

Lab #6: Design Example: Keypad Scanner and Encoder - Part 1 (120 pts)

Objective

The objective of lab assignments 6 through 10 are to systematically design and implement an FPGA-based keypad scanner. The keypad scanner will utilize a FIFO for data storage and retrieval, a display mux, and the seven-segment displays, slide switches, and LEDs of the Digilab XLA prototyping board.

The top level block diagram of the system is shown in Figure 1 and the I/O in Table 1. When a button of the hex keypad is pressed the system must decode the button and store the data in an internal FIFO. The *read* button will be used to read data from the FIFO and display the data on the seven-segment displays. The *mode_toggle* input will be used to toggle between display states so that more than eight signals can be presented for view on the LED's. The LEDs will display the status of the FIFO and other information. The hardware prototype will be verified to operate with the Grayhill 072 hex Keypad. A system partition is shown in Figure 2. As input *mode_toggle* is depressed output *DGrp* will toggle between DGrp1, DGrp2, and DGrp3 as denoted in Table 2.

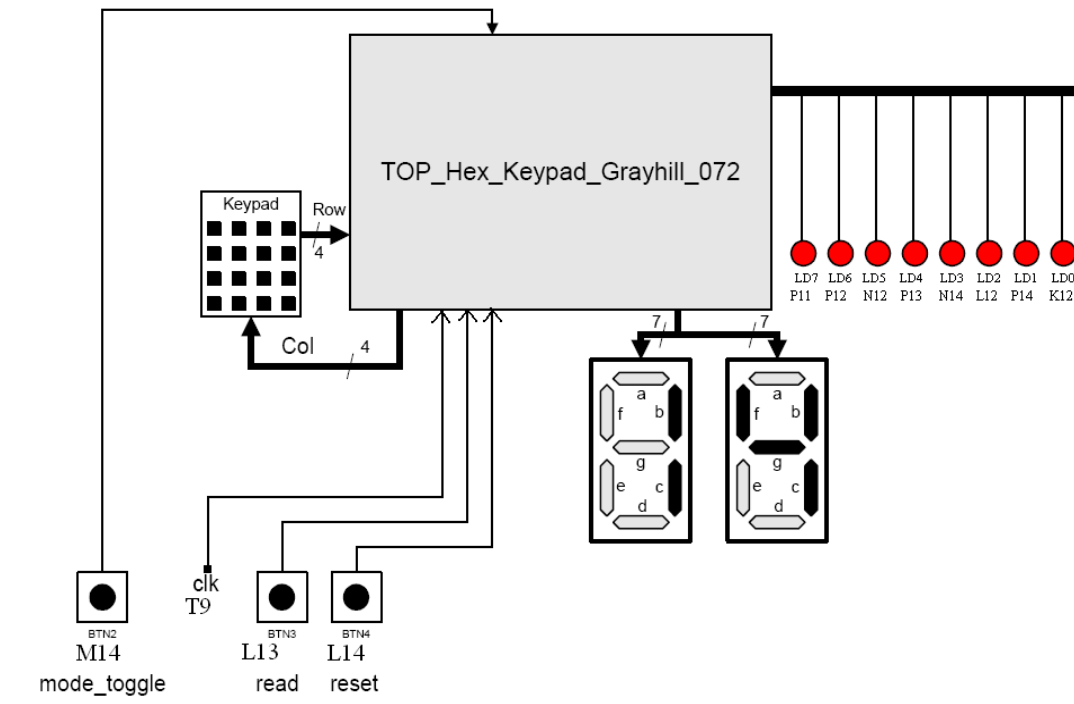


Figure 1: Keypad scanner top level

Name	Direction	Width	Purpose
clk	input		50MHz clock
reset	input		Asynchronous active high reset
read	input		Read from the fifo.

mode_toggle	input		Toggle between LED display states
Row	input	[3:0]	Keypad row
Col	output	[3:0]	Keypad column
seven_seg	output	[6:0]	7-segment character - active low
seven_seg_0_en	output		7-segment character 0 enable – active low
seven_seg_1_en	output		7-segment character 1 enable – active low
seven_seg_2_en	output		7-segment character 2 enable – active low
seven_seg_3_en	output		7-segment character 3 enable – active low
DGrp	output	[7:0]	8 LED's

