ECE4530 Fall 2017
Homework 7: SignalTap II - Logic Analyzer inside your FPGA

Assignment posted on 7 November
Solutions due on 14 November 5:00PM

Homework 7 is an assignment on the use of SignalTap II, a logic-analyzer tool for your FPGA. A logic analyzer is a digital measurement tool that captures the trace of digital signals in real time on the FPGA. It is useful for low-level debugging of hardware, such as verifying if a hardware interface operates correctly. In this Homework, you will:

- prepare a setup using SignalTap II;
- implement a small coprocessor on the FPGA;
- drive the coprocessor from C;
- and capture the interface signals of the Avalon bus in SignalTap II.

The following materials will be useful in preparation of the Homework.

- Altera has a short video tutorial about SignalTap II at `https://www.youtube.com/watch?v=vhkzxCEXuaA`.
- A comprehensive manual of SignalTap II can be find online at `https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/qts/qts_qii5v3.pdf`
- This Homework also directly builds on the system we discussed in Lecture 17, and the coprocessor we discussed in Lecture 18.

Initial hardware and software setup

1. The design that we will analyze in this Homework is the baseline design of Lecture 17, with the `mymul` 64-bit multiplier coprocessor attached to the lightweight bus. You will be given the `mymul.v` design as part of the Lecture 18 materials, and the baseline design as part of the Lecture 17 materials. Use QSYS to create a new component (`mymul`) and attach that to the lightweight HPS-to-FPGA bridge.

2. Generate HDL. Close QSYS and synthesize the design in Quartus. Generate a bitstream.

3. Open a command shell and navigate to `conversion` directory next to where you generated the bitstream.

4. Type `make`. This generates a header include file `hps_0.h` and and rbf file `baseline.rbf`.

5. Finally, go to the software directory, and compile the sample program `piodemo`. This program repeatedly scans the switches and reflects the switch status on the LEDs. Note that the sample program does not use the `mymul` coprocessor; that will come later.

6. To test if the system is operating correctly, copy `piodemo` and `baseline.rbf` to the DE1-SoC board. If the board does not have the `hps_config_fpga`, you need to copy that as well. There is a copy in the `conversion` subdirectory. Copy the files with `scp`. The exact command will depend on your network configuration, but could look similar to the following.

   ```
   scp -P50444 hps_config_fpga  root@192.168.1.120:/home/root
   scp -P50444 baseline.rbf  root@192.168.1.120:/home/root
   scp -P50444 ../software/piodemo  root@192.168.1.120:/home/root
   ```

7. Log in to the board, configure the bitstream, and run the program. You should see activity on the LED display. Refer to the main program in C for the exact behavior of the program: you can use switches to turn on or tunr off the LEDs, and when all even switches are on, the program will terminate.

   ```
   root@socfpga:~# ./hps_config_fpga baseline.rbf
   root@socfpga:~# ./piodemo
   ```

8. You are now ready to hook up the SignalTap II Logic Analyzer

Configure the SignalTap II Logic Analyzer

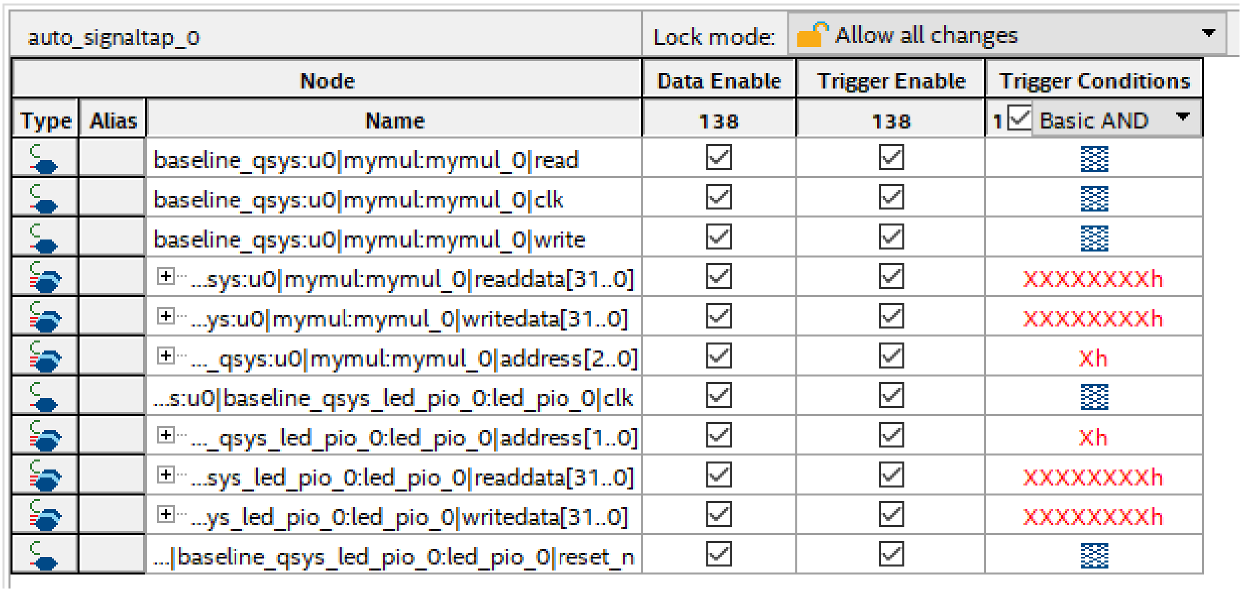
1. In a nutshell, we will do the following operations. We will configure the Logic Analyzer to capture the bus interface of the onchip memory, and the bus interface of the 64 bit multiplier. Then, we will write a test program that exercises the multiplier. Then, we will use SignalTap to capture the transactions on the bus. The answer to this homework will consist of the test program in C, and a screen capture of the SignalTap II waveform window showing the requested transactions.

