CPE 324-01: Advanced Logic Design Laboratory

Lab04

Scanning Keypad Interface

Submitted by: Esther Shore

Date of Experiment(s): February 13, 2024 and February 15, 2024

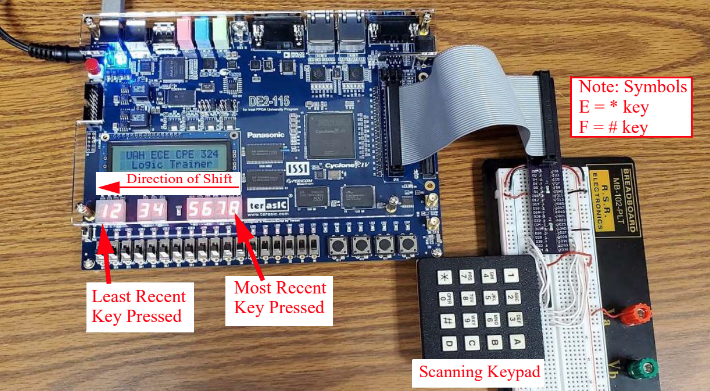
Report Deadline: February 20, 2024

1. Introduction

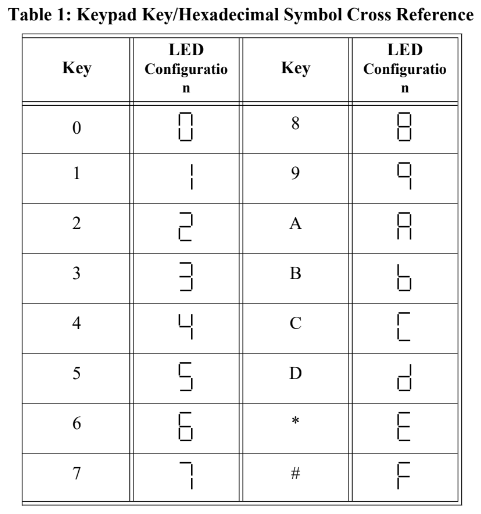
This laboratory experiment focuses on the intricate challenges of interfacing mechanical devices, like a keypad, with high-speed digital hardware. We address key issues including the key bounce problem and the implementation of time multiplexing to minimize I/O connections. Our objective is to design a sequential system capable of decoding inputs from a 16-key extended touchtone keypad and displaying the corresponding hexadecimal symbols on the Terasic DE2-115 Platform's seven-segment LED. Through this project, we aim to gain practical insights into digital system design while honing our skills in FPGA-based design methodologies.

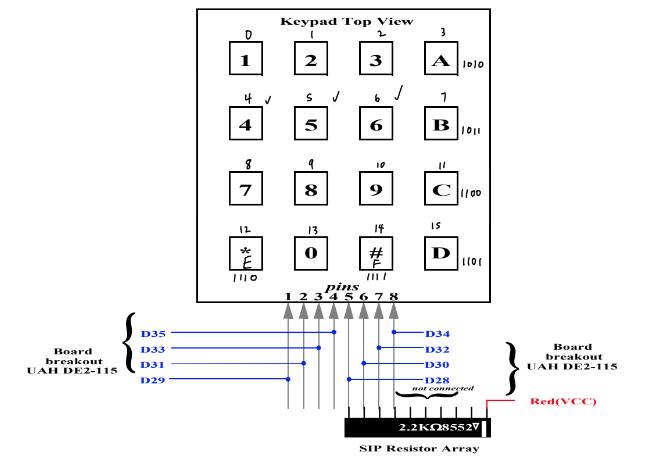
1. Experiment Description

The overarching design goal is displayed in Figure 1 below. The result should accept key presses on the scanning keypad and display the corresponding key pressed on the rightmost seven-segment display, shifting the prior stored values over to the left.



