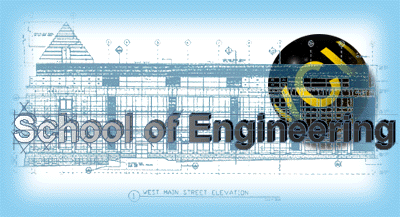
### EGRE 427 Lab/Homework No. 2

Title FPGA-based Hardware Design  
 Using VHDL

Name Chad Chapman

#### Date Performed 23-Dec-10

##### Date Submitted 25-Dec-10



I pledge this work to be my own \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Combination Lock FSM**

Upon inspection of the state transition diagram and the corresponding code we clearly show that this machine is a Mealy machine. The FSM gets it’s input from a signal named ‘btn\_pressed’ which is of type STD\_LOGIC\_VECTOR(3 DOWNTO 0). The input is directly received from the signal ‘sync\_out’ which resides inside of the architecture for Electronic\_Combination\_Lock. The input at each state is explicitly specified for each case and maps to an output within the process Output\_Function.

When we are inside of the finite state machine we go from one state to another until we reach the states labeled UNLOCKED and ERROR. If we are in either of these two states we prevent the FSM from going to the next desired state by using a signal called ‘enable’. The conditions are described by the following figures:

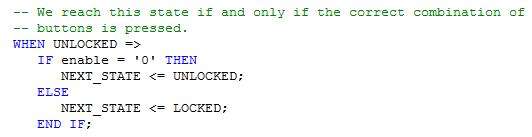


Figure 1 – Logic for UNLOCKED.

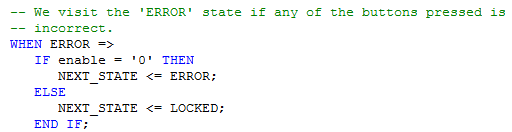


Figure 2 – Logic for ERROR.

As shown in Figure 1, when enable is set to ‘0’ we want to stay in the state called UNLOCKED and the same idea apply as well for Figure 2. While we are at these two states the finite state machine that produces the required output to the LED’s is performing some kind of operation. Although not explicitly stated in the VHDL file, when enable is set to ‘1’ we want the FSM to go to the next desired state. We have labeled the transitions from UNLOCKED to LOCKED and from ERROR to LOCKED to show that this is the behavior we expect. More details concerning the signal called enable will be discussed in the section for LED\_CONROL\_FSM.

The output for this machine will be string “01” when the FSM is still accepting inputs from the buttons on the FPGA. When the output string is “10” this information is provided to the FSM that controls the output for the LED’s when the correct sequence of buttons is pressed by the user. If the output string is “11” we have reached the ERROR state and the FSM that controls the LED’s will provide the required output.

**LED\_Control\_FSM**

The LED\_Control\_FSM is a bit more complex than the CombinationLockFSM. Now we are using a counter and status type to help model the behavior of the FSM. Figures 3 and 4 shows this how it is written in the .VHD file.



Figure 3 – TYPE status\_type.

The TYPE called ‘status\_type keeps track of the state of the CombinationLockFSM and it’s current\_status is determined by the signal led\_select. This allows us to choose the next path to take in the LED\_Control\_FSM without having to use any nested ‘if then’ statements and it helps us determine the value enable\_out is going to assume with respect to status. By choosing the next value enable\_out is going to take, status\_type plays a role in the behavior of the CombinationLockFSM when it is at the states LOCKED, UNLOCKED, and ERROR.



Figure 4 – SUBTYPE counter.

The SUBTYPE counter is used to keep track of the time since the status changed from LOCKED to UNLOCKED and from LOCKED to INCORRECT. This allows us to set or clear the enable\_out bit when ‘present\_status’ is not set to LOCKED since we no longer care what values led\_select holds. The logic for ‘status\_type’ UNLOCKED and INCORECT compares the current value held by counter to determine when ‘present\_status’ needs to be assigned a new value. The logic used is shown in the figures below.

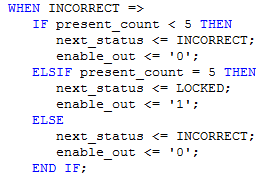
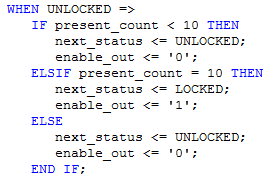


Figure 5 – Logic for UNLOCKED Figure 6 – Logic for INCORRECT

In Figure 5, we test to see when ‘present\_count’ is less than 10. We repeat this comparison every second until . Once this happens we change the ‘enable\_out’ bit to’ 1’ so that the CombinationLockFSM may continue to accept input signals originating from the buttons. If the ‘enable\_out’ bit remains at ‘0’ and the LED\_Control\_FSM continues to generate output for the LED’s. Similar combinational logic is used for when   
. Instead in this case we increment count until . All of this information from the TYPE status\_type is used in state to determine which path to take in the ‘Transition\_Function.’

VHDL and UCF files

The VHDL files that are written or modified and the UCF file are included in this section. The UCF file used in our design is shown below in Figure 7.

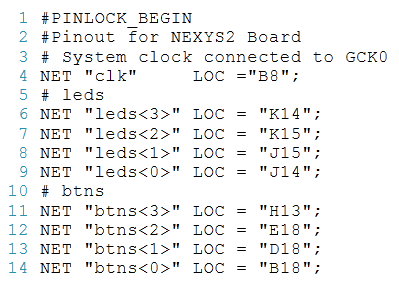


Figure 7 – CombinationLockPins.ucf

The .vhd files attached after this page are the following in the order shown.

1. Electronic\_Combination\_Lock.vhd
2. LEDTimer.vhd
3. CombinationLockFSM.vhd
4. LED\_Control\_FSM.vhd

The only thing we need to discuss about LEDTimer.vhd is that the constant used in the generic map is different from the one in clock\_divider.vhd in order to create a time pulse signal to be used by the LED\_Control\_FSM in order to time the outputs correctly.