Home Alarm Control

EE/CS 499/599 Verilog Design Project

Table of Contents

Introduction 3
Design Specifications 4
  Overview 4
  Arming the system 5
  Alarm 5
  Disarming the system 6
  Status LEDs 6
  Sensors 6
  System output 6
  Password 6
  Clock and Reset 7
Alarm System Block Diagram 8
Lab Summary 9
Getting Started 10
Lab 1 – Introduction 11
Lab 2 – Motion Sensor Decode 13
Lab 3 – Window Sensor Decode 15
Lab 4 – Keypad Encode 17
Lab 5 – Keypad Pulse Generator 19
Lab 6 – Half and Full Second Pulse Generator 21
Lab 7 – Alarm and Armed LED, and Beep Control 23
Lab 8 – Timer 25
Lab 9 – Password Check 27
Lab 10 – Finite State Machine 29
Lab 11 – Bringing it all together 32
Lab 12 – Design Enhancement 34

Introduction

The lab exercises are designed to further help you familiarize yourself with the Verilog language at various levels of abstraction, including: RTL code and behavioral test benches. In this lab, you will design, synthesize, and test a control circuit for a home alarm system.

Design Specifications

Overview

The home alarm system consists of 10 window sensors, 4 door sensors, 4 motion sensors, a 12-key keypad with status LEDs (see figure 1), a telephone dialer (to the home security provider), piezoelectric beeper (built in), and an external siren.

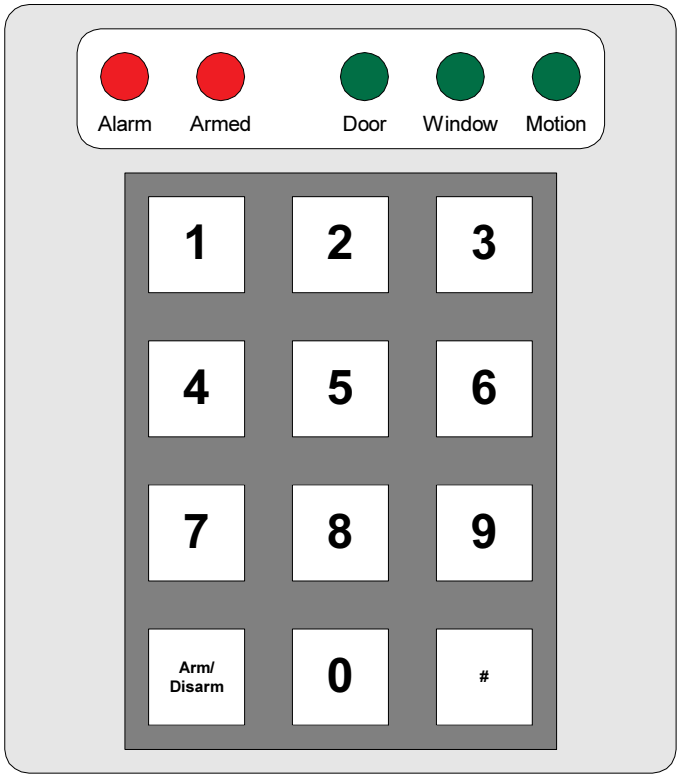


Figure 1. Home alarm system keypad and status LEDs

This design is for the central control circuit (see figure 2) of the home alarm system. The circuit will receive the window, door, and motion sensors, and the 12-key keypad as inputs, and will provide control signals for the status LEDs, the telephone dialer, the beeper, and the audible alarm.

The circuit will control all aspects of the alarm system. The system allows the user to arm and disarm the system. The system will give the user status of the armed state of the system as well as the status of the sensors.

