1 HLAB2: The Keyboard

Aim:

1. To implement your VERILOG deserialiser developed in the prelab exercise to read characters from a keyboard.

2. Meet our first finite state machine (FSM).

3. Learn how to develop for the keyboard.

4. Obtain experience in combining several simple VERILOG modules to form a complex digital system.

5. Look at some VERILOG code that produces spurious latches.

Your name, student ID, the date and the name of the module MUST be entered into every module that you produce

1.1 Introduction

In this lab we are going to communicate with the FPGA via the PC keyboard. You will use your prelab2’s deserialiser to do the keyboard scan code acquisition. In the RPC (reverse polish calculator) and the WAT (wireless audio terminal) projects a PC keyboard will be used to communicate with the FPGA. The Spartan 3E starter board comes with a PS/2 port just like that used on a PC. In the prelab we developed a deserialiser which will allow us to acquire keyboard data in this lab.

This lab takes us two steps further in our understanding of sequential circuits:

1. Look at serial communications which can be modelled by a shift register sequential circuit.

2. Implement a simple finite state machine which consists of a couple of combinational circuits and a sequential circuit.

In each case the sequential circuit is an obvious extension of a D type flip-flop. A shift register is simply a concatenation of D flip-flops. A FSM the sequential circuit is even simpler,

always @(posedge sys_clk) begin
    state <= nextstate;
end
1.2 The Keyboard Scan Code Finite State Machine

You should have read a bit about keyboards by now. In case you have not, here is a quick explanation. Keyboards run a finite state machine (FSM) in which serial characters referred to as SCAN codes are transmitted over PS/2. These scan codes represent information about the key pressed and when it is released. There are both normal and extended SCAN codes. When a key is pressed a key stroke or SCAN code representing that key is transmitted. When the key is released a special escape character $F0$ is transmitted followed by a repeat of the SCAN code. Fig. 1 shows the keyboard numeric key pad with the SCAN codes written inside the key squares they represent. It is the task of your deserialiser to catch these SCAN codes and convert them into bytes from which 5-bit words can be extracted.

![Keyboard Numeric Key Pad]

**Figure 1:** The keyboard numeric key pad. The SCAN codes appear on the keys they represent.

Fig. 2 shows GTKwave traces simulating what happens when the zero and one keys on the keyboard number pad are pressed. The variable `keyStroke` is the detected SCAN code and `keyID` is the actual value of the key. Note that the SCAN code is not the same as the `keyID` of the key. The VERILOG file `KeyStroke2KeyID.v` contains the list of their correspondences for the case of the IBM number pad.
Figure 2: GTKwave traces of the keyboard scan code *keystroke* when the zero and one keys on the number pad are pressed.

The VERILOG file *KeyStroke2KeyID.v* also contains the FSM. The FSM is shown in Fig. 3. The FSM receives SCAN codes *keyStroke* from the keyboard and produces character outputs, *keyID*. The reset button *rst* sets the state machine to the idle state $Si$ (the FSM should eventually arrive here without a reset because all other states default to $Si$). For normal key characters, when the key is released an $F0$ comes from the keyboard and the state machine moves to the $S0$ state where it outputs the *keyID*. The FSM also treats extended SCAN codes $^1$.

```

Figure 3: The Keyboard scan code state machine

^1In this lab we will just implement the right hand number pad of an IBM keyboard: in the WAT project you may wish to send alpha characters as well so the SCAN code list in 'KeyStroke2KeyID' will have to be augmented.

5
1.3 The Keyboard VERILOG Sources

Download the HLAB2 source code from,


The ZIP file contains everything you need except for the deserialiser. Fig. 4 shows the system arrangement. In the RPC you will display results on an LCD and in the WAT project on a VGA monitor. The keyboard is common to both projects. For now we tackle the keyboard and display the results on the LEDs on the Spartan 3E FPGA board.

![Keyboard - FPGA - LED system](image)

Figure 4: The Keyboard - FPGA - LED system

As can be seen from Fig. 4 there are two lines from the keyboard for PS/2 serial communications. These are PS2_data and PS2_clk. There are five output lines to the S3E LEDs because this is all that is required by the numeric keypad. Take a look at the Kbd2LEDS.ucf and read the relevant sections of S3EStarter_user230.pdf to understand how the input and output interfaces work.

Fig. 5 shows the block schematic. The top module Kbd2LEDS.v does the interfacing to the outside world and wires up all the submodules including your serial_rcvr.v. It also uses the following code to capture the keyID for transmission to the LEDs.

```vhdl
always @(posedge sys_clk) begin
  if(~|(keyID^NOKEY)) LEDS <= LEDS;
  else LEDS <= keyID;
end
```

The variable NOKEY which has the value 1E represents undefined key values. Only valid keys can be displayed on LCDs or VGAs. The key value NOKEY occurs during times when the deserialiser is acquiring data but the done flag is not set as can be seen from the traces of Fig. 4.
The module `SwDebounce.v` debounces the `rst` switch. Note that this is a very simple and token debounce which merely synchronises `rst` to the system clock. We will learn some proper debouncing circuits later on.

```
always @(posedge sys_clk) rstdb <= rst;
```

`ps2_clk_syn.v` synchronises the `PS2_clk` to the system clock as follows. This is a satisfactory solution that is a requirement of good sequential design practice.

```
always @(posedge sys_clk) ps2_clk_syn <= ps2_clk;
```

`KeyStroke2KeyID.v` not only contains the relationship between the `KeyStrokes` (SCAN codes) and the `keyIDs` but also implements the keyboard FSM.

There is also a testbench to simulate the outside world `TB_kbd_to_display.v`.

### 1.4 Exercises

**Exercise 1: (3 marks)** Use ICARUS VERILOG, GTKWAVE and `TB_kbd_to_display.v` to simulate the keyboard. Your results should reproduce those of Fig. 2 if you have a properly debugged deserialiser.

**Exercise 2: (5 marks)** If you have a working simulation then open ISE and run the project. Plug in a keyboard and check that the correct LEDS are illuminated when you type the keys 1,2,3,4,5,6,7,8,9 of the numeric key pad.

**Exercise 3: (2 marks)** In the top module, `kBD2leds.V` replace the following code,

```
always @(posedge sys_clk) begin
    if(~|(keyID^NOKEY)) LEDS <= LEDS;
    else LEDS <= keyID;
end
```

with the following combinational formulation,
always @(keyID) begin
if(~|(keyID^NOKEY)) LEDS = LEDS;
else LEDS = keyID;
end

Rebuild the ISE project. Do you get any warnings about latches? If so explain why they have occurred.