Contents

1 HLAB1: Xilinx ISE WebPACK and the Spartan 3E Starter Board 3
  1.1 Aims ........................................................................................................... 3
  1.2 Instructions ................................................................................................ 3
  1.3 Equipment .................................................................................................. 3
  1.4 Electronic Safety ......................................................................................... 4
  1.5 Preliminaries ............................................................................................... 4
  1.6 A Counter (6 marks) .................................................................................. 4
      1.6.1 ISE Implementation of the Counter ...................................................... 7
  1.7 Implementation of a Two bit ADDER (8 marks) ......................................... 9
      1.7.1 The 1-bit adder ..................................................................................... 10
      1.7.2 Top Modules ......................................................................................... 11
      1.7.3 ISE Implementation (6 marks) ............................................................... 12
  1.8 Implementation of a 1-Bit MUX (6 marks) .............................................. 15
1 HLAB1: Xilinx ISE WebPACK and the Spartan 3E Starter Board

1.1 Aims

- To build on skills in using Xilinx ISE. Specifically managing multiple files and simple coding examples.
- By the end of the lab students should be comfortable creating a simple working system from scratch, both configuring the project and writing the code without assistance.
- Learn to use XILINX ISE software and FPGA hardware.
- Use VERILOG to create an ADDER and a MUX combinational circuits.
- Learn how to draw block schematics and how to design VERILOG from them.
- Learn how to implement a counter.
- Learn how to compose simple user constraint files.
- Use iMPACT to download to hardware XILINX FPGA hardware

1.2 Instructions

Read through these notes and do any additional reading from the lecture notes to help your understanding.

Use your lab notebook to enter results and answer the questions. Also date and record your procedures for future reference.

Remember to save all your files to your personal USB flash stick for future use. This includes ISE project folders so you can run them at home.

1.3 Equipment

Xilinx ISE WebPACK 9.2i running on a Windows PC and a SPARTAN 3E Starter FPGA board.

Location: Telecoms and signals lab, Ian Ross room 103.
1.4 Electronic Safety

CAREFUL NOT TO TOUCH THE EXPOSED ELECTRONIC CIRCUITS WITH YOUR HANDS. LIKE MOST ELECTRONIC CIRCUITS THEY CAN BE DESTROYED BY E.S.D. (ELECTROSTATIC DAMAGE).

BEFORE COMING INTO CONTACT WITH THE SPARTAN 3E CIRCUIT BOARD, BE SURE TO EARTH YOURSELF BY TOUCHING AN EARTHED OBJECT SUCH AS THE METAL SCREWS ON THE BACK OF A PC.

1.5 Preliminaries

In this course we will develop complex digital systems in Field Programmable Gate Array (FPGA) technology. FPGAs are programmable logic devices (PLD) in which highly complex digital logic circuits can be implemented. The FPGA we will use is the Xilinx Spartan 3E XC3S500E chip. FPGAs come in many formats and the present is an FG320 (320 pin ball grid array) package of speed grade -5. As FPGAs are SDRAM based they are volatile meaning that they lose their configuration when power is removed. This is handy because it allows us to reuse the FPGA for many designs. It is also possible to store the FPGA configuration in an on-board flash PROM so that the FPGA can be reconfigured automatically at power up. Some other types of PLDs cannot be reused. The XC3S500E FPGA is the central device on the SPARTAN 3E development board (S3E board). The S3E board also has many peripherals that can be controlled by the FPGA. Take a moment to have a look at them.

A datasheet for the XC3S500E FPGA and the user guide for the S3E board can be downloaded from http://engnet.anu.edu.au/DEcourses/engn3213/Documents/FPGA/. Be sure to keep them handy on your computer.

There is also an expansion breadboard that plugs into the S3E board. It will be useful in the wireless project for the addition of analog circuitry.

1.6 A Counter (6 marks)

This is your first VERILOG implementation in hardware. The exercise is a typical “Hello FPGA World” program that lights up the eight LEDs (Light Emitting Diodes) on the S3E board (bottom right hand corner of Fig. 1). The exercise not only illustrates most of what you need to do get a design working on the S3E but also shows you a simple way to implement an modulo up-counter.