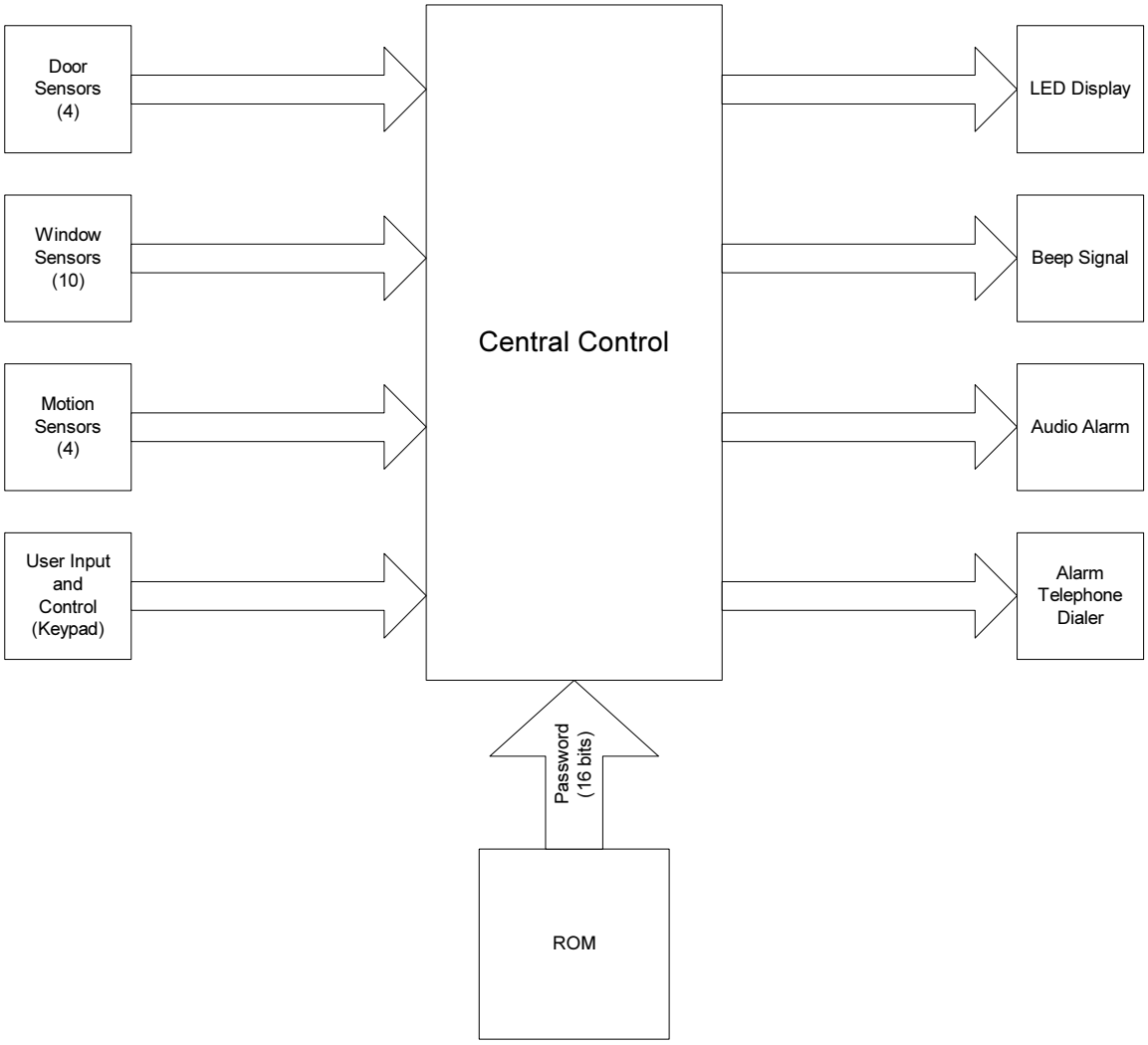


Figure 2. Central control circuit block diagram.

Arming the system

When the system is idle (disarmed), the user will press Arm/Disarm button to arm the system. After the Arm/Disarm button is pressed, a 4-digit password will be required. Once the password is verified, there will be a 10 second delay until the system is armed. During the 10-second delay, the armed LED will flash and the beeper will pulse. The system will stay in the armed state until the Arm/Disarm button is pressed or one of the sensors is activated.

Alarm

The system will go into a pre-alarm state 10 seconds after the keypad is pressed or one of the sensors is activated. The 10-second buffer allows the user time to disarm the system

before the alarm activity begins. If the system is not disarmed within 10 seconds, the system will go into full alarm activity.

Disarming the system

To disarm the system, the user must press the Arm/Disarm button followed by the 4-digit password. The system will return to an idle state once the password is verified. If the system is armed, the system will go in to an alarm state after 10 seconds if the correct password is not entered.