2. Before you start, make sure that you connect your laptop to the FPGA board using a USB cable and the USB Blaster port. Signaltap requires access to the USB Blaster port and the JTAG interface behind it.

3. Open the SignalTap II Logic Analyzer in Quartus, under Tools.

4. Double-click in the data window to add signal nodes to capture. This opens the Node Finder. To identify the signals we're looking for, you need to select and filter the 'Signal Tap II: pre-synthesis' nodes. Look for the following entities: mymul_0 and led_pio_0. Navigating the Node Finder requires a bit of patience because of the large amount of signals.

5. For mymul_0, select the following signals for capture: clk, read, write, readdata, write-data and address. For led_pio_0, select the following signals for capture: clk, write_n, address, readdata, writedata. Add those nodes and close the node finder. If all went well, the data pane will now list the signals as shown below.



auto_signaltap_0 — Lock mode: Allow all changes

Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	138	138	1 ✓ Basic AND
		baseline_qsys:u0|mymul:mymul_0|read	✓	✓	▨
		baseline_qsys:u0|mymul:mymul_0|clk	✓	✓	▨
		baseline_qsys:u0|mymul:mymul_0|write	✓	✓	▨
		...sys:u0|mymul:mymul_0|readdata[31..0]	✓	✓	XXXXXXXXh
		...ys:u0|mymul:mymul_0|writedata[31..0]	✓	✓	XXXXXXXXh
		..._qsys:u0|mymul:mymul_0|address[2..0]	✓	✓	Xh
		...s:u0|baseline_qsys_led_pio_0:led_pio_0|clk	✓	✓	▨
		..._qsys_led_pio_0:led_pio_0|address[1..0]	✓	✓	Xh
		...sys_led_pio_0:led_pio_0|readdata[31..0]	✓	✓	XXXXXXXXh
		...ys_led_pio_0:led_pio_0|writedata[31..0]	✓	✓	XXXXXXXXh
		...|baseline_qsys_led_pio_0:led_pio_0|reset_n	✓	✓	▨

6. Next, add a clock to the SignalTap. Add the system pin CLOCK_50 as the clock signal, and select a buffer depth of 1K signals.

7. Then click the big play 'play' button to recompile the bitstream. This may take a while (monitor the progress in Quartus). Eventually, it finishes and you can re-download the bitstream through SignalTap (no need to use hps_config_fpga). Press the 'SOF Manager' download button in the upper right of the SignalTap window to re-download the bitstream.

8. To write the driver program, you will need to figure out how to access the mymul module and the PIO port. Study the memory map of the system (for example, using QSYS) to find what memory regions these modules use. Use the `mmap` system call, following the example of the LEDs, to map the real hardware addresses to a virtual pointer.

9. Write a driver function `mymul_hw` which has the following interface.

```
unsigned long long mymul(volatile unsigned *base,
                         unsigned a,
                         unsigned b) {
    // base is the address of the first memory-mapped register of multiplier
    // study mymul.v to write the rest of the function:
    // - load argument a in a memory-mapped register
    // - load argument b in a memory-mapped register
    // - drive the memory-mapped control signal
    // - read result z from a memory-mapped register
    //   since z is 64 bit, this may require two reads over a 32-bit bus
}
```

10. The actual testbench, which uses a pointer `led_pio` and a function `mymul_hw` to call the multiplier, must look exactly as follows.

