Running Simulink-based Designs on SRC-6

David Meixner, Volodymyr Kindratenko§, David Pointer
National Center for Supercomputing Applications (NCSA), University of Illinois at Urbana-Champaign (UIUC)
E-mail addresses: dmeixner@uiuc.edu, kindr@ncsa.uiuc.edu, pointer@ncsa.uiuc.edu

Introduction

This paper provides details on how to program an SRC-6 reconfigurable computer [1] using MathWorks Simulink® [2] with Xilinx System Generator™ for DSP [3] and Xilinx Blockset [4]. SRC-6 is programmed in SRC's MAP C programming language [5]. Code development on the SRC-6 platform closely resembles code development on a conventional microprocessor-based system, except for explicit code to support data transfer between the system memory and FPGA-controlled memory. SRC's Carte™ development environment [5] allows the developer to bring in third party subroutines, called macros, that can be used to extend the functionality of the original language. These macros are typically implemented in Verilog Hardware Description Language (HDL) and are brought into the MAP C program via a configuration file that defines the interface between the macros and MAP C language. We propose a method of using the Verilog source for SRC macros generated from the MathWork's Simulink-based designs.

The ability to introduce Simulink-based designs into the Carte programming environment opens up new possibilities in programming the SRC-6 system. Thus, one can explore system-level designs that use FPGA resources (e.g., BRAM) directly. One can also use IP cores, such as FFT and CORDIC algorithms, provided by Xilinx without the need to re-write them in MAP C. Another advantage in using Simulink-based designs is the ability to use fixed point arithmetic and define arbitrary sized data types, which is not directly available in the MAP C environment. The latter ability may lead to reduced FPGA resources utilization as one can avoid the need to use larger numerical types for problems that require a reduced numerical range.

Simulink model design

The Simulink model one wishes to incorporate into SRC-6's Carte framework should be created using the Xilinx Blockset for Simulink. Figure 1 shows a simple Simulink example which takes three inputs: *a*, *b*, and *c*, and outputs *q=(a+b)*c*. For this example, all signals have a bit width of 40 and a binary point of 30. The input and output ports are '*gateway in*' and '*gateway out*' blocks, respectively. They should be labeled in lowercase letters using the same names as the variables to be used in the MAP C code. Note that Xilinx System Generator will convert all variables to lowercase. The '*gateway in*' blocks also allow specification of the bit width and binary point of the inputs; more on this follows.

§corresponding author

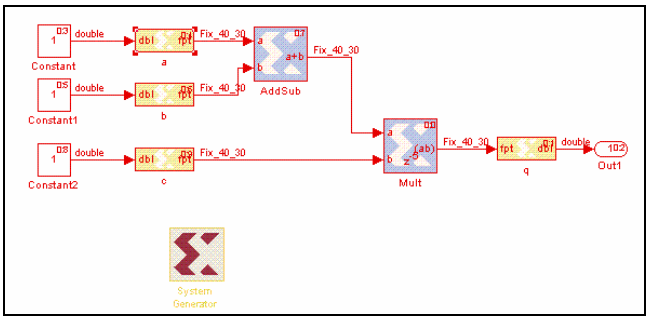

Figure 1: Simple Simulink Example, addmult.mdl.

The next step is to set up and run the System Generator. Its parameters should be set up as follows:

- Compilation Type: HDL Netlist
- Part: the FPGA to be used, e.g., Virtex2 xc2v6000-4ff1517 for MAPC
- Target Directory: the directory where the Verilog files will be output
- Hardware Description Language: Verilog
- FPGA Clock period (ns): 10

Once the model generation is complete, several files will be created (here <*design*> is the name of the model):

- <*design*>_files.v – contains most of the HDL for the design.
- <*design*>_clk_wrapper.v – an HDL wrapper that drives clocks and clock enables.
- conv_pkg.v – contains constants and functions used by <*design*>_fils.v.
- .edn files – implementation of the parts of the design.

In the above example, the following files were produced: addmult_files.v, addmult_clk_wrapper.v, conv_pkg.v, adder_subtracter_virtex2_7_0_84f1dba84ee809b9.edn, and multiplier_virtex2_7_0_b018b3a1b259a550.edn. These files need to be copied into the macro directory of the MAP C implementation.

