bit floating point. This model runs the fastest and produces results that are close to the ones produced by Mathematica. This first version of the FinSimMath description of the filter shall contain also the generation of input with which to test the computations. We will not go into detail here but the reader can look at www.fintronic.com/buterworth.html where such a high level FinSimMath description is presented for an IIR Filter (scalar product where coefficients are multiplied with sampled inputs from previous time points as well as from future time points).

Subsequent versions may contain more than one thread. Multiple threads are supported by the always block available in Verilog/FinSimMath. Also, subsequent versions may contain multiple modules, corresponding to the partitioning of the hardware implementation.

2.3 FinSimMath description using multiple modules

An important point to remember is that modules containing high level descriptions as well as modules containing low level descriptions communicate with each other using ports that contain one or more Verilog bits. Some collection of bits may be interpreted according to a certain format and size of fields by using the system function $VpCopyReg2Vp, as can be seen in the modules describing computational resources (adder, multipliers, etc.). Due to this feature, high level modules can be replaced by low level modules and vice versa, resulting in very fast simulations, because most of the modules will be simulated at the high level.

The description of the design consists of a top module, the test bench, that instantiates the device under test (the description of the FIR filter in this example), as well as the modules generating the input to the filter. In turn the description of the filter consists of instantiation of various module some of them being computational resources such as adders and multipliers and others being pipeline delays (delaying data by a certain number of clock cycles.

2.3 FinSimMath Gate Level Description using adders and multipliers

2.3.1 Introduction

An implementation of the FIR filter using adders and multipliers is available in firGTL_mlt.v which is described in www.fintronic.com/firGTL_mlt.pdf .

2.3.2 Design Issues addressed

This description addressed several design issues:

a) The high level mathematical description assumes that all the data is available in an input array. In getting a step closer to the actual implementation, we need to address the fact that the data arrives one sample at a time and that the filter needs to process the data as soon as possible in order to provide the result as soon as possible. Therefore at each internal clock cycle data must be rovided by the test bench modeling the data that will reach the actual circuit. Note that the data must be provided in several samples at a time because the sampling rate is higher than the rate of the internal clock. This code can be found starting on line 184.

   Given that the filter produces several output values at each cycle of the internal clock the test bench must serialize the output of the filter so that one value is produced at each cycle of the output clock. The code in the test bench producing the serialization of the output starts on line 240.

b) The filter must be partitioned in several sub modules. A memory model is described starting on line 426. This model needs more work in order to use an existing memory block model because of the large number of data read ports. What is needed is to multiplex an actual memory block using a faster clock or to replicate several memory blocks having fewer data read ports or a combination of the two solutions. The memory stores a history of samples and is instantiated in module fir which instantiates also as many processing units as necessary to process all samples that need to be processed at one time.

c) The processing units must be partitioned into basic resources such as adders and multipliers. This code for one processing unit can be can be found in module tu, starting with line 1232 and instances of processing unit can be found in module fir.

d) The exact interface for each module, including clocks, reset, data enable where necessary, and data must be determined based on the given design methodology used.

e) Given that the coefficients are symmetric in this case, the number of multiplications is reduced by first adding the samples corresponding to symmetric coefficients and then performing a reduced number of multiplications.

f) Given that the coefficients are constant, i.e. the filter being implemented uses always the same coefficients, the constants can be wired at the inputs of the multipliers rather than being stored in registers.

2.3.3 Test Bench issues

This description addressed several test bench issues:

a) The test bench instantiates a module providing input having the frequency that will not be attenuated and another module providing a waveform that is the sum of two frequencies: one which will be attenuated and the other one identical to the one produced by the previously discussed module, frequency that will not be attenuated, and which represents the ideal output

of the filter. The two modules are described starting with lines 3 and 51 respectively and are instantiated in the test bench on lines 182 and 181, respectively.

An important aspect is that the model needs to be such that the sampled values should be at times consistent with the sampling rate and with the timescale. FinFilter does that automatically, but in general this has to be done manually.

b) The clock generation for the sampling clock, the internal clock and the output clock is generated inside the test bench beginning with line 139. It has to be consistent with the timescale and the respective clock rates.

c) The test bench produces a graph that shows the amplitude response of the filter. The code of the task computing the amplitude response starts on line 158 and the task is invoked on line 346.

d) The module implementing the filter is instantiated in the test bench on line 180.

e) The test bench contains code that displays graphically the input, output, performs FFT and displays graphically the results of the FFT. This code starts on line 349.

f) The test bench contains code that compares the high level bit-accurate computation to the low level computation. This code starts on line 374.

2.3.4 Bit accurate models of resources.

This code starts on line 473. Note that the interfaces of the bit accurate models of resources consist only simple Verilog bits which makes it straight forward to replace them with low level implementations of the resources.

2.4 FinSimMath Gate Level Description using only adder instead of multipliers.

2.4.1 Introduction

An implementation of the FIR filter using adders and multipliers is available in firGTL_add.v described in www.fintronic.com/firGTL_add.html . In addition to the issues addressed in the firGTL_mlt.v, firGTL_add.v addresses an additional design optimization issue.

Given that the coefficients are constants, i.e. the filter being implemented uses always the same coefficients, the multiplication with such constants can be replaced by just the necessary additions and or subtractions. Under certain conditions the implementation cost is reduced significantly with this approach.

Starting with line 1310 one can find the implementation of modules that are used in replacing the multiplications in the scalar product. These modules are used in the processing unit (module tu) which is described starting with line 1392.

2.5 FinSimMath Gate Level Description using only adders and decimation with interpolation.

2.5.1 Introduction

An implementation of the FIR filter using adders and no multipliers and which is using decimation and interpolation is available in firGTL_add_decimate_interpolate.v, described in www.fintronic.com/firGTL_add_decimate_interpolate.html .

In addition to the issues addressed in the firGTL_mlt.v, and firGTL_add.v , firGTL_add_decimate_interpolate.v addresses an additional design optimization issue.

This description addressed the issue of decimation and interpolation. The differences with the previous version (described in www.fintronic.com/firGTL_add.pdf) are only in module fir, where one processing unit is decimated and interpolation hardware is added, beginning with line 1478.