Status LEDs

The circuit will control the system status LEDs. The window, door, and motion LEDs will illuminate whenever the sensor is activated regardless of the state of the system. The Armed LED will blink during the arm countdown, and will fully illuminate when the system is armed. The Alarm LED will blink during the disarm countdown, and will fully illuminate if the system enters the full alarm state.

Piezoelectric Beeper

The Beeper will pulse during the arm and disarm countdown. The Beeper will be on when the system is in the alarm state.

Sensors

The door, window, and motion sensors will provide the system with a logic ‘0’ when they are inactive, and a logic ‘1’ when they are active.

System output

The circuit will provide a logic ‘1’ on the Audio Alarm and Telephone Dialer output pins when the system is in the alarm state. Otherwise, these pins will maintain a logic ‘0’.

Password

For design simplicity, the system will store the password information in an external 1x16 ROM, and the password will be set to a hex value of 1234.

Clock and Reset

The system clock input will connect to a 100 Hz oscillator, and reset will be active high. The system will be reset during installation only. Users will not be able to reset the system.

Alarm System Block Diagram

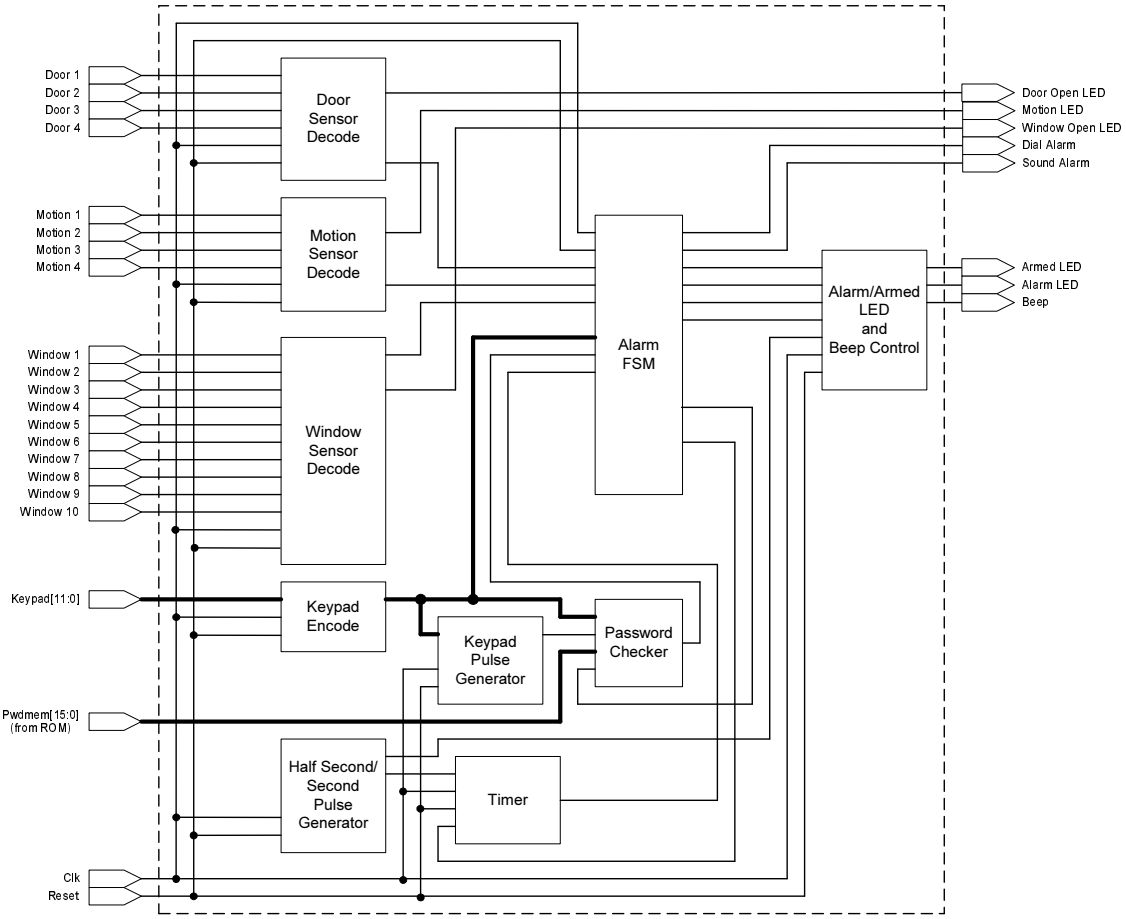


Figure 3. Alarm System block diagram.

