COMPARISON OF HIGH-LEVEL FPGA DESIGN TOOLS FOR A BPSK SIGNAL DETECTION APPLICATION

Jan Frigo (Los Alamos National Laboratory, Los Alamos, NM, USA, jfrigo@lanl.gov);
Tom Braun (Los Alamos National Laboratory, Los Alamos, NM, USA, tbraun@lanl.gov);
Joe Arrowood (Los Alamos National Laboratory, Los Alamos, NM, USA, arrowood@lanl.gov);
Maya Gokhale (Los Alamos National Laboratory, Los Alamos, NM, USA, maya@lanl.gov);

ABSTRACT

We investigate two approaches using high-level tools to map a signal processing algorithm to reconfigurable hardware. Our application presents the first step of a phase modulation sorter that locates binary phase shift keying (BPSK) signals in wide-band data. The first approach uses the Xilinx System Generator tool to model the system within Matlab/Simulink. These system modules are synthesized to hardware as Xilinx IP cores. The second approach uses the Streams-C language and compiler to write a high-level program (using mostly C code) consisting of software and hardware processes. We will discuss our experiences using these high-level tools in terms of their ease of use, and the accuracy of their functional simulators, and generated hardware. The designs utilized two Virtex 1000E FPGAs. The System Generator version of the application used 80% of the available area with a placement speed of 70 MHz, compared to 60% area utilization and a 60 MHz speed for Streams-C. We estimate a productivity improvement of 2X to 4X over manual hardware design depending on the complexity of the modification, and the stage of system development.

1. INTRODUCTION

The motivation for this study comes from the need to have a compliant hardware development system that allows developers the flexibility to change portions of the algorithm, while not impacting the schedule of the project or "shutting down" the system while these modifications are in-progress. There are numerous commercially available development tools for reconfigurable computing (RCC) systems aimed at reducing design time and making the algorithm-to-hardware transition less cumbersome. Some tools focus on dataflow design, have a graphical interface for algorithm development and target special-purpose hardware boards [1], while others require the developer to translate their algorithm into a cycle-by-cycle Glike representation and generate synthesizable VHDL (or Verilog) or a net list [2]. Still others [3] have algorithm development in a subset of C or Fortran, include mapping to specialized hardware boards and allow for high-level resource allocation on the hardware. We chose to evaluate Xilinx's System Generator for DSP [4] and the open source sc2 Streams-C compiler [5][1] in this study.

System Generator is a Xilinx software tool for designing, simulating, and implementing high performance FPGA-based DSP systems, and exists as a plug-in for Mathwork's Simulink. The VHDL output of the tool is a component similar to a Xilinx Coregen core, and can be instanced in any design or board model containing Xilinx Virtex or Virtex II FPGAs. System Generator allows bit-true simulation and system design to be done in Simulink, without the need for a HDL simulator.

The sc2 Streams-C tool targets algorithm mapping to hardware/software systems. Streams-C provides language-level support (a subset of C) for stream-oriented computation. The Streams-C programming model is that of communicating processes. A system consists of a collection of *processes* that communicate using *streams* and *signals*. Processes can run either in software on conventional processors or in hardware on FPGAs. The sc2 synthesis compiler compiles hardware processes into RTL VHDL. The software libraries provide a functional simulator and a runtime library. The sc2 compiler currently, targets the Annapolis Micro Systems Firebird board.

2. BPSK APPLICATION

BPSK is a form of Phase Shift Keying (PSK) modulation. In BPSK modulation, the phase of the RF carrier is shifted 180 degrees in accordance with a digital bit stream. A generalized representation may be written as

$$s(t) = A\cos[w_c t + pp(t) + q_0]$$

Where $p(t)$ is a binary switching function with possible states of 0 or 1 that represents the digital modulation. The term q_0 dictates the initial phase of the signal. A "one" causes a phase transition while a "zero" does not.