Table 1: Keypad scanner I/O

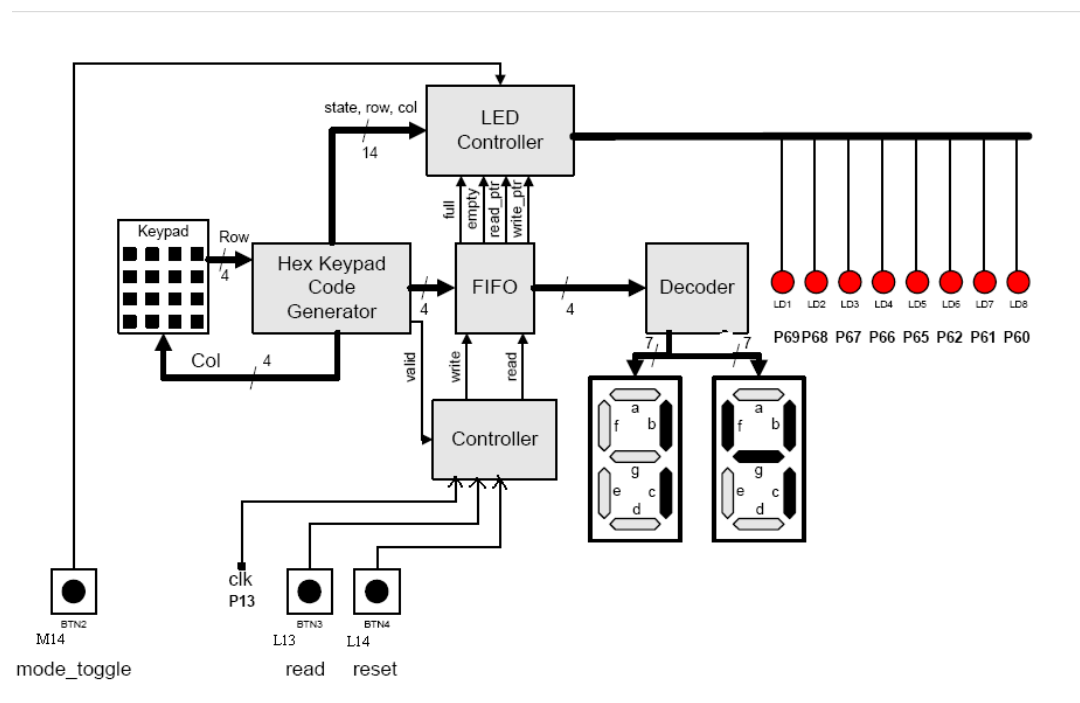


Figure 2: Partition of keypad scanner

Data Group	System Debug Info
DGrp1	{1'b0, read_ptr[2:0], 1'b0, write_ptr[2:0]}
DGrp2	{<your choice>, empty, full}
DGrp3	{Row, Col}

Table 2: LED System Debug info

The objective of this lab is to design and test the keypad scanner/encoder. **This lab will be completed individually.**

Keypad Scanner

Keypad scanners are used to enter data manually in digital telephones, computer keyboards and other digital systems. Telephones have a decimal keypad, and computer keyboards usually have a hexadecimal keypad. A keypad scanner responds to a pressed key and forms a code that uniquely identifies the key that is pressed. It must take into account the asynchronous nature of the inputs and deal with switch debounce. Also, in some applications, it must not interpret a key to be pressed repeatedly if it is pressed once and held down. We will use a synchronizer to synchronize the asynchronous inputs and a debounce circuit to debounce the synchronized input.

Let's consider a scheme for designing a scanner/decoder for the hexadecimal keypad circuit shown in Figure 3. A pressed button establishes a connection between a row and a column at the location of the button. Each row is connected to a pull-down resistor, and is connected to a column when a keypad button is pressed. If a column line is connected to the supply voltage, a row that is connected to that column by a pressed button will be pulled to the supply voltage too; otherwise the row line is pulled down to 0 V. The keypad code generator unit has control over the column lines, and will exercise that control to detect the location of a pressed button.

