SIEMENS EDA

Edge Detect Lab #3 Integration

Lab Workbook

Catapult 10.6a

SIEMENS

Support Center: support.sw.siemens.com
Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Table of Contents

Before you Begin

Do the following after logging into the AWS Virtual Machine (VM):

- Open a terminal by right-clicking on the desktop and select “Terminal”
- Enter “ls catapult_edge_detect”
  - If the folder is already there you have already copied the files over and can proceed to the rest of the lab
- Copy the lab files to the user home directory by entering the following at the terminal prompt:
  - cp /project/catapult_edge_detect_106.tar .
  - tar -xvf catapult_edge_detect_106.tar
- Source the setup script in the terminal
  - Type “source /project/setup.sh” and hit return

You are now ready to proceed with the lab exercises.

NOTE: *If you find yourself with a need to retrieve the original data for a module, you can always download the lab data to another directory within your student directory*

Lab #3: Integration

Objectives

At the end of this lab you should be able to do the following:

- Add a catapult generated component to a processor design
- Verify that the register interface can be accessed correctly
- Exercise the component and compare the results with a reference software implementation in the context of the processor design

Introduction

This lab will walk you through adding the Catapult component to the processor design. This involves modifying the AXI interconnect for the design, instantiating the catapult component, and verifying that the coding was performed correctly. In this case the catapult component has both a master and slave interface. The slave interface is used for the processor to set configuration inputs and synchronize activity of the component. The master interface is used by the component to access the memory area shared between the processor and the slave.

- Modify the AXI bus to add a new master port and slave port
- Instantiate the catapult component, using the register interface wrapper
- Verify the communications paths by reading and writing registers in the register interface.
- Verify that the component is working in the context of the processor design

Add Accelerator to the Design

Go to the directory for lab 3:

```
% cd ~/catapult_edge_detect/system_designs/labs/lab3
```

Change directories into the “hardware_sources” directory. Edit the file systemc_subsystem.h. This is the implementation of the main AXI complex, and the instantiation of the shared memory and peripherals.

You will find the code to be added is commented out (this is not a typing exercise).

At lines 27 to 31, add the signal bundles for the AXI segments to connect the master and slave ports of the accelerator:

```
25    //== Local signals
26
27  //r_chan<>        CCS_INIT_S1(r_acc_regs);
28  //w_chan<>        CCS_INIT_S1(w_acc_regs);
29
30  //r_chan<>        CCS_INIT_S1(r_acc_master);
31  //w_chan<>        CCS_INIT_S1(w_acc_master);
32
33    r_chan<>        CCS_INIT_S1(r_memory);
34    w_chan<>        CCS_INIT_S1(w_memory);
35
```

At line 45, add the instantiation of the accelerator:

```
42    //== Instances
43
44    fabric          CCS_INIT_S1(io_matrix);
45  //accel_if        CCS_INIT_S1(go_fast);
46    sys_ram         CCS_INIT_S1(shared_memory);
47    uart_if         CCS_INIT_S1(uart);
48    clock_timer     CCS_INIT_S1(timer);
49
```

At lines 74 to 79, connect the signals to the ports of the accelerator, clock and reset, as well as the AXI master and slave segments:

```
71      shared_memory.r_slave0 (r_memory);
72      shared_memory.w_slave0 (w_memory);
73
74  //  go_fast.clk            (clk);
75  //  go_fast.rst_bar        (reset_bar);
76  //  go_fast.r_reg_if       (r_acc_regs);
77  //  go_fast.w_reg_if       (w_acc_regs);
78  //  go_fast.r_mem_if       (r_acc_master);
79  //  go_fast.w_mem_if       (w_acc_master);
80
81      uart.clk               (clk);
82      uart.rst_bar           (reset_bar);
```

Next edit the file proc_fabric.h.