Lab Summary

Lab 1: Introduction – Examine Verilog RTL Code (part of this design), and introduce yourself to the Simulator and Synthesizer.

Lab 2: Motion Decode – Write your first Verilog code. Create RTL code for a 4 input OR gate with a registered output.

Lab 3: Window Decode – Write RTL code for a 10 input OR gate with a registered output. You'll take a different approach from lab 2 and the example, and you'll have to add stimulus to the testbench.

Lab 4: Keypad Encode – Take the raw 12-bit data from the keypad and encode it into a 4-bit decimal representation.

Lab 5: Keypad Pulse Generator – Take the output from your creation in lab 4 and generate a pulse whenever a key on the keypad is pressed.

Lab 6: Half-Second and Second Pulse Generator – Using the system clock create pulses at every half-second and second.

Lab 7: Alarm LED, Armed LED, and Beep Control – Create a control block to enable LED flashing and beep pulsing.

Lab 8: Timer – Create a 10 second countdown timer.

Lab 9: Password Check – Take the password information from the ROM and compare it with the password entered on the keypad.

Lab 10: Alarm Finite State Machine – The brains of the design. Create a 8 state FSM to control the system.

Lab 11: Bringing it all together – Bring all of the design pieces together to create the top-level design. Write system tasks to test the design.

Lab 12: Design Enhancement (Grad Students) – Add password storage capability to the design.

Getting Started

Important information before you begin the lab exercises

1. It is imperative that you use the specified module names, port names, and file names as outlined in this lab. Failure to do so will result in a loss of points.

2. The Verilog code for each block in the design will be named after the design block. For example, the Door Decode block will be called doordecode.v. The testbench for each design block will be named the same as the design block with a "_tb" suffix. For example, the testbench for the Door Decode Verilog code will be called doordecode_tb.v.

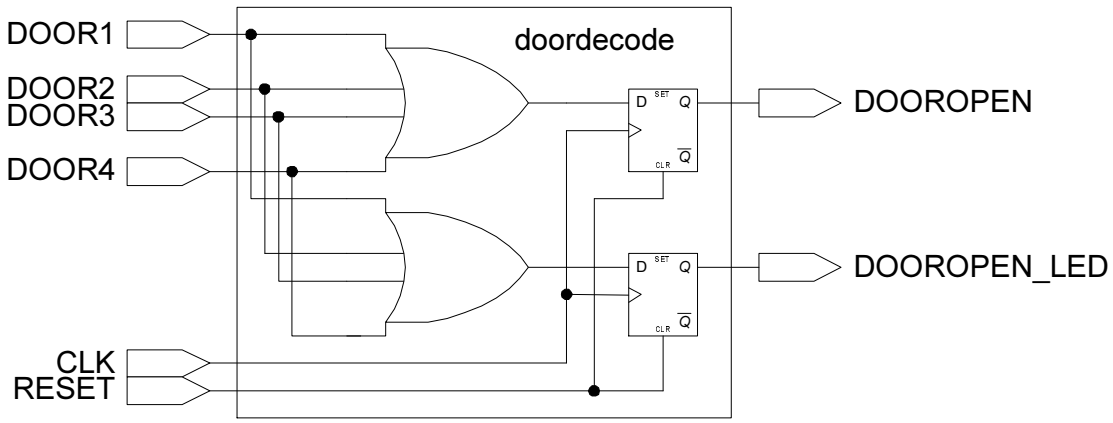
3. Design Synthesis:
    a. If you are using the VLSI/Cadence Lab, you'll synthesize the design into the 0.35u AMIS process, using the AMI350LXSC 3.3 Volt libraries. The Cadence synthesis tool is called Ambit Build Gates. The libraries and the synthesis run scripts will be provided.

    b. If you're using the Xilinx ISE Software. You will synthesize to a CPLD, using the XST tool built-in to the Xilinx ISE software.

Lab 1 – Introduction

Goal - This lab will introduce you to Verilog RTL code, the Verilog simulator, and the Synthesis tool.