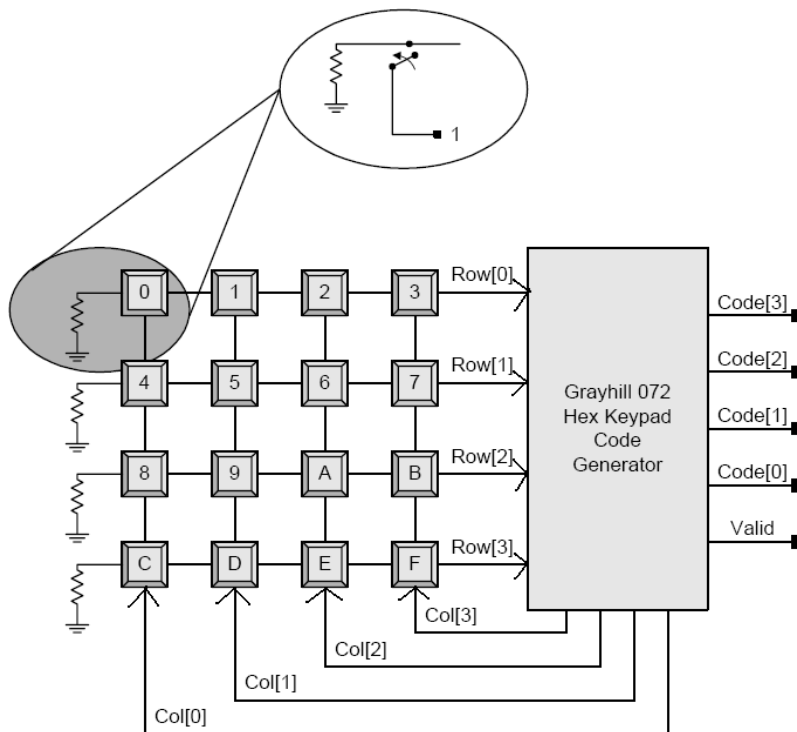


Figure 3: Scanner/encoder for the Grayhill 072 hexadecimal keypad circuit

A keypad code generator must implement a decoding scheme that (1) detects whether a button is pressed, (2) identifies the button that is pressed, and (3) generates an output consisting of the unique code of the button. The decoding scheme will be implemented as a synchronous sequential machine with the

button codes shown in Table 3. The outputs of the machine are the column lines, the code lines, and a signal, *Valid*, that indicates a valid code. Note that for each key press *Valid* will only be asserted for 1 clock cycle to indicate a valid signal on Code. This is true even if a key is pressed and held down.

Key	Row[3:0]	Col[3:0]	Code
0	0 0 0 1	0 0 0 1	0 0 0 0
1	0 0 0 1	0 0 1 0	0 0 0 1
2	0 0 0 1	0 1 0 0	0 0 1 0
3	0 0 0 1	1 0 0 0	0 0 1 1
4	0 0 1 0	0 0 0 1	0 1 0 0
5	0 0 1 0	0 0 1 0	0 1 0 1
6	0 0 1 0	0 1 0 0	0 1 1 0
7	0 0 1 0	1 0 0 0	0 1 1 1
8	0 1 0 0	0 0 0 1	1 0 0 0
9	0 1 0 0	0 0 1 0	1 0 0 1
A	0 1 0 0	0 1 0 0	1 0 1 0
B	0 1 0 0	1 0 0 0	1 0 1 1
C	1 0 0 0	0 0 0 1	1 1 0 0
D	1 0 0 0	0 0 1 0	1 1 0 1
E	1 0 0 0	0 1 0 0	1 1 1 0
F	1 0 0 0	1 0 0 0	1 1 1 1

Table 3: Keypad codes for a hexadecimal scanner/encoder

The pin-out of the ribbon cable for the keypad is Figure 4. The column (Col) is scanned by driving a 1 across each column and observing the value of Row. From this information it can be determined what button was pressed. For example if Col[2] is driven to 1 and a 1 is observed on row[1] key 6 has been pressed.

