

EGCP 1010
Digital Logic Design (DLD)
Laboratory #7

Introduction to the Quartus II Tools and
PS/2 Keyboard Interface

Prepared By:
Alex Laird

on

November 10, 2007

Lab Partner:
Ryan Morehart

Objective:

The goal of this laboratory is to provide the student with an introductory knowledge of the powerful commercial Altera Computer Aided Design tool. It will also teach the student the basic workings of the UP1 Programmable Logic Board and the PS/2 Keyboard.

Procedure and Results:

In the Pre Lab, it was learned that there are two signal lines used to transmit the code from a PS/2 Keyboard: A clock line and a data line. Eleven (11) clock cycles occur, sending one bit at a time over the data line. Each bit is sent individually on the clock.

Each make code requires eleven (11) clock cycles to complete. Eight (8) bits are the make code for that particular key on the PS/2 Keyboard, the remaining three (3) bits are added as a Parity after the key code, a high (1) Stop bit at the end of the make code, after the Parity bit, and a low (0) Start bit at the beginning of the make code.

560 microseconds are required for the key code, and a total time of 735 microseconds are required for the complete make code. This is determined by the fact that each clock cycle requires 70 microseconds to complete, and there are eleven (11) clock cycles. However, the very first clock cycle, for the Start bit, only remains low for half of a clock cycle, so only 35 microseconds are needed for that clock.

If the key "Y" is pressed on a standard PS/2 Keyboard, the following complete make code is sent: 35. When the "Y" key is released, the following break code is sent: F0 35.

When breaking (releasing) a key, F0 is always sent prior to the code for the key. When a key is released, the make code for that key is sent after the code F0, which means whatever make code follows this, break it. As a result, two complete codes must be sent in order to break a key, which requires a total time of 1,470 microseconds. The timing diagram for the "Y" key being pressed and then released is shown below in Figure 1.