At lines 28 and 29, add the slave AXI signal bundles to connect the interconnect to the accelerator slave port:

```
22    r_master<> CCS_INIT_S1(r_mem);
23    w_master<> CCS_INIT_S1(w_mem);
24    r_master<> CCS_INIT_S1(r_uart);
25    w_master<> CCS_INIT_S1(w_uart);
26    r_master<> CCS_INIT_S1(r_timer);
27    w_master<> CCS_INIT_S1(w_timer);
28  //r_master<> CCS_INIT_S1(r_reg);
29  //w_master<> CCS_INIT_S1(w_reg);
30
```

At lines 37 and 38, add the master AXI signal bundles to connect the interconnect to the accelerator master port:

```
31    // ports to the masters
32    //  master 0 = cpu
33    //  master 1 = accelerator
34
35    r_slave<>  CCS_INIT_S1(r_cpu);
36    w_slave<>  CCS_INIT_S1(w_cpu);
37  //r_slave<>  CCS_INIT_S1(r_acc);
38  //w_slave<>  CCS_INIT_S1(w_acc);
39
```

At lines 42 and 43, change the initialized value of “numSlaves” to 4 and the initialized value of “numMasters” to 2:

```
40     static const int numAddrBitsToInspect = 32;
41
42     static const int numSlaves            =  3;  // will be 4 with accelerator added
43     static const int numMasters           =  1;  // will be 2 with accelerator added
44
45     sc_signal<NVUINTW(numAddrBitsToInspect)> addrBound[numSlaves][2];
46
```

At line 48, add an AXI splitter for the new master:

```
47     AxiSplitter<sysbus_axi4_config, numSlaves, numAddrBitsToInspect, false, true> CCS_INIT_S1(cpu_router);
48   //AxiSplitter<sysbus_axi4_config, numSlaves, numAddrBitsToInspect, false, true> CCS_INIT_S1(acc_router);
49
```

At line 53, add the Arbiter for the register interface:

```
50     AxiArbiter<sysbus_axi4_config, numMasters, 4> CCS_INIT_S1(mem_arbiter);
51     AxiArbiter<sysbus_axi4_config, numMasters, 4> CCS_INIT_S1(uart_arbiter);
52     AxiArbiter<sysbus_axi4_config, numMasters, 4> CCS_INIT_S1(timer_arbiter);
53   //AxiArbiter<sysbus_axi4_config, numMasters, 4> CCS_INIT_S1(reg_arbiter);
54
```

At lines 64 to 71, add the signal bundles to connect the accelerator master to the slaves in the design:

```
62     typename axi::axi4<sysbus_axi4_config>::write::template chan<> CCS_INIT_S1(w_cpu2reg);
63
64   //typename axi::axi4<sysbus_axi4_config>::read::template chan<>  CCS_INIT_S1(r_acc2mem);
65   //typename axi::axi4<sysbus_axi4_config>::write::template chan<> CCS_INIT_S1(w_acc2mem);
66   //typename axi::axi4<sysbus_axi4_config>::read::template chan<>  CCS_INIT_S1(r_acc2uart);
67   //typename axi::axi4<sysbus_axi4_config>::write::template chan<> CCS_INIT_S1(w_acc2uart);
68   //typename axi::axi4<sysbus_axi4_config>::read::template chan<>  CCS_INIT_S1(r_acc2timer);
69   //typename axi::axi4<sysbus_axi4_config>::write::template chan<> CCS_INIT_S1(w_acc2timer);
70   //typename axi::axi4<sysbus_axi4_config>::read::template chan<>  CCS_INIT_S1(r_acc2reg);
71   //typename axi::axi4<sysbus_axi4_config>::write::template chan<> CCS_INIT_S1(w_acc2reg);
72
73     SC_CTOR(fabric)
74     {
```

At lines 84 and 85, add the address ranges for the new peripheral, from 0x60000000 to 0x6000FFFF. This is where the registers of the accelerator can be accessed:

```
75        addrBound[0][0] = 0x70000000;
76        addrBound[0][1] = 0x7fffffff;
77
78        addrBound[1][0] = 0x60080000;
79        addrBound[1][1] = 0x6008ffff;
80
81        addrBound[2][0] = 0x600A0000;
82        addrBound[2][1] = 0x600Affff;
83
84    //  addrBound[3][0] = 0x60000000;
85    //  addrBound[3][1] = 0x6000ffff;
```