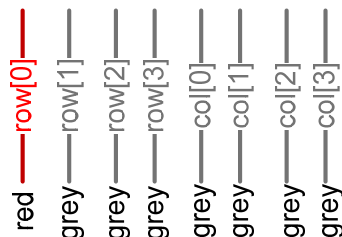


Figure 4: Pinout for keypad

Synchronization Circuit

The input from the keypad, *Row*, is asynchronous. Any flip-flop that uses input *Row* will have to meet that flip-flop's setup or hold time requirement as depicted in Figure 5. When a flip-flop's setup/hold time requirement is not met the flip-flop's output may be unknown for a period of time.

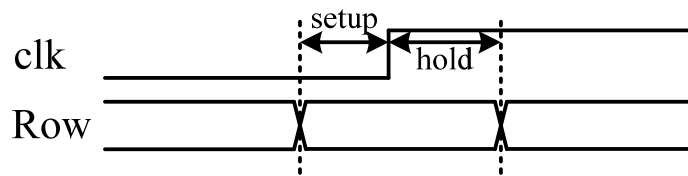


Figure 5: Setup/hold time for a flip-flop

To combat this problem a synchronization circuit will be used as depicted in Figure 6 with the behavior as depicted in Figure 7. The *clk* signal is connected to the high speed system clock.

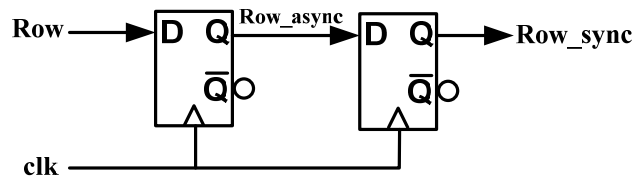


Figure 6: Synchronization Circuit

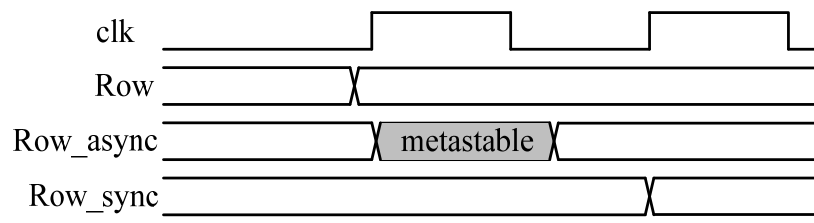


Figure 7: Synchronization Circuit timing

Debounce Circuit

The Row input can bounce for many milliseconds. There are many possibilities for de-bounce circuits. One possibility is to create a very slow divided down clock (~100Hz) and if Row is the same value on two clock edges then Row is considered to have stabilized. Another possibility that does not require dividing down the clock is a counter that will count the number of times a potential transition has occurred. For example, if the current de-bounced value is a zero, the counter will count the number of sequential ones detected. If a zero is detected the counter is reset. When the counter reaches a set value the de-bounced value will be set to a 1. Then the counter begins to look for a zero. Sample code is below:

```
module debounce (
    input wire clk,
    input wire reset, // Active high
    input wire data_in, // Input to debounce

    output reg data_debounced // Data debounced
);
```

```
reg [15:0] change_reg;

always @(posedge clk or posedge reset) begin
    if (reset)
        change_reg <= #1 'b0;
    // If we are at a 1 and get a 1 reset the change reg
    else if (data_debounced && data_in)
        change_reg <= #1 'b0;
    // If we are at a 0 and get a 0 reset the change reg
    else if (!data_debounced && !data_in)
        change_reg <= #1 'b0;
    // If we are at a 1 and get a 0 increment the change reg
    else if (data_debounced && !data_in)
        change_reg <= #1 change_reg + 1'b1;
    // If we are at a 0 and get a 1 increment the change reg
    else if (!data_debounced && data_in)
        change_reg <= #1 change_reg + 1'b1;
end

// If change_reg is MAX, toggle data_debounced
always @(posedge clk or posedge reset) begin
    if (reset)
        data_debounced <= #1 'b0;
    else if (change_reg == 16'hFFFF)
        data_debounced <= #1 !data_debounced;
end
endmodule
```