Background - This design is from the door sensor decode. It is a simple 4-input OR gate with registered outputs. When a door sensor is activated (door open), the DOOROPEN and DOOROPEN_LED signals will be a logic '1' after the next positive clock edge.



Do: Create Verilog Code –

1) Create the design in the diagram using a combination of dataflow modeling and behavioral modeling.
2) Name the design file doordecode.v.
3) Use the port names shown above and call the module doordecode.
4) Create a Verilog testbench to thoroughly test the design, name the testbench file doordecode_tb.v.

Do: Run a simulation and view the waveforms –

1) Simulate the design using the ModelSim or Cadence simulator.
2) Look over the waveforms and make sure that the simulation results meet your expectations.

Do: Synthesize the RTL code –

1) Synthesize the RTL code (doordecode.v), using either Ambit or XST. Make sure the RTL code synthesizes without any errors.

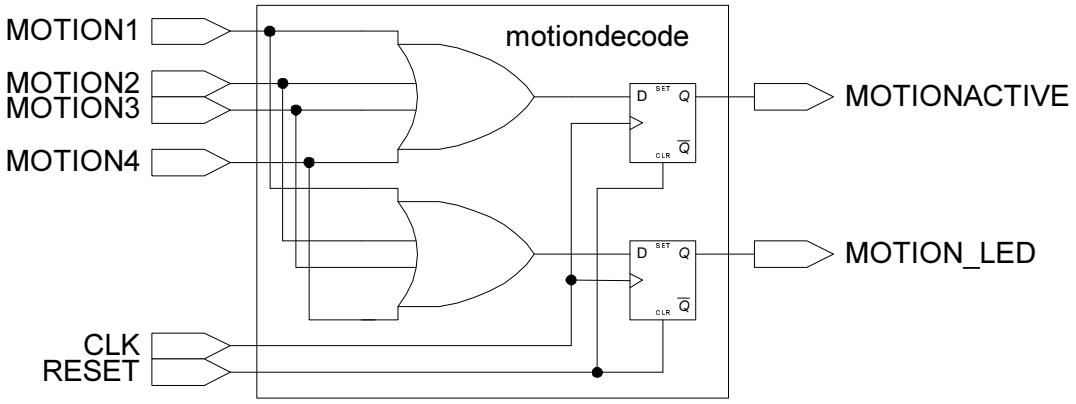
Submit:

1) The doordecode.v file.
2) The doordecode_tb.v file.
3) The log file from the synthesis.
4) The log file from the simulation.

Lab 2 – Motion Sensor Decode

Goal – In this lab, you'll write Verilog RTL code that will take 4 inputs from the motion sensors and give two outputs: MOTIONACTIVE, and MOTION_LED.

Background - This block is essentially the same as the 'door sensor decode' design. It is a simple 4-input OR gate with registered outputs. When a motion sensor is activated, MOTIONACTIVE and MOTION_LED signals will be high after the next positive clock edge.



Do: Write the Verilog Code –

1) Create the design in the diagram, name the design file motiondecode.v.
2) Use the port names shown above and call the module motiondecode.
3) Create a Verilog testbench to thoroughly test the design, name the testbench file motiondecode_tb.v.
4) Instead of using dataflow modeling as in lab 1, use two procedural blocks to create your design **(points will be deducted if you use anything other than two procedural blocks!).**

Do: Run a simulation and view the waveforms –

1) Simulate the design.

Do: Synthesize the RTL code –

1) Synthesize the RTL code (motiondecode.v).