At lines 97 and 98, set the new address bounds in the cpu_router.addrBound array:

```
91        cpu_router.addrBound[0][0](addrBound[0][0]);
92        cpu_router.addrBound[0][1](addrBound[0][1]);
93        cpu_router.addrBound[1][0](addrBound[1][0]);
94        cpu_router.addrBound[1][1](addrBound[1][1]);
95        cpu_router.addrBound[2][0](addrBound[2][0]);
96        cpu_router.addrBound[2][1](addrBound[2][1]);
97    //  cpu_router.addrBound[3][0](addrBound[3][0]);
98    //  cpu_router.addrBound[3][1](addrBound[3][1]);
```

At lines 121 to 125, add the connection to the slave port on the accelerator to the CPU master splitter:

```
120
121   //  cpu_router.axi_rd_s_ar[3](r_cpu2reg.ar);
122   //  cpu_router.axi_rd_s_r[3] (r_cpu2reg.r);
123   //  cpu_router.axi_wr_s_aw[3](w_cpu2reg.aw);
124   //  cpu_router.axi_wr_s_w[3] (w_cpu2reg.w);
125   //  cpu_router.axi_wr_s_b[3] (w_cpu2reg.b);
126
127
```

At lines 128 to 160, add the router for the accelerator master port:

```
127
128   //  acc_router.clk(clk);
129   //  acc_router.reset_bar(rst_bar);
130
131   //  acc_router.addrBound[0][0](addrBound[0][0]);
132   //  acc_router.addrBound[0][1](addrBound[0][1]);
133   //  acc_router.addrBound[1][0](addrBound[1][0]);
134   //  acc_router.addrBound[1][1](addrBound[1][1]);
135   //  acc_router.addrBound[2][0](addrBound[1][0]);
136   //  acc_router.addrBound[2][1](addrBound[1][1]);
```

At lines 178 to 182, 197 to 201, and 216 to 220, connect the slave side of the accelerator to the arbiters:

```
176       mem_arbiter.axi_wr_m_b[0] (w_cpu2mem.b);
177
178   //  mem_arbiter.axi_rd_m_ar[1](r_acc2mem.ar);
179   //  mem_arbiter.axi_rd_m_r[1] (r_acc2mem.r);
180   //  mem_arbiter.axi_wr_m_aw[1](w_acc2mem.aw);
181   //  mem_arbiter.axi_wr_m_w[1] (w_acc2mem.w);
182   //  mem_arbiter.axi_wr_m_b[1] (w_acc2mem.b);
183
184       mem_arbiter.axi_rd_s(r_mem);
```

At lines 226 to 242, add an arbiter for the accelerator:

```
226   //  reg_arbiter.clk(clk);
227   //  reg_arbiter.reset_bar(rst_bar);
228
229   //  reg_arbiter.axi_rd_m_ar[0](r_cpu2reg.ar);
230   //  reg_arbiter.axi_rd_m_r[0] (r_cpu2reg.r);
231   //  reg_arbiter.axi_wr_m_aw[0](w_cpu2reg.aw);
232   //  reg_arbiter.axi_wr_m_w[0] (w_cpu2reg.w);
233   //  reg_arbiter.axi_wr_m_b[0] (w_cpu2reg.b);
234
235   //  reg_arbiter.axi_rd_m_ar[1](r_acc2reg.ar);
236   //  reg_arbiter.axi_rd_m_r[1] (r_acc2reg.r);
237   //  reg_arbiter.axi_wr_m_aw[1](w_acc2reg.aw);
238   //  reg_arbiter.axi_wr_m_w[1] (w_acc2reg.w);
239   //  reg_arbiter.axi_wr_m_b[1] (w_acc2reg.b);
240
241   //  reg_arbiter.axi_rd_s(r_reg);
242   //  reg_arbiter.axi_wr_s(w_reg);
```

Pro-tip: find and replace the “//” at the start of a line with 2 spaces (“ “)

At this point, the accelerator is added to the design.