First plug in the S3E board. You will need to plug in the power supply (top left hand corner of Fig. 1) and connect a USB cable to the USB port (just below the ethernet

1These are large documents and there is probably no need to print them
connector half way down the left hand side of Fig. 1. The USB cable allows you to download fuse files using JTAG².

The following is the VERILOG code for the led flasher.

```
'timescale 1ns / 1ps

// Company: Australian National University
// Engineer: Gerard Borg
// Create Date: 13:53:15 08/30/2009
// Design Name: ledflash
// Project Name: ledflash
// Target Devices: XC3S500E
// Tool versions: ISE WebPACK 9.2i
// Description: flash 8 leds on the S3E board

// Dependencies:

² JTAG stands for "joint task action group" and is a protocol for in-situ testing of electronic circuits (see http://www.inaccessnetworks.com/projects/ianjtag/jtag-intro/jtag-intro.html)
Here are the most important points about this code.

1. The 'timescale' parameter sets the time step for simulation. It is common to see it in VERILOG for synthesis but it is only used by the simulator.

2. The // starts a comment (ignored by the compiler). Note the nature of the documentation at the top of the module.

3. Every VERILOG module starts with the module declaration and a set of input and output arguments (wires). The sysclk is the system clock. It is an input and is the ultimate source of all timing in the module. It is the main FPGA clock ultimately provided by an external crystal. You can read all about the S3E’s XC3S500E clock capabilities in Chapter 3 of the S3E user guide. The output LEDS is an 8 bit wire bus that lights up the physical LEDs on the board.

4. reg [23 : 0] counter is a 24 bit register which is the modulo up-counter. The counter works by adding a 1 bit to counter at every positive edge of sysclk. The first always block shows how this is done. Note that non-blocking <= assignment has to be used as a combinational = makes no sense when a register is incremented by adding 1 to itself.

5. The second always block shows one way the eight bits of LEDS can be assigned.

Explain in your log books how the modulo up-counter works. Why does it overflow? (1 mark).

Explain exactly why counter = counter + 1 might be unimplementable in hardware (1 mark).
If the sysclk is 50MHz then at what rate is the slowest LED flashing? (1 mark).

Propose a different approach to using an 'if' statement for the purpose of assigning the LED values (1 mark).

Draw the block schematic of the module (1 mark).

1.6.1 ISE Implementation of the Counter

Start ISE and implement the design. Follow the steps in Fig. 2 starting at the top left and moving across the page and down.

I believe that you should always go to ISE 'VERILOG ready' so open WORDPAD and enter the above VERILOG code: call the file 'ledflash.v' and make sure it does not have any MS '.TXT' ending.

In order to save us entering constraints through the ISE interface, create 'ledflash.ucf', the user constraints file and enter the code below. The UCF file gives ISE information about the design. Here it defines the FPGA pins on which the S3E LEDs are located, the IOSTANDARD voltage level of 'low voltage TTL' and SLEW RATE. FPGAs provide considerable control over digital standards and the UCF allows you to express it. Timing constraints, area constraints and even IP deployment constraints can be set by the UCF file.

Surprisingly, just these two files are all you need for a complete design.

```
NET "clk" LOC = "C9" ;
NET "LEDS[0]" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS[1]" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS[2]" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS[3]" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS[4]" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS[5]" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS[6]" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS[7]" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

1. First close any existing project and start a new project.
2. Make sure that you enter the correct values in the 'DEVICE PROPERTIES' as shown.
3. Skip 'NEW SOURCE' and go to 'ADD SOURCE'. Add the two ledflash files.
4. Next highlight the verilog modules and 'IMPLEMENT TOP MODULE'. ISE will now run through all the steps automatically to implement the design.