One characteristic of BPSK modulation is that squaring the data will produce a strong peak in the frequency domain, while the data itself contains no such peak. This is distinct

[1] The sc2 Streams-C compiler was developed at LANL.

from other forms of communication modulation such as QPSK (Quadrature Phase Shift Keying), QAM (Quadrature Amplitude Modulation) or FSK (Frequency Shift Keying). Squaring the above equation produces

$$s^2(t) = \frac{A}{2} + \frac{A}{2}\cos[2w_c t + 2pp(t) + 2q_0]$$

which simplifies to

$$s^2(t) = \frac{A}{2} + \frac{A}{2}\cos[2w_c t + 2q_0]$$

The switching function, $p(t)$, is now multiplied by 2π and thus does not provide a phase shift. Therefore, this squaring technique removes the digital modulation and the spectral spreading associated with it in favor of an easily identifiable spectral peak at double the carrier frequency. The test application will exploit this characteristic to search input data for the presence of a BPSK signal. Spectral plots shown in Figure 1 illustrate this property.

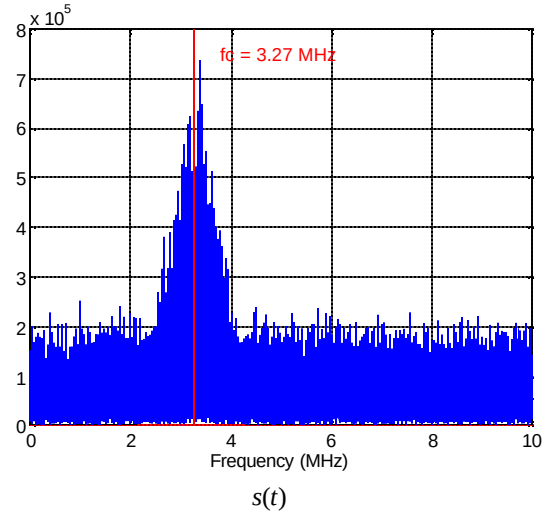

Figure 1. Spectral plots of $s(t)$ and $s^2(t)$

This application, therefore, consists of four parts: 1) calculating a FFT on the data and the square of the data. Only the magnitude of the output is considered, and DC and Nyquist bands are ignored. 2) Four FFTs are accumulated on a frequency bin by frequency bin basis to reduce the probability of false detection due to noise. 3) We threshold the accumulated FFT output based on its mean to find frequencies that have strong peaks. 4) Peaks are compared between the squared and non-squared data to determine if a peak in the squared data is a result of a BPSK signal or a continuous wave. The algorithm flow is shown in Figure 2.

This application was chosen for testing due to its complexity, as it requires a large amount of processing for the FFT, immediate data storage and accumulation of the FFT outputs, and control logic to handle the threshold and peak detection. For instance, due to aliasing and quantization, every time a peak is detected in the squared data, six frequency bins of non-squared data must be checked. Also, for comparison purposes we have a hand-coded (VHDL) version of this application.

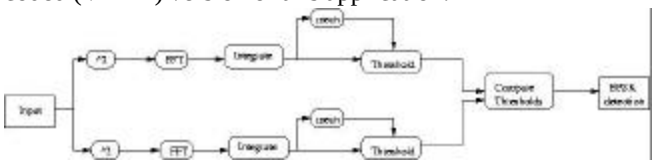

Figure 2. BPSK algorithm

3. SYSTEM GENERATOR TOOL

3.1. Introduction

System Generator is a Xilinx product that acts as an additional blockset for Mathwork's Simulink [6]. Simulink is a block diagram design tool for Matlab that allows time-based simulation and graphical design. Models (algorithms) are constructed via a graphical user interface (GUI) by dragging different blocks onto the workspace and connecting them. Most of the Matlab functions are available for use within Simulink. The system uses the standard Matlab workspace for file I/O.

The System Generator blocks closely correspond to Xilinx IP cores available in their Coregen product. It generates VHDL for each model by instancing the cores in automatically generated wrappers. These wrappers are then connected together. Thus, only blocks in the System Generator blockset can be converted to VHDL, although the whole range of Simulink blocks and Matlab functions may be used for testing and verification inside the Simulink simulation environment.

The System Generator blockset includes block implementation of most low-level and intermediate-level FPGA functions, such as adders, multipliers, muxes, registers, and counters. Selected higher-level building blocks are also available, such as FIR filters (including

support for Matlab's fdatool), FFTs, a CORDIC-based Cartesian to Polar converter, convolution encoder/decoder, and a soft microprocessor core.