The software interface to the accelerator is defined in the file “magnitude_angle_if.h”. Since the embedded code will be run both as a SystemC testbench and on the embedded processor, a set of accessor C preprocessor macros are defined in this file. The accessors define SET_ and GET_ macros for the registers on the accelerator. And there are TB_READ_ and TB_WRITE macros are defined for accessing shared memory.

In SystemC, these macros translate to matchlib AXI transactions. When compiled for the rocket core, they will translate to pointer dereferences to the addresses of the registers. In this way a single test program can be written to run as stimulus from either SystemC or the embedded core without any changes to the code.

Verify the Accelerator Instantiation

Each of the registers in the interface of the accelerator can be read and written from the testbench/processor. To verify that the accelerator has been instantiated correctly a test function called “check_register_access()” has been added to the file “common_stim_results.h”. This file is included in both the SystemC testbench (host_code_tb.h) and the embedded code (sw.edge/main.c).

Modify the SystemC testbench to call this function. Uncomment the call to this function on line 45 of the file host_code_tb.h:

```
38    void sw_thread()
39    {
40      w_master.reset();
41      r_master.reset();
42
43      wait();
44
45  //   check_register_access();
46  /*
47      // inputs are preloaded into memory at 0x70000000
48
49      load_data(sw_image, (unsigned char *) 0x70000000);
50
```

Change directories to the host code design directory:

```
% cd ../host_code_design
```

Then make and run the design and check to see that the testbench is able to correctly read and write the registers on the interface of the accelerator:

```
% make
```

The compilation process will take a few minutes

You should see output that looks like this:

```
Environment variables are set correctly

 All tool output is redirected to "make.out"

MKDIR   ./object_dir
CXX     ../hardware_sources/testbench.cpp
LINK    -o testbench
make -C ../../utils
make[1]: Entering directory `/home/user/labs/utils'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/home/user/labs/utils'
./sim_sc

             SystemC 2.3.0-ASI --- Aug 22 2020 22:15:37
        Copyright (c) 1996-2012 by all Contributors,
        ALL RIGHTS RESERVED

Opening file "../../data/image.mem"
Reading image...
96 values loaded
Memory initialization complete.
Connections Clock: testbench.clk Period: 10 ns
WARNING: Default time step is used for VCD tracing.
register access test passed!

Info: /OSCI/SystemC: Simulation stopped by user.
Simulation PASSED
```

You specifically want to look for “register access test passed!” and “Simulation PASSED”. These indicate that the registers can be properly accessed.

Next, run the same testbench from the Rocket core. Change directories into the “processor_design/sw.edge”:

```
% cd ../processor_design
```

Edit the file “main.c”. Uncomment the call to “check_register_access()” on line 25:

```
21    static unsigned char  data_in[IMAGE_SIZE];
22    static float          sw_data_out[2 * IMAGE_SIZE];
23    static float          hw_data_out[2 * IMAGE_SIZE];
24
25  //check_register_access();
26
27  /*
28    printf("loading data... \n");
29
30    load_data_array(data_in);
```

Build the software:

```
% make
```

Change directories to the “processor_design” directory:

```
% cd ..
```

Build and run the design.

```
% make
```

The compilation process will take a few minutes.