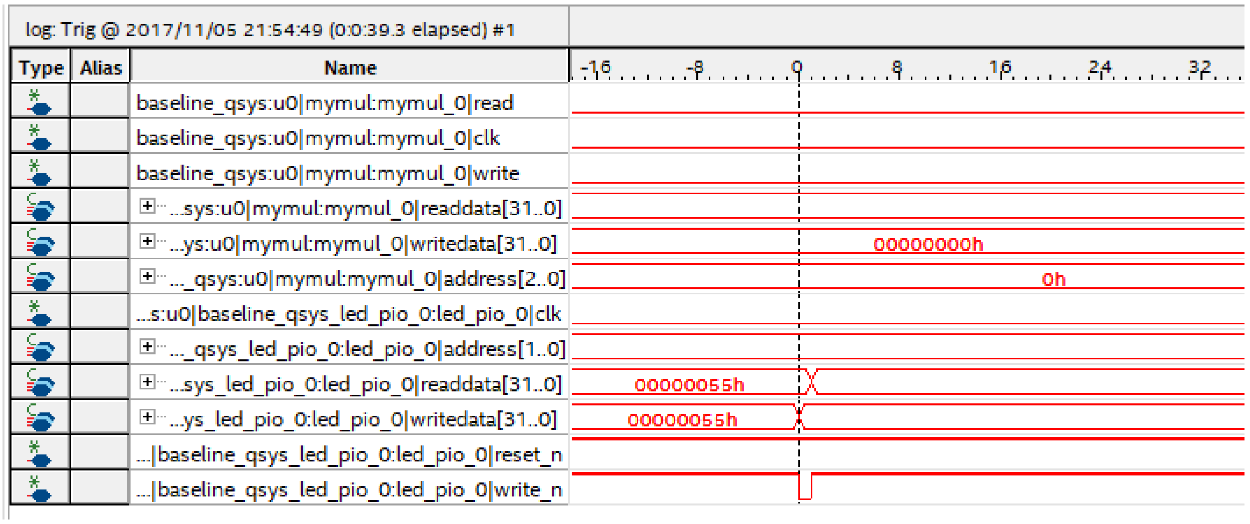
```
unsigned long c = 1;

while (1) {
  *led_pio = *switch_pio;
  c = mymul_hw(*switch_pio, c);
  if (*switch_pio == 0x2AA)
    break;
  printf("Multiplier: %x\n", c);
}
```

11. After you compile the program, you can download the binary to the board (eg using `scp`) and run it. You will see a stream of values as shown below. When all switches are off, the multiplier value is 0 and stays zero. Otherwise, it will print a sequence of multiplications.

```
Multiplier: e44c2de9
Multiplier: 757ce58d
Multiplier: 4b707bc1
Multiplier: 79326ac5
Multiplier: 5dfc15d9
Multiplier: d5ec6d3d
Multiplier: 2d9e2231
Multiplier: e416aaf5
Multiplier: 747156c9
Multiplier: 4636b1ed
Multiplier: 5f1179a1
```

12. The assignment using SignalTap is as follows. You have to select the trigger conditions and capture a waveform. Capturing a waveform can be initiated by clicking the small play button in the signaltap window, after the proper trigger conditions are set. For each of the following captures, you have to make a screenshot of the captured waveform, and of the trigger conditions.

13. The first transaction to capture is the point when the led_pio is written with the data 0x155. That is, all odd LEDs are set. The following gives an idea of the waveform you should try to capture.



log: Trig @ 2017/11/05 21:54:49 (0:0:39.3 elapsed) #1

Type	Alias	Name	-16 -8 0 8 16 24 32
		baseline_qsys:u0|mymul:mymul_0|read	
		baseline_qsys:u0|mymul:mymul_0|clk	
		baseline_qsys:u0|mymul:mymul_0|write	
		...sys:u0|mymul:mymul_0|readdata[31..0]	
		...ys:u0|mymul:mymul_0|writedata[31..0]	00000000h
		..._qsys:u0|mymul:mymul_0|address[2..0]	0h
		...s:u0|baseline_qsys_led_pio_0:led_pio_0|clk	
		..._qsys_led_pio_0:led_pio_0|address[1..0]	
		...sys_led_pio_0:led_pio_0|readdata[31..0]	00000055h
		...ys_led_pio_0:led_pio_0|writedata[31..0]	00000055h
		...|baseline_qsys_led_pio_0:led_pio_0|reset_n	
		...|baseline_qsys_led_pio_0:led_pio_0|write_n	

14. The second transaction to capture is the point when the multiplier control register is written with a logic-1. You can capture any multiplication you like. Prepare a similar screenshot as shown for the first transaction, and include it with the solution of the homework.

15. The third transaction to capture is the point when the multiplier computes 0x100 as the output data. You may have to consider for a bit how you can bring the system

in this configuration. Using a proper setting of trigger conditions in SignalTap II, manipulation of the switches, and running of piodemo, you can get the exact output needed quickly.

What to turn in

On the root directory of the git repository, include a PDF with the following information

1. A listing of the C program you used to drive the signaltap transactions
2. For each of the three transactions requested, a screenshot showing the waveform window (first shot) and the trigger conditions (second shot).