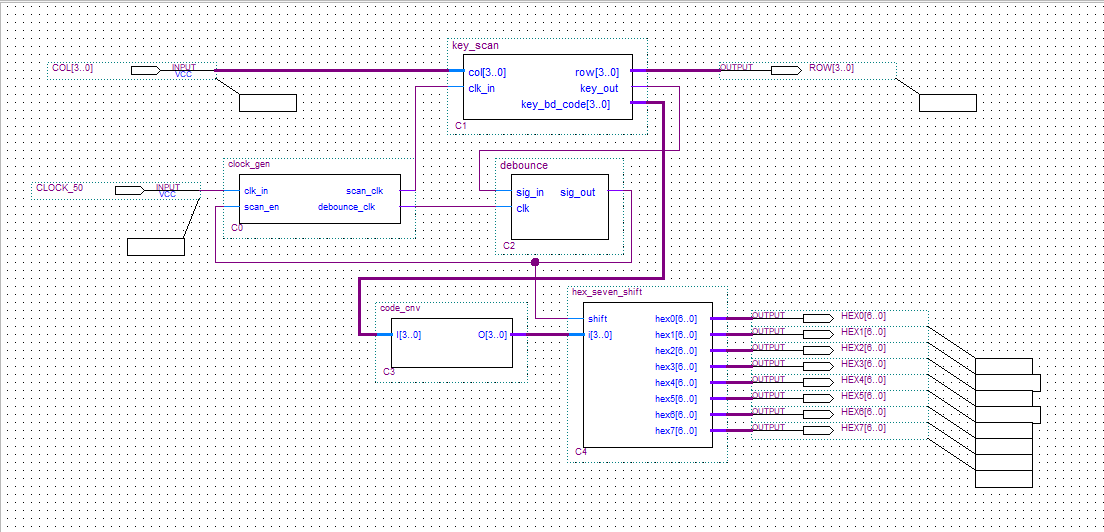
**Figure 1.** Design Implementation

In order to set up this design we had to understand the functionality of the scanning keypad organized with normally open switches into a two-dimensional matrix array as well as the external hardware configuration with the SIP resistor array as shown in Figure 2. A buzzed SIP resistor array contains a set of closely matched resistors that share a common internal connection point.



**Figure 2.** Keypad Wiring Diagram

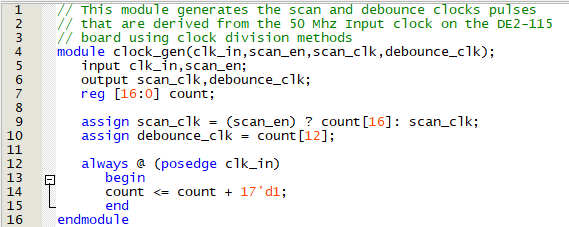
The top-level software design is shown in Figure 3, composed of five main modules implemented in either Verilog HDL or schematic capture. The clock\_gen and hex\_seven\_shift modules had already been created while the key\_scan, debounce, and code\_cnv modules had to be designed.



**Figure 3.** Top-Level View of the Scanning Keypad Design

2.1 Phase I: Creation of the key\_scan module

The creation of the key\_scan module, which relies on the clk\_in input forces us to examine the functionality of the clock\_gen module in Figure 4. It generated two clocks at different speeds, scan\_clk and debounce\_clk.



**Figure 4.** clock\_gen module

The key\_scan module must monitor the keypad’s four columns and drive the four rows accordingly. Upon detecting a key press, it outputs a unique 4-bit keyboard code and transitions key\_out from 1 to 0. The scanning process halts while the key is pressed due to the connection setup, where a low key\_out signal inhibits scan\_clk generation.