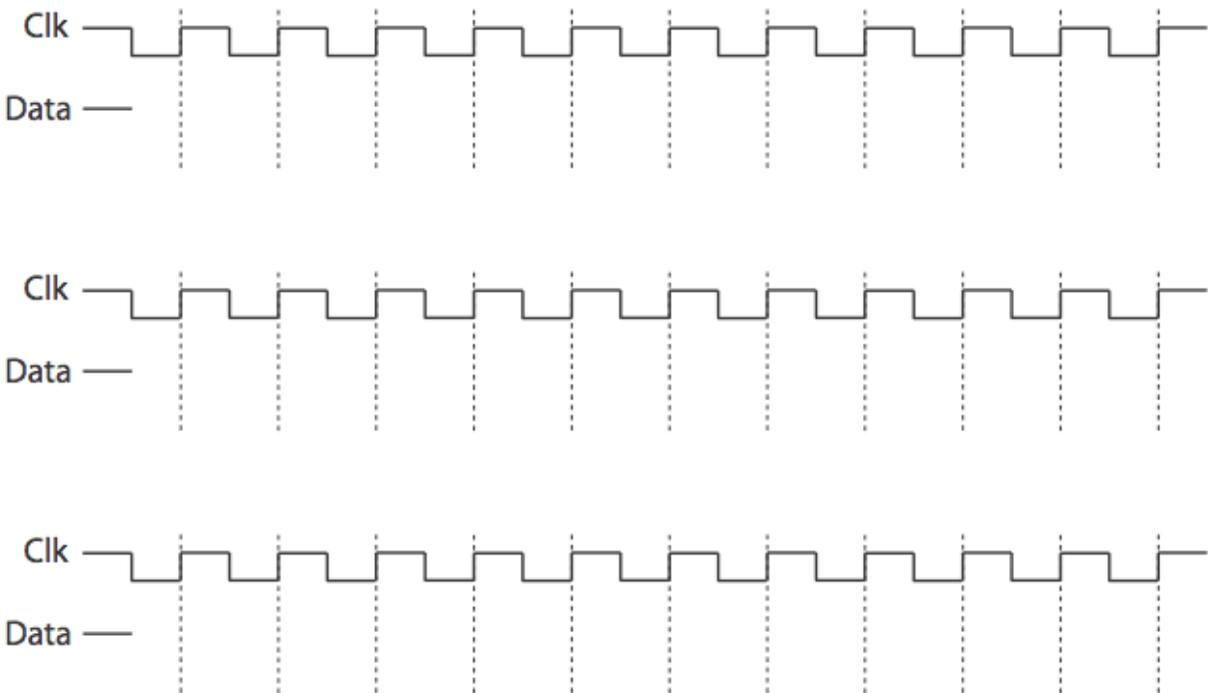


Figure 1: Complete Make and Break Code for Y Key

In Iad, we followed the procedure to implement something similar to this on a UP1 Programmable Logic Board. We programmed the UP1 Board to display the key code for whatever key is pressed on the PS/2 Keyboard, which was connected to the UP1 Board.

To start, we opened Altera Tools (Quartus II) and created a new document which we saved into a folder we had created for Lab 7. We created the new project and selected “Block Diagram/Schematic File” as the document type. We named it “Keyboard”. We changed the Target Device Family to “FLEX10K” and the Speed Grade to 4. We also selected “EPF10K20RC240-4 before clicking “Finish” on the Project Options.

By double-clicking in the schematic window, we added an Input symbol and an Output symbol, placed on opposite sides of the schematic window. We then added a LPM_Shiftreg. For the LPM_Shiftreg, we specified that it should have an output bus of eleven (11) bits and be a right-shifting register.

We located the file Seven_Seg_Driver.vhd on the S Drive and added it to our project. This is the driver for the seven segment display on the UP1 Board. The contents of this file can be found if you are within the Cedarville network and the S Drive is mounted. Navigate to S:\Dept\EG\Courses\EGCP_101DLD\Qurtus Files\Seven_Seg_Driver.vhd. While in the file Seven_Seg_Driver, we selected the File menu, then Create Symbol for

Current File from the Create/Update submenu. Going back to the Keyboard schematic file on our project, we added the Seven_Seg_Driver symbol to our schematic.

We copied and pasted the Input symbol and named one Keyboard_CLK and the other Keyboard_Data. We connected these to the clock input and the shiftin input of the LPM_Shiftreg symbol. We ran the output Data[10..0], which defined that Data would count down from 10 to 0. We copied and pasted the Seven_Seg_Driver so there were two of them. The first one was connected the input to Data[8..5]. The second one we connected the input to Data[4..1]. We copied and pasted the Output symbol and connected each of them to the output of the Seven_Seg_Driver symbols. We named the first one 7_Seg_Left[6..0] and the second one 7_Seg_Right[6..0].

Notice that Data[0] should always be zero (0) because it is the initial condition and starting bit. Data[9] is the Parity bit, so since the PS/2 Keyboard uses Odd Parity, it will be zero (0) if the number of bits in the make code is odd, and it will be one (1) if the number of bits in the make code is even. Data[10] will always be high (1), specifying that it is the Stop bit. Data bits eight (8) to one (1) are Keyboard Data we wish to display on our Seven Segment Display. We will connect Data bits eight (8) to five (5) to the Left Display, and Data bits four (4) to one (1) to the Right Display.

We saved the file and pressed the Compile button, enabling us to specify pin numbers and also ensuring we had not programmed any errors thus far.

Under the Assignment menu, we selected Device and made sure “Reserve all unused pins as inputs, tri-stated” was selected for unused pins. From the Assignment menu, we selected Pins and the Assignment Editor appeared. The pins were assigned to the following inputs and outputs as shown in Table 1.