---

3 Test benches may also be required if you wish to do your simulations in ISE. However there is no need to do simulation in ISE if you are happy with your non-ISE simulations such as those in ICARUS VERILOG. It is good practice to verify your designs by a correctly functioning simulation before going to hardware.
5. If all goes well as shown in the fifth figure, there will be green ticks beside all the successfully completed processes. Note the command console at the bottom where WARNINGS and ERRORS are posted. ISE won’t let you proceed with a design if there are ERRORS and in any case you should investigate all WARNINGS. Very often this requires you to visit Xilinx’s support site. Sometimes WARNINGS are just ISE bugs.

6. After a successful implementation there are two major outcomes. (a) A bit or fuse file 'ledflash.bit' is produced for download into the FPGA. (b) A complete design report is produced as well as numerous other useful bits of information revealed by tools such as the floorplanner. Take some time to snoop around and see if you can understand what is going on.

7. The next step is to invoke IMPACT and download the fuse file. For this to work the USB cable has to be connected to the board and the USB green light indicator on the board must be illuminated. Also you may have to right click on PROPERTIES over 'Configure Device (IMPACT)' and make sure that the USB connection is selected. If you are using a parallel port JTAG cable then select LPT instead.

8. In the sixth figure the button 'Configure devices using Boundary Scan (JTAG)' should already be checked so click 'OK'.

9. The seventh figure shows the results of the boundary scan. There are three devices detected by JTAG. In addition to the FPGA there is an XCF04S Platform Flash PROM (programmable read only memory) and an XC2C64A CoolRunnerTM-II CPLD (configurable programmable logic device). These devices also 'talk JTAG' and can be configured using ISE. In this exercise we bypass them.

10. The XC3S500E is highlighted when IMPACT starts up. Ignore the warning about the JTAGClk clock as shown in the eighth figure by clicking 'OK'.

11. As shown in the ninth figure bypass the other two devices.

12. Return to the XC3S500E icon by right clicking on it and choose 'PROGRAM'. Navigate to 'ledflash.bit' and load it into the FPGA as shown in the tenth figure.

13. Finally click OK in the pop-up window 'PROGRAMMING PROPERTIES' as shown in the eleventh figure.

14. After a few seconds the FPGA will be programmed and as shown in the twelfth figure a 'PROGRAM SUCCEEDED' message is posted.

15. At this point the eight LEDs should be flashing. Try to understand what is happening with the LEDs in terms of the VERILOG.

4ISE WebPACK actually points you automatically to the URL
Figure 2: The sequence of steps for implementing the ledflasher design

If you got this to work then be awarded 1 mark.

1.7 Implementation of a Two bit ADDER (8 marks)

In this exercise we are going to combine two 1-bit adders to make one two bit adder whose results will be displayed in 3 bits on the LEDs LD0-LD2 with the LSB on LD0. The inputs can be set using the switches SW0 - SW3 at the bottom right hand corner of the S3E board. In order to do this we will have to learn about top modules and how to design using hierarchical VERILOG.
1.7.1 The 1-bit adder

Open your 1-bit adder project from CLAB1. Recall that the truth table of the 1-bit full adder is as follows.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Cin</th>
<th>Z</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The truth table of the 1-bit full adder

The VERILOG for the 1-bit full adder is as follows.

```verilog
/*******************************************************************************
			timescale 1 ns/1 ps
	module add1(X, Y, Cin, Z, Cout);

	input  X;
	input  Y;
	input  Cin;

	output Z;
	output Cout;

	reg    Z;
	reg    Cout;

	always @(X, Y, Cin) begin

	    Z    = ??;  

	    Cout = ??;  

end

endmodule
/*******************************************************************************

If you haven’t already done so, complete the ?? code in the module (2 marks).
1.7.2 Top Modules

The top module is the module in your project whose inputs and outputs interface to the outside world (i.e. connect to the IO PINS of the FPGA). These interfaces are VERILOG *wires*. A block schematic is a block diagram of your digital hardware showing all the interconnects. Since different blocks within the schematic also need to be interconnected there must be a way of interfacing VERILOG submodules as well. The VERILOG programming technique that allows this is termed *module instantiation* and is the key to hierarchical VERILOG.

![Block schematic of the two bit adder](image)

Figure 3: Block schematic of the two bit adder

Fig. 3 shows the block schematic of the two bit adder. Note the two 1-bit adders and the manner in which they are interconnected. This device performs two bit binary addition with carry on two 2-bit numbers X and Y entered by adjusting the sliding switches on the S3E board. Note how it deals with the carry-in for the LSB and how the carry-out leads to an extra third bit in the output. The addition of two 2-bit numbers requires 3 bits for the result.

Given a 1-bit adder here is how to make a VERILOG top module to implement a two bit adder.