For simulation, all Xilinx blocks act like native Simulink blocks. Simulation of the Xilinx blocks is both bit and cycle true. For synthesis, project files may be created for a user-specified (XST, Leonardo, or Synplicity) synthesis tool. Additionally, project files for HDL simulation (ModelSim CAD tool) are produced.

The VHDL produced by System Generator may be instanced as a component in a larger design. Alternatively, pin constraints may be specified for each port in the Simulink model in order to map it directly to the FPGA. Because of the reliance on Xilinx IP cores, the tool will only generate code for Xilinx Virtex, Virtex2, and Spartan FPGAs. Many of the high level blocks require the Virtex2.

Figure 3 shows the general design flow for using System Generator for hardware design. An example design-flow shown in Figure 4 is the mean calculation used in the BPSK discrimination algorithm.

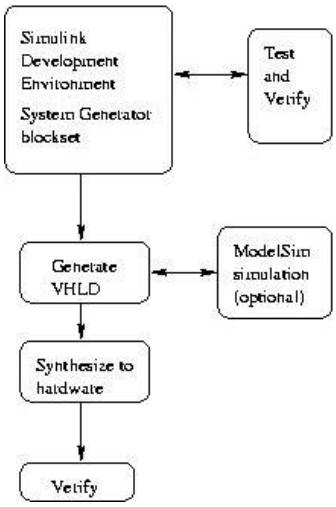

Figure 3. System Generator Design Flow

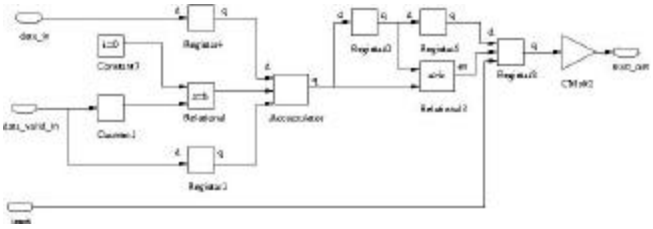

Figure 4. Example of System Generator Algorithm

3.2. System Generator: BPSK application mapping to hardware

The BPSK algorithm described in Section 2 was implemented with the System Generator blockset, including the FFT. Due to constraints with the tool (described in Section 3.3), it was necessary to build all the components from low-level blocks, such as adders, muxes, and multipliers. Subsystems were made for each of the components shown in Figure 2, and these were connected together to produce a design in Simulink. Emphasis was put on making the design capable of handling data in real time. Thus, the algorithm is fully pipelined to run continuously on the hardware (or for a given number of seconds in simulation), with a continuous stream of input data.

The performance results in Table 1. show the placed and routed design generated by System Generator was split across two Xilinx Virtex 1000E FPGAs and each chip met the clock speed constraint of 70 MHz. It used just over 19500 slices, which accounts for approximately 80% of the total available area. Once the learning curve on the tool was overcome, the total development time to map our BPSK algorithm to reconfigurable hardware was approximately 2 to 3 weeks. We estimate that a hand-coded version of this application in VHDL requires about 4 to 8 weeks of development time, thus, a productivity speed up of 2X to 4X.

3.3. Evaluation of System Generator

One might view System Generator as a high-level design tool in which high-level function blocks are connected together and the tool handles the remaining details. This view is inconsistent with our experience. While the design entry method is high-level, the designs themselves typically are not. Many of the high-level blocks were too constrained for use in our application. For example, the existing FFT block provided with System Generator can only handle 16-bit input and 16-bit output, and requires a continuous stream of input data without a data valid indicator. HDL designers must address these same constraints with respect to the Xilinx FFT core. Nevertheless, this caused us to build many designs entirely from the basic building blocks, blocks similar to the functionality already available in VHDL or Verilog. In addition, the amount of control logic necessary to handle delays and clocking often requires low-level manipulation by the user.