To	Location
7_Seg_Left	
7_Seg_Left[0]	PIN_13
7_Seg_Left[1]	PIN_12
7_Seg_Left[2]	PIN_11
7_Seg_Left[3]	PIN_9
7_Seg_Left[4]	PIN_8
7_Seg_Left[5]	PIN_7
7_Seg_Left[6]	PIN_6
7_Seg_Right	

To	Location
7_Seg_Right[0]	PIN_24
7_Seg_Right[1]	PIN_23
7_Seg_Right[2]	PIN_21
7_Seg_Right[3]	PIN_20
7_Seg_Right[4]	PIN_19
7_Seg_Right[5]	PIN_18
7_Seg_Right[6]	PIN_17
Keyboard_CLK	PIN_30
Keyboard_Data	PIN_31

Table 1: Pin Assignments

Since the UP1 Board is low active (0), and the Seven_Seg_Driver is high active (1), we must change the port inversion to all ports. To do this, we right-clicked on the Seven_Seg_Driver and selected Properties. Under the Ports tab, we selected LED[6..0] and selected the “All” radio button for Inversion.

We recompiled the project and double checked all our our schematics and pin assignments. We then selected the “Programmer” button to program the UP1 Board with our schematic. We ensured the power was connected and switched on to the UP1 Board and that it was connected to the computer. We checked the “Program/Configure” box next on our Keyboard project, and then click “Start” to initiate the programming of the UP1 Board.

After we ensured the UP1 Board did what it was supposed to (displayed the key code of the pressed key on the keyboard), we reprogrammed it to show that key code and THEN count down, in hex, to zero.

To do this, we added two LPM_Counter symbols to the Keyboard schematic. We made the first one an eight (8) bit wide, down only counter that counted down. We selected VHDL Output, Cnt Enable and Asynchronous load. For the second counter, we selected VHDL Output and made it 23 bits wide. We left the second counter as an up counter.

We added an OR8 symbol and a NOT gate. We connected the NOT gate output to the aload of the down counter. The input of the NOT gate was connected to Keyboard_CLK. For the data[7..0] input for down counter we connected it to Data[9..2], and we connected to the clock input to Clk[22]. The output, q[7..0], is attached to Count[7..0].

The OR8 gate had eight (8) inputs coming into it, which was an array of Count[7] down to Count[0]. The output was attached to cnt_en on the downward counting counter.

The input to the up counting counter was connect to another Input symbol which we named 25MHZ. We tied the output to that symbol to Clk[22..0].

The final change that was made was that the inputs to the Seven_Seg_Driver symbols were changed. The Data[8..5] input was changed to Count[7..4], and Data[4..1] was changed to Count[3..0].

We compiled the project and then entered the Pin Assignment Editor again to assign our new pins. The only pin that needed assignment was the input for 25MHZ, which was assigned PIN_91.

Finally, the program was compiled and the UP1 Board was programmed in the same way it had been programmed before. The final schematic is shown below in Figure 2.

