San Jose State University
Department of Electrical Engineering
EE178, Spring 2006, Crabill

Laboratory Assignment #5

Objectives

The objective of this lab is to implement a simple PS/2 keyboard host controller. Standard PS/2 keyboards communicate with a host controller in another device using a two wire serial interface. As a result of completing this lab, you should have some appreciation for how PS/2 keyboards work. Your design will accept transmitted data from a PS/2 keyboard and queue it in a FIFO so that it may be read out by a user at a different rate. Conceptually, this design has a great deal in common with the receive portion of the UART you used in a previous lab. Figure 1 shows a block diagram of the project. The omnipresent clock and reset signals have been omitted.

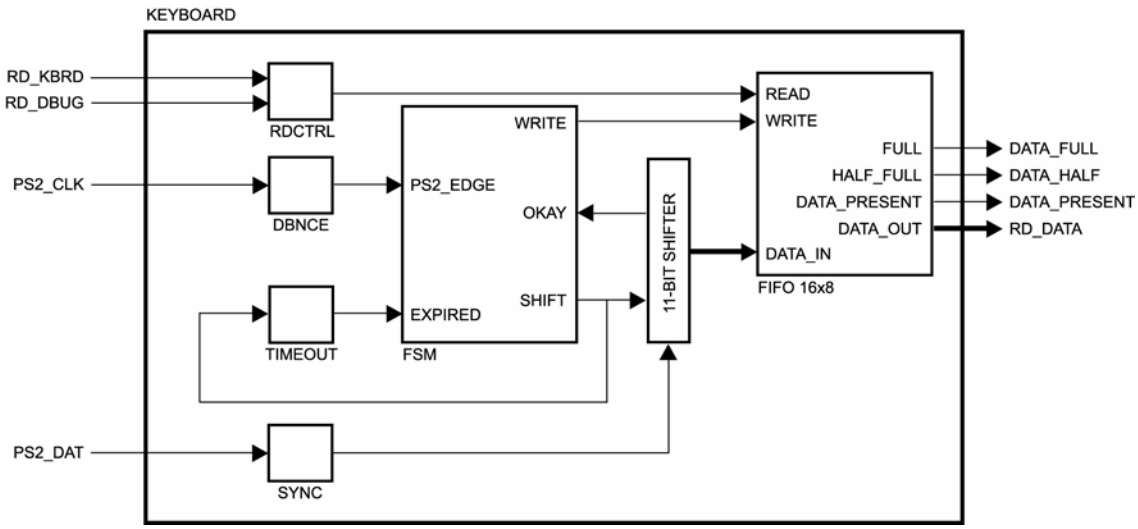

Figure 1: Project Block Diagram

The host controller will be implemented using the datapath and control method of design. Many designs can be divided into two major sections, datapath and control. The datapath section performs operations on data – for example, arithmetic computations or storing values. The control section generates signals to ensure that such operations are performed by the datapath section at the right time and in the right order. Typically, the control section is implemented with a finite state machine, although it may also be implemented with random logic or micro-programmed sequencers. For this lab, you will implement the control section with a finite state machine, using the techniques discussed in class.

When you successfully complete this lab, you will have developed a piece of intellectual property that you might be able to re-use in the future.

Bibliography

The information about PS/2 keyboard to host communication, including the table of scan codes provided in the downloadable support package for this lab, was obtained from web pages at http://www.computer-engineering.org written by Adam Chapweske. I encourage you to review these pages, as the information contained in this lab handout is only a basic summary.

PS/2 Keyboard to Host Communication

Standard PS/2 keyboards communicate with a host controller in another device using a two wire serial interface. These two wires are for data and clock; both are open-collector, which means that devices may either drive the wire low or not drive it at all. When a wire is not actively driven, it is passively pulled up by a pull-up resistor that is part of the host controller.

Data is transmitted serially from the keyboard to the host. Both the clock and data signals are driven by the keyboard. The clock signal may have a frequency from 10 KHz to 30 KHz. The data signal is valid on the falling edge of the clock signal. Data bytes are sent in frames of eleven bits, as shown in Figure 2.

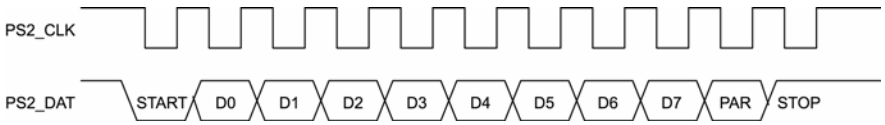

Figure 2: PS/2 Keyboard to Host Communication

The start bit is always zero. The start bit is followed by eight data bits, with the least significant bit transmitted first. Next, a single parity bit is transmitted. The parity bit is one if there is an even number of ones in the eight data bits, otherwise it is zero. The number of ones in the eight data bits plus the parity bit should always be an odd number. This is called odd parity and is used for error detection. The host controller should check the parity of what it receives and if it detects an error, the data should be discarded. Finally, the last bit in the frame is the stop bit which is always one.

Whenever a key is pressed, the keyboard transmits the "make code" for that key. If the key is held for an extended period of time (called the typematic delay), the keyboard repeatedly transmits the "make code" at a fixed frequency (called the typematic rate). Whenever a key is released, the keyboard transmits the "break code" for that key. Collectively, these codes are called "scan codes". Note that scan codes can be one or more bytes long, and are not related to ASCII character codes. See the table of scan codes provided in the downloadable support package for this lab.

At power-on or software reset, the keyboard performs a diagnostic self-test referred to as BAT (Basic Assurance Test). Upon the completion of this test, the keyboard transmits a status code to the host.

Project Description and Requirements

In this design, you are not allowed to use latches. You are allowed to use only one clock and only one asynchronous reset signal. The clock must be the 50 MHz clock signal available from the oscillator on the Spartan-3 Starter Kit board. **You will receive zero points if you do not follow these requirements.**

As shown in Figure 1, the design has a number of inputs. In addition to the clock and reset, there are inputs for the PS/2 keyboard signals and inputs to read the FIFO.

clk	clock signal, 50 MHz from oscillator
rst	reset signal
rd_kbrd	FIFO read enable (for synchronous use only)
rd_dbug	FIFO read enable (for asynchronous pushbutton use only, synchronized and debounced)
ps2_clk	PS/2 keyboard clock signal; open collector, must be pulled up externally
ps2_dat	PS/2 keyboard data signal; open collector, must be pulled up externally

Also shown in Figure 1 are the outputs. The outputs consist of the FIFO data output and three FIFO status signals that indicate how much data is currently stored in the FIFO.

rd_data [7:0]	FIFO data output
data_present	FIFO status indicating one or more pieces of data present
data_half	FIFO status indicating the buffer is at least half full
data_full	FIFO status indicating the buffer is completely full

The user interface consists of an eight-bit data output, three status signals, and two read enable inputs. For this lab, only the synchronized and de-bounced read enable is used. The other read enable input is pulled down and unused. The desired behavior of this interface is as follows:

- When data is present (indicated by data_present) the rd_data output should indicate what is at the head of the FIFO. If data is not present, the rd_data output is undefined.
- When data is present, the user may read the data from rd_data and then assert rd_dbug or rd_kbrd to advance the FIFO to the next location.

The PS/2 keyboard host controller must reliably receive scan codes transmitted by the keyboard. Received data should not be dropped or written into the FIFO multiple times.

Project Design

There is a downloadable support package for this lab. In it, you are provided with a test bench, the top level keyboard module, and the FIFO module. Your task is to design the control logic, which is a finite state machine. Use the block diagram as a tool to understand the datapath for the design. The major sections are described below, followed by some general observations.

Keyboard Module

You are provided with the top level keyboard module. This module contains the datapath and an instantiation of the control. The datapath consists of an 11-bit shift register, a counter/timer, and a FIFO. The control is responsible for making the datapath operate properly to achieve the desired result.

This module also contains some synchronizers and de-bouncing circuits to process the input signals from the PS/2 keyboard and the user.

Shift Register

The datapath shift register is an 11-bit shift register that may be enabled to shift by the control. Associated with the shift register is a small circuit that checks the validity of the received data. The output of this circuit is used by the control. It is only meaningful to check the validity of the received data after all 11 bits have been received, because intermediate shift results may actually appear to have valid start, stop, and parity, even though the complete set of 11 bits has not yet been shifted into the shift register. Eight of the 11 bits in the shift register are brought out as the received data byte, and are connected to the FIFO module data input.

Counter / Timer

The datapath counter/timer is a simple counter that tracks clock cycles. The purpose of this function is to count the time elapsed after a bit of data is shifted into the shift register. Every time a shift occurs, the counter is reset to zero and then begins to increment. If another shift does not occur before the counter reaches its terminal count, a status signal is asserted to indicate this condition. This allows the control to abort the reception of a frame if the keyboard clock signal stops (e.g. the keyboard is unplugged in the middle of a transmission or some other unexpected transmission error).

FIFO Module

To write data to the FIFO, present the data to be written and assert the write enable. At the next rising edge of the clock, the data will be written. For every rising edge of the clock that the write enable is asserted, a piece of data is written into the FIFO.

If the FIFO has data in it, the value at the head of the FIFO is present on the FIFO data output. To read data from the FIFO, assert the read enable. At the next rising edge of the clock, capture the data output; the FIFO will subsequently advance to the next piece of data stored in the FIFO.

Control Module

The control module will contain a finite state machine of your own design. This finite state machine may be either Mealy or Moore; it doesn't matter as long as it satisfies the project requirement. The interface between the control module and the datapath consists of these signals:

ps2_edge	FSM input: indicates a negative edge has occurred on the keyboard clock signal.
shift	FSM output: enables shift register and clears timer/counter.
okay	FSM input: indicates the contents of the shift register has valid start, stop, and parity.
write	FSM output: enables the FIFO to store data presented at its data input.
expired	FSM input: indicates that the timer/counter has reached its terminal count.

It is important to remember that the signals listed above are inputs and outputs to the finite state machine. The finite state machine must be clocked by the system clock. You may not use the ps2_edge signal as a clock in your finite state machine design. Further, it is important to understand that the okay signal does not indicate that 11 bits have been received; it only indicates that the contents of the shift register have a valid start, stop, and parity. The okay signal should be evaluated only after you have shifted 11 bits into the shift register.

General Observations

Before you begin writing any code, you must sit down with scratch paper and draw a state diagram of a finite state machine that will satisfy the design requirements. Once you have a possible solution, write a description of it in Verilog-HDL and proceed to test it in simulation.

When designing synchronous systems, it can be important for asynchronous inputs to be synchronized to the system clock. In this design, ps2_clk, ps2_dat, and rd_dbug are asynchronous to the system clock. Synchronization can be done using a circuit called a synchronizer, shown in Figure 3.

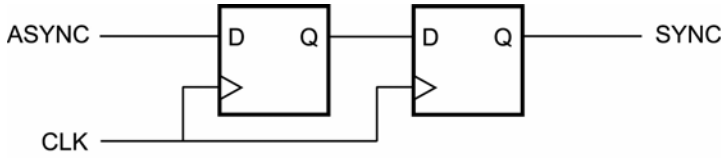

Figure 3: Synchronizer

As a specific example, consider rd_dbug. This input is asynchronous, but will interface with synchronous logic in your design, specifically the FIFO. For this reason, rd_dbug must be synchronized. Mechanical switches, in addition to being asynchronous, also exhibit a behavior called "bounce". When contact is made in the switch, it is possible for the mechanical contacts to bounce a few times. Outside the switch, from an electrical point of view, this can look like repeated closing and opening of the switch. For this reason, rd_dbug must also be debounced so that one button press does not result in multiple FIFO reads.

There is a second FIFO read enable, rd_kbrd. This signal does not pass through a synchronizer or de-bouncer and is therefore not appropriate for use with a button or switch, but would be appropriate for interfacing to other digital logic that is synchronous to the system clock. In your hardware testing, you will not actively use rd_kbrd.

Project Verification

You must perform some minimal functional simulation of the design. This is important for two reasons. First, it will give you confidence your design is working properly before you implement it. Second, if the design does not behave as expected when you download it, you will have a mechanism to quickly create additional test cases to help debug the problem. The instructor will not help you debug logic problems (incorrect design behavior) unless you have a state diagram and are able to run a simulation.

A simple test bench is included with the downloadable support package for this lab. The test bench can be used for functional simulation as well as timing simulation after the implementation step. Feel free to enhance the basic test bench as you see fit.

Project Synthesis

Synthesize your design exactly as you have done before. Do not forget to check the synthesis report. This report will tell you how many clocks exist in your design, under the "clock information" summary. If you have more than one clock, you need to go back and correct your design. Also check to see if any latches were used, you should not have any. These can be found in the "cell usage" summary. If you see anything starting with LD (Latch, D-type) then you need to go back and correct your design.

Project Implementation

Before you implement your design, you will need to add a constraints file and edit the I/O locations and properties. Use the details provided in Figure 4 to constrain your design. You will notice that the clock input is properly constrained, and button 3 is used as the reset. The data output from the keyboard controller drives the LEDs, and the PS/2 signals are assigned to the correct connector on the Spartan-3 Starter Kit board.

The de-bounced read enable is connected to button 1. This is used to read the data from the FIFO. The unused synchronous read enable has a pull-down and is assigned to the anode control for digit three of the seven-segment display. This serves two purposes; it permanently disables this read enable and also activates one digit of the seven-segment display so that individual segments may be used for the FIFO status signals. The three FIFO status signals are assigned to the segment controls to produce a "gas gauge" effect so that you can visually tell how much room is remaining in the FIFO.

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination
rst	Input	L14	BANK3	LVCMOS33	N/A	3.30		
rd_kbrd	Input	E13	BANK2	LVCMOS33	N/A	3.30		PULLDOWN
rd_dbug	Input	M13	BANK3	LVCMOS33	N/A	3.30		
rd_data<7>	Output	P11	BANK4	LVCMOS33	N/A	3.30		
rd_data<6>	Output	P12	BANK4	LVCMOS33	N/A	3.30		
rd_data<5>	Output	N12	BANK4	LVCMOS33	N/A	3.30		
rd_data<4>	Output	P13	BANK4	LVCMOS33	N/A	3.30		
rd_data<3>	Output	N14	BANK3	LVCMOS33	N/A	3.30		
rd_data<2>	Output	L12	BANK3	LVCMOS33	N/A	3.30		
rd_data<1>	Output	P14	BANK3	LVCMOS33	N/A	3.30		
rd_data<0>	Output	K12	BANK3	LVCMOS33	N/A	3.30		
ps2_dat	Input	M15	BANK3	LVCMOS33	N/A	3.30		PULLUP
ps2_clk	Input	M16	BANK3	LVCMOS33	N/A	3.30		PULLUP
data_present	Output	E14	BANK2	LVCMOS33	N/A	3.30		
data_half	Output	N16	BANK3	LVCMOS33	N/A	3.30		
data_full	Output	P15	BANK3	LVCMOS33	N/A	3.30		
clk	Input	T9	BANK4	LVCMOS33	N/A	3.30		

Figure 4: I/O Constraints

For this design, you are required to add timing constraints. After selecting the constraint file in Project Navigator, use one of the available processes to launch the constraints editor. Enter the constraints as shown in Figure 5. These constraints tell the implementation tools that your design must run at 50 MHz and that the input and output timing is bounded. The specific values used for the input and output timing are arbitrary in this instance, but represent reasonable (less than one full clock period) values. If your design fails to meet the timing requirements, consult the instructor for advice.

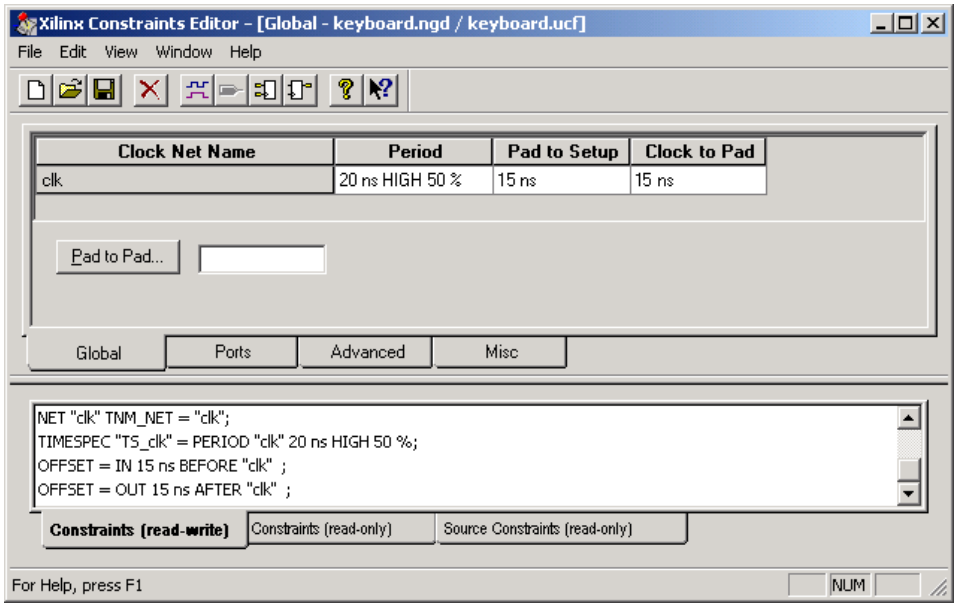

Figure 5: Timing Constraint Entry

You will notice there is a three-pin header next to the PS/2 connector on the Spartan-3 Starter Kit board. This header is used to select the PS/2 keyboard power supply voltage. Verify that a jumper is installed to select 5.0 volts; if not, install the required jumper. Do not use 3.3 volts for the keyboard supply. Verify your design in hardware. When you are satisfied with the results, generate a programming file for the PROM and then load your design into the PROM.

Laboratory Hand-In Requirements

This lab requires a written report. The report must be submitted as a professional-looking document in a single electronic file. A Microsoft Word DOC file is preferred, although an Adobe PDF file is acceptable. Paper submissions are not accepted. The body of the report must be written in English and contain the following sections:

- Title page containing your name, the lab title, and the date.
- Introduction containing a brief summary of the problem you set out to solve and your final results. For example, "The objective of Lab 1 was to implement a two-input XOR gate. I described the XOR gate, implemented it, and successfully demonstrated its operation in hardware."
- Design details documenting how you achieved your result. This is the most important part of the lab report. Illustrate that you understood the problem and explain how you solved it. In this section, you should consider using block diagrams, simple waveforms, FSM state diagrams, tables, figures, and other forms of graphical communication to clarify and strengthen your textual description.
- Simulation strategy and results documenting how you verified your design. You may include simple waveforms or other relevant test bench output.
- Final results. Describe the ultimate result of your efforts. Here, you should also include information such as maximum frequency and resource usage of your implementation.
- Conclusion containing a brief summary and what you thought of the lab. Evaluate what worked well, what did not, and how you might do it again, if you had the chance to start over. Please provide feedback on your perception of the lab difficulty and how you suggest the lab could be improved for next semester. For example, "I successfully described, implemented, and demonstrated the two-input XOR gate. My description of the logic using a continuous assignment was too terse. Next time I would describe the behavior with a verbose procedural assignment. This lab was tedious and boring, next semester you should use a two-input AND gate instead, that would really spice it up!"

Do not include project source code listings in the report. If you wish to refer to specific portions of the source code, cite a file name and a line number. This allows the reader to refer to specific portions of the source code without unnecessarily bloating the body of the report.

Once you have completed a working design and written your lab report, prepare for the submission process. You are required to demonstrate a working design which has been programmed into the PROM. Within six hours of your demonstration, you are required to submit your entire project directory and lab report in the form of a compressed ZIP archive. Your lab report must be in the project directory with the file name l5_yourlastname.doc or l5_yourlastname.pdf. Use WinZIP to archive the entire project directory, and name the archive l5_yourlastname.zip. For example, if I were to make a submission, it would be l5_crabill.zip, containing a report file l5_crabill.doc. Then email the archive to the instructor. Only WinZIP archives will be accepted. If your archive is too large, you may remove:

- The xst subdirectory (temporary synthesis files)
- The work subdirectory (temporary simulation files)
- Any file with a .wlf extension (simulation waveform files)

Demonstrations must be made on or before the due date. If your circuit is not completely functional by the due date, you should still write a report documenting what you have accomplished and turn in what you have to receive partial credit.