sc2 Hardware Library Reference Manual

Jan Frigo
Los Alamos National Laboratory
MSD440
Los Alamos, NM 87545
jfrigo@lanl.gov

Contents

1. Introduction **3**
1.1. Scope of this Reference Manual . . . 3

2. Streams-C Hardware Library Components **3**

3. Functional Streams-C Hardware Library Description **4**
3.1. Process Component . . . 4
3.2. Stream Components . . . 4
3.3. Signal and Parameter Components . . . 4

4. Memory Interfaces **4**
4.1. External Memory . . . 4
4.2. Block RAM . . . 6

5. Using sc2 **7**
5.1. Tools Required . . . 9
5.2. VHDL simulation . . . 9
5.3. VHDL synthesis . . . 10
5.4. Hardware bit stream generation . . . 10

6. Hardware Implementation Notes **11**

7. Examples **11**

8. Retarget Hardware Notes **11**

9. Hardware-Bugs **12**

10 Style Issues **12**
10.1 Optimization Hints . . . 12
10.2 Simulation vs. Synthesis . . . 12

List of Figures

1 Hardware Process Component . . . 5
2 Hardware Streams Component . . . 5
3 Streams-C External Memory Interface . . . 6
4 Streams-C External Memory Types . . . 7
5 Streams-C Block RAM Types . . . 7
6 Streams-C Block RAM Memory Interface . . . 8
7 Streams-C Dual Port Block RAM Memory Interface . . . 8
8 Example Architecture File . . . 12

1. Introduction

sc2 is a new implementation of the Streams-C language and compiler. 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 compiler is used to compile FPGA processes in hardware. The compiler translates a subset of C into Register-Transfer-Level (RTL) VHDL that is synthesizable on FPGAs. The sc2 compiler synthesizes hardware circuits for one or more FPGAs as well as a set of communicating processes on conventional processors. The language extensions allow pipelined stream computation, so that the generated hardware/software is capable of pipelining a computation across multiple FPGAs and the conventional processor.

Our programming model is targeted at stream-oriented FPGA applications. 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.

This sc2 release provides hardware libraries for the Annapolis Micro Systems (AMS) Firebird board, which contains one Xilinx Virtex-E FPGA on a 64-bit PCI bus. This manual describes the Streams-C hardware library used to facilitate the hardware interface between the Streams-C hardware processes and the target hardware board. It describes how to compile a VHDL simulation of the hardware processes and how to generate an executable (bit stream) for the hardware.

1.1. Scope of this Reference Manual

The purpose of this reference manual is to enable the user to generate a bit stream for hardware and describe the hardware interface between the sc2 generated hardware processes and the AMS Firebird board. It is assumed the user is familiar with VHDL or other hardware description languages.

2. Streams-C Hardware Library Components

- Hardware Streams Components - high bandwidth synchronous communications (tested 32-bit data streams)

  StrmFifoWrite - software to hardware fifo module

  StrmFifoRead - hardware to software fifo module

  StrmIntraRead - hardware to hardware streams fifo module

  StrmIntraWrite - hardware to hardware streams fifo module

  Fifo16 - internal fifo used in the above stream fifo modules.

- Hardware Signal Components - low bandwidth asynchronous communications (tested 32-bit data signals)

  Sig_Recv - software to hardware process signal receive

  Sig_Send - hardware to software process signal send

- External Memory Component - tested 64-bit memory data

  Sc_Mem - interface from hardware process to the 64 bit memories

  Sc_Mem4 - interface from hardware process to the 32-bit external memory

- Block Ram Components - tested 32 and 64-bit blockram data

  Sc_BRAM - 8, 16, 32, 64-bit block rams connected to hardware processes

  LAD64_Mux_BlockRam64 - Annapolis VHDL module for 64-bit by 256 deep, dual port ram connected to software process via the LAD bus and hardware process.

  LAD64_Mux_BlockRam - Annapolis VHDL module for 32-bit by 256 deep, dual port ram connected to software process via the LAD bus and hardware process.

- Pipeline Control Components

  Indefinite - controls pipeline for indefinite loops (*while*)

  Definite - controls pipeline for definite loops (*for*)

Note: tested both *for* and *while* loops, have noted that an indefinite loop (*while*) has problems with terminate in some schedules the last two pieces of data are missing, fixed the library VHDL.

3. Functional Streams-C Hardware Library Description

3.1. Process Component

The process module has two main components, a datapath process and a sequencer. The datapath process entity consists of a datapath entity and a pipeline control entity (if a *while* or *for* loop is pipelined within the process). The sequencer is a state machine for sequencing through the instruction set of the process module. If a stream, signal, external memory, or blockram is used in the process, the signal interface for these components is included on the process module's port. See figure 1.