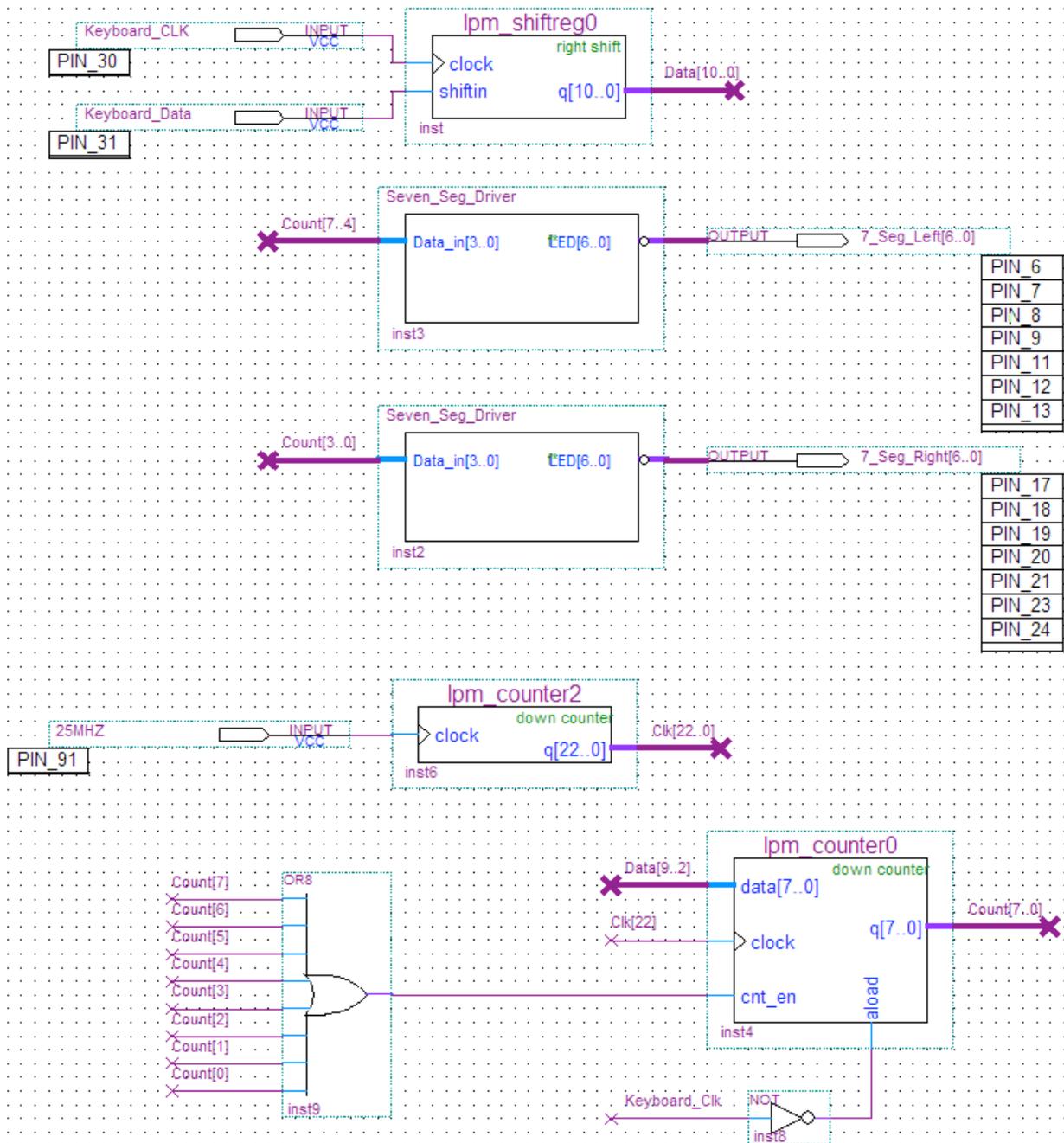


Figure 2: Final Schematic for Keyboard

For the remaining time in the lab, my partner and I discussed our final project, Lab 8, and what we might do for it. We decided to do a circuit similar to a Morse Code interpreter. Using the two buttons on the UP1 Board, the user would press them in such a combination, one representing short and one representing long, and after three combinations, the Seven Segment Display would show what letter was inputed.

We decided that as the user pressed a key, the Display would show how many button presses had occurred up to that point. Then, on the third button press, the Display would display the letter.

The left button will represent a short signal, and the right button will represent long. For combinations, they are shown below in Table 2.

Combination	Display
...	A
.._	B
._.	C
._	D
..	E
._.	F
..	7
..	8

Table 2: Morse Code Combinations

Conclusion and Suggestions:

This lab gave a wonderful overview of the Altera Tools and exposed the student to the ease of creating something graphically, on a computer, and then programming a chip using a powerful too like the Quartus Tools.

As far as the shift and counting registers, this was nothing new to us, but it was an easier way to expose us to the Altera Tools.

Questions:

1.) What is the frequency of the clk[22] pin of the large clock divider of the Final Keyboard Countdown Design?

The frequency for the clock at Count #22 is 3.0011, according to Table 2 in the lab handout.

2.) Describe how the aload (asynchronous load) and Cnt_enb (Count Enable) logic perform their functions of counting down the keyboard to zero.

Asynchronous load makes it load whenever the clock for the keyboard is low. It actually holds the incorrect value until the entire make code has been put into the shiftreg.

The count enable only allows the counter to count when it is closest to zero, otherwise it might roll over to FF or something.

3.) What does FPGA stand for?

Field-Programmable Gate Array