

Worcester Polytechnic Institute

Electrical and Computer Engineering Department

EE3801 – Lab4 (Keyboard Interface)

Copyright: April, 2005

Written by Ahmad Hatami
Modified by Samant Kakarla

Modified by: S. Jarvis (11/2006)

Please have all individual components created in this lab available to all parties working together to complete the lab. Make sure everyone also has the components available individually from previous labs. It is a good idea to backup your data on another media apart from the network space you have available.

Objective

In this lab you will learn how to design a system that provides an interface between Spartan-3 starter kit board and a PS2 style keyboard. Your system will display the scan code sequence, generated by the keyboard when each key is pressed.

Big Picture

Each PS2 style keyboard (Figure 1) is a large matrix of keys, all scanned by an embedded chip (keyboard encoder) to generate a unique scan code for each key pressed or released. The keyboard encoder debounces the inputs and provides buffering for a fast writer. A host computer contains a keyboard controller that provides the interface for a two way communication between the host and the keyboard encoder. In order to reduce the number of connection lines between a keyboard and a processing host, a two wire PS2 serial bus communication is used. All communication between the host and the keyboard uses an IBM protocol. In this experiment we use the keyboard as an input device with its basic default features.



Figure 1

Make/Break Codes

Whenever a key is pressed, a sequence of bits is sent serially through the data line from the keyboard to the host. This sequence of bits is called a *Make Code*. Whenever a key is released a separate set of bit sequence is sent by the keyboard to the host which is called a *Break Code*. For most keys on a standard PS2 keyboard, the last byte in each make and break code are identical. Each make/break code uniquely identifies the associated key. It should be noted that the make/break code does not represent the ASCII character on the keyboard, and it is the responsibility of the host to map the corresponding make/break code to an equivalent ASCII character. If a key is pressed and held, the keyboards repeatedly sends the make code every 100 millisecond or so. When a key is released the keyboard sends “F0” key-up code, followed by the break code of the released key. Some keys; called extended keys; send an “E0” ahead of the make code and sometimes they send more than one make code. When an extended key is released an “E0 F0” is followed

KEY	MAKE	BREAK	KEY	MAKE	BREAK	KEY	MAKE	BREAK
A	1C	F0,1C	9	46	F0,46	[54	F0,54
B	32	F0,32	`	0E	F0,0E	INSERT	E0,70	E0,F0,70
C	21	F0,21	-	4E	F0,4E	HOME	E0,6C	E0,F0,6C
D	23	F0,23	=	55	F0,55	PG UP	E0,7D	E0,F0,7D
E	24	F0,24	\	5D	F0,5D	DELETE	E0,71	E0,F0,71
F	2B	F0,2B	BKSP	66	F0,66	END	E0,69	E0,F0,69
G	34	F0,34	SPACE	29	F0,29	PG DN	E0,7A	E0,F0,7A
H	33	F0,33	TAB	0D	F0,0D	↑	E0,75	E0,F0,75
I	43	F0,43	CAPS	58	F0,58	←	E0,6B	E0,F0,6B
J	3B	F0,3B	L SHFT	12	F0,12	↓	E0,72	E0,F0,72
K	42	F0,42	L CTRL	14	F0,14	→	E0,74	E0,F0,74
L	4B	F0,4B	L GUI	E0,1F	E0,F0,1F	NUM	77	F0,77
M	3A	F0,3A	L ALT	11	F0,11	KP /	E0,4A	E0,F0,4A
N	31	F0,31	R SHFT	59	F0,59	KP *	7C	F0,7C
O	44	F0,44	R CTRL	E0,14	E0,F0,14	KP -	7B	F0,7B
P	4D	F0,4D	R GUI	E0,27	E0,F0,27	KP +	79	F0,79
Q	15	F0,15	R ALT	E0,11	E0,F0,11	KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D	APPS	E0,2F	E0,F0,2F	KP .	71	F0,71
S	1B	F0,1B	ENTER	5A	F0,5A	KP 0	70	F0,70
T	2C	F0,2C	ESC	76	F0,76	KP 1	69	F0,69
U	3C	F0,3C	F1	05	F0,05	KP 2	72	F0,72
V	2A	F0,2A	F2	06	F0,06	KP 3	7A	F0,7A
W	1D	F0,1D	F3	04	F0,04	KP 4	6B	F0,6B
X	22	F0,22	F4	0C	F0,0C	KP 5	73	F0,73
Y	35	F0,35	F5	03	F0,03	KP 6	74	F0,74
Z	1A	F0,1A	F6	0B	F0,0B	KP 7	6C	F0,6C
0	45	F0,45	F7	83	F0,83	KP 8	75	F0,75
1	16	F0,16	F8	0A	F0,0A	KP 9	7D	F0,7D
2	1E	F0,1E	F9	01	F0,01]	5B	F0,5B
3	26	F0,26	F10	09	F0,09	;	4C	F0,4C
4	25	F0,25	F11	78	F0,78	'	52	F0,52
5	2E	F0,2E	F12	07	F0,07	,	41	F0,41
6	36	F0,36	PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12	.	49	F0,49
7	3D	F0,3D	SCROLL	7E	F0,7E	/	4A	F0,4A
8	3E	F0,3E	PAUSE	E1,14,77, E1,F0,14, F0,77	-NONE-			

Table 1

by the key's break code. Although the host can send data to the keyboard but we will not use that function in this lab. Table 1 shows the make/break code for a generic PS2 style keyboard.

Keyboard to Host Communication Protocol

The keyboard sends data to the host only when both the data and clock lines are high. Because the host is the “bus master”, the keyboard checks whether the host is sending data before driving the bus. The clock line can be used as a “clear to send” signal. If the host pulls the clock line low, the keyboard must not send any data until the clock is released. The keyboard sends data to the host in 11-bit words that contain a ‘0’ start bit, followed by eight bits of make/break code (LSB first), followed by an odd parity bit and terminated with a ‘1’ stop bit. When the keyboard sends data, it generates 11 clock transitions at around 20 to 30 kHz, and data is valid on the falling edge of the clock as shown in Figure 2. After each transmission the data and clock lines are returned to the inactive state until the next byte is sent. The parity bit is set if there is an even number of 1's in the data bits and reset (0) if there is an odd number of 1's in the data bits. The number of 1's in the data bits plus the parity bit always add up to an odd number (odd parity.) This is used for error detection. The keyboard/mouse must check this bit and if incorrect it should respond as if it had received an invalid command. Data sent from the keyboard to the host is read on the *falling* edge of the clock signal; data sent from the host to the keyboard is read on the *rising* edge of the clock.

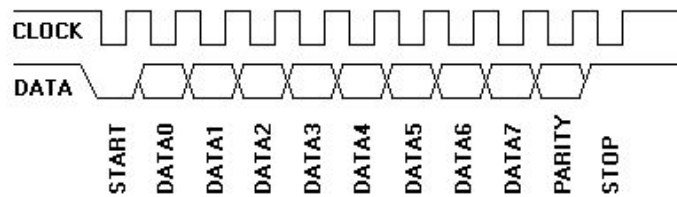


Figure 2

As an example Figure 3 shows the complete make/break bit sequence for transmitting the character ‘a’ (1C F0 1C) by a keyboard to a host. This bit sequence is depicted between the markers ‘O’ and ‘X’.

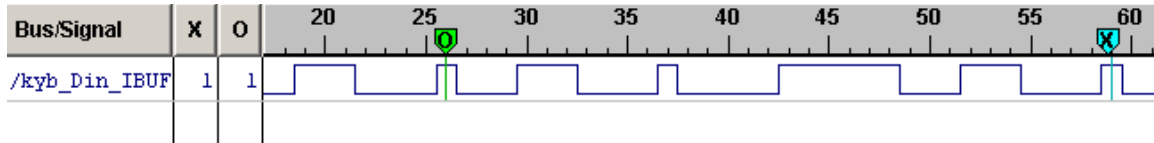


Figure 3

The Spartan-3 starter kit board uses a PS/2 mouse/keyboard port with the standard 6-pin mini-DIN for this interface as depicted in Figure 4. Only pins 1 and five of the connector

are directly attached to the FPGA IO ports. Table 2 shows the pin assignments in a PS2 style connector.

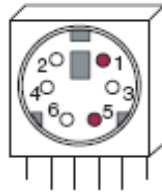


Figure 4

PS/2 DIN Pin	Signal	FPGA Pin
1	DATA (PS2D)	M15
2	Reserved	—
3	GND	GND
4	Voltage Supply	—
5	CLK (PS2C)	M16
6	Reserved	—

Table 2

Project

The overall design of your system is shown in Figure 5. Here we describe all the major components and signals in this system.

Keyboard Driver

This component provides an interface between a PS2 keyboard and Spartan3 board. It receives each scan code along with a clock signal transmitted by the keyboard. Upon receiving a character it activates a control signal (*Data_Ready*) which notifies the system of availability of new data, delivers the newly received character to the display driver, computes the parity of the received character and in case of a parity error it activates the parity signal.

Process Data

This unit resembles a data processing unit by displaying the received character on a seven segment display unit for three seconds. This unit resets the driver after three seconds and informs the keyboard that the system is ready to receive the next character.

Seven Segment Display Unit

This unit is the same display unit that you have used in previous labs.

Signals

Din: This signal is used by the keyboard to transmit information to the board as described in the previous section. When there is no data to transmit the keyboard keeps this signal at high.

Kyb_Clk: This is the clock signal between the board and the keyboard. Note that this is a bidirectional signal controlled by both keyboard and the board. Board is the master in driving this signal. If the board applies a low on this signal keyboard stops any further transmission. More detail is provided in the previous section.

Reset_Driver: Activating this signal resets the keyboard driver. In order to prevent any data loss, during the reset period the keyboard must not transmit any information.

Parity_Error: By default there is an encapsulated odd parity on any transmitted data. If the computed parity on the received character is even this signal is activated indicating a transmission error.

Data_Ready: Upon reception of a character this signal is activated by the controller. This signal initiates the processing of the received character by the processing unit (Process Data).

Clk: Internal Clock of the system. You can use the 10 KHz clock similar to our previous labs.

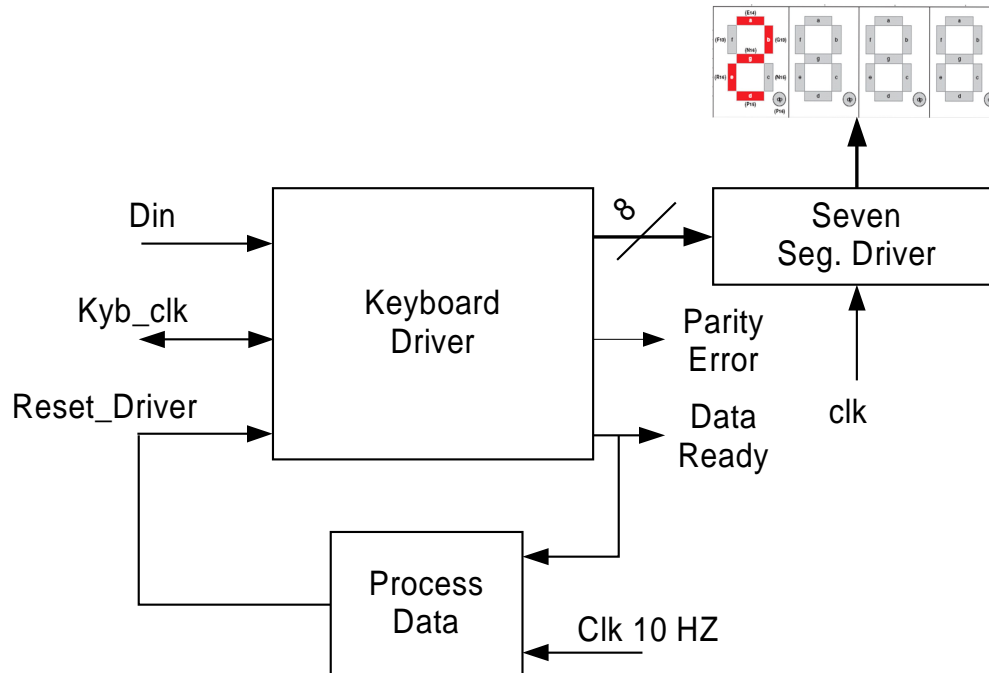


Figure 5

Design

In the following sections you are going to design the keyboard driver component by component.

Keyboard Driver

We divide the keyboard driver unit into three different subunits as suggested by Figure 6.

Keyboard Clock Manager

This component is responsible for generating the clock signal for other components of the driver based on the clock signal provided by the keyboard. The keyboard provides a clock signal when it starts to transmit data to the FPGA board. Note that the clock signal shared between the board and the keyboard is a bidirectional connection in which the Spartan3 board is the master. The complete design of this unit is provided in Figure 7. Create the multiplexer in VHDL for this part of the design and incorporate it into your schematic.

1. Create a new schematic for the clock manager unit in your project.
2. Save your design.
3. In your report you must explain how this unit works, and provide some simulation waveforms to verify its functionality.
4. After testing this design create a symbol and save that in your library.

Note that the output of this unit (*Rx_Clk*) is a clock signal for all the remaining components of the driver.

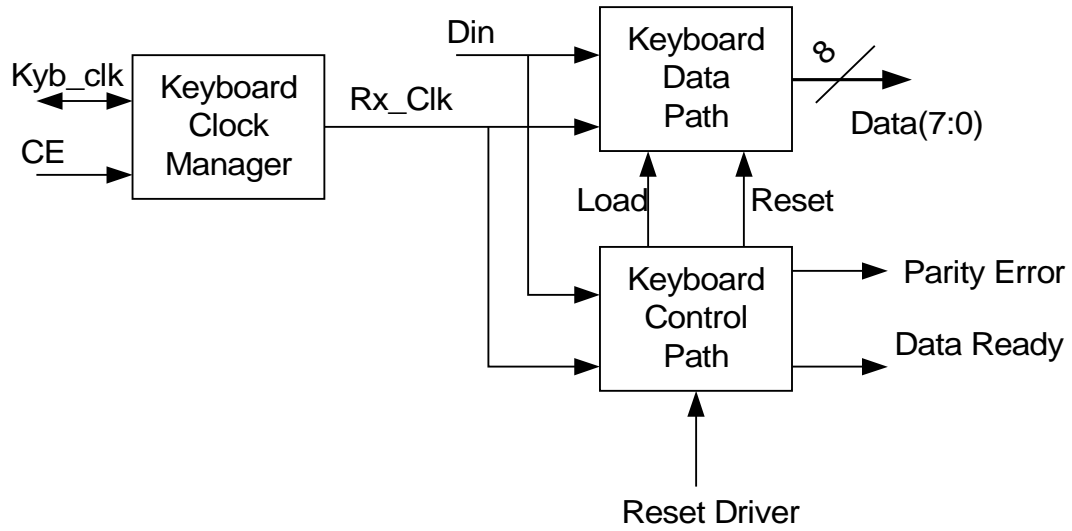


Figure 6

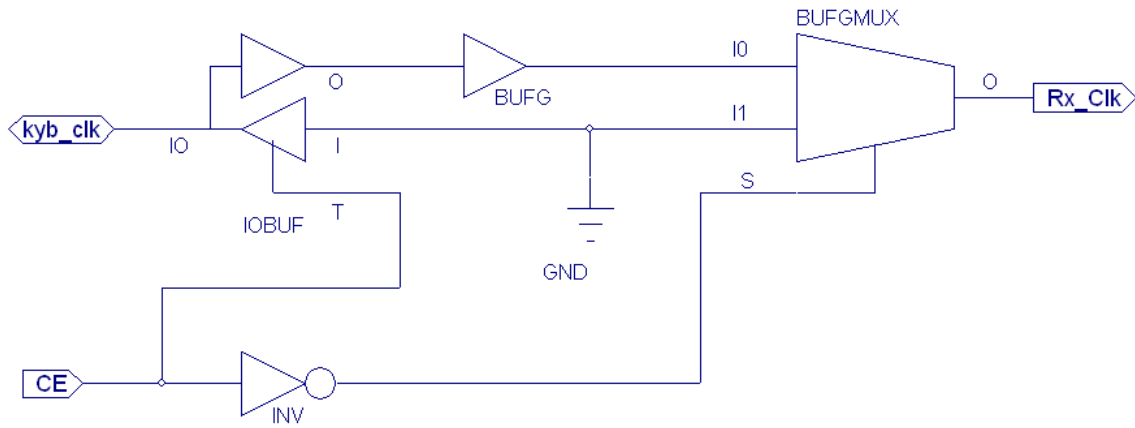


Figure 7

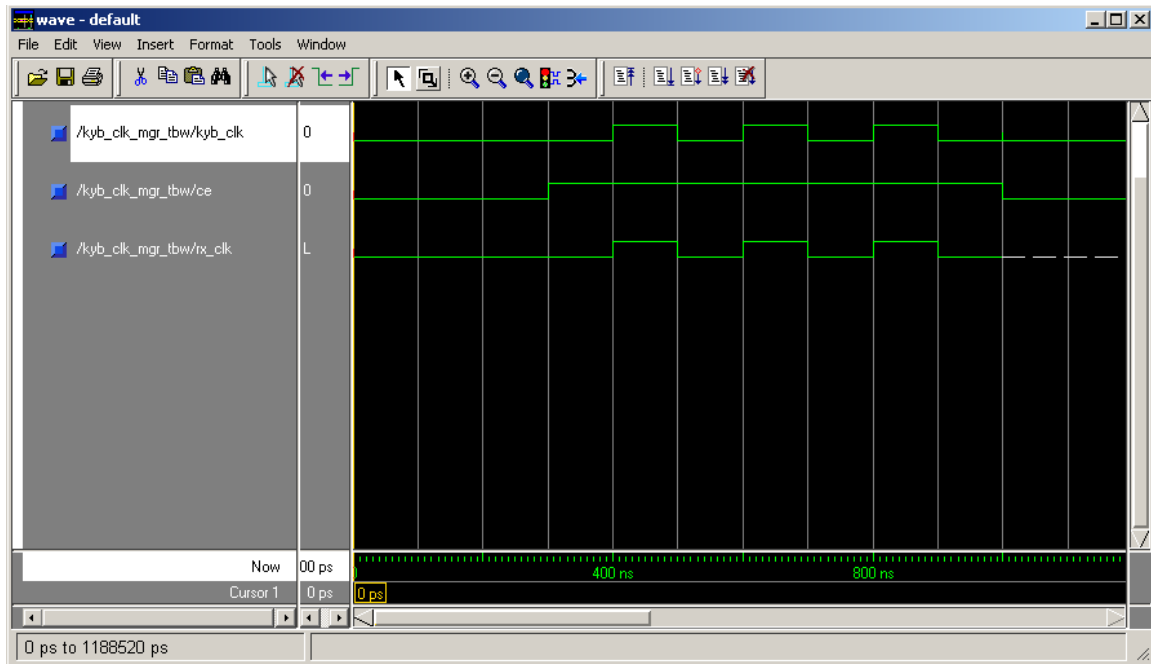


Figure 8

Communicating with Keyboard

In this section you will connect a keyboard to a simple system and observe the signals generated by a keyboard.

5. Create a new schematic similar to Figure 9 and add it to your project.
6. Check your schematic and save this file.
7. Create a user constraint file for the newly created file.(Table 3)
8. Generate a programming file and download your design into the board.
9. **Show your design to the TA**, get a keyboard and connect it to the PS2 port available on the board.
10. Put SW0 in high position and press a couple of keys on the keyboard and observe your Spartan 3 board.
11. Explain your observations and include them in your report.
12. Now put SW0 in low position.
13. Start typing some characters on the keyboard.
14. Explain your observations.
15. Return SW0 to its original high position. Explain your observations. **Show TA.**

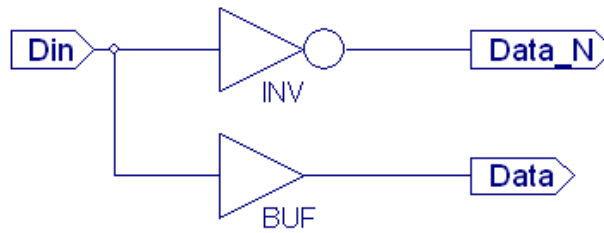


Figure 9

<pre> NET "Din" LOC = "M15"; NET "kyb_clk" LOC = "M16"; NET "CE" LOC = "F12"; NET "Data_N" LOC = "K12"; NET "Data" LOC = "P14"; NET "Rx_Clk" LOC = "P11"; </pre>

Table 3


The circuit you just used demonstrated that signals were generated by the keyboard but we were not able to capture the codes generated by the keyboard. In the following section you will observe the keyboard signals in more detail using either the ChipScope Pro Analyzer, which is an on chip debug and verification tool provided by Xilinx, or the Agilent MSO6012A Logic Analyzers located in AK-113. (Note that ChipScope pro is not available on the free version of the Webpack so you need to complete this section in the lab whether you use Chip Scope or the Agilent analyzer.)

The instructions below are for using ChipScope. As you might imagine, having experience with the Xilinx ISE, ChipScope can be a bit painful (i.e. buggy) to use. The Agilent logic analyzers are a bit more straight forward. See the link to the Agilent User Guide on the Class web page. One of the challenges of using the Agilent system will be finding the places to attach the leads. The pin locations are given by the constraints file

(Table 3). Check the Spartan 3 board schematic (see Users Guide) to find probe-able points.

You DO NOT need to do these steps if using the Agilent logic analyzer. If you use the Aligent scopes, save a screen dump of the display using the usb port.

16. Select the file you just created as your top level design in project navigator.
17. Add a new source file to your project as suggested by Figure 10.
18. Follow the remaining dialogue boxes as shown in Figure 11 and Figure 12.
19. Note that a new file is added to your project with a *.cdc extension.
20. Double Click on the newly generated file to open the Chipscope pro core inserter (Figure 13). You can insert different cores into your design to probe signals and their associated timings by this tool.
21. Make sure that Spartan3 is selected as the device family and click on next.
22. Click on “New ILA Unit” to create add a logic analyzer core into your design. (Figure 14)
23. Note that a new ILA core is added in this dialogue box. (U0:ILA in Figure 15)
24. Click on next.
25. In “Trigger Parameter” tab enter the parameters as shown in Figure 16 and click on next.
26. In “Capture Parameters” tab enter the parameters as shown in Figure 17 and click next.
27. On the “Net Connection” tab click on “Modify Connections”. (Figure 18)
28. Note that in this new dialogue box there is a list of all the available net names in your design. In this dialogue box you will assign a clock signal and all the signals that you want to probe to different channels. Assign “Rx_Clk_OBUF” as your clock signal and click on “Make Connections”.
29. Select the “Trigger Data Signals” tab, and complete the selections as shown in Figure 19 and click on OK.
30. Click on “Return to Project Navigator” and click on yes to save your changes.
31. Make sure that your top design is selected in project navigator and click on “Configure Device IMPACT”(Figure 22). This generates a new bit file accompanied with the ILA core that you just added and load the new bit file into the board.
32. Download the board with the new bit file like you have done before.
33. Close the IMPACT window. The board should behave as before.
34. Note that in your project navigator a new icon is added at the bottom of the process window as shown in Figure 22. Double click on “Analyze Design Using Chipscope”. This launches the ChipScope pro analyzer.
35. Select “JTAG Chain”, Xilinx Parallel Cable from the menu bar and click on OK (Figure 23).
36. It should detect the two available devices in your board (Figure 24). Click on OK.
37. Your screen should be similar to Figure 25.

38. Select File, Import file, and click on “Select File” and select the *.cdc file you created in previous sections as shown in Figure 26. Click on “Open” and “OK”. This will assign the net names you have defined to the physical nets on the board (Figure 27).
 39. On the trigger setup window you can define different trigger conditions for initiating a data capture. Enlarge that window and enter the values as shown in Figure 28. With this setting data capture starts when SW0 on the board is in high position. Note that since the actual capturing starts on the rising edges of the clock (*Rx_Clk*) which is provided by the keyboard when you press a key on the keyboard.
 40. Click on  icon.
 41. Make sure the SW0 is set to the high position.
 42. Press the character “A” on the keyboard repeatedly until the data capture buffer becomes full.
 43. Maximize the waveform window you should see a waveform similar to (Figure 29).
 44. Zoom in the signal waveform. Note that during the capture you may lose some of the initial samples. You should be able to see signals similar to Figure 30.
 45. **Show your results to the TA.** Compare the signals you observe with your expectations based on Table 1 and justify that.
46. Why the *Rx_Clk* signal is always constant although you have assigned that as a clock signal?
 47. Close the analyzer window.

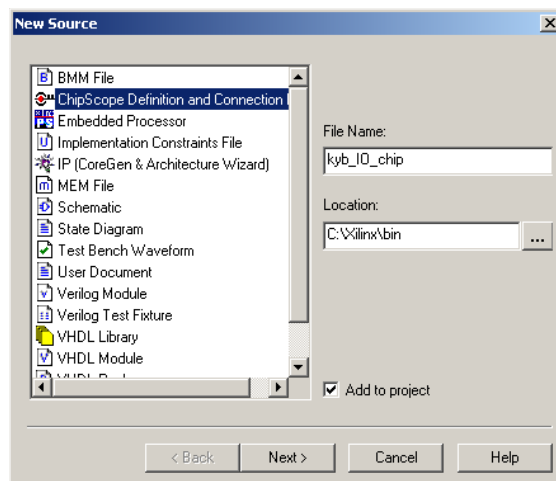


Figure 10

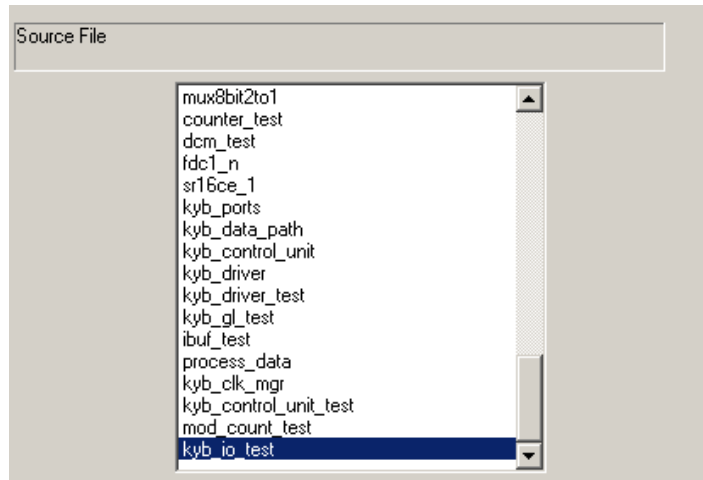


Figure 11

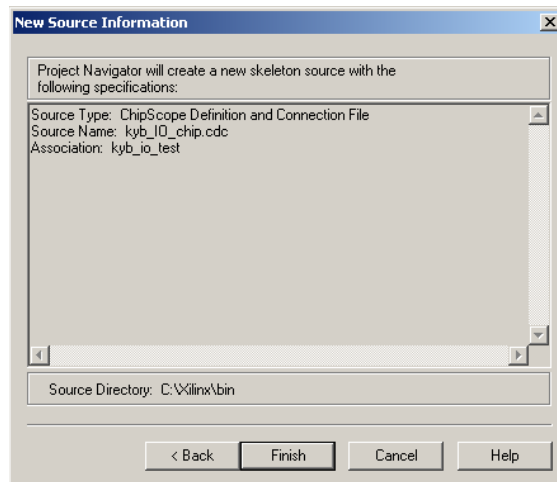


Figure 12

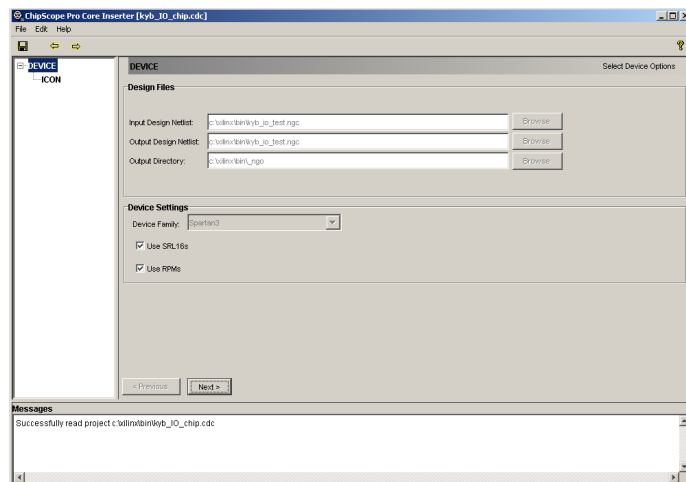


Figure 13

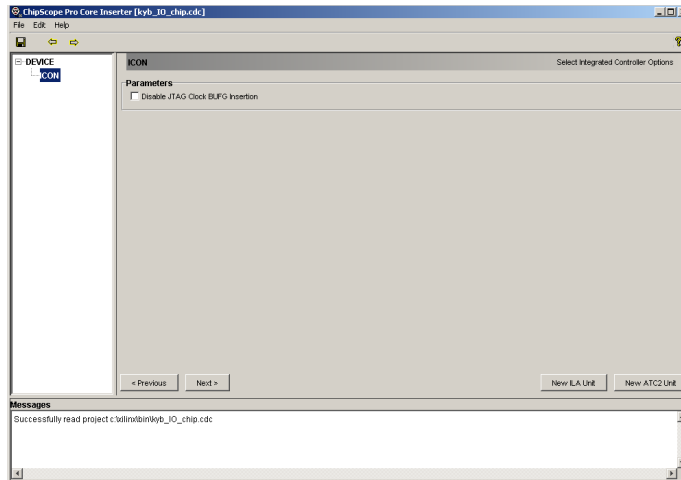


Figure 14

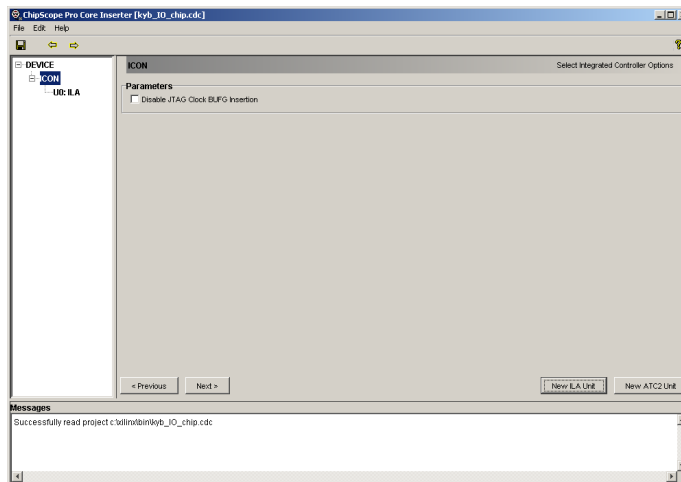


Figure 15

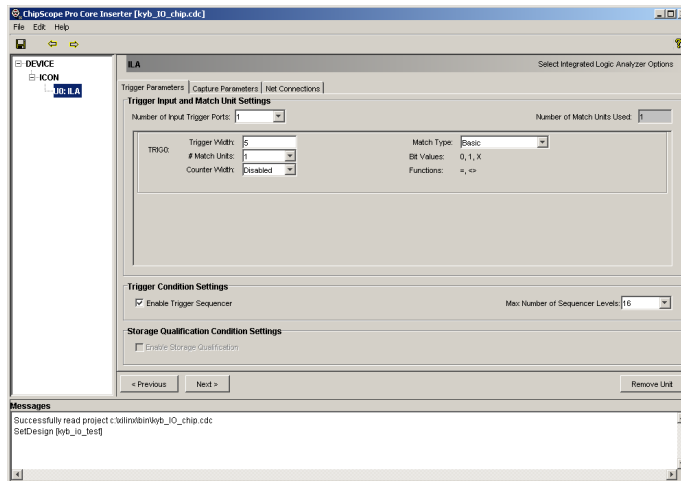


Figure 16

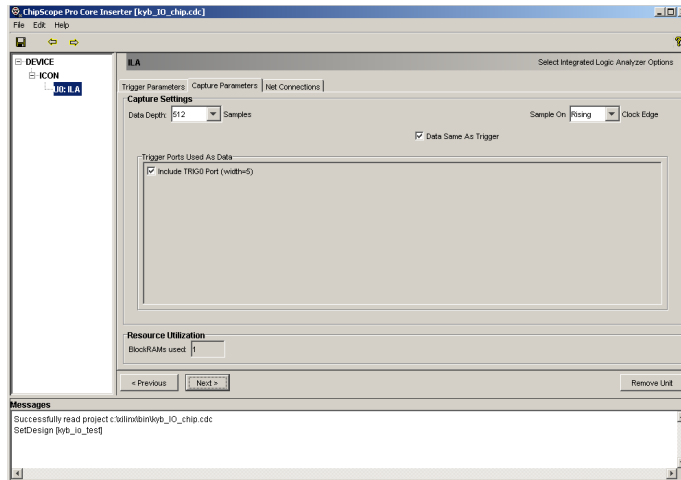


Figure 17

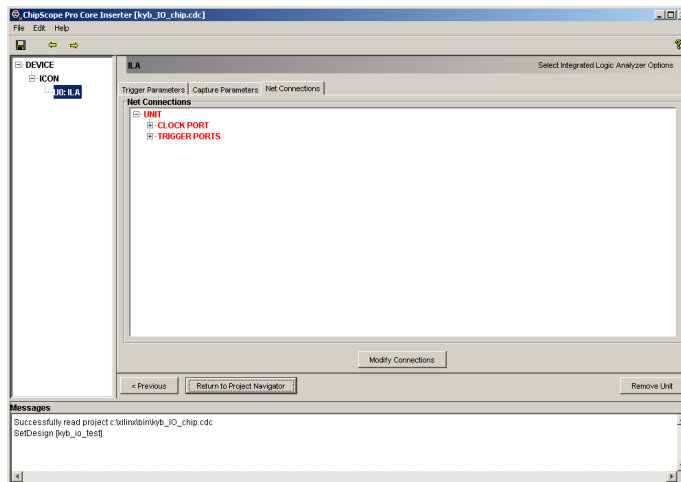


Figure 18

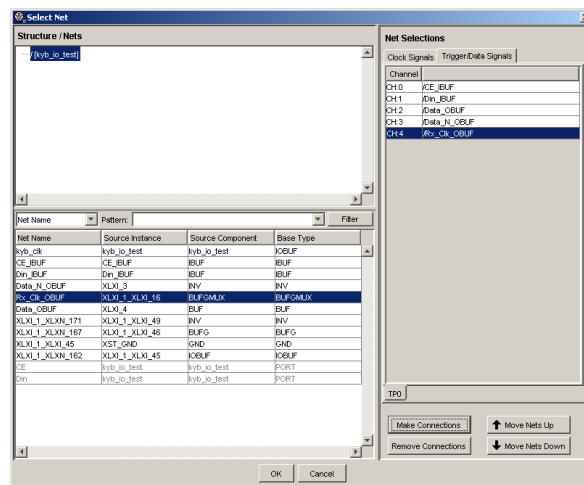


Figure 19

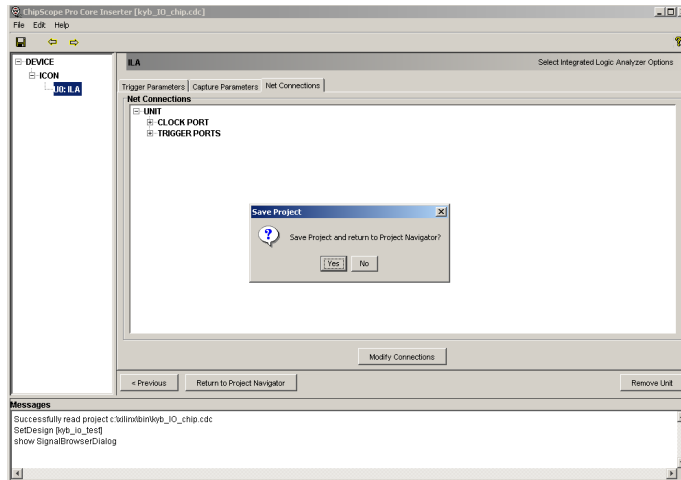


Figure 20

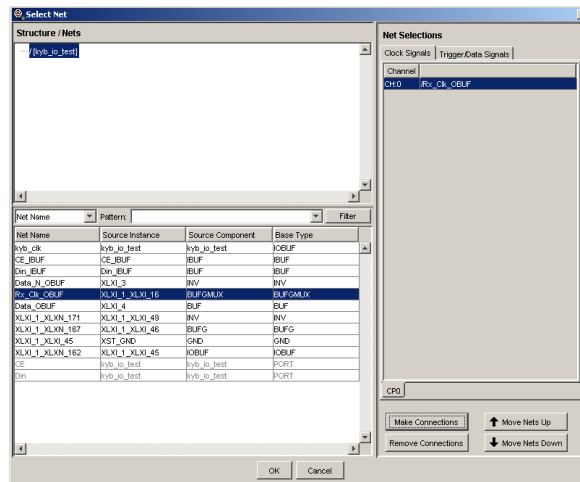


Figure 21

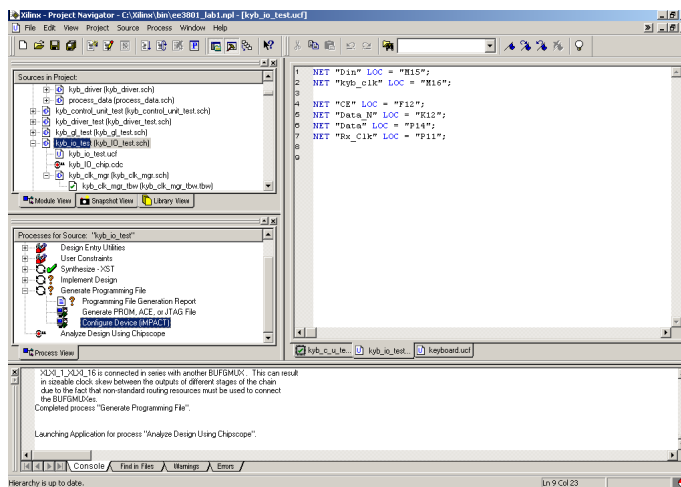


Figure 22

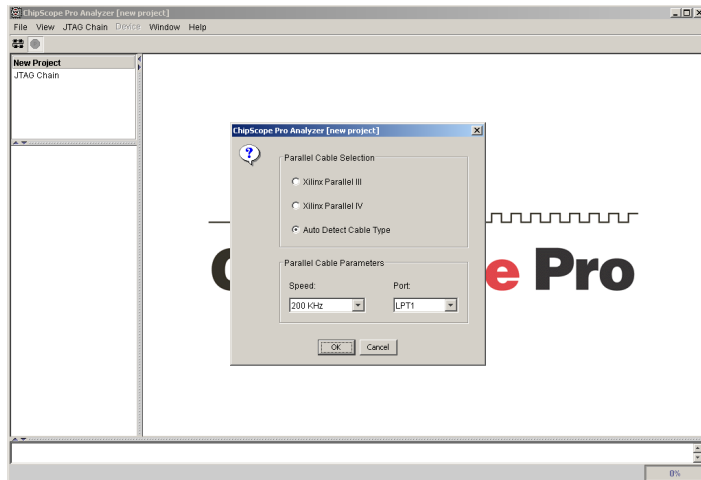


Figure 23

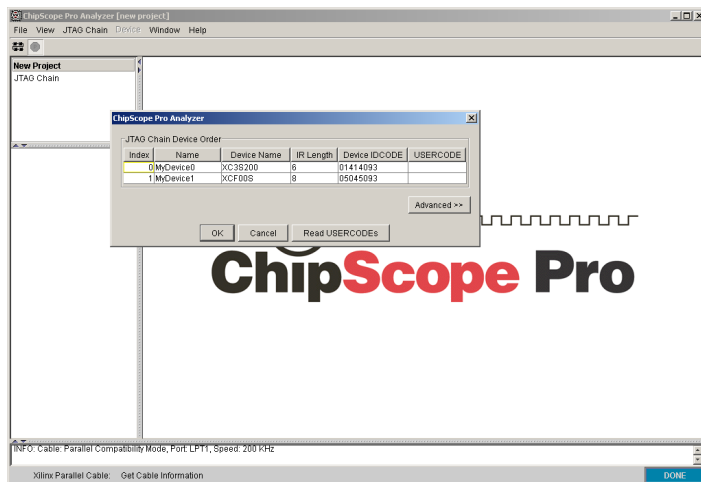


Figure 24

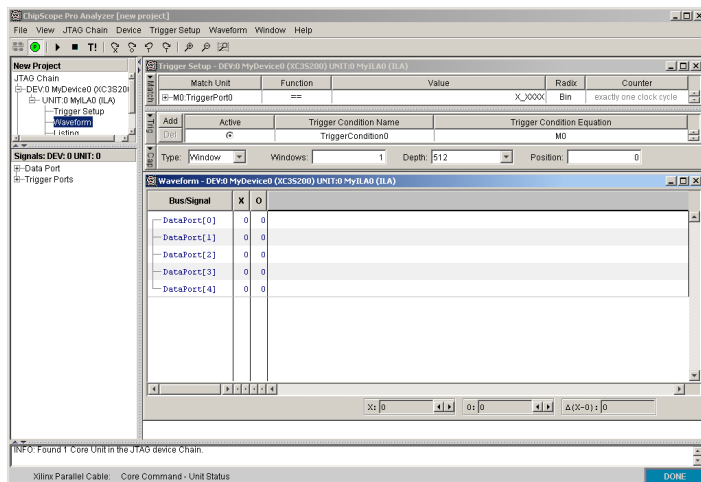


Figure 25

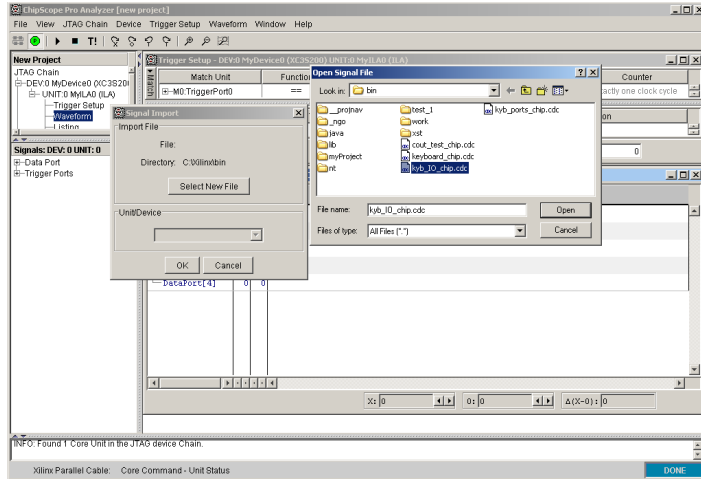


Figure 26

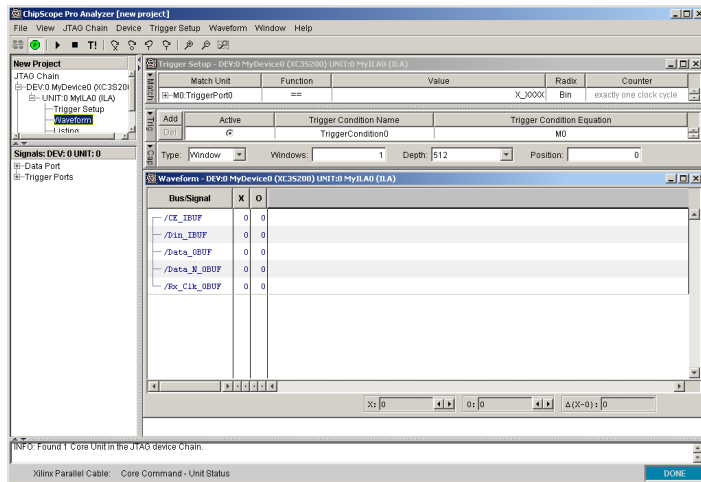


Figure 27

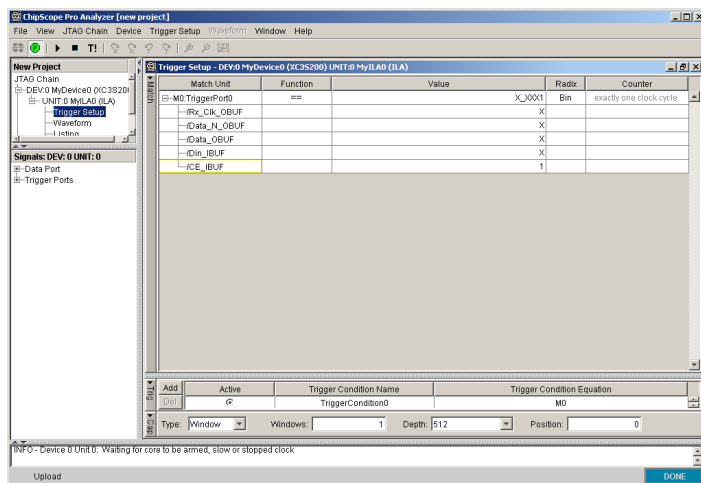


Figure 28

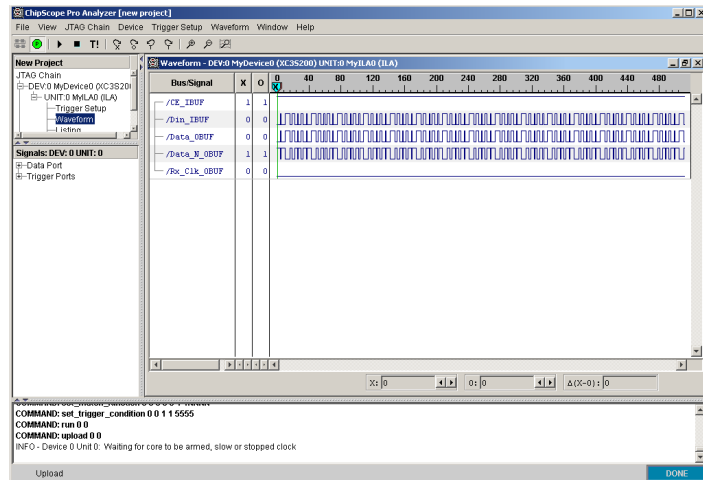


Figure 29

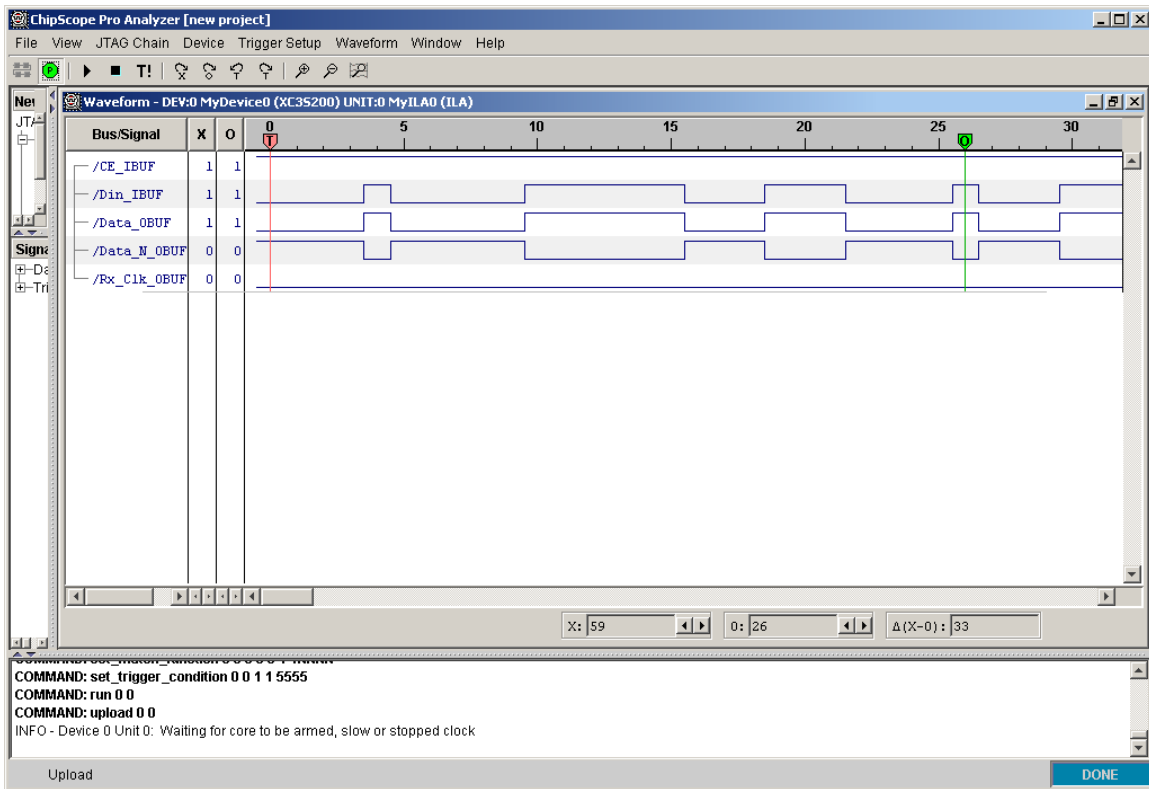


Figure 30

Keyboard Control Path

This unit controls the operation of data path based on its internal state machine, generates a control flag when a new character arrives or a transmission error occurs. We have divided this unit into two smaller units, kyb_control unit and parity unit as depicted in Figure 31. We have also included a test bench wave form for the control path in Figure

35 that shows the timing relationship between these units. This timing diagram simulates the transmission of two instances of character 'A' (0 0011 1000 0 1) with no transmission error and with an error (0 0011 1010 0 1) respectively. Here we need to consider the following points:

- A. In practice the clock signal is provided by the keyboard and as soon as the stop bit arrives the clock signal will be deactivated by the keyboard.
- B. The keyboard changes the data line on the rising edge of the clock and the board must read the data on the falling edge of the clock. It is essential to follow the clock edges as shown in the picture.

In the next section you will start your design by first designing the parity unit and keyboard control unit. Make sure that each part works correctly before connecting it to other components in your system.

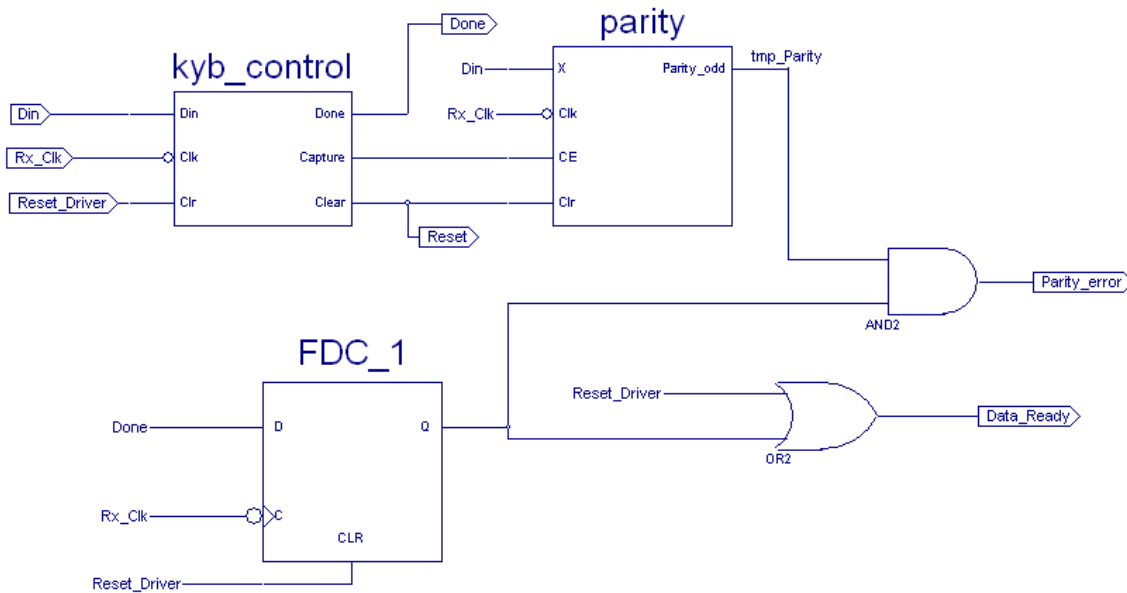


Figure 31

Parity Unit: This unit receives serial information through its Din port and generates an odd parity signal based on that. If the number of 1's in the received sequence of bits is an odd number the output will be a zero, and if the number of 1's received is an even number it generates a one. Here we explain the IO interface of this component as shown in Figure 32.

Din: The serial input bit sequence.

Rx_Clk: The clock signal provided for this unit, note that for this application this block must be a falling edge triggered device.

CE: Clock enable when one this unit observes the *Din* signal otherwise it ignores any changes on the *Din* signal.

CLR: Asynchronous clear for this unit.

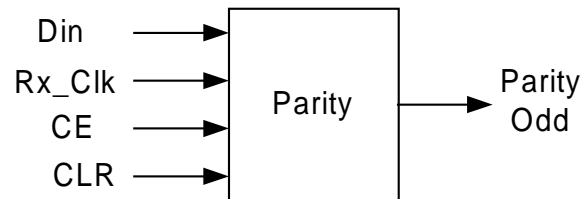


Figure 32

48. Design a sequential system for this component.
49. Create a new schematic for your design and add it to your project.
50. Enter your design and save the file.
51. Create a test bench waveform and make sure that your circuit works properly.
Note that your design should be falling edge triggered.
52. Create a symbol for your design and add it to your symbol library.

Kyb_control Unit: This unit is the main state machine for the keyboard driver (Figure 33). We have provided the state machine for this unit in Figure 34.

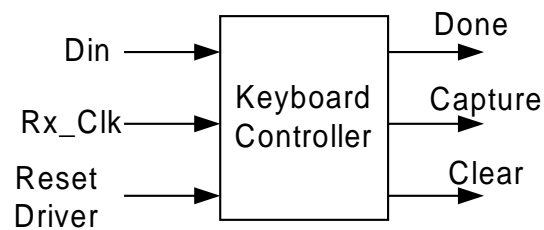


Figure 33

Din: The serial data bits received from the keyboard data port.

Rx_Clk: The clock signal provided for this unit; note that your design must be a falling edge triggered system.

Reset_Driver: Asynchronous reset for this block. Activating this signal resets the controller and initializes the driver for receiving new data.

Done: When the controller receives the last bit of a character (Parity bit), it activates this signal for one clock cycle.

Capture: This signal is activated after the controller detects the start bit and remains active until the controller captures the parity bit of the received sequence. This signal must be falling edge triggered.

Clear: In order to receive a new character the controller needs to clear the contents of the data registers. The clear signal conveys that command to the data path.

53. Design a sequential system for the state transition diagram of Figure 34.
54. Create a new schematic file and enter your design for this system and save your design. Note that your system must use negative edge triggered flip flops.
55. Create a test bench waveform and verify your design.
56. Create a symbol for this unit.
57. Why do we need to use the additional flip flop, the two input 'OR' and the two input 'AND' gate in the control path?

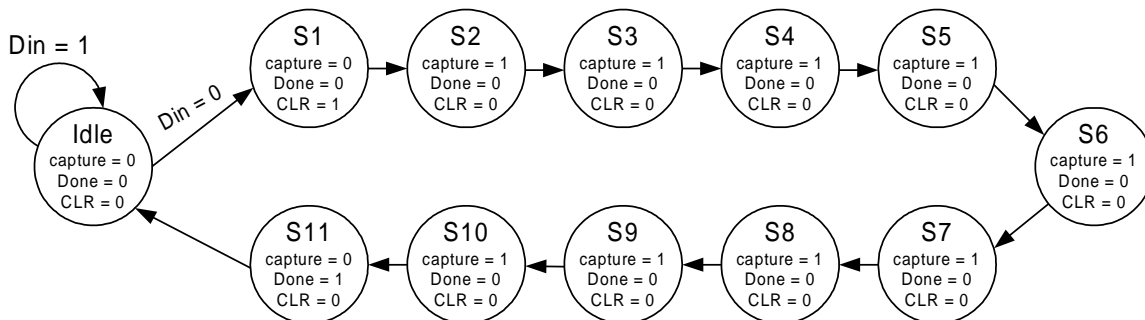


Figure 34

58. Create a new schematic file similar to Figure 31 in your project and save your file.
59. Create a test bench waveform similar to Figure 35 and make sure that your design generates valid results.
60. Create a symbol for the control unit and save it.

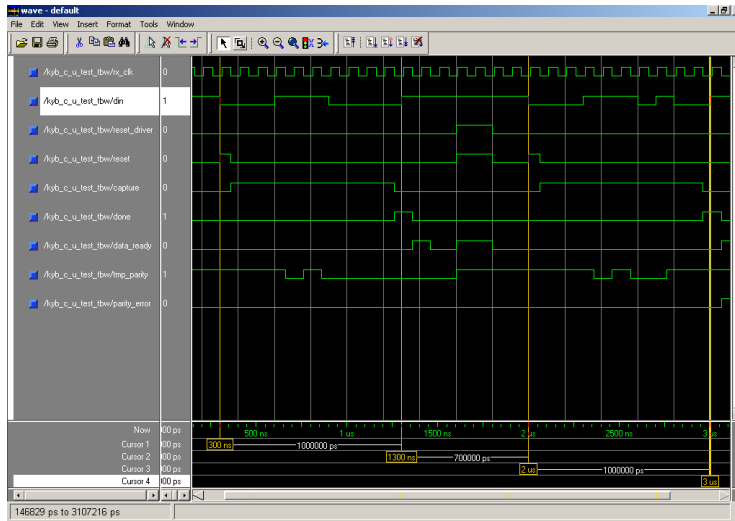


Figure 35

Keyboard Data Path

Figure 36 shows the interface of the data path unit. This unit consists of two internal components: a 16-bit negative edge triggered shift register (M) and an 8-bit positive edge triggered register (R). When the keyboard starts transmitting a bit sequence, the control path activates the shift register M to capture the incoming sequence (Figure 37). Note that each transmission contains eleven bits (start bit + 8-bits of data + parity + stop bit). The shift register captures the whole sequences and at the end of the transmission passes the 8-bits of data to register R. The clock signal is provided by the keyboard, thus the timing is extremely important in this design. The content of the shift register must be loaded into register R on the rising clock edge of the clock (stop bit) as shown in Figure 35. Note that *Done* signal is activated at the falling edge of the clock followed by a rising edge of the clock that must be used to load the data into R. Here we describe the IO interface of the data path unit in detail.

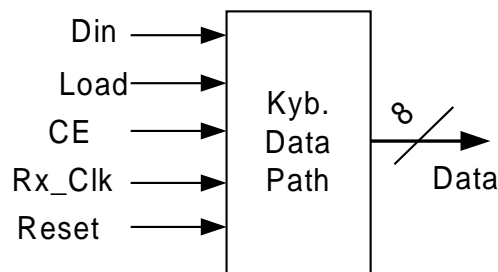


Figure 36

Din: The serial data bits received from the keyboard data port.

Load: Activating this port loads the content of the shift register into the device register. This signal will be activated by control path as soon as a character arrives. This is connected to the *Done* port of the control path.

CE: Clock enable input for the data path unit which controls the clock enable of the shift register M. The clock enable of the register R is controlled by the control path as shown in Figure 37.

Rx_Clk: The clock signal provided for this unit. Note that M is triggered on the falling edge of this clock while R is triggered on the rising edge of this clock.

Reset: Asynchronous reset for this block. Note that this should be driven by the control path.

Data: An 8-bit output transmitted by the keyboard.

61. You need to create a 16 bit negative edge triggered shift register for this design.
62. Create a new schematic in your project.
63. Use the internal architecture of (*SR16CE*); provided by Xilinx; as a base line and replace all the flip flops with negative edge triggered flip flops.
64. Check your schematic and save your design.
65. Create a test bench and make sure that your shift register's functions correctly.
66. Create a symbol for your shift register.
67. You can use an *SR8CLE* which is an 8-bit rising edge triggered shift register as register R in your design. Make sure to ground its SLI input.
68. Use Figure 36 and Figure 37 as a guideline to design the control path for this system. Note that the receiver receives the bits in the reverse order.

Figure 38 shows a sample timing diagram when a character 'A' is received by the driver for all the signals in the driver.

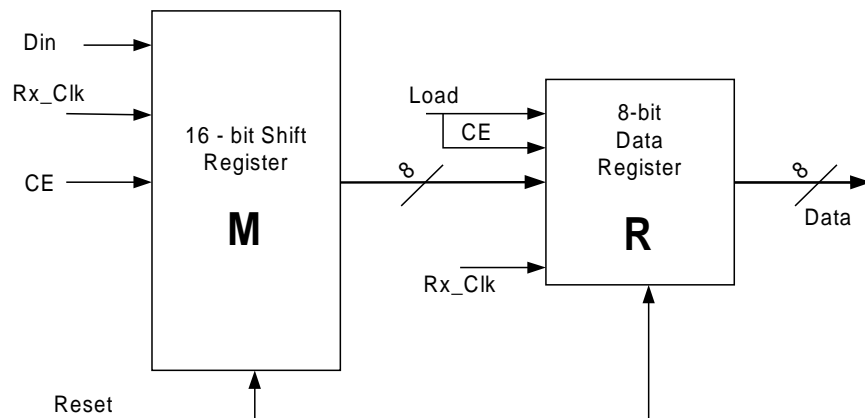


Figure 37

69. Make sure that there is the correct communication between data path and control path in your design.
70. Create a symbol for the data path you just designed.
71. Create a new schematic and based on Figure 6 complete the design of your keyboard driver.
72. Check your schematic, verify that, and save it.
73. Create a symbol for your design.

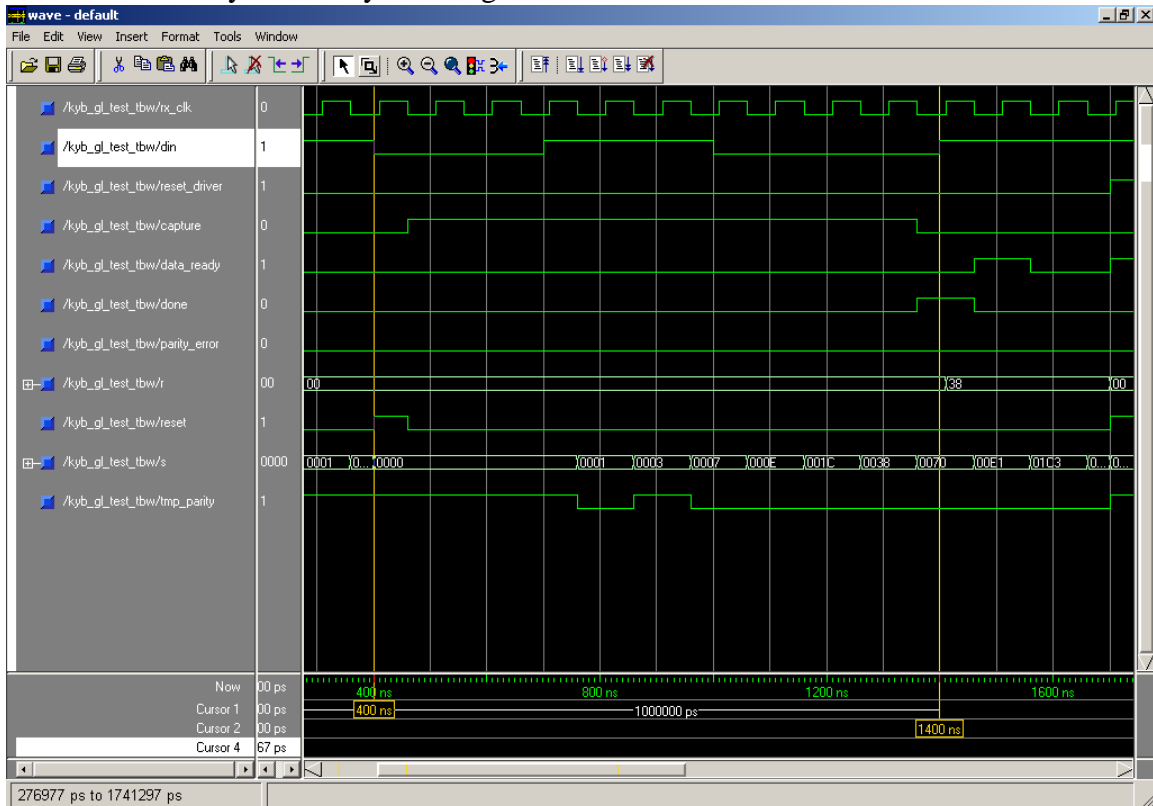


Figure 38

Process Data Unit

In any device driver there is a flag bit which notifies a processor that a new data has arrived from a particular device which needs to be processed. In this design the process consist of displaying your received data on a seven segment display for 3 seconds. Note that during the time that your system is processing the new data the driver is not able to receive any data so any new data must be queued somewhere in the system. Fortunately the keyboard processor is equipped with an internal queue for this purpose. The process data unit is a simple state machine which displays the output of the keyboard driver on

the seven segment display for 3 seconds and resets the driver after that and make it ready to receive a new set of data.

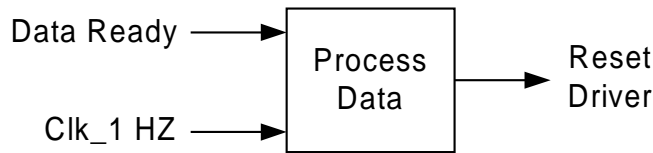


Figure 39

74. Based on the above description design a state machine for this purpose (Figure 40).

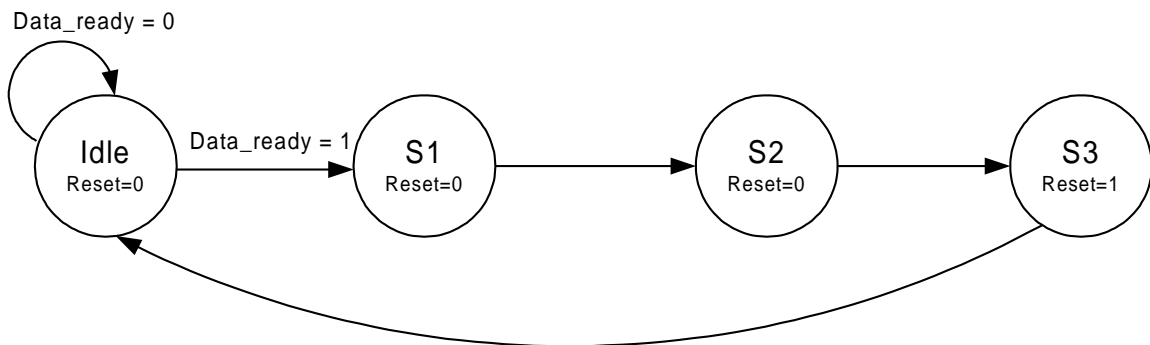


Figure 40

Putting the system Together

75. Create a new schematic and use Figure 41 as a guideline to put the system together.
76. Check the schematic and save your file.
77. The user constraints file for this design is provided in Table 4.
78. Generate the programming file and download that into the board.
79. Press a key on the keyboard and verify the output with the expected output from Table 1.
80. **Demonstrate your system to the TA.**
81. This concludes the labs for 3801. Good luck on your final report and exam. Have a well-earned break! Congratulations!!

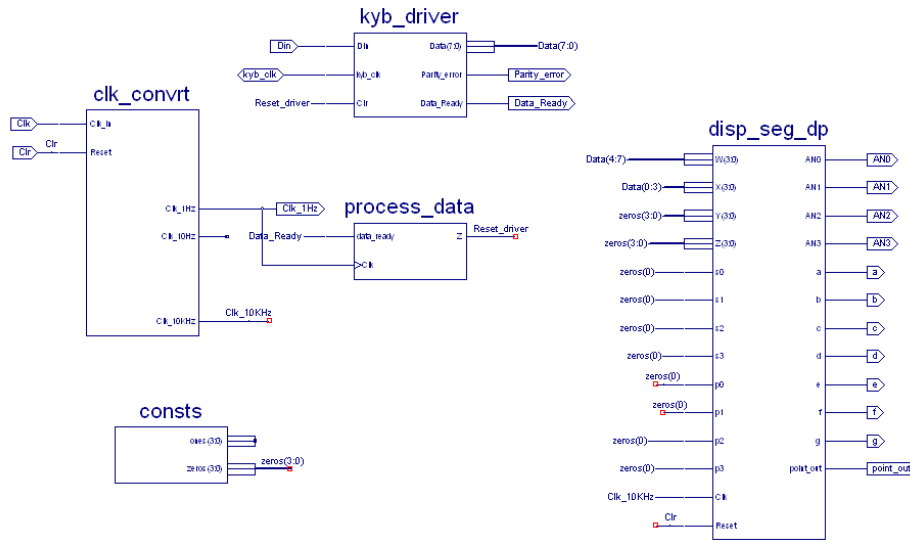


Figure 41

```

NET "Clk" LOC = "T9";
NET "Din" LOC = "M15";
NET "kyb_clk" LOC = "M16";

NET "a" LOC = "E14";
NET "b" LOC = "G13";
NET "c" LOC = "N15";
NET "d" LOC = "P15";
NET "e" LOC = "R16";
NET "f" LOC = "F13";
NET "g" LOC = "N16";
NET "Point_out" LOC = "P16";

NET "AN0" LOC = "D14";
NET "AN1" LOC = "G14";
NET "AN2" LOC = "F14";
NET "AN3" LOC = "E13";

NET "Clr" LOC = "L14";
NET "Parity_error" LOC = "P11";

NET "Data_Ready" LOC = "K12";
NET "Clk_1Hz" LOC = "P12";

```

Table 4

References

- [1] Xilinx, 'Spartan-3 Starter Kit Board User Guide'
- [2] <http://www.computer-engineering.org/ps2protocol/>

Keyboard Interface Sign-Off Sheet

Make sure lab instructors initialize each part. Attach this page to the end of your report.

Your Name:

Lab Partner:

Date Performed:

Demonstrated that the circuit works correctly:

- Keyboard Communication works _____
- Keyboard inputs codes(ChipScope or Agilent): _____
- Overall Keyboard System works: _____