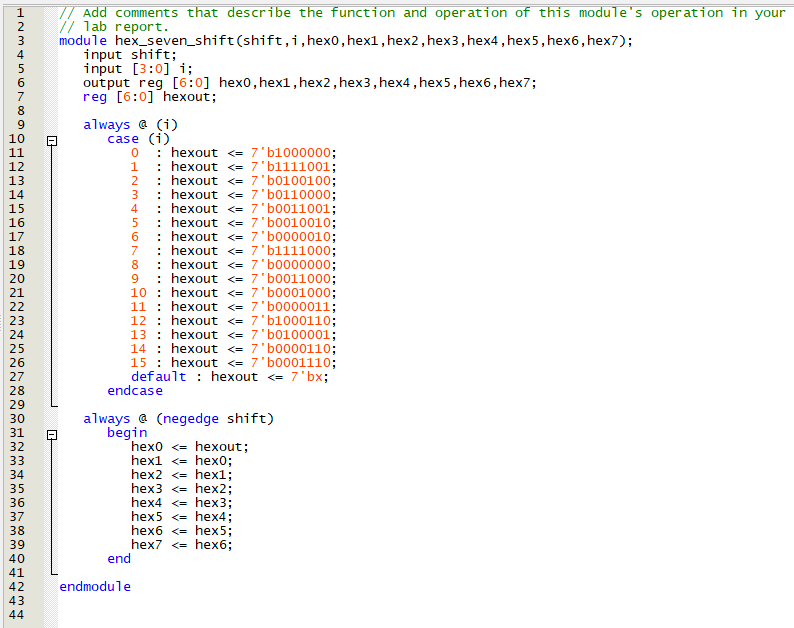
2.2 Phase II: Creation of the debounce module

The debounce module functions as a glitch filtering module to ensure clean transitions during switch transition periods. It connects sig\_in to sig\_out, enabling incremental validation of the overall design during FPGA prototyping.

2.3 Phase III: Creation of the code\_cnv module

The code\_cnv module serves as a 4-bit to 4-bit code conversion module, intended to translate the unique 4-bit keyboard code into the format required to display the proper seven-segment output, effectively translating from the key’s index value to the desired display value.

The resulting output from the code\_cnv module feeds the hex\_seven\_shift module as seen in Figure 5 below. This module captures the input 4-bit value and converts it into hexadecimal format suitable for display on the seven-segment display. It then displays this value on the least significant seven-segment port, shifting the previous seven-segment values one position to the left, preserving the last eight values entered.