MAP C design

First, <*design*>_clk_wrapper.v file needs to be modified to include <*deisgn*>_files.v in addition to conv_pkg.v:

```
--System Generator code here--

`include "conv_pkg.v"
`include "addmult_files.v" /* this is the line that was added */

module addmult_clk_wrapper (a, b, c, ce, ce_clr, clk, q);

--System Generator code here--
```

Next, a black box definition for the design is created. It includes the inputs and outputs defined in the Simulink model, as well as the signals *ce*, *ce_clr*, and *clk*, created by the System Generator. If the design does not have any delays, the signals *ce*, *ce_clr*, and *clk* will not be generated, and should therefore not be included. For this example, the multiplier has a delay of 5, so the clock signals are generated. The black box definition appears as follows (remember that we are using 40-bit fixed point numbers):

```
module addmult_clk_wrapper (a, b, c, ce, ce_clr, clk, q);
   input [39:0] a;
   input [39:0] b;
   input [39:0] c;
   input ce;
   input ce_clr;
   input clk;
   output [39:0] q;
endmodule
```

The next step is to create the info file. This file links the operators and calls from the source program to macros and signal names in the HDL code. Note that the inputs and outputs are 64-bits in the source code, but we only use the bottom 40-bits in our HDL code. The *clk* signal should always be mapped to CLOCK. Lastly, the debug function performs the same operation as our Verilog code, except the result is shifted 20 bits to the right as is required for proper fixed point multiplication. The resultant info file is as follows:

```
BEGIN_DEF "addmult"
   MACRO = "addmult_clk_wrapper";
   STATEFUL = NO;
   EXTERNAL = NO;
   PIPELINED = YES;
   LATENCY = 5;

   INPUTS = 3:
     I0 = INT 64 BITS (a[39:0])  // explicit input
     I1 = INT 64 BITS (b[39:0])  // explicit input
     I2 = INT 64 BITS (c[39:0])  // explicit input
     ;

   OUTPUTS = 1:
     O0 = INT 64 BITS (q[39:0])  // explicit output
     ;

   IN_SIGNAL : 1 BITS "ce"     = "1'b1";
   IN_SIGNAL : 1 BITS "ce_clr" = "1'b0";
   IN_SIGNAL : 1 BITS "clk"    = "CLOCK";

   DEBUG_HEADER = #
      void addmult__dbg (long long v0, long long v1, long
long v2, long long *res);
   #;

   DEBUG_FUNC = #
      void addmult__dbg (long long v0, long long v1, long
long v2, long long *res) { *res = (v0+v1)*v2 >> 20; }
   #;

END_DEF
```

Finally, we add the path for <*design*>_clk_wrapper.v to the MACROS line of the Makefile. For this example, the line would read:

```
MACROS = macros/addmult_clk_wrapper.v
```

Now the resulting macro can be called from MAP C code:

```
addmult(a, b, c, &q);
```

Numerical conversion

Since Simulink performs all calculations with fixed point arithmetic, floating point numbers must be converted to fixed point representation in order to take advantage of variable resolution. This conversion can occur either on the microprocessor side before the data is transferred from the main memory to the MAP, or on the MAP before the data is used by the Simulink-based design. We have implemented subroutines both in C for the execution in the microprocessor-based code and in MAP C for the inclusion in the MAP's algorithm implementation.

When converting between floating point and fixed point numerical representations, there can be a loss of numerical resolution. The size and fractional precision of the fixed point numbers determine its precision, so a higher precision requires a larger size. This is a tradeoff that must be taken into consideration by the programmer.

Conclusions and future work

We have demonstrated how a Simulink-based design created with the help of Xilinx System Generator and Xilinx Blockset can be integrated with the native SRC's MAP C code. The Simulink example shown in Figure 1 was tested from within a MAP C code and was found to execute correctly. We are now in the process of testing more complex Simulink-based designs, such as FIR filter and FFT. Our goal is to enable a rapid development and integration of Simulink-based DSP models on SRC's Compact MAP Processor [6].

Acknowledgements

This work was funded by the National Science Foundation grant SCI 05-25308. We would like to thank Jon Huppenthal, David Caliga, Dan Poznanovic, and Jeff Hammes, all from SRC Computers Inc., for their help and support with SRC-6 system. Special thanks to Trish Barker from NCSA's Office of Public Affairs for help in preparing this publication.

References

[1] SRC Computers Inc., Colorado Springs, CO, SRC Systems and Servers Datasheet, 2005.

[2] http://www.mathworks.com/products/simulink/

[3] http://www.xilinx.com/ise/optional_prod/system_generator.htm

[4] http://www.xilinx.com/products/software/sysgen/blockset.htm

[5] SRC Computers Inc., Colorado Springs, CO, SRC C Programming Environment v 2.1 Guide, 2005.

[6] SRC Computers Inc., Colorado Springs, CO, Portable MAPstation™ Datasheet, 2005.