You should see something like this:

```
  All tool output is redirected to "make.out"

MKDIR   marker_dir/marker.mark
        checking SYSTEMC_HOME
        checking CONNECTIONS_HOME
        checking MATCHLIB_HOME
        checking AC_TYPES
        checking AC_STIMUTILS
        Environment variables are set correctly
VLIB    ./work
VMAP    -work ./work
GCC     ../../terminal/terminal_emulator.c
SCCOM   ../hardware_sources/systemc_subsystem_wrapper.cpp
SCCOM   -link
VLOG    ../hardware_sources/testbench.sv
VLOG    ../hardware_sources/top.sv
MAKE    rocket_design
make[1]: Entering directory
`/home/user/labs/lab3/processor_design'
MAKE    rocket_lib
make[2]: Entering directory
`/home/user/labs/lab3/processor_design'
```

```
VLIB    ./rocket_lib
VMAP    rocket_lib ./rocket_lib
VLOG    ../../rocket_core/vsim/generated-
src/freechips.rocketchip.system.DefaultConfig.v
make[2]: Leaving directory
`/home/user/labs/lab3/processor_design'
VLOG    ../../rocket_design/addr_gen.sv
VLOG    ../../rocket_design/axi_addr_latch.sv
VLOG    ../../rocket_design/axi_byte_enables.sv
VLOG    ../../rocket_design/axi_data_latch.sv
VLOG    ../../rocket_design/axi_matrix.sv
VLOG    ../../rocket_design/axi_segment_arbiter.sv
VLOG    ../../rocket_design/axi_slave_if.sv
VLOG    ../../rocket_design/axi_slave_segment.sv
VLOG    ../../rocket_design/bus_fifo.sv
VLOG    ../../rocket_design/mux.sv
VLOG    ../../rocket_design/ready_gen.sv
VLOG    ../../rocket_design/rocket_subsystem.sv
VLOG    ../../rocket_design/sram.sv
make[1]: Leaving directory
`/home/user/labs/lab3/processor_design'
VOPT    testbench -o testbench_opt

>>----> Design compiled

make -C ../../utils
make[1]: Entering directory `/home/user/labs/utils'
./make_stim_response     ./image/people_gray.bmp
Loading Input File
Running
Finished
mv image.mem             ../data
mv expected_results.bin  ../data
make[1]: Leaving directory `/home/user/labs/utils'
VSIM    -work ./work -voptargs=+acc -L rocket_lib -do run.do -c
testbench_opt
register access test passed!
Finished
```

You are specifically looking for the line: “register access test passed!”. If you see this, then the RTL processor model can correctly access the accelerator.

Run the Accelerator

Now that we have proven we can access the accelerator from the processor and testbench, let's use the accelerator in line detect operation.

We will start in the host_code_design, and then move to the processor_design.

Change directories to the host_code_design:

```
% cd ../host_code_design
```

Edit the file ../hardware_sources/host_code_tb.h. Uncomment the code from lines 47 to 54. If the call to check_register_access() is uncommented from earlier in the lab, comment out this line:

```
42
43      wait();
44
45  //   check_register_access();
46  /*
47      // inputs are preloaded into memory at 0x70000000
48
49      load_data(sw_image, (unsigned char *) 0x70000000);
50
51      edge_detect_hw((unsigned char *) 0x70000000, hw_data_out);
52      edge_detect_sw(sw_image, sw_data_out);
53
54      check_results(sw_data_out, hw_data_out);
55  */
56      sc_stop();
57    }
58
```

This testbench code will run a software version of the algorithm for reference and then exercise the hardware definition (from magnitude_angle.h). It will compare the results between the two, and report any discrepancies.

Make and run the design:

```
% make
```

The compilation process will take a few minutes.

You should see output that looks like this:

```
Environment variables are set correctly

  All tool output is redirected to "make.out"

MKDIR   ./object_dir
CXX     ../hardware_sources/testbench.cpp
LINK    -o testbench
make -C ../../utils
make[1]: Entering directory `/home/user/labs/utils'
./make_stim_response     ./image/people_gray.bmp
Loading Input File
Running
Finished
mv image.mem             ../data
mv expected_results.bin  ../data
make[1]: Leaving directory `/home/user/labs/utils'
./sim_sc

             SystemC 2.3.0-ASI --- Aug 22 2020 22:15:37
        Copyright (c) 1996-2012 by all Contributors,
        ALL RIGHTS RESERVED

Opening file "../../data/image.mem"
Reading image...
96 values loaded
Memory initialization complete.
Connections Clock: testbench.clk Period: 10 ns
WARNING: Default time step is used for VCD tracing.
Magnitude avg error = 0.925227
Angle avg error = 0.016451
Passed

Info: /OSCI/SystemC: Simulation stopped by user.
Simulation PASSED
```

You are specifically looking for the message “Simulation PASSED”.

