Lab Worksheet 04

Using Custom Accelerator within a System Building Tool.

Objective: To use your custom square-root accelerator hardware to improve the execution time of a software loop that computes many square-roots.

Part 1:

Objective: To measure approximately the rate for computing square roots in software..

1) Review (and rebuild if necessary) the circuit that you built in LWS 01.
2) Edit your C-code: add a function
   ```
   int    sqrtSW( int x );
   ```
   that implements your square root algorithm from LSW 03 using C-code.
3) In the main() code, call the sqrtSW() with and place the result on the red LEDs
   check that you have the correct result for x = 25, 225 and 62025.
4) Devise and experiment to determine approximately the computation rate. (in sqrts/sec)
   You may find it useful to clear the LEDs after the main loop so that you know that the computation is complete. The answer for my code was around 8,000,000 sqrts/sec.
5) Write the loop so that x increments from 1 to 400,000,000 in steps of 1023. How long does this take?

Part 2:

Objective: **Control And Status Registers:**

The objective of this problem is to create your own components that can be used in an Qsys sub-system. You are to create your own versions of the parallel input and output ports that were used in Tutorial ***Introduction_to_the_Altera_Qsys_Tool-V15.0(33).pdf***.

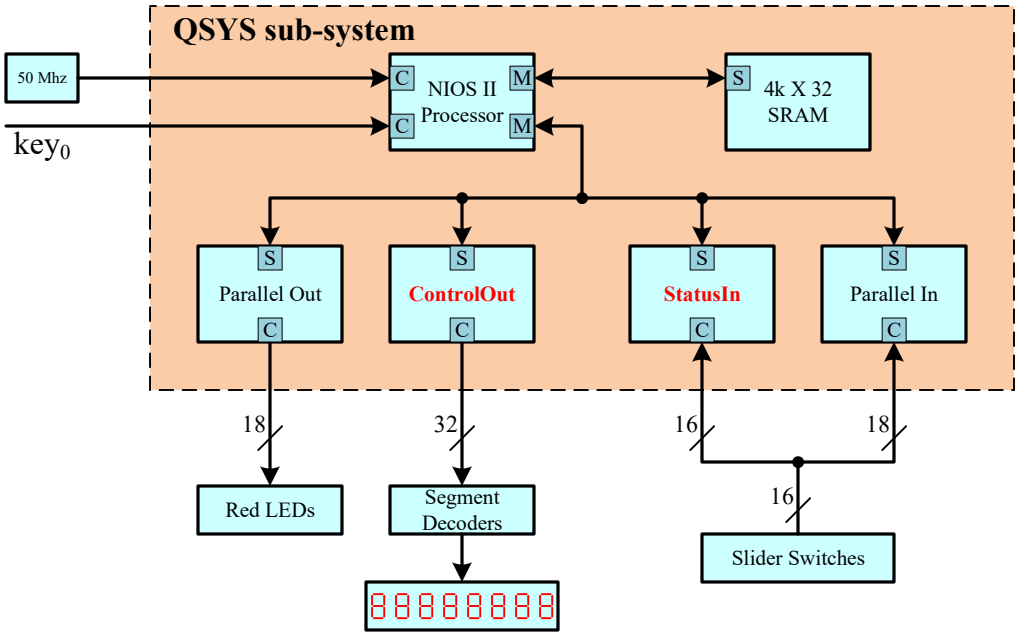
Specs:

Entity ControlOut:

- The processor can only write to this device.
- Has 32 outputs that are conduits to the upper design.
- The byte-enables should be used.
- One wait-state to be generated by the interconnect fabric.

Entity StatusIn:

- The processor can only read from this device.
- Has 16 inputs that are ***conduits*** from the upper design.
- The byte-enables may or may not be used.
- One wait-state to be generated by the interconnect fabric.



Part 2:

a) Reading Resources:

Attached .pdfs contain relevant pages extracted from

Avalon Interface Specifications(58).pdf

Qsys Components(38).pdf

b) Create the system described in the previous figure **without** your components, ***ControlOut*** and ***StatusIn***. Test that everything works correctly by executing a simple loop that reads 32-bit words from the switches and then writes the 32-bit pattern to the Red LEDs.

c) Add the 7-segment decoders. Test that everything is fine by connecting them to the lower 16-bits of the parallel port.

d) Now it's time to build the output only entity called ControlOut. Think carefully about which Avalon signals you will require for this memory mapped slave. Create a new directory inside your Quartus project directory called IP and then another directory within it called IP/ControlOut. Create a VHDL file called ControlOut.vhd inside this sub-directory, write the Entity declaration and add the file to the project.

e) Start up the Qsys and double click the "new component". On the second tab (HDL files) add your vhdl file. The builder will check the signals and syntax. Now look at the tabs labelled "signals" and "interface". The software should have recognised that you were attempting to create a single memory-mapped slave-port. Notice that you can edit your VHDL Entity declaration and then re-analyse the file without exiting the Component Editor.

f) Make sure that your Entity includes the following signals.