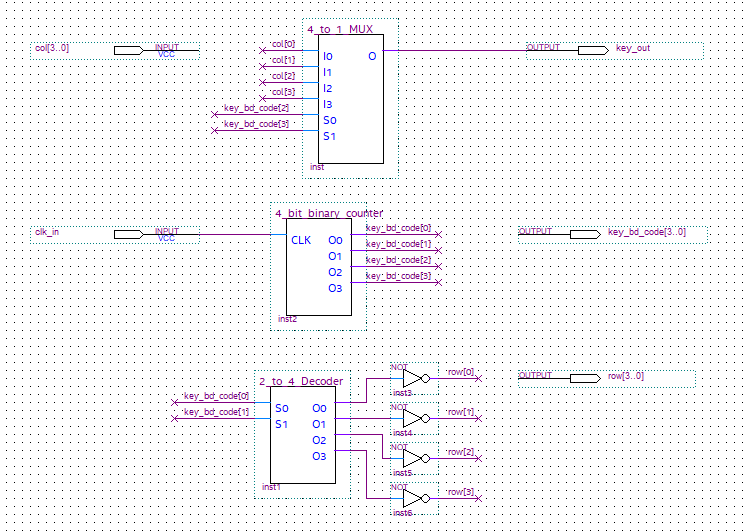


**Figure 5.** hex\_seven\_shift module

1. Demonstration

3.1 Phase I: Creation of the key\_scan module

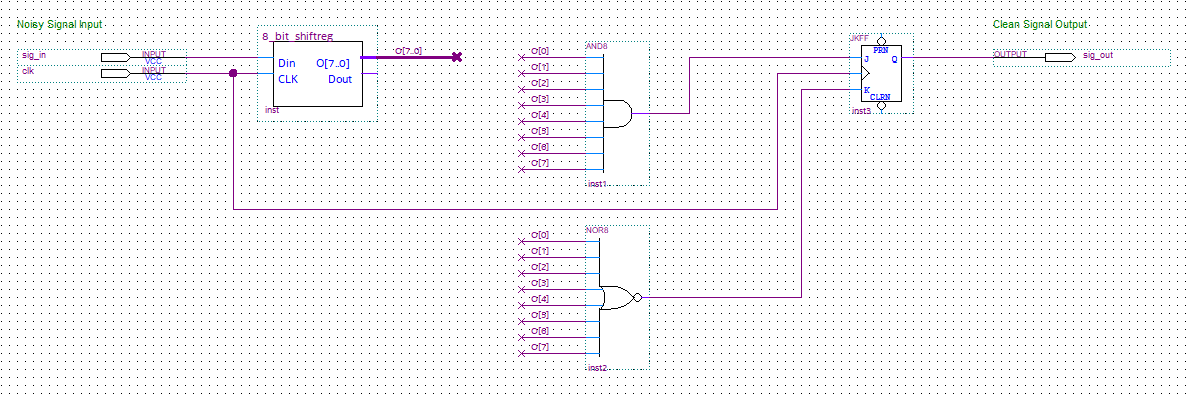
To create the key\_scan module using schematic capture design entry techniques, we utilized a 4-bit binary counter, a 4-to-1 multiplexer, a 2-to-4 decoder, and basic logic gates to generate a unique 4-bit code associated with the pressed key. The multiplexer determines if a key has been pressed by combining the column inputs. The counter outputs are used to drive the rate at which the columns are scanned and the output decoded. The resulting diagram is shown in Figure 6 below. The outputs from the decoder are inverted since logic high is inactive for the switches on the keypad.



**Figure 6.** key\_scan.bdf

3.2 Phase II: Creation of the debounce module

The debounce module as shown in Figure 7 is designed to filter out mechanical switch bounce, ensuring a single logic transition occurs when appropriate. It utilizes an 8-bit shift register to check the state for eight consecutive clock cycles. The bits are then AND-ed or NOR-ed together to check if they are all 1’s or all 0’s respectively, confirming the desired state. These outputs are then directed into a J-K flip-flop to set the output signal accordingly based on whether the eight clock cycles were all LOW or all HIGH.



**Figure 7.** debounce.bdf

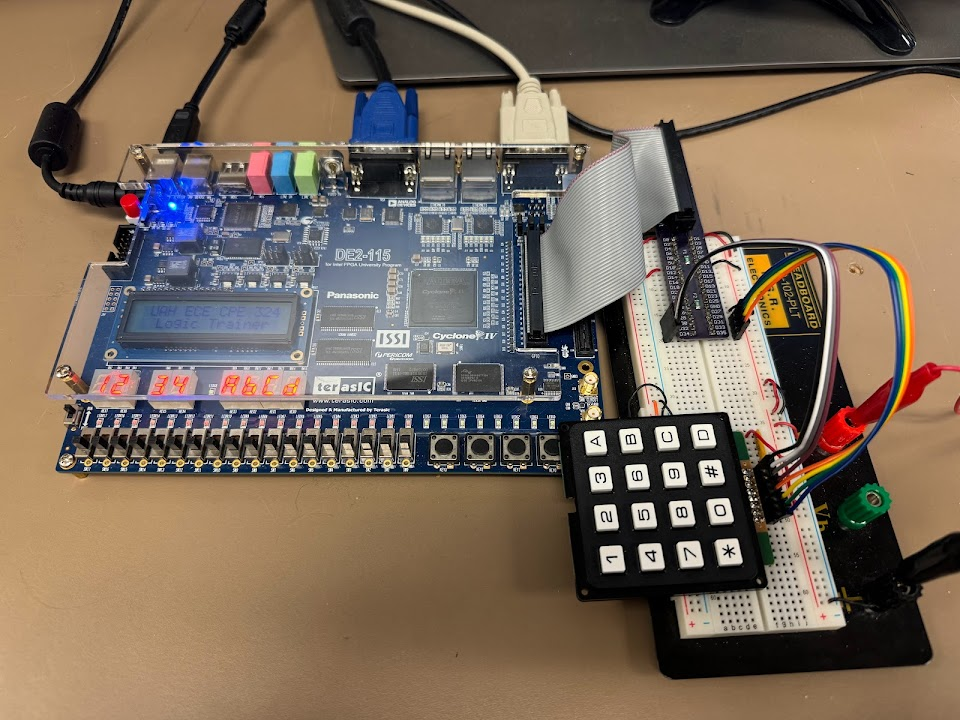
3.3 Phase III: Creation of the code\_cnv module