With this working correctly, we can now use the RTL Rocket core in the design.

Change directory to the processor design:

```
% cd ../processor_design
```

The enter the software directory:

```
% cd sw.edge
```

Edit the file main.c. Uncomment the lines from 28 to 37. If line 25 is uncommented from earlier in the lab, comment out this line:

```
23    static float          hw_data_out[2 * IMAGE_SIZE];
24
25  //check_register_access();
26
27  /*
28    printf("loading data... \n");
29
30    load_data_array(data_in);
31
32    printf("Running... \n");
33
34    edge_detect_sw(data_in, sw_data_out);
35    edge_detect_hw(data_in, hw_data_out);
36
37    check_results(sw_data_out, hw_data_out);
38  */
39
40    printf("Finished \n");
```

Build the software image:

```
% make
```

You should see output that looks like this:

```
CC      main.c
LD      --script link.ld -o edge_detect.x
OC      edge_detect.x -O binary edge_detect.bin
OD      edge_detect.bin > edge_detect.mem
```

Change directories back to the processor design

```
% cd ..
```

Now build and run the design:

```
% make
```

The compilation process will take a few minutes.

You should see output that looks like this:

```
  All tool output is redirected to "make.out"

MKDIR   marker_dir/marker.mark
        checking SYSTEMC_HOME
        checking CONNECTIONS_HOME
        checking MATCHLIB_HOME
        checking AC_TYPES
        checking AC_STIMUTILS
        Environment variables are set correctly
VLIB    ./work
VMAP    -work ./work
GCC     ../../terminal/terminal_emulator.c
SCCOM   ../hardware_sources/systemc_subsystem_wrapper.cpp
SCCOM   -link
VLOG    ../hardware_sources/testbench.sv
VLOG    ../hardware_sources/top.sv
MAKE    rocket_design
make[1]: Entering directory
`/home/user/labs/lab3/processor_design'
MAKE    rocket_lib
make[2]: Entering directory
`/home/user/labs/lab3/processor_design'
make[2]: `rocket_lib' is up to date.
make[2]: Leaving directory
`/home/user/labs/lab3/processor_design'
VLOG    ../../rocket_design/addr_gen.sv
VLOG    ../../rocket_design/axi_addr_latch.sv
VLOG    ../../rocket_design/axi_byte_enables.sv
VLOG    ../../rocket_design/axi_data_latch.sv
VLOG    ../../rocket_design/axi_matrix.sv
VLOG    ../../rocket_design/axi_segment_arbiter.sv
VLOG    ../../rocket_design/axi_slave_if.sv
VLOG    ../../rocket_design/axi_slave_segment.sv
VLOG    ../../rocket_design/bus_fifo.sv
VLOG    ../../rocket_design/mux.sv
VLOG    ../../rocket_design/ready_gen.sv
VLOG    ../../rocket_design/rocket_subsystem.sv
VLOG    ../../rocket_design/sram.sv
make[1]: Leaving directory
`/home/user/labs/lab3/processor_design'
VOPT    testbench -o testbench_opt
```

```
>>----> Design compiled

make -C ../../utils
make[1]: Entering directory `/home/user/labs/utils'
./make_stim_response     ./image/people_gray.bmp
Loading Input File
Running
Finished
mv image.mem             ../data
mv expected_results.bin  ../data
make[1]: Leaving directory `/home/user/labs/utils'
VSIM    -work ./work -voptargs=+acc -L rocket_lib -do run.do -c
testbench_opt
loading data...
Running...
SW magnitude angle compute time: 48091
Vertical derivative compute time: 7480
Horizontal derivative compute time: 8250
HW magnitude angle compute time: 481
Magnitude avg error = 0.708333
Angle avg error = 0.000000
Passed
Finished
```

You are specifically looking for the line “Passed” near the end of the transcript.

In this case, the magnitude angle calculation now runs about 100 time faster. But this means that the differential computations, which were less than 10% of the total computations, are now 93% of the load in performing the edge detection.