Testbench

Using the testbench depicted in Figure 8, verify the functionality of *Hex_Keypad_Grayhill_072*. The scanner is to be tested within a Verilog environment, not on a physical prototyping board with a physical keypad. Therefore, the testbench shown in Figure 8 must include (1) a signal generator that will simulate the assertion of a key, and (2) a module, *Row_Signal*, that will assert a row line corresponding to the asserted key, and (3) *Hex_Keypad_Grayhill_072*, the DUT. After the model of the keypad scanner has been verified, it can serve as a user interface in simulating other systems, and can also be used in a physical environment with confidence that it should function correctly, which greatly reduces the scope of a search for the source of an error in the operation of a prototype. The code for module *Row_Signal* is in Figure 9

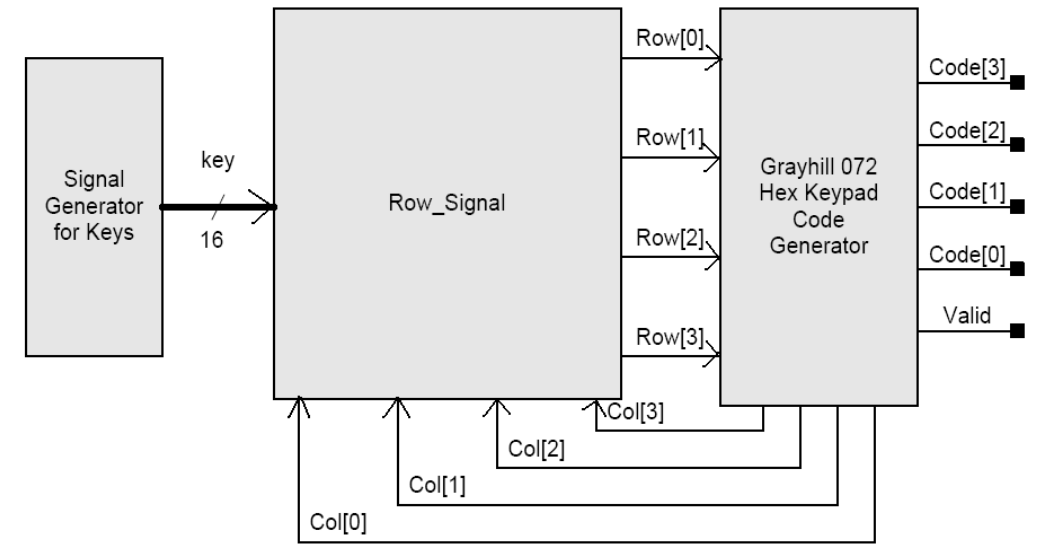


Figure 8: Sample Testbench

```

module Row_Signal (Row, Key, Col);
  output [3:0] Row;
  input [15:0] Key;
  input [3:0] Col;
  reg Row;

  // Scan for row of the asserted key

  always @ (Key or Col) begin //Asynchronous behavior for key assertion
    Row[0] = Key[0] && Col[0] || Key[1] && Col[1] || Key[2] && Col[2] || Key[3] && Col[3];
    Row[1] = Key[4] && Col[0] || Key[5] && Col[1] || Key[6] && Col[2] || Key[7] && Col[3];
    Row[2] = Key[8] && Col[0] || Key[9] && Col[1] || Key[10] && Col[2] || Key[11] &&
    Col[3];
    Row[3] = Key[12] && Col[0] || Key[13] && Col[1] || Key[14] && Col[2] || Key[15] &&
    Col[3];
  end
endmodule

```

Figure 9: Code for Row_Signal

Deliverables

1. I/O table for the Scanner/Encoder
2. State transition/output diagram for the Scanner/Encoder
3. Source code of Scanner/Encoder
4. Test plan
5. Source code of final self checking testbench
6. Demonstration of the operation of the test-bench to the lab instructor.

Grading

1. Demonstrations:

- a. Correct operation of self checking test-bench – 30pts
 - b. Correct operation of the Scanner/Encoder - 30 pts each
- 2. Report:
 - a. Completeness of test plan.– 30pts
 - b. Final report including test plan.– 30pts