We created the code\_cnv implemented using Verilog HDL as shown in Figure 8. It uses an @always procedural construct with a simple case statement mapping inputs to outputs, ensuring that the correct output symbols are generated upon each key press.



**Figure 8.** code\_cnv.v

Our final project result with the seven-segment display showing the output “1234abcd” from the corresponding scanning keypad inputs is exhibited in Figure 9.



**Figure 9.** Scanning Keypad Interface Demonstration

1. Experimental Results

4.1 Phase I: Creation of the key\_scan module

The scan\_clk output is activated only when its enabling signal, scan\_en, is high, while debounce\_clk operates continuously, allowing separate clocking signals for key scanning and debounce logic, with the scan process paused during keypress events. The module effectively translated key presses into a unique code associated with each pressed key, although occasional incorrect values were permissible at this phase before implementing the debounce module.

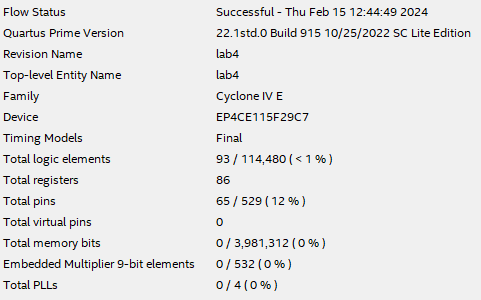
4.2 Phase II: Creation of the debounce module

Utilizing shift register, J-K flip-flop, AND, and NOR gates ensured that only a single logic transition occurred despite noisy input signals on key presses. Checking that inputs remained stable for eight clock cycles before any output transformation resulted in the debounce module successfully reducing mechanical switch bounce effects.

4.3 Phase III: Creation of the code\_cnv module

We observed the successful translation of input codes into the correct output symbols as specified in Table 1. The module accurately produced the desired output symbols corresponding to each key press.

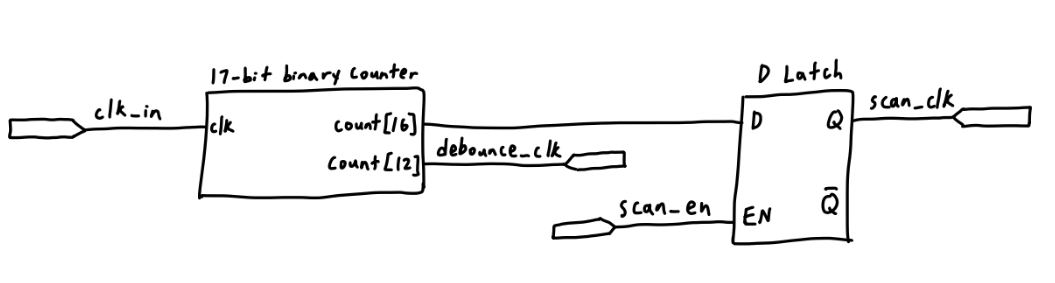
Following each phase, successful compilation was ensured with final results shown in Figure 10 below.



**Figure 10.** Compilation Output

Post-Lab Questions

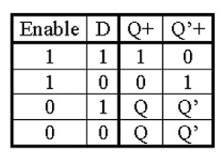
1. Schematic *clock\_gen* module using a single binary counter and a D latch is displayed in Figure 11 below.