3.2. Stream Components

The hardware stream component is used for high bandwidth, synchronous communication between the processes. These are parameterized modules with respect to data register width and fifo depth. Data width can be 16-bit, 32-bit or 64-bit. The fifo depth allowable currently is 16. The modules have separate Data, Enable and Ready signals as shown in figure 2 the Enable is an input which means the data on the input or output port is valid. The Ready is an output signal indicating the module is ready to receive data.

3.3. Signal and Parameter Components

The hardware signal component is used for low bandwidth, asynchronous coordination or flow control between processes. Signals are parameterizable with respect to data width. The convention for Enable, Ready, and Data is shown in figure 2. The Enable is an input to the module, SigSend, which means the data on the input port is valid. The Ready is an output signal indicating the module, SigRecv, has valid data on the output port.

The hardware parameter component is used for passing a variable between processes and to initiate a hardware process. Parameters are parameterizable with respect to data width. The modules have separate Data and Enable signals. The Enable is an output that means the data on the output port is valid. A hardware parameter is implemented as an input signal by the sc2 compiler.

4. Memory Interfaces

4.1. External Memory

When Memory is invoked in a hardware process, if a pragma does not exist to indicate a memory type, the compiler default is 64-bit external memory (Mem_0). When memory is used, the ports for address, data, and enables - MAR, MDR_IN, MDR_OUT, RD_EN, WR_EN respectivley appear on the hardware process entity. They are connected through the hardware streams library to the Firebird board. The hardware processes and streams components for external memory, SC_Mem (64-bit memories), SC_Mem4 (32-bit memory) connect to the Annapolis hardware with a Mem64_Mux_Priority_IF component. This Mem64_Mux allows multiple connections to external memory - one to connect to the LAD bus, allowing memory to be loaded from the host, and one or more as needed to connect to hardware processes. The Mem64_Mux_Priority_IF component connects to the LAD64_Mem64_Bridge component which facilitates loading to external memory via the LAD bus/PCI bus. Dual port rams (64-bit x 256 words deep) are contained within the LAD64_Mem64_Bridge component (each Bridge component uses 2 blocks of the Virtex Block RAM). The LAD64_Mux_IF is the mux interface to the LAD bus which connects to the PCI controller and the host as shown in figure 3. The 32-bit external memory, mem4, uses a separate set of Annapolis components - SC_Mem4, Mem32_Mux_Priority_IF, LAD64_Mem32_Bridge, LAD64_Mux_IF. Figure 4 describes the external memory types. Also reference /streamsc/apps/arch/Firebird.def

Notes:

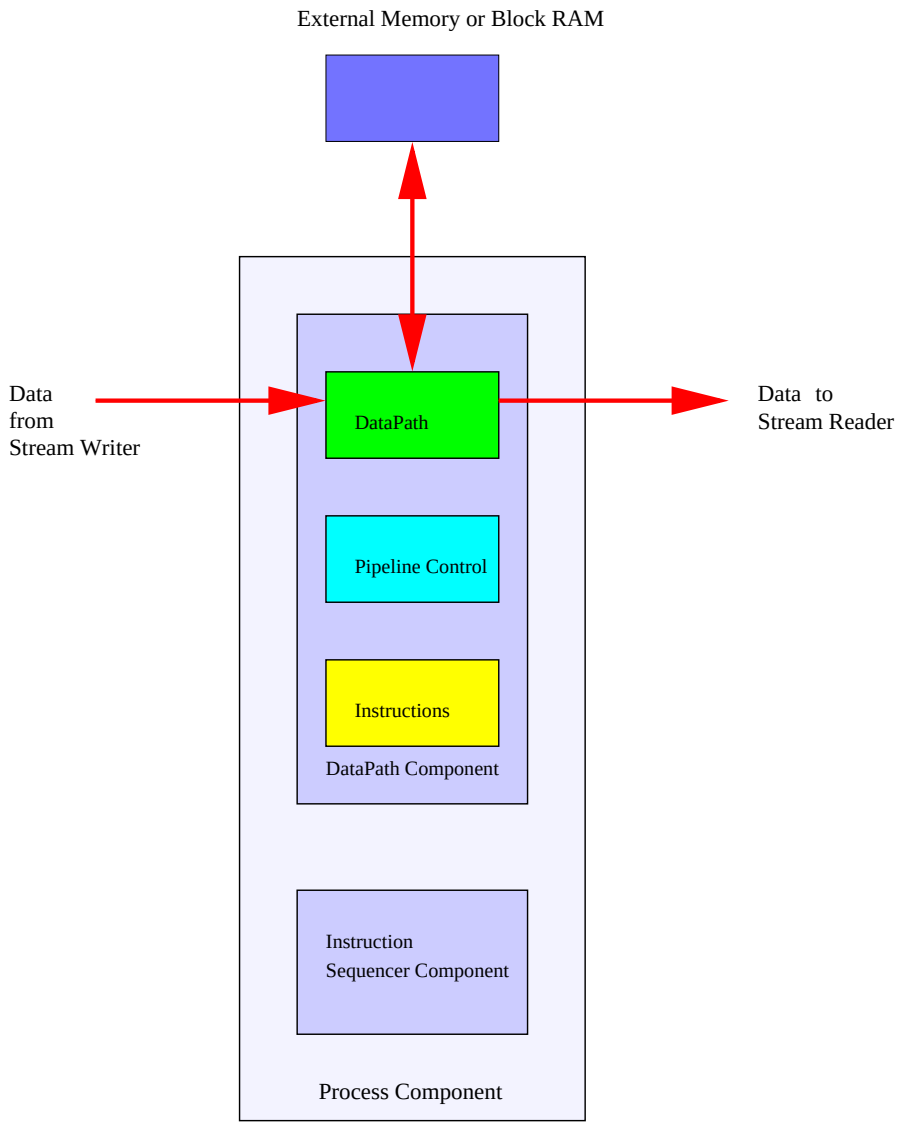

Figure 1. Hardware Process Component

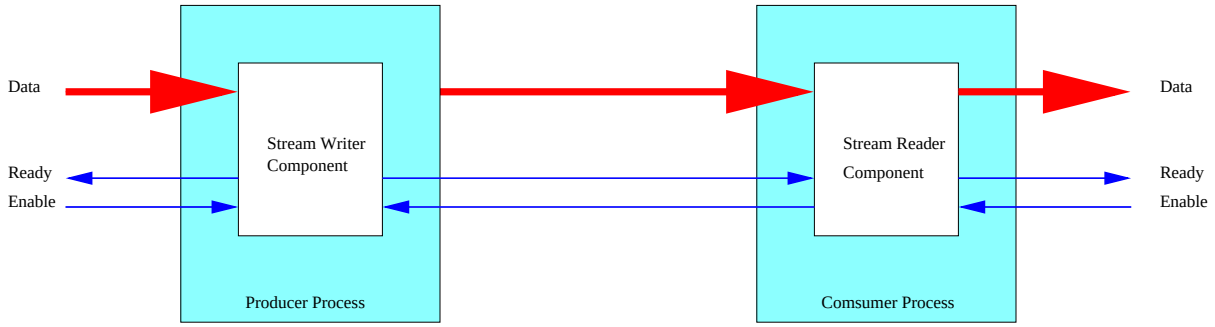

Figure 2. Hardware Streams Component

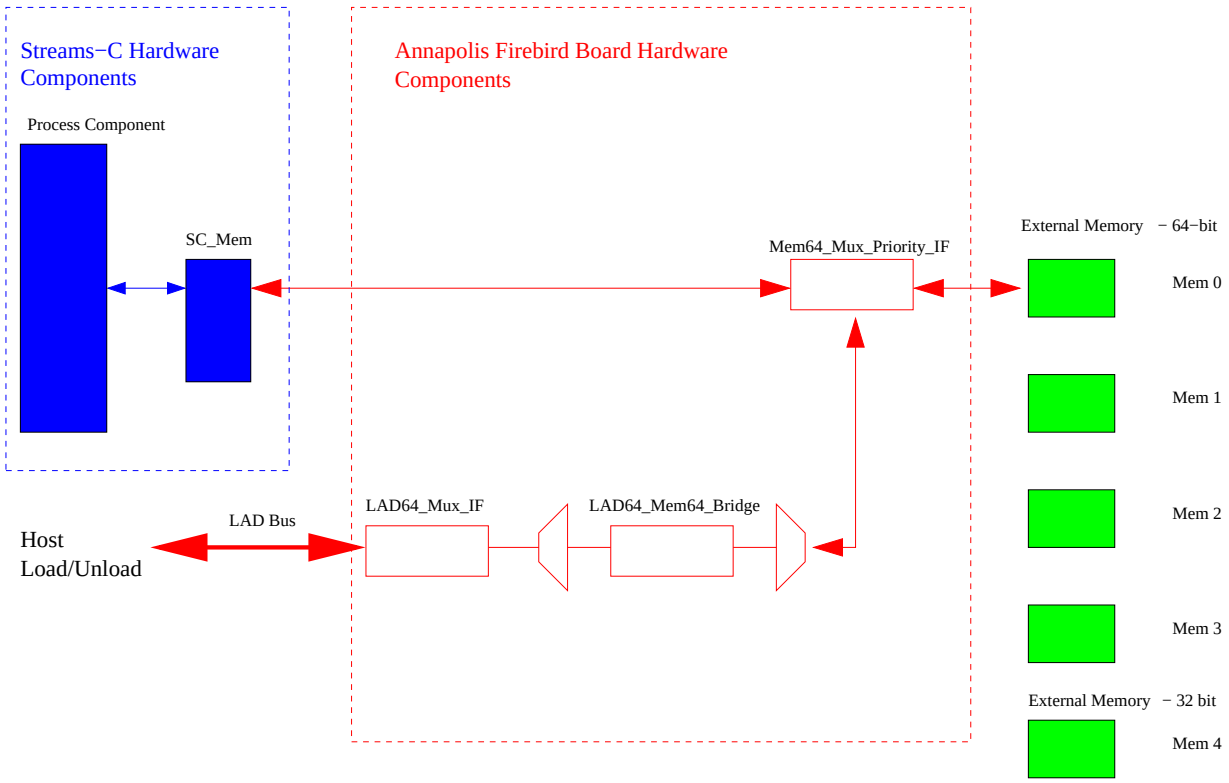

Figure 3. Streams-C External Memory Interface

- The Streams-C examples tested todate use a LAD bus speed of 66 MHz. The LAD bus is adjustable to either 33MHz or 66MHz with Annapolis PCI controller version 3.0.
- It is recommended to set the memory clock, SC_MCLK, for the run time simulation, sim_rt, to match the place and route maximum frequency.

4.2. Block RAM

For local data storage, the Streams-C hardware libraries allow the user to connect to block RAM in two ways: a write/read to a stream or a host software process load/unload block RAM data, examples strm3 and mem3 respectively. See figures 6 and 7 for the Streams-C hardware implementation of block RAM.

Streams-C uses a pragma statement for selection of a block RAM type. See figure 5 for the types of block RAM the Streams-C hardware library provides. (These type definitions are located in /streamsc/apps/arch/Firebird.def) The pragma statement for a block ram type, B_L, of size 64-bit x 256 is shown below where A is the name of the array in the user program and the maximum size of A is 256. A read from block RAM can be placed in a local variable, data, or in an output stream. These statements are identical to those used for external memory. A write and read with respect to block RAM is equivalent to the following C statements:

```
#pragma SC memory B_L A
data = sc_stream_read(input_stream);
A[i] = data; //write data to block ram
data = A[i]; //read from block ram and place in a variable, data
for(i=i-1; i>=0; i--)
   sc_stream_write(output_stream, A[i]);//read from block ram and place in output_stream
```

The hardware streams library components that support block RAM load/unload from the host are LAD64_Mux_BlockRam64 (64-bit x 256), and LAD64_Mux_BlockRam (32-bit x 256). These library components use dual ported RAM with one port

type	width (bits)	size
mem_0	64	1000000
mem_1	64	1000000
mem_2	64	1000000
mem_3	64	1000000
mem_4	32	500000

Figure 4. Streams-C External Memory Types

Single Port type	width (bits)	size
B_A	8	128
B_B	8	256
B_C	8	512
B_D	16	128
B_E	16	256
B_F	16	512
B_G	32	128
B_H	32	256
B_I	32	512
B_J	64	128
B_K	64	256
B_L	64	512
B_M	1	4096
B_N	2	2048
B_O	4	1024
Dual Port type	**width (bits)**	**size**
DP_1	32	256
DP_2	64	256

Figure 5. Streams-C Block RAM Types

connected to the LAD bus and the other port connected to the hardware process port(s). For an example of how to load and unload block RAM from the host software process, reference the mem3 example.

Note: StreamsC expresses memory latency of a read(load) as the delay after the read is issued, until the data is available on the port's data-register. For example, a one-cycle latency means the data is available the cycle after the read is issued:

Latency one:

tick 1 - set mar and enable bit

tick 2 - copy mdr into user register.

With this definition, latency is always at least one. For the user who wishes to define a new hardware target board, read(load) latency is the delay required for the data to be available in the port's data-register. For the AMS Firebird board, an external memory write executes in one cycle, a read executes in 6 cycles. The Virtex-E chip executes a write and a read in one cycle using block RAM. See /streamsc/apps/arch/Firebird_mem.def for definitions of load and store latencies.

5. Using sc2

The generated VHDL command, "make filename_all.vhd" generates the following VHDL output files: filename_all.vhd, filename_arch.vhd. The filename_all.vhd file contains the generated VHDL for the hardware processes which consists of a datapath, sequencer, and indefinite or definite hardware library component (if a *for* or *while* loop exists) for each hardware process defined in the Streams-C program, filename.sc The architecture file, filename_arch.vhd, uses the Streams-C hardware library to connect the hardware processes and any external memory or block ram to the target hardware board. This file is specific to AMS Firebird board, but can be modified to target a user defined board. (see section 8).

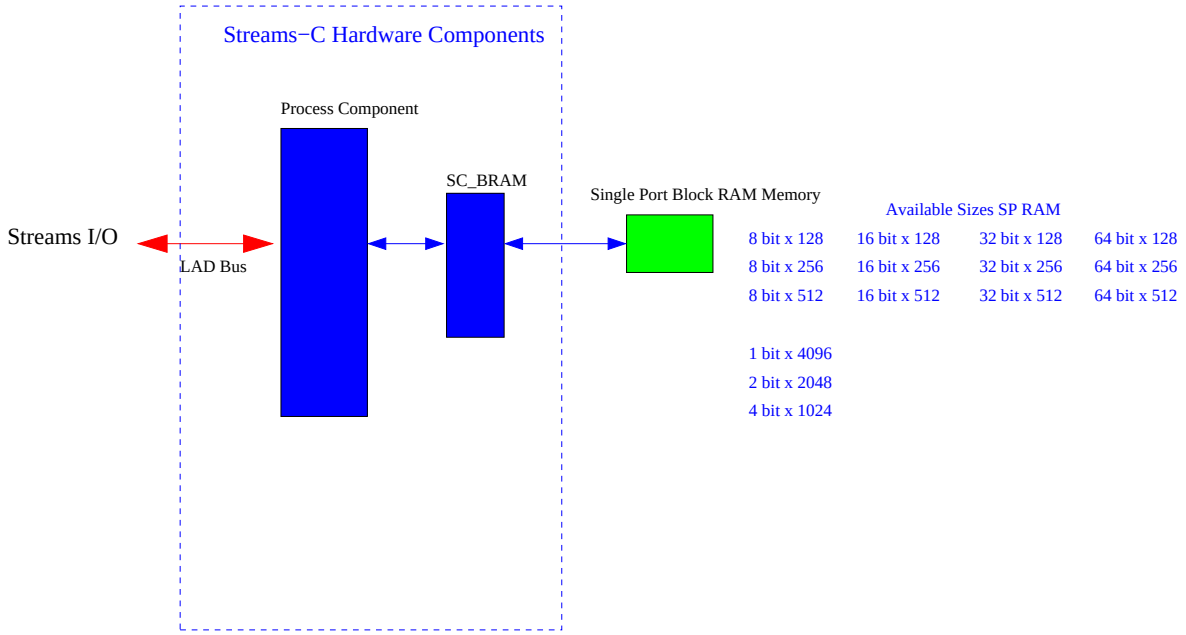

Figure 6. Streams-C Block RAM Memory Interface

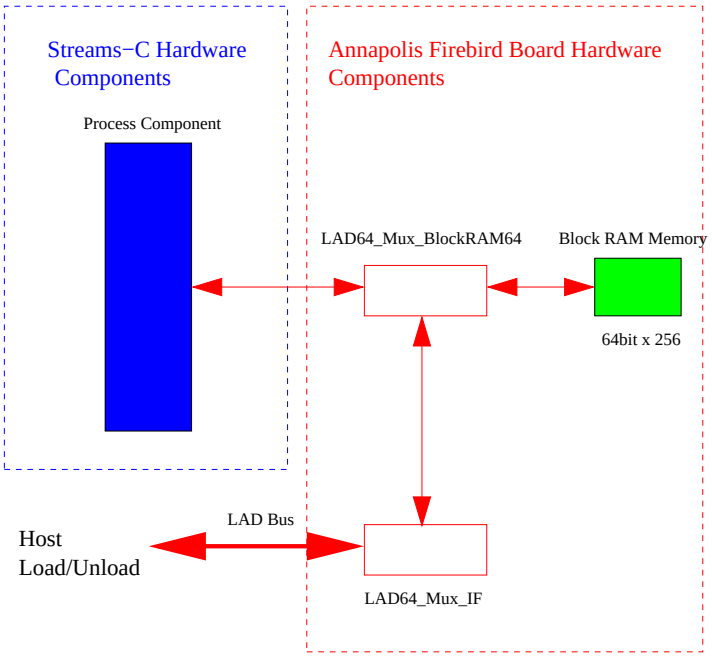

Figure 7. Streams-C Dual Port Block RAM Memory Interface

The following sections describe how to compile a VHDL simulation using ModelSim PE 5.5e, how to compile the VHDL project files for synthesis, and how to generate a hardware bit stream.

5.1. Tools Required

Modelsim PE 5.5e
Xilinx Core Gen 3.3.07i
Xilinx 4.1 Tools
Synplify 7.0
Annapolis VDHL version 3.1
Annapolis host API, version 5.0.0

5.2. VHDL simulation

- *Generate the Xilinx Core Libraries*

  Prior to compiling for Modelsim the Xilinx core library must be generated using Xilinx Core Gen 3.3.07i tool. How do I compile Xilinx Core Libraries on Modelsim? Go to the Xilinx website, Technical Answer Database: 8066 and Technical Answer Database: 2561

  http://support.xilinx.com/xlnx/xil_ans_display.jsp?iLanguageID=1&iCountryID=1&getPagePath=8066

  http://support.xilinx.com/xlnx/xil_ans_display.jsp?iLanguageID=1&iCountryID=1&getPagePath=2561

  Read these two Xilinx Techincal Answers carefully. Briefly, the compilation involves setting the correct paths to the library, setting some environment variables and compiling a vcom.do file. The compilation generates the behavioral models for the Xilinx Cores used with the Modelsim simulator. The /streamsc/vhdl_lib/sc_xilinx/ directory includes the vcom.do file needed to generate the libraries. Be sure you are using Xilinx Core Gen 3.3.07i.

  Note: If you are currently Xilinx technology user, please contact the authors for the pre-compiled Xilinx core libraries.

- *Set Library and Project Paths*

  Once the core libraries are compiled, edit the mti_vcom.do file to set paths to the Xilinx Core libraries, Annapolis hardware libraries, Streams-C hardware library, and your project directory. Make sure you have the Annapolis Board VHDL models installed in a local directory.

  set MODEL_TECH ”D:/ProgramFiles/modeltech”

  set ANNAPOLIS_BASE ”D:/Annapolis”

  set PROJECT_BASE ”F:/Firebird/strm/project”

  set SC_LIBRARY_BASE ”F:/streamsc/vhdl_lib”

  – Xilinx Logiblox support for RAM blocks vmap xilinxcorelib ”d:/xilinx/vhdl/src/XilinxCoreLib/xilinxcorelib”

  Place the Xilinx .mif files located in streamsc/vhdl_lib/sc_xilinx/sim in your PROJECT_BASE directory.

- *Edit your Project filename*

  vcom -93 -explicit -work PE0_Lib $PROJECT_BASE/filename_all.vhd

  vcom -93 -explicit -work PE0_Lib $PROJECT_BASE/filename_arch.vhd

- *Create your Host VHDL program*

  There are example host programs in the strm and memory examples, such as strm1, strm2, etc. These will show how to simulate a write/read to/from streams, signals, external memory and block ram for Modelsim PE. Construct a host program named, host_filename_arch.vhd, and edit the mti_vcom.do file line to include it in the simulation.

  vcom -93 -explicit -work system $PROJECT_BASE/host_filename_arch.vhd

- *Compile - Annapolis board libraries, Streams-C hardware libraries and Streams-C generated project files*

  mti_vcom.do

- *Load system configuration*

  Once your project has compiled, go to the ModelSim *System* library and load system_cfg, the top level configuration module.

  Note: The Annapolis libraries only need to be completed once for each project directory, so after a successful compile comment out the line –do $FIREBIRD_PCI_BASE/vhdl/system_vcom.do

5.3. VHDL synthesis

- *Set environment variables and edit project filenames*

  The VHDL synthesis tool used with Streams-C is Synplify 7.0. The vhdl_lib/sc_xilinx/syn directory has a synthesis project file, pe0.prj, which sets all the defaults needed for the synthesis tool. Edit the location of the paths in the pe0.prj file and edit the names of your project files

  –environment variables

  set PROJECT_BASE ”F:/firebird/strm/project”

  set ANNAPOLIS_BASE ”D:annapolis”

  set SC2_LIB_BASE ”F:/streamsc/vhdl_lib”

  –project files

  add_file -vhdl -lib PE0_Lib ”$PROJECT_BASE/filename_all.vhd”

  add_file -vhdl -lib PE0_Lib ”$PROJECT_BASE/filename_arch.vhd”

- *Set Clock frequency*

  Set the Clock frequency option to a desired design speed.

  set_option -frequency 66.000

- *Run the pe0.prj project in Synplify to generate a pe0.edf file*

5.4. Hardware bit stream generation

- *Copy your pe0.edf file to $PROJECT_BASE/pnr directory.*

  where $PROJECT_BASE is the location of your project files.

- *Set path for Annapolis VHDL mddels.*

  The /streamsc/vhdl_lib/sc_xilinx/pnr directory contains a makefile file. Edit this makefile to set the path for the location of the Annapolis VHDL models.

  set ANNAPOLIS_BASE ”D:/Annapolis”

- *Place and Route the design*

  Place the Xilinx .edn files located in streamsc/vhdl_lib/sc_xilinx/pnr in your PROJECT_BASE directory.

  Open a cygwin bash shell, and cd to your project directory then type, ”$ANNAPOLIS_BASE/shared/bin/make pe0.hex

  where $ANNAPOLIS_BASE is the location of the Annapolis VHDL models. This command creates the bit stream - places and routes the hardware design. This step will take quite a bit of time depending on the size of the project. Decrease the design clock frequency if faster routing time is desired. The constraints files, .ucf, etc. for place and route are defaults set by Annapolis MicroSystems Inc.

- *Convert pe0.hex to pe0.x86*

  Once the .hex file has been generated, type ”D:/Annapolis/shared/bin/peutils pe0.hex and choose option 0) x86, to convert the pe0.hex file into the correct format. You have created a pe0.x86 binary file, the hardware bit stream. Use this bit stream with the sim_rt simulation to run the application on the hardware.

The .par log files from Xilinx reports the maximum frequency achieved during place and route. The user should set SC_MCLK, the desired frequency for running the bit stream on hardware, by setting constants in the software include in the .sc program:

#define SC_MCLK 20.0

#define SC_UCLK 50.0

A default is set to 20 MHz by the preprocessor if nothing is defined. Streams-C hardware components use the MCLK, the user design speed, and the KCLK, the LAD Bus speed. UCLK does not affect the speed of the hardware design for Streams-C.

6. Hardware Implementation Notes

- Be sure to check the version of the simulation, synthesis and place and route tools you are using with Streams-C. Streams-C was tested and developed with the following CAD tools: Modelsim PE 5.5e –for VHDL simulation

  Xilinx Core Gen 3.3.07i – to generate the behaviour models for VHDL simulation and the Xilinx cores for place and route

  Xilinx 4.1 Tools – to generate a bit stream

  Synplify 7.0 – to generate a net list, pe0.edf file

  Annapolis VDHL version 3.1 –for simulation, synthesis and place and route

  Annapolis host API ,version 5.0.0

- Be sure to check the log files after place and route to determine the actual speed at which your design was routed. This value needs to be entered with the sim_rt, host run-time program.

7. Examples

The program strm2.sc has two software processes and two hardware processes. The first software process host1, with run function host1_run opens an output stream and writes a sequence of integers to the stream. The bound on the loop ("iterations") is set by the input argument to the program invocation (eg. the invocation "strm2_sim 400" causes a sequence of integers from 0 to 399 to be written to the output stream).

The stream sent by host1 goes to a hardware process controller with run function controller_run. This process simply forwards the stream to the next hardware process, pe0_proc_run. pe0_proc_run has two phases. First it copies its input stream to memory. (Note: no #pragma statement is given for memory - the default is an external 64-bit memory, mem 0.) When the whole stream has been read into memory, it reads back the data in reverse order and writes to its output stream. The final software process, host2, using run function host2_run, reads the stream from pe0_proc_run and prints out the data received from the stream.

The two hardware processes are generated by the sc2 compiler and placed in the strm2_all.vhd file. The hardware processes, controller_run, and pe0_proc_run are connected through the Hardware Streams Library components to the Annapolis board models in order to facilitate the hardware interface to the host workstation and memory. The strm2_arch.vhd file represented in figure 8 contains the "system connectivity".

The streamsc/vhdl_lib directory contains all of the Streams-C hardware library components. See section 3.2 for a description of the hardware library components.

8. Retarget Hardware Notes

This strm2_all.vhd file is designed to be "stand-alone" and can be retargeted to a different user system. The architecture file and the associated Streams-C hardware library modules for streams, signals, parameters and memory need to be redesigned for the user system. The Streams-C memory types in the streamsc/apps/arch/Firebird.def and Firebird_mem.def files need to be modified as well.

The process declarations and connects can reflect multiple FPGA's via the naming convention, PE0, PE1, etc. in the PE.def file.

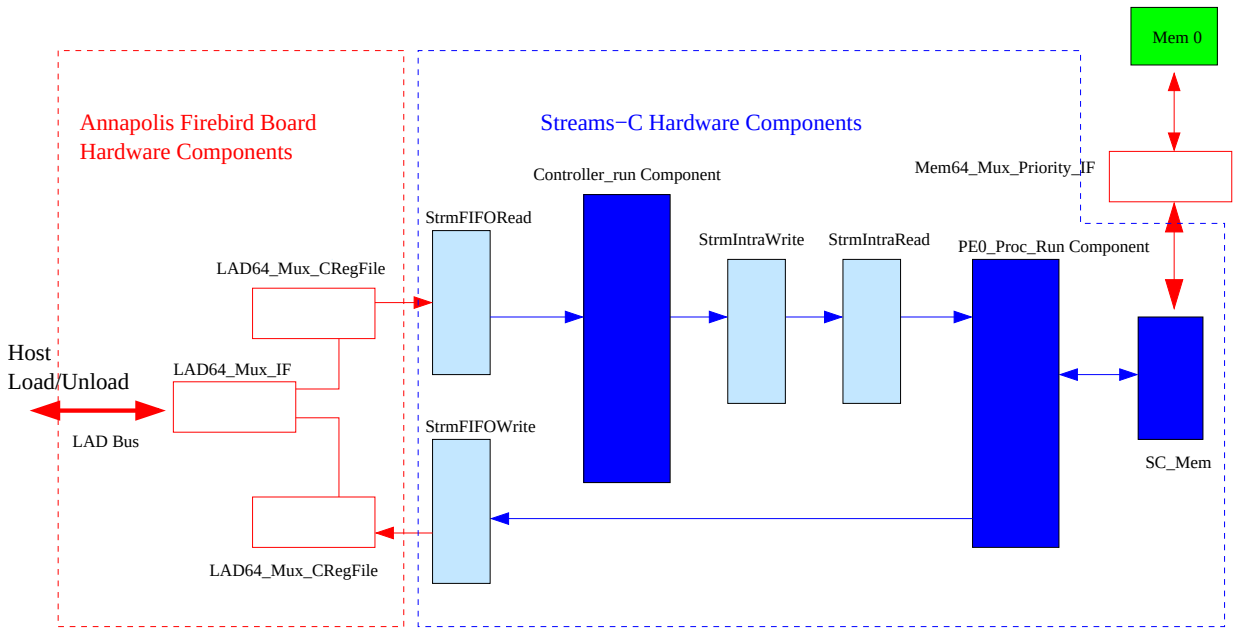

Figure 8. Example Architecture File

9. Hardware-Bugs

- In the indefinite pipeline, at times one or two pieces of data was lost at the end of a stream.
- Do not use the #pragma SC pipeline pragma in hardware processes which contain sc_wait() calls.
- Arrays of processes are not implemented in hardware yet.
- Parameters are not implemented in hardware yet.

10. Style Issues

10.1. Optimization Hints

- Use functions sc_bit_insert() and sc_bit_extract() instead of the shift left or shift right (”<<” or ”>>”) operators.
- Pointers are not permitted. Indirect reference must be accomplished through array reference.
- If most of the 160 blocks of block RAMs are used for a memory intensive application, the overall timing of the design may decrease. The place and route tool routes these blocks automatically and timing could be affected.

10.2. Simulation vs. Synthesis

- Occasionally it is useful to write the code in one way for simulation and slightly different for synthesis. The IF_SIM macro is provided for this purpose.
- We have found that C compiler bugs sometimes cause large locally allocated arrays to get corrupted. Thus to circumvent the bug in simulation, the array is globally allocated during simulation and locally allocated for synthesis.

- The user may set SC_MCLK, the desired frequency for running the bit stream, by using a software include in the .sc program. A default is set to 20 MHz by the preprocessor if nothing is defined. Streams-C hardware components use the MCLK, the user design speed, and the KCLK, the LAD Bus speed. UCLK does not affect the speed of the user design for Streams-C.
- Consider your application usage of streams and design speed issues. The LAD bus runs at 33MHz/66MHz on the Firebird, so heavy demands on streams may result in slower overall system speed. See example strm4.
- For mulitiplication of two or more operands, the result is the size of the larger of the operands unless the each operand is cast.
- The sc2 compiler can pipeline blocks that contain control flow statements, i.e. if-else statements.