Questions:

How does the circuit differ if you use writebyteenable instead of byteenable?

Notice the warning that arises if you include a read signal.

	Interface type	*Signal Type*
a clock	clock in	clock
an active high reset	reset in	reset
your 32-bit output	conduit	export
two address lines	MM slave	address
an active high write	MM slave	write
4 byte enables	MM slave	byteenable
32-bit data lines	MM slave	writedata

Part 2:

g) Now set the parameters for the known interfaces. Make sure that there are 4 interfaces, a memory-mapped slave, a clock_in, a reset_in and a conduit.

- Rename these interfaces S0, C0, R0 and OutputData respectively.
- Check that the clock_in interface, C0 is correct. You may assign the known parameter for the clock frequency of 50 Mhz. This parameter is used when you connect the component into the system. The builder will check that the clock frequencies match. Test this.
- Check that the conduit interface is correct.
- Assign C0 to the reset_in interface, R0 and C0 to the MM slave, S0. If C0 and R0 do not appear in the drop-down boxes of S0 (associated clock and associated reset) then exit the component editor saving the component. Now right click on the new component and choose "edit" to restart the component editor. The interfaces C0 and R0 should now appear in the drop-downs.
- Assign ONE wait cycle for writing. Notice what happens if you include a WaitRequest signal in your interface.
- The Explicit Address Span. Notice that the units are words (32-bits) Ignore this for now.

Click finish. Notice that your component appears in the list of IP on the left.

h) Now you can create the Architecture for the device.
Design a 32-bit register:
The register contents should always be available at the Conduit Outputs.
The register should be cleared when the active high reset signal asserts.
The ByteEnable signals should be used for correctly writing to the register.
You should assume that the Master always inserts One Wait Cycle.

i) Add your new component to the Qsys system that you are building. Auto assign the base address and check the memory map. Generate the Qsys sub-system.
Connect the 7-segment decoders to the 32-bit output. Compile.

Questions:

- Try rebuilding your component with 6 address line and see what happens to the memory map.
- Try completely removing the address lines, rebuild the component, add it to the system and check the memory map. Notice that the component occupies 4 memory locations in the NIOS II address space.
- What happens if you explicitly specify an address span that is smaller that the span inferred from the address lines?

Part 2:

j) Modify your program so that we can verify that the address decoding and the byteEnables work correctly with `stbio`, `sthio` and `stwio`.

Tests:

- Verify that `stbio` works correctly in the lowest 4 bytes. What happens if you execute `stbio` using the 12 upper byte locations?
- Verify that `sthio` works correctly in the lowest 2 half-words. What happens if you execute `sthio` using the upper locations? What happens if you execute `sthio` with a misaligned address?
- Verify that `stwio` works correctly in the lowest word. What happens if you execute `stwio` using the upper locations? What happens if you execute `stwio` with a misaligned address?
- Does the Reset Work? Do you need to invert the pushbutton iKEY(0)?

k) Now it's time to build the input only entity called ***StatusIn***. Think carefully about which Avalon signals you will require for this memory mapped slave. Create a new directory inside your Quartus project IP directory called IP/StatusIn. Create a VHDL file called StatusIn.vhd inside this sub-directory, write the Entity declaration and add the file to the project.

l) Using the same procedures as before, create the component and add it to the SOPC system. Think carefully about the following.

- Should the action of loading the StatusRegister be related to the action of reading the register with the Avalon Slave Interface?
- What happens to the Slaves Data Lines when the interface is idle?

m) Modify your program so that we can verify that the address decoding and the byteEnables work correctly with `ldbio`, `ldhio` and `ldwio`.

Tests:

- Verify that `ldbio` works correctly in the lowest 4 bytes. What happens if you execute `ldbio` using the 12 upper byte locations?
- Verify that `ldhio` works correctly in the lowest 2 half-words. What happens if you execute `ldhio` using the upper locations? What happens if you execute `ldhio` with a misaligned address?
- Verify that `ldwio` works correctly in the lowest word. What happens if you execute `ldwio` using the upper locations? What happens if you execute `ldwio` with a misaligned address?

Part 3:

Objective: **Accelerating computation with sqrt hardware.:**
To interface the circuit that calculated sqrt() form the previous lab to a NIOS subsystem.

1) Create a new version of the square root circuit from the previous lab. This new circuit should be identical except that it also contain 4 registers acting as the FEC and BEC. There are a Control Register, Status register, Data Input Register and Data Output Register.

2) Using the knowledge that you have gained from part 2, connect these registers to an Avalon Memory Mapped Slave Port. Now add your component to Qsys.

4) Write a new C function called

```
int     sqrtHW( int x);
```

Write C-Code so that the start/done handshaking is performed by SW instructions. Transfer the value X to the Data Input Register and wait until the sqrt is calculated. Return the value read from the Data Output Register.

5) Change the main() code used in part 1 to now call sqrtHW(). verify that the computation produces the correct result and measure the approximate computation rate. (sqrt/sec)