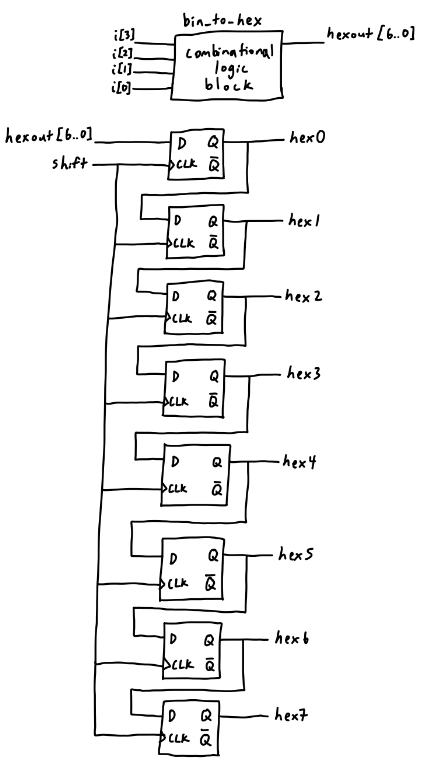
**Figure 11.** schematic clock\_gen module

This will utilize the specific bit of the output count to drive the input to the latch as well as the debounce clock signal. It will then combine the scan\_clk bit output with the scan\_en input using a D latch to generate the scan\_clk signal.

**Table 2.** D-Latch Logic



1. Given that the clk\_in input is a 50 MHz clock source, the frequency of the scan\_clk and debounce\_clk signals that are produced is determined by the corresponding bit of the output count. Since the scan\_clk output is dependent on the 17th bit, the frequency can be calculated as 50 MHz/2^17 = 381.47 Hz. It follows that the debounce clock frequency is 50 MHz/2^13 = 6,103.52 Hz since it is based on the 13th bit of count.
2. For the clock\_gen module, when the scan\_en signal is high, it sets scan\_clk to the value in the 17th bit of count, otherwise, the scan\_clk output remains unchanged. The scan\_en signal does not affect the debounce\_clk.
3. We needed a debounce circuit in this design in order to ensure proper detection of the switch presses from the keypad. In Phase I, when we did not have it, the noise from the switch press sometimes led to unexpected results, with the display showing differently than what we anticipated due to an inconsistent signal. The debounce module installs a delay, requiring the requested state to be active for a certain number of clock cycles before registering as a valid press, improving reliability of our design.
4. When choosing frequencies of clock\_gen module output clock signals, one should consider the desired speed of scanning and debounce clock. The scan clock determines the speed that we check if a key is pressed on the keypad. The debounce clock determines the duration for filtering a noisy signal to a clean signal. These values may not be appropriate for all mechanical push button normally open/momentarily closed devices because different systems have varying requirements based on the type of system and selected hardware.
5. In the Verilog HDL model for hex\_seven\_shift, based on the binary input code, the corresponding hexout value for the rightmost seven-segment display is assigned and the previous hex values displayed are shifted over. The purpose of the shift signal is to indicate when the hex displays are loaded. It is active while the key is pressed as detected by debounce and scan enabled, but the shift is not triggered until the falling edge once the key is released. Equivalent block diagram *hex\_seven\_shift* module using eight 7-input flip-flops and a single combinational block that converts from 4-bit binary to hexadecimal is drawn in Figure 12.



**Figure 12.** block diagram hex\_seven\_shift module

Each input D and output Q is 7 bits, where the appropriate individual bit is extracted at each stage from the D flip-flop output Q.

1. The code\_cnv module was implemented by checking each binary input case and assigning the output to the desired hex output in binary using a case statement. This module is a combinational design since it only depends on the current input.
2. Conclusion

Through successful completion of this scanning keypad interface laboratory experiment, we demonstrated our ability to interface a mechanical device with high-speed digital hardware. We learned to design and develop block diagram file and Verilog HDL modules through phased development and addressed key issues such as key bounce and signal conversion, improving the accuracy and reliability of our results. This underscored the importance of meticulous design methodologies and enhanced our FPGA-based development proficiency.