```verilog
`timescale 1ns / 1ps
```
First of all note that X and Y are 2-bit wires (2-bit buses) and Z is a 3-bit wire (3-bit bus) as required. The IO pins of the FPGA are connected to the sliding switches, SW0/SW1 for X and SW2/SW3 for Y and to LD0/LD1/LD2 for Z.

There are two new and interesting aspects to the top module, add2. First is that the two 1-bit adders labelled a1 and a2 actually use the same VERILOG 1-bit adder, add1. The only difference is their arguments. Compare the add1 module arguments to those used by a1 and a2 to understand how this works. Thus we see that the add1 module can be reused as expected.

The second is the **assign** statement. The **assign** hard wires the 3-bit Z bus to the individual 1-bit LEDs LD2, LD1, LD0. **ASSIGNS** are an alternative to the combinational **ALWAYS** block shown commented out in add2. The **ALWAYS** block is a simpler approach for complex combinational systems such as large look-up tables (LUTs). Note that the **assign** requires Z to be a **reg** as explained in CLAB1.

**1.7.3 ISE Implementation (6 marks)**

Use the information in Fig. 4 to make a suitable UCF file. add2.ucf, like the one used in the LED flasher.
Figure 4: The sliding switches and LEDs. To the right are the UCF file snippets

1. Complete the code in add1.v and add2.v, open the CMD prompt and check your code compiles free of errors with ICARUS VERILOG:

```
iverilog add1.v add2.v
```

2. Open ISE and follow the same steps as for the led flasher. Be sure to close the old project and start a new one. Call it ‘add2’.

3. Add the three files add1.v, add2.v and add2.ucf to the project. The steps are shown in Fig. 5. When you are done entering the information, the ISE source pane should look as shown in the second figure in the sequence. Be sure that you have the correct file hierarchy with add2.v on top (top module), followed by two instances a1 and a2 of add1.v in turn followed by the UCF file. Note that ISE has to guess the structure from your code and if it gets it wrong then there is probably something wrong with your VERILOG code structure.
4. Implement the code as before. When done open the device constraints and check that the UCF file produced the correct pinouts. Remember that the sliding switches SW0 - SW3 are used to enter the values of X and Y in binary and the LEDs LDS0-LD2 to display Z.

5. When implementing the code it is possible that it will produce the benign WARNING shown in the last figure of Fig. 5. You may ignore it.

6. Run iMPACT to download the fuse file into the FPGA.

7. Spend some time browsing the design report, the RTL schematic (Fig. 6) and the floorplanner. From Fig. 6 you need to double click on the add2 block schematic to dig deeper to the more detailed schematic on the right. Note the use of LUTs to handle the adder truth table. Why this is the case should be clear from the structure of a CLB mentioned in lectures. Confirm this by studying the floorplanner.

8. Try out the adder. Does it work? Document the results in your log book.
If you successfully completed the above steps and your two bit adder works then be awarded 6 marks.

1.8 Implementation of a 1-Bit MUX (6 marks)

In this section you are on your own. Take the 1-bit MUX of CLAB1 and implement the schematic shown in Fig. 7 on the S3E board. Design it so that SW0 is Sel., SW1 is X, SW2 is Y and LD0 is Z. Either compose a suitable UCF file or use the constraints editor to assign the package pins.
Figure 7: Block schematic of the 1-bit MUX

Try out the MUX. Does it work? Document the results in your log book.

If you successfully completed this exercise and your one bit MUX works then be awarded 6 marks.