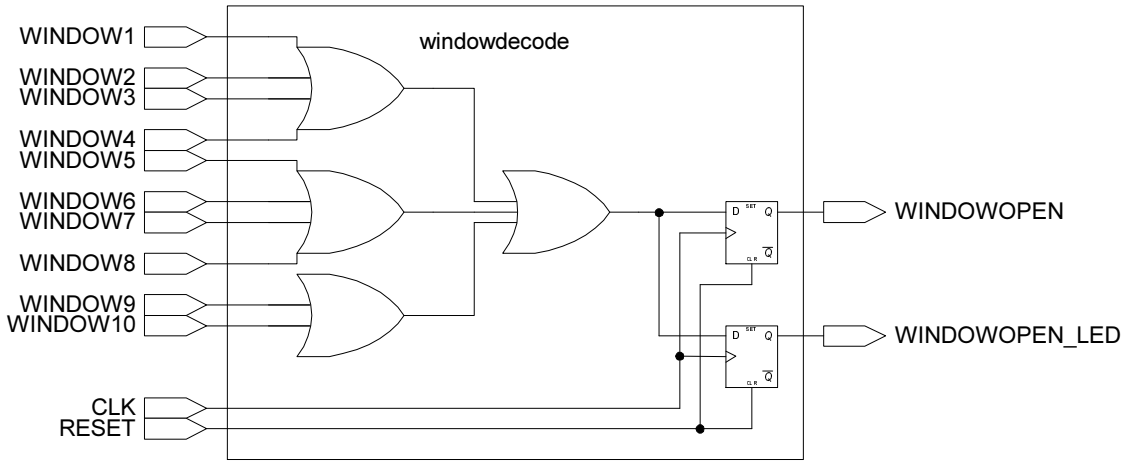
Submit:

1) The doordecode.v file.
2) The doordecode_tb.v file.
3) The log file from the synthesis.
4) The log file from the simulation.

Lab 3 – Window Sensor Decode

Goal – In this lab you'll Verilog RTL code that will take 10 inputs from the window sensors and give two outputs: WINDOWOPEN, and WINDOWOPEN_LED.

Background - This block is similar to the 'door sensor decode', and the 'motion sensor decode' designs. The design is a 10-input logic OR. When a motion sensor is activated, WINDOWOPEN and WINDOWOPEN_LED signals will be high after the next positive clock edge.



Do: Write the Verilog Code –

1) Create the design in the diagram, name the design file windowdecode.v.
2) Use the port names shown above and call the module windowdecode.
3) Create a Verilog testbench to thoroughly test the design, name the testbench file windowdecode_tb.v.
4) Use one procedural block to create your design **(points will be deducted if you use anything other than one procedural block!).**

Do: Add stimulus, run the simulation, view the waveforms –

1) Simulate the design.

Do: Synthesize the RTL code –

1) Synthesize the RTL code (windowdecode.v).

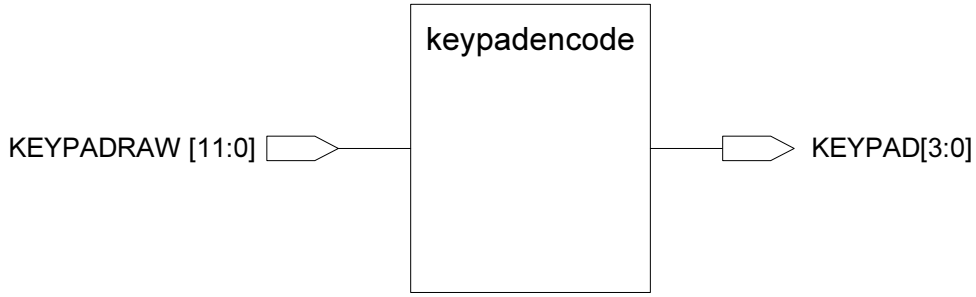
Submit:

1) The windowdecode.v file.
2) The windowdecode_tb.v file.
3) The log file from the synthesis.
4) The log file from the simulation.

Lab 4 – Keypad Encode

Goal – In this lab you'll write and verify Verilog RTL code that will take raw 12-bit data from the keypad and encode it into a 4-bit decimal representation.

Background - This block is a simple encoder that takes 12 inputs from the 12-key keypad, and gives as an output a binary-coded-decimal (BCD) value from 0 to 12. When one key is pressed the associated bit on KEYPADRAW will become a logic '1'. For example when the number 7 is pressed on the keypad, KEYPADRAW[7] will be a logic '1', and the KEYPAD should become '0111' (binary value 7). When more than one key is pressed at a time, or no keys are pressed, the default BCD is 13 (binary 1101). The arm/disarm key is encoded as 10, and the unused key (#) is encoded as 11 (reserved for possible future use).



Do: Write the Verilog Code –

1) Create the design described above, name the design file keypadencode.v.
2) Use the port names shown above and call the module keypadencode.
3) Create a Verilