THE UNIVERSITY OF ALABAMA IN HUNTSVILLE
Department of Electrical and Computer Engineering

CPE 324 Advanced Logic Design Laboratory

Laboratory Assignment #4
Scanning Keypad Interface

(10% of Final Grade)

Purpose

The purpose of this laboratory is to give each student the opportunity to examine the lower-level issues involved when one interfaces a mechanical device such as a simple keypad to high-speed digital hardware. The specific issues being exposed include the key bounce problem and the use of time multiplexing to minimize the number of I/O connections between the peripheral device and external electronics. This laboratory experience also provides students with the opportunity to utilize hybrid schematic capture and hardware description language design entry paradigms within the same design. It also requires that students be able to integrate their own design elements into a larger system that is composed of additional intellectual property modules.

Design Problem

The specific problem is to develop a sequential design that will decode a 16 key extended touch-tone^{tm} type scanning keypad and then display the hexadecimal symbol that corresponds to the specific key being pressed on the right-most seven segment hexadecimal LED of the Altera DE2-115 Platform. The symbols for the last eight keys that have been pressed should be retained and displayed right to left from the most recent to least recent. An implied constraint is that the symbol associated with each key should be recorded only once as the key is pressed (not when it is released). Each key should be debounced meaning that only <u>one</u> value should be displayed each time the key is pressed. Figure 1 illustrates this basic configuration and Table 1 represents the specified hexadecimal symbol that is associated with each key.

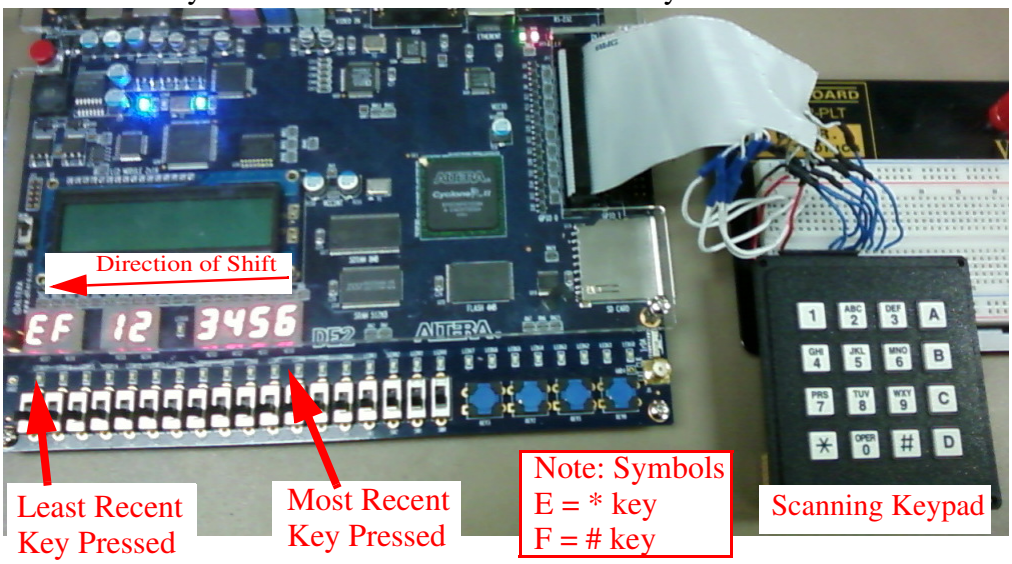

Figure 1: Design Implementation

Table 1: Keypad Key/Hexadecimal Symbol Cross Reference

Key	LED Configuration	Key	LED Configuration
0		8	
1		9	
2		A	
3		B	
4		C	
5		D	
6		*	
7		#	

Background

Scanning keypads (and keyboards) are designed with the idea of reducing the number of I/O lines that enter and exit the keypad. If each key on a keypad (or keyboard for that matter) acted as a separate switch the number of lines that would have to be run into the external digital logic would equal the number of keys on the keypad plus one (assuming a common ground is used). As the number of keys increases, a better alternative is to set up a matrix of switches as shown in Figure 2. This arrangement requires that the total of number of pins exiting the keypad be approximately twice the square root of the total number of keys. (A more exact expression is given by the equation $no\ pins\ = ceil(\sqrt{keys}) + int(\sqrt{keys} + 0.5) \approx 2 \times ceil(\sqrt{keys})$, where *keys* is the number of keys on the keypad (or keyboard), *int* is the next lowest integer value, and *ceil* is the next highest integer value). Thus for a 100 key keyboard a nonscanning implementation would require 101 lines be brought out into the external logic, whereas a scanning keypad implementation would only require that only 20 lines be used.

Section 4.11 (pages 255-264) of the text contain an example of a Verilog HDL only implementation of a scanning keypad design that is created using a finite state machine approach. In this implementation an Algorithmic State Machine, ASM is created and then implemented using Verilog in a higher behavioral level then the approach to be taken in this laboratory. Reading this section, though, will be useful in gaining a more detailed understanding of the keypad scanning process and the key debounce problem as well as illustrating a valid alternate solution to this problem. It should be noted that a major difference between the scanning keypad configuration in this laboratory is that the design in the text utilizes pull-down resistors and the specified design configuration for this design should use a pull-up resistor based configuration.

Specified DE2-115/Peripheral Configuration.

Scanning keypads organize the normally open switches into a two-dimensional array where each switch is located at a row/column junction point as shown if Figure 2

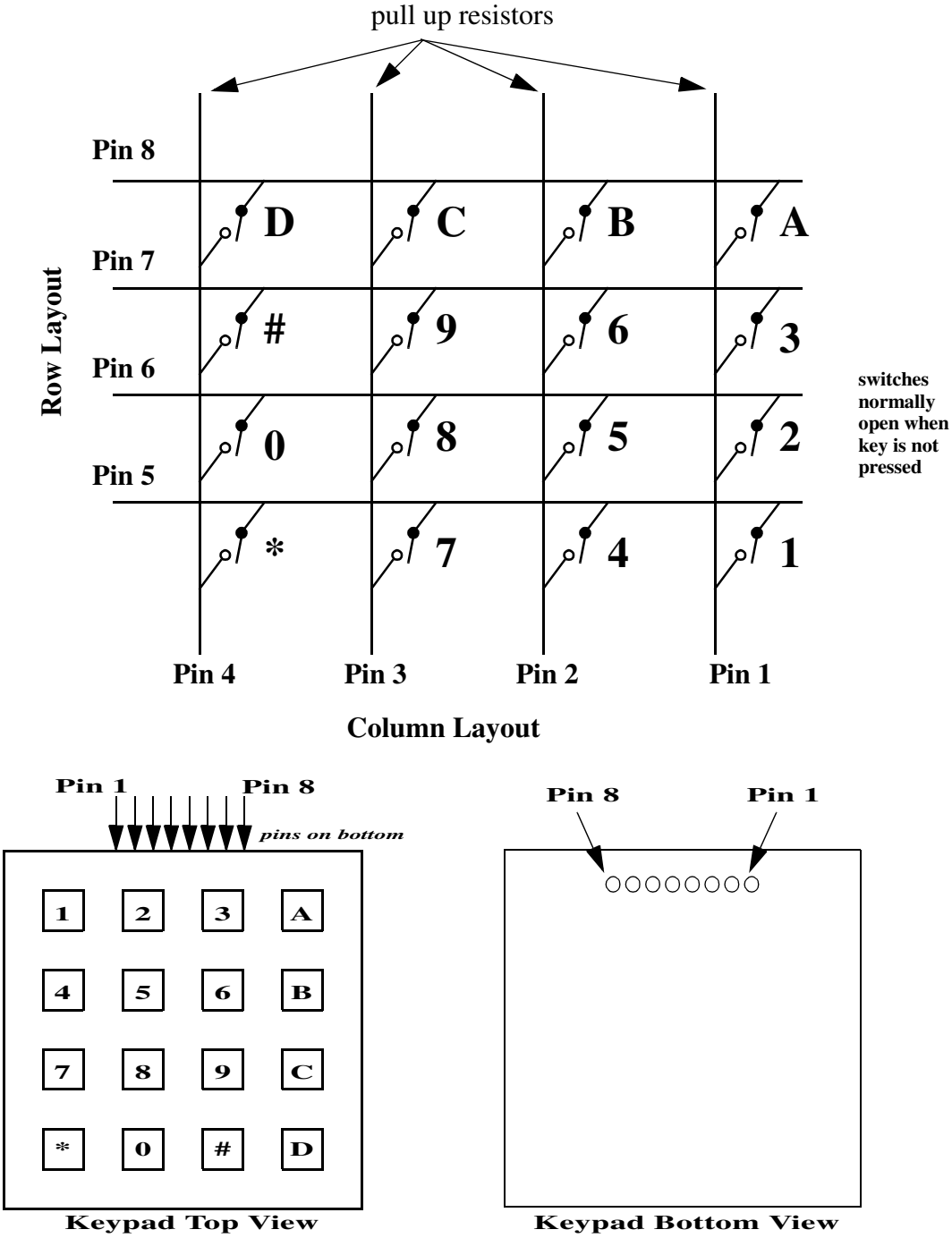

Figure 2: Keypad Switch Layout and Pin Numbering

This is accomplished using multi-level traces within the circuit board. The general concept for a scanning keypad interface logic is that it produces a selecting logic signal on one row at a time and if any of the switches that are connected to the selected row are pressed then a defining logic value can be observed on the column that is associated with that key. As in the case in this labora-

tory when a positive logic system and pull-up resistors are used along the columns (as indicated in Figure 2) this means that the normal value one would observe along any of the columns is a logic 1. For the keypad scanning logic to select a row it would need to drive it to a logic 0. When a key is pressed and its row is selected then current will flow from VCC into the driving logic through the pull-up resistor that is associated with the column. The voltage drop across this resister would limit the current flowing into the driving logic and resulting in a voltage level that is near zero which would be interpreted as a logic low. In the case where a row is not selected it will be driven to a logic 1. This means that there will be virtually no voltage drop across the resister and a logic 1 will be observed along columns (unless another row is selected and one or more of its associated keys are pressed). If the logic is designed to drive one row at a time to a logic 0 with the other rows being held to a logic 1 and if the same logic can be made to selectively determine which columns along the selected rows of switches is at a logic 0 then one can determine which key along the two-dimensional array is pressed. Scanning keyboards interface circuits that utilize pull-up resistors progressively scan through each row by activating it with a logic 0. Then they scan through each column to see if any of the keys along this selected row have been driven to a logic 0. Each time a row is selected and a column monitored a unique code is given which is sometimes called the keyboard code. This code is usually specific to the keypad or keyboard but can be easily mapped to well known standardized codes such as ascii or unicode.

External Hardware Configuration

All designs in this laboratory should incorporate the external wiring configuration that is shown below in Figure 3.

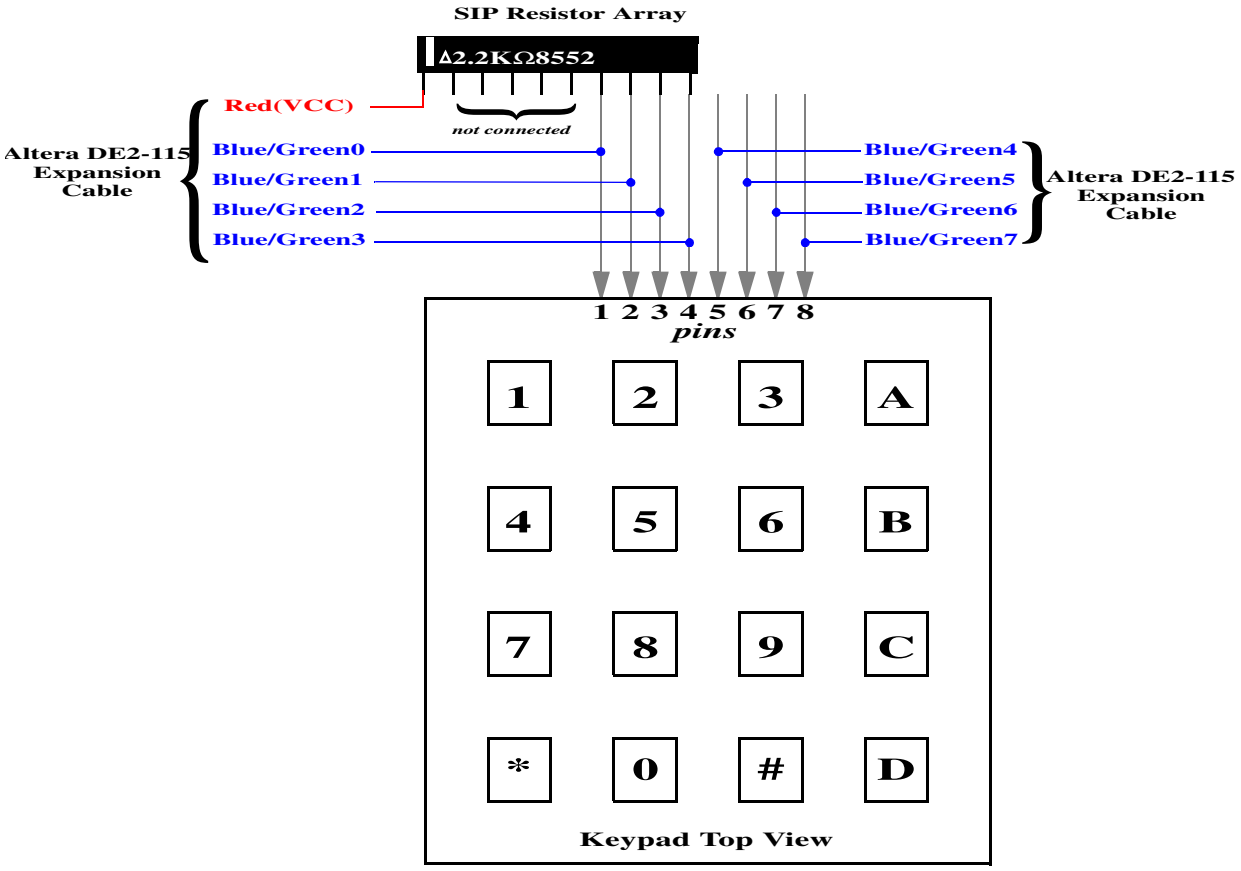

Figure 3: DE2-115 Keypad Wiring Diagram

The eight *blue/green* lines should be connected to the eight inputs pins of the keypad as shown in the figure (*blue/green0* wire connected to keypad pin1 *blue/green1* wire connected to keypad pin 2, etc.). To simplify this implementation a bussed SIP resister array is used to provide separate pull-up resistors for each column. A bussed SIP resister array contains a set of closely matched resistors that share a common internal connection point. This point is on the end of the SIP that has a distinct marking. Figure 4 shows the configuration of the SIP resistor used in this lab.

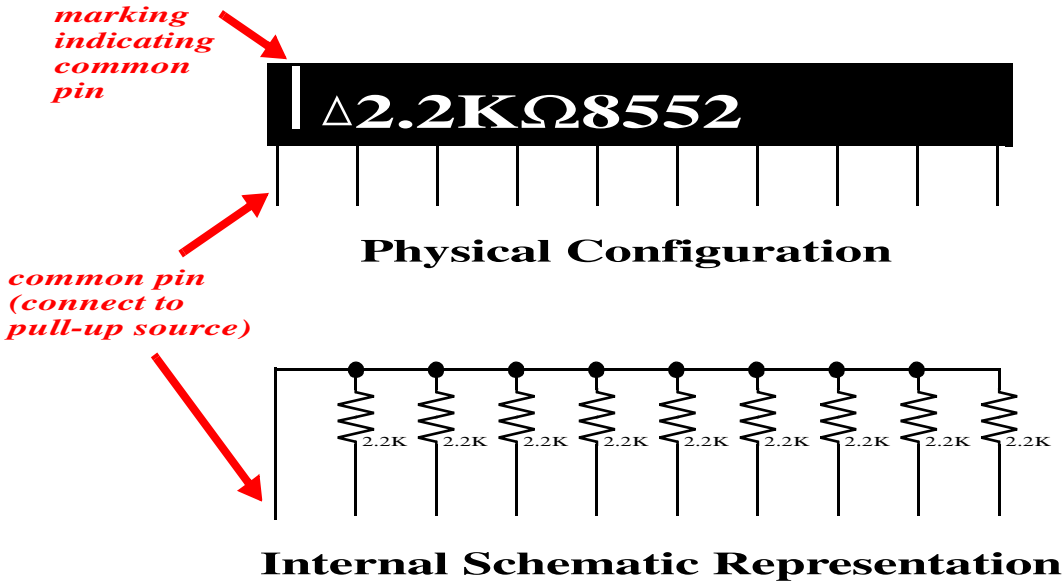

Figure 4: Bussed 2.2K Single In-Line Package, SIP Resistor

To complete the configuration the SIP resistor needs to be placed so that the last four pins are connected to pins 1-4 of the keypad as shown in Figure 3. The first pin near the marking line should then be connected to the red wire coming out of the DE2-115 expansion cable. All the other lines including the ground should be disconnected but secured on the breadboard to insure that no unintentional contact is made to other electrical elements.

Top Level Design

The top-level of the design is shown in Figure 5. It is composed of five main modules (labeled C0 through C4) some of which are to be implemented in Verilog HDL and others to be implemented using schematic capture.

The modules C0 and C5 are already implemented with their functionality being verified through unit testing. The module C0 is basically a clock division component that produces two clocks that are derived from a global base clock. The slower clock, ***scan_clk***, only produces active transitions whenever its enabling signal, ***scan_en***, is high. The other clock, ***debounce_clk***, continues indefinitely. The purpose of the C0 module is to supply separate clocking signals to the key scan and debounce logic while temporarily stopping the key scan process whenever a key is being pressed.

The purpose of the C5 *hex_seven_shift* module is to latch in a 4-bit value, convert it to hexadecimal form for display on a 7-segment display, and output this value on the least significant seven-segment port, while shifting the old seven-segment values one position to the left, thereby retaining the last 8 values that have been entered. Latching occurs on the high to low transition of the ***shift*** input signal.

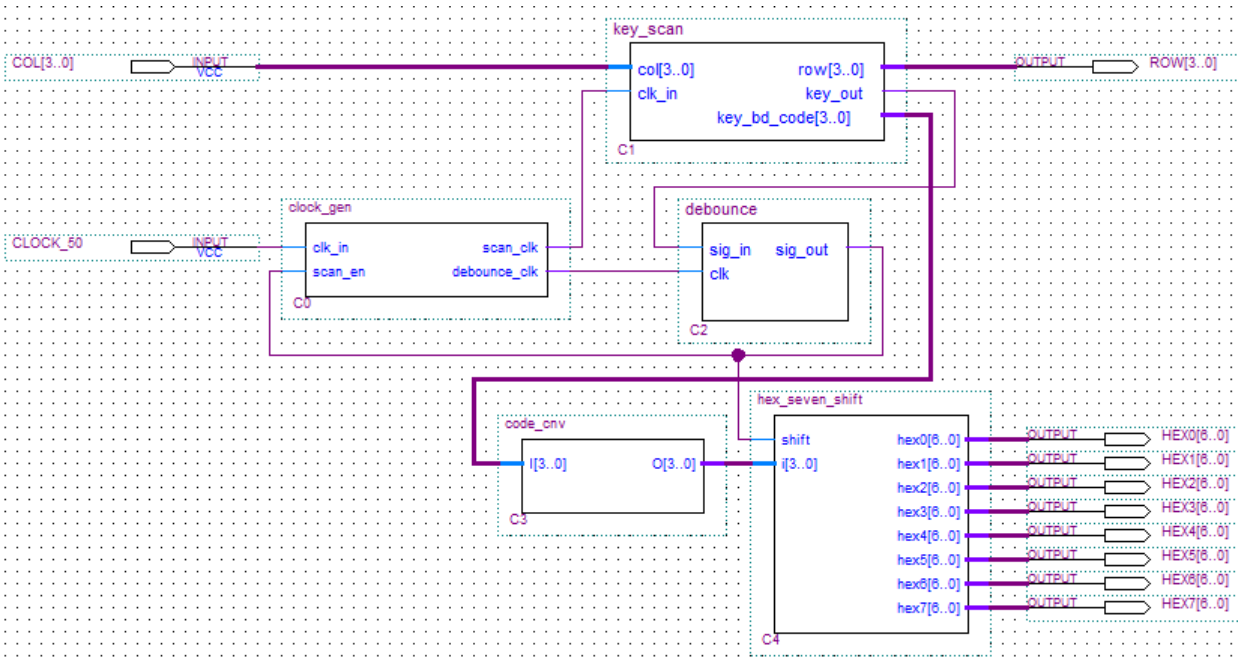

Figure 5: Top-level View of the Scanning Keypad Design

The remaining modules are to be designed by the students in the order specified in the assignment section. Module C1, the *key_scan* module is the heart of the design. Its function is to monitor the four columns of the keypad and drive the four rows in the manner discussed previously. Whenever a key press is detected a unique 4-bit keyboard code that is associated with the key being pressed should be present on the ***key_bd_code*** output and the **key_out** output should transition from a logic 1 to 0. The module should then stop the scanning process and hold these values as long as the key is pressed. This function is greatly facilitated by the fact that the modules are connected in such a way that whenever the ***key_out*** signal is low it inhibits the ***scan_clk*** clock generation that feeds the *key_scan* module.

Module C2 (*debounce*) is in effect a glitch filtering module that is designed to produce clean high-to-low and low-to-high transitions during initial switch transition periods. This clean signal is then used to trigger when new data is displayed and shifting occurs. In the initial design this module is provided as a wrapper module with the signal ***sig_in*** simply being connected to ***sig_out***.This allows for the design to be validated incrementally during FPGA prototpying, making it possible to defer the creation of this module until later in the design process after other modules are verified.

Module C3 (*code_cnv*) is a 4-bit to 4-bit code conversion module. Its function is to convert from the unique 4-bit keyboard code produced by the *key_bd_code* module into the 4-bit code needed by the *hex_seven_shift* module in order to produce the pattern shown in Table 1. In the initial design this module is present on the top level but actually performs no code conversion but simply passes the uncovered code straight through it from its input to its output much like a bus. As in the case of module C2 the initial state of this module allows validation of other portions of the design to proceed the development of this element.

Laboratory Assignment:

This laboratory assignment is composed of three phases, with each phase culminating with the current state of the design being validated by prototyping it on the Altera DE2-115 platform. Students must successfully complete and demonstrate the valid functionality of a given phase to their design to the lab instructor before proceeding to the next phase. Demonstration will involve configuring the Altera DE2-115 board and verifying that the specified design objectives were met when the specific module was integrated into the overall design. To facilitate design entry a template of the design is provided on the course Canvas website. This template should be downloaded into the *lab4* folder on each student's usb drive.

Phase I: Creation of the *key_scan* module using schematic capture design entry techniques

In this phase, students are to develop the main part of the keyboard scanning design by transversing into the **key_scan.bdf** module and adding the necessary logic. The design must contain a *binary counter*, a *multiplexer*, a *decoder*, and basic logic gates (*AND, OR, NOT*). A successfully designed module will result in the overall design producing a unique code that is associated with the key that is pressed to appear on one or more 7-segment LED's on the DE2-115. Occasional incorrect values are allowable at this stage but a clear output pattern should be observable. Students must obtain the laboratory instructor's approval before going on to Phase II of this laboratory.

Phase II: Creation of the *debounce* module using schematic capture design entry techniques

In this phase, students are to develop the basic glitch filtering circuit that reduces the effect of mechanical conductor vibrations. Whenever the two electrical conducting contacts that are present in a mechanical switch are placed in contact with one another they tend to bounce back and forth causing the electrical connection between them to be made and broken many times before the connection becomes stable. The same is generally true when the two contacts are separated from one another. A debounce circuit is one that filters this activity in a manner that only a single logic transition will be present. Often this clean output is important to prevent multiple triggering of electronic devices.

There are several ways the effects of switch bounce can be removed. One way is to introduce a delay period after an initial transition from one logic level to another during which time the input is no longer monitored. This has the disadvantage that responsiveness can be compromised if the period is too large and that noise can cause false triggers. The manner that switch bounce is to be filtered in this laboratory assignment involves the idea of creating a sequential circuit that is clocked at a rate that is likely to record the multiple triggerings that are associated with switch bounce. The idea is that this debounce circuit will remember a set of values from the past and only pass on a transition from low-to-high or high-to-low when the input being monitored has been stable at its new logic level for that set number of clock cycles.

In this phase of the assignment, students are to develop a design using *D-flip flop*, or a *shift register* elements, simple *AND*, *OR*, or *NOT* gates and a J-K flip flop. The design should function in the following manner. When an input changes state from high-to-low or low-to-high it must remain at the new state consistently for eight clock cycles before its output makes a similar transformation.

Figure 6 illustrates the desired functionality of the debounce module for both low-to-high and high-to-low transitions. Often simulation is used to verify waveform friendly component elements such as this module but this is not required for this assignment because of the relative simplicity

of the design and the time constraints associated with the course.

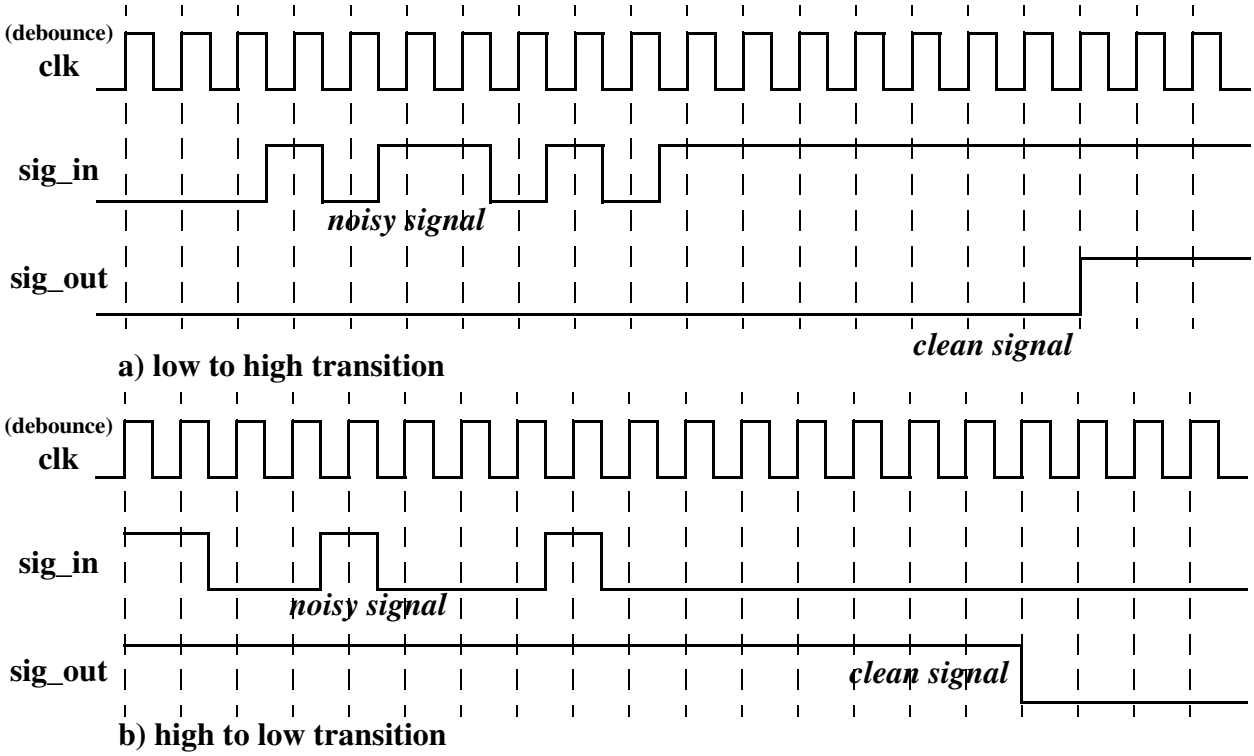

Figure 6: Example Waveforms for *debounce* Module

Upon completion of the debounce module the entire design should be implemented again on the DE2-115 and demonstrated to the laboratory instructor. A successful design will produce a unique symbol that is associated with the key that is pressed as in the previous case, but this time only one such value should appear for each key that is pressed and the value displayed should be consistent. If not students should first evaluate the correctness of the *debounce* module they have implementation and may also have to revisit the *key_scanner* module if they believe it is a source of an undetected design error. Student's must obtain the laboratory instructor's approval before moving on to the final phase.

Phase III: Creation of the *code_cnv* module using Verilog HDL

The code converter module, *code_cnv*, template is written in a manner that already utilizes the Verilog HDL. It employs the @always procedural construct that contains a simple *case* statement which maps the input over to the output. Students are to modify this file so that the output symbols shown in Table 1 are produced for each key that is pressed. Students must demonstrate this to their lab instructor before proceeding to answer the post lab questions.

Post-Lab Questions:

Answer the following questions and include your answers in the appropriate section of the final laboratory report.

1. Examine the Verilog HDL model of the *clock_gen* module that is shown below. Show an equivalent design for this module that uses a single binary counter and some gating (and/or latching). Do this as an academic exercise only you are not expected to use Quartus II components.

```
// This module generates the scan and debounce clocks pulses
// that are derived from the 50 Mhz Input clock on the DE2-115
// board using clock division methods
module clock_gen(clk_in,scan_en,scan_clk,debounce_clk);
   input clk_in,scan_en;
   output scan_clk,debounce_clk;
   reg [16:0] count;

   assign scan_clk = (scan_en) ? count[16]: scan_clk;
   assign debounce_clk = count[12];

   always @ (posedge clk_in)
      begin
      count <= count + 1;
      end
endmodule
```

2. For the *clock_gen* module above, given that the ***clk_in*** input is a 50 Mhz clock source what is the frequency of the ***scan_clk*** and ***debounce_clk*** signals that are produced?
3. For the *clock_gen* module, how does the ***scan_en*** signal affect the ***scan_clk***? How does it affect the ***debounce_clk***?
4. Why do we need a debounce circuit in this design? What happened in Phase 1 when you did not have it? In your report, show your design for the *debounce* module and describe how it works.
5. What factors should one consider when choosing the frequencies of *clock_gen* module output clock signals? Would these values be appropriate for all mechanical push button normally open/monetarily closed devices? Why or why not?

6. The Verilog HDL model for *hex_seven_shift* module is shown below.

```
module hex_seven_shift(shift,i,hex0,hex1,hex2,hex3,hex4,hex5,hex6,hex7);
   input shift;
   input [3:0] i;
   output reg [6:0] hex0,hex1,hex2,hex3,hex4,hex5,hex6,hex7;
   reg [6:0] hexout;

   always @ (i)
      case (i)
         0  : hexout <= 7'b1000000;
         1  : hexout <= 7'b1111001;
         2  : hexout <= 7'b0100100;
         3  : hexout <= 7'b0110000;
         4  : hexout <= 7'b0011001;
         5  : hexout <= 7'b0010010;
         6  : hexout <= 7'b0000010;
         7  : hexout <= 7'b1111000;
         8  : hexout <= 7'b0000000;
         9  : hexout <= 7'b0011000;
         10 : hexout <= 7'b0001000;
         11 : hexout <= 7'b0000011;
         12 : hexout <= 7'b1000110;
         13 : hexout <= 7'b0100001;
         14 : hexout <= 7'b0000110;
         15 : hexout <= 7'b0001110;
         default : hexout <= 7'bx;
      endcase

   always @ (negedge shift)
      begin
         hex0 <= hexout;
         hex1 <= hex0;
         hex2 <= hex1;
         hex3 <= hex2;
         hex4 <= hex3;
         hex5 <= hex4;
         hex6 <= hex5;
         hex7 <= hex6;
      end

endmodule
```

Describe how the design works. What is the purpose of the ***shift*** signal? When is it active? Draw a block diagram of an equivalent logic network using eight 7-input FLIP-FLOPs and a single combinational block that converts from 4-bit binary to hexadecimal. Do this as an academic exercise only you are not expected to use Quartus II components.

7. Describe how you implemented your *code_cnv* module. Is this module a combinational or sequential design?

DE2-115 Pin Connection Information.

Table 2: Keyboard Row/Column Connections

Keypad Pin Number	Design Signal Name (design inputs)	DE2-115 Pin Location	Expansion Cable Wire	Keypad Pin Number	Design Signal Name (design outputs)	DE2-115 Pin Location	Expansion Cable Wire
1	COL[0]	PIN_AC19	blue/green0	5	ROW[0]	PIN_AF24	blue/green4
2	COL[1]	PIN_AF16	blue/green1	6	ROW[1]	PIN_AE21	blue/green5
3	COL[2]	PIN_AD19	blue/green2	7	ROW[2]	PIN_AF25	blue/green6
4	COL[3]	PIN_AF15	blue/green3	8	ROW[3]	PIN_AC22	blue/green7

Table 3: DE2-115 Pin Locations for 7-Segment Hexadecimal LED Displays HEX0 -- HEX7

Design Signal Name	Direction	DE2-115 Pin Location	Design Signal Name	Direction	DE2-115 Pin Location
HEX0[6]	Output	PIN_H22	HEX4[6]	Output	PIN_AE18
HEX0[5]	Output	PIN_J22	HEX4[5]	Output	PIN_AF19
HEX0[4]	Output	PIN_L25	HEX4[4]	Output	PIN_AE19
HEX0[3]	Output	PIN_L26	HEX4[3]	Output	PIN_AH21
HEX0[2]	Output	PIN_E17	HEX4[2]	Output	PIN_AG21
HEX0[1]	Output	PIN_F22	HEX4[1]	Output	PIN_AA19
HEX0[0]	Output	PIN_G18	HEX4[0]	Output	PIN_AB19
HEX1[6]	Output	PIN_U24	HEX5[6]	Output	PIN_AH18
HEX1[5]	Output	PIN_U23	HEX5[5]	Output	PIN_AF18
HEX1[4]	Output	PIN_W25	HEX5[4]	Output	PIN_AG19
HEX1[3]	Output	PIN_W22	HEX5[3]	Output	PIN_AH19
HEX1[2]	Output	PIN_W21	HEX5[2]	Output	PIN_AB18
HEX1[1]	Output	PIN_Y22	HEX5[1]	Output	PIN_AC18
HEX1[0]	Output	PIN_M24	HEX5[0]	Output	PIN_AD18
HEX2[6]	Output	PIN_W28	HEX6[6]	Output	PIN_AC17
HEX2[5]	Output	PIN_W27	HEX6[5]	Output	PIN_AA15
HEX2[4]	Output	PIN_Y26	HEX6[4]	Output	PIN_AB15
HEX2[3]	Output	PIN_W26	HEX6[3]	Output	PIN_AB17
HEX2[2]	Output	PIN_Y25	HEX6[2]	Output	PIN_AA16
HEX2[1]	Output	PIN_AA26	HEX6[1]	Output	PIN_AB16
HEX2[0]	Output	PIN_AA25	HEX6[0]	Output	PIN_AA17
HEX3[6]	Output	PIN_Y19	HEX7[6]	Output	PIN_AA14
HEX3[5]	Output	PIN_AF23	HEX7[5]	Output	PIN_AG18
HEX3[4]	Output	PIN_AD24	HEX7[4]	Output	PIN_AF17
HEX3[3]	Output	PIN_AA21	HEX7[3]	Output	PIN_AH17
HEX3[2]	Output	PIN_AB20	HEX7[2]	Output	PIN_AG17
HEX3[1]	Output	PIN_U21	HEX7[1]	Output	PIN_AE17
HEX3[0]	Output	PIN_V21	HEX7[0]	Output	PIN_AD17

System Clock: Design Signal Name: **CLOCK_50** -- **Input** -- Pin Location: **PIN_Y2**