However, there are definite advantages to using System Generator instead of conventional HDL languages. Some of the high-level blocks in System Generator are accurate. The FIR filter is an appropriate example, which not only allows input of coefficients or the dynamic design of a filter in Matlab, but also will automatically analyze the coefficients for symmetry and optimize accordingly. Even where high-level blocks are not available, System Generator provides an efficient GUI for design entry. The design also makes it trivial to save common pieces of code (blocks) for later use, and it is a straightforward process to design these blocks to configure themselves automatically for varying bit widths and number of iterations. Thus, changes to an algorithm can be made quickly and easily with System Generator. Testing and verification is convenient with System Generator, since

the data is directly available in the Matlab workspace with all the Matlab data visualization tools. Finally, using System Generator inside Simulink for bit and cycle true simulations is an order of magnitude faster than running the same simulation through an HDL simulator. It could be argued that the decrease in time for testing and verification alone is worth the migration to System Generator. We estimate a 2X to 4X productivity improvement using System Generator over conventional HDL language development methods due to System Generator's design environment and simulation speed. We found the tool provides a nice balance between the amount of control capable in the design processes and the advanced design entry and data testing properties one would expect in a high-level tool.

4. STREAMS-C COMPILER

4.1. Introduction

Approaches for reducing design time for RCC applications have ranged from high-level optimization schemes [7] [8], to low-level [9], technology-specific, optimized designs. The Streams-C approach [10] targets algorithm mapping to hardware/software systems. Streams-C provides language-level support for stream-oriented computation. Characteristics of stream-oriented computing include high-data-rate flow of one or more data sources, fixed size, small stream payload (one byte to one word), compute-intensive operations, usually low precision fixed point on the data stream, access to small local memories holding coefficients and other constants, and occasional synchronization between computational phases.

The Streams-C programming model is that of communicating processes. A system consists of a collection of *processes* that communicate using *streams* and *signals*. Processes can run either in software on conventional processors (SP) or in hardware on FPGA processors (HP). The sc2 synthesis compiler compiles FPGA processes in hardware. The compiler translates a subset of C (e.g. generalized pointers or recursion are not supported) into Register-Transfer-Level (RTL) VHDL that is synthesizable on FPGAs. The compiler can pipeline loops, so that the generated hardware/software is capable of pipelining a streamed computation across multiple FPGAs and the conventional processor. In addition, the compiler can unroll loops.

A software library using POSIX threads provides concurrent processes and stream support in software. Thus the software libraries support a dual function: when all processes are mapped to software, the system provides a functional simulation environment for the hardware/software program. The library also provides a convenient, lightweight mechanism for parallel programming in software. When processes are mapped to a combination of software and hardware, the software libraries are used for communication among software processes and between software and hardware processes. Hardware libraries for the Annapolis Micro Systems (AMS) Firebird board, which contains one Xilinx Virtex-E FPGA on a 64-bit PCI bus, are used for communication among hardware processes and for the hardware side of communication to software processes. Figure 5 shows the software development flow for applications using the Streams-C compiler.

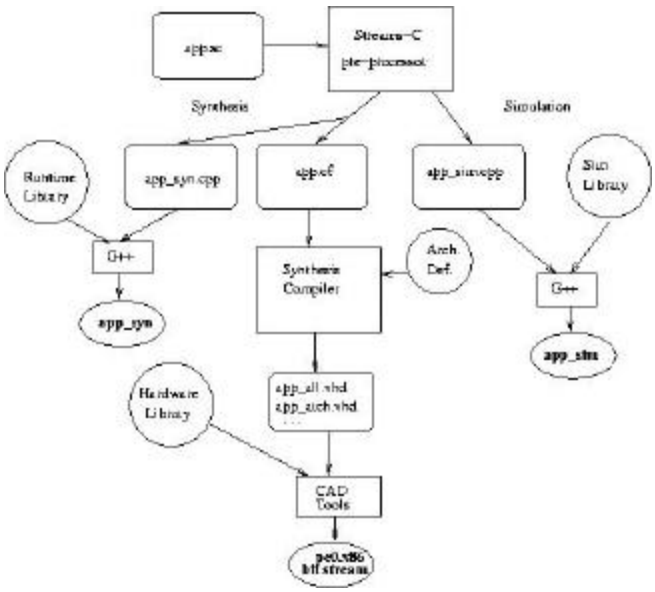

Figure 5. Streams-C compiler structure

4.2. Streams-C: BPSK application mapping to hardware

The Streams-C version of the BPSK algorithm as shown in Figure 6, has one software process, host1, and three hardware processes, fft, data_run, and squared_data_run. The host1 process fills external on-board memory with input data and initiates all the hardware processes once the memory load is complete. Host1 then waits for an input signal from the data_run process to indicate signal detection is complete. The fft hardware process reads data from memory and computes the Fast Fourier Transform (FFT) of the input and the squared input. The IP core for the FFT calculation was inserted into a Streams-C hardware process. The data and squared data outputs for a block (N) of FFTs are summed and stored in dual-port block ram. A signal containing the accumulated sum of the data and the squared data is sent to the data_run process and the squared_data_run process respectively. These signals initiate the other hardware processes to threshold the accumulated output data (in dual port ram). In this step the data is converted to a "one" if it is above the threshold and a "zero" if it is below. The squared_data_run process streams the output of the threshold operation to the data_run

process. Here the final stage of peak detection is performed and a signal is sent to the host1 process in order to read back signal detection frequencies from external on-board memory. This process repeats for Q iterations. For this example, Q = 3 and N = 4. Figure 7 shows the threshold and peak detection Streams-C code in the data_run hardware process.

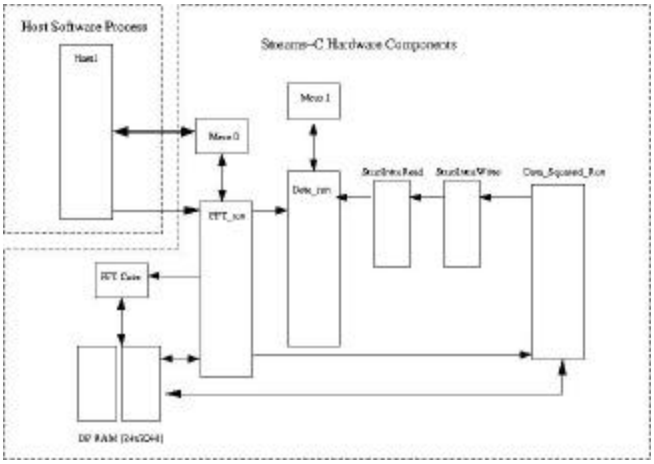

Figure 6. Streams-C mapping to hardware

4.3. Evaluation of Streams-C

The Streams-C compiler has a functional simulator that allows a software only simulation of the algorithm to check for correctness. The Streams-C language code is readable and very similar to C-code. This part of the tool is easy to use and fairly straightforward. For applications with multiple hardware and software processes, bit-accurate functional simulation limits the use of bit-widths to 8, 16, 32, and 64, however, a simple #ifdef structure allows you to separate software-only simulation code from hardware-only synthesis code in the process functions. As well, the design space in this part of development is flexible, allowing the user to iteratively interchange hardware and software processes in order to accommodate modifications in the project development cycle.

The next part of the compiler is the hardware generator that produces synthesizable RTL. In this phase of the process, sections of the peak detection algorithm had to be rewritten to generate synthesizable VHDL. Verification of the generated VHDL with ModelTech's Modelsim HDL simulator gives a clock-accurate representation of the design. The sc2 Streams-C compiler does not generate the VHLD behavioral host simulation code so this portion of the development is manual and time consuming. Also, the VHDL module for the FFT IP core had to be manually inserted into the top-level architecture file. The compiler did allow for accurate representations of variable data widths in hardware synthesis. The hardware libraries support Xilinx Virtex FPGAs, but the conversion to a different technology (such as Altera) is straightforward through the use of configuration statements in the hardware library.

The Streams-C generated design (see Table 1) used two Xilinx Virtex 1000E FPGAs and 14,250 slices, approximately 60% of the total area with a speed of 60 MHz. (The performance results were generated by Synplicity 7.1 and Xilinx ISE 5.2i.) The design took approximately 2 weeks to complete. The productivity improvement using the sc2 Streams-C compiler is approximately 2X to 4X over manual methods of implementing the application on a RCC system due to the fact that changes to the algorithm can be easily accommodated and the compiler provides automatic mapping to hardware.

```
#pragma SC memory mem_1 data_out
# pragma SC memory mem_1 event_data
# pragma SC memory DP_FFT_1 x

for(j=0; j<Q; j++){
// external IP core generates 4 ffts
// fft IP core 'finished' processing return the sum of 4 ffts
  sum = sc_wait(input_signal);

// calculate the threshold
Threshold = (sum/(sc_int24)2048)*(sc_int24)4;

  for(i=0; i<L; i++){
   if (x[i] < Threshold )
    data_out[i] = 0;
   else
    data_out[i] = 1; //strong value
}
// check for BPSK
// if squared_data = 1 check six frequency bins of the
// non squared data for a "1"
  for(j=0; j<L; j++){
    squared_data = sc_stream_read(input_stream);
    if (squared_data == (sc_int2)1){
      tmp0 = data_out[j];
      tmp1 = data_out[j/(sc_int32)2 + (sc_int32)1];
      tmp2 = data_out[j/(sc_int32)2 - (sc_int32)1];
      tmp3 = data_out[(FFTSize-j)/(sc_int32)2];
      tmp4 = data_out[(FFTSize-j)/(sc_int32)2 + (sc_int32)1];
      tmp5 = data_out[(FFTSize-j)/(sc_int32)2 - (sc_int32)1];
      check = sc_catenate(tmp5,tmp4,tmp3,tmp2,tmp1,tmp0);
       if ((sc_int12)check != (sc_int12)0){
         event_data[k] = (sc_int32)j; //peak detection
         k++;
}
```

Figure 7. Streams-C example code

5. SUMMARY

We investigated two high-level reconfigurable computing system development tools for mapping our BPSK algorithm to hardware, the Xilinx System Generator Tool, and the sc2 Streams-C compiler. Each tool has a useful functional simulator, and generates synthesizable hardware. Streams-C has an automatic target hardware board and produces the necessary framework to directly synthesize a design to this board and additional targets may also be defined. System Generator on the other hand, does not target a special-purpose board, thus, requiring manual mapping to a board. Both tools required "debugging", as some of the core components in System Generator did not function accurately, and some of the generated VHDL from the Streams-C compiler was not accurate. Clock accurate hardware simulation was cumbersome in Streams-C because the HDL host simulation code must be written by hand. In most cases System Generator clock-accurate simulation was easy, although it took time to learn which blocks had bugs (i.e. did not produce accurate hardware). In terms of performance, System Generator produced a design capable of running at a faster speed than the design produced by Streams-C, and therefore a design capable of more processing in the same amount of time. (This is attributed to the native use of pre-optimized cores in System Generator, whereas Streams-C relies on automatically generated VHDL). The Streams-C design, however, was smaller than the System Generator design, which would allow for smaller, less expensive FPGAs to be used.

An advanced tool requires a certain amount of "learning curve" and this coupled with the maturity of the tool can lead to a considerable amount of lost productivity. Over time, we found productivity is 2X to 4X higher compared to manual HDL design method. However, the tools have different trade-offs between placement speed and area, utilization of specific FPGA technology, and ease of use. Developers wishing to convert HDL development to a high-level tool should carefully examine their priorities on these issues before choosing a tool.

Table 1. Comparison of Performance Results

	Time to complete (weeks)	Area (Slices)	Speed (MHz)
System Generator	2 - 3	80%	70
Streams-C Compiler	2	60%	60

6. REFERENCES

[1] Star Bridge Systems, Inc., Midvale, Utah, *Viva*, http://www.starbridgesystems.com/products4.html

[2] Celoxica, Abingdon, Oxfordshire, UK, *DK Design Suite*, http://www.celoxica.com/methodology/c2rtl.asp

[3] SRC Computers, Inc., Colorado Springs, CO, *MAP Processor*, http://www.srccomp.com

[4] Xilinx Inc., *System Generator for DPS*, http://www.xilinx.com/ipcenter/ipevaluation/sysgen_evaluation.htm

[5] Los Alamos National Laboratory, Los Alamos, NM, *sc2 Streams-C compiler*, http://www.streams-c.lanl.gov

[6] The Mathworks Inc., Natick, MA, *Matlab/Simulink*, http://www.mathworks.com

[7] T. Maruyama and T. Defacto, "A design environment for adaptive computing technology," Proceedings of the 6th Reconfigurable Architectures Workshop (RAW '99), 1999.

[8] M. Weinhardt and W. Luk, "Pipeline vectorization for reconfigurable systems," FCCM 99, April 1999.

[9] Xilinx, http://www.xilinx.com/xilinxonline/jbits.htm, 1999.

[10] M. Gokhale, J. Stone, J. Frigo and C. Ahrens, "Streams-C: Stream-Oriented C Programming for FPGAs", http://rcc.lanl.gov/Tools/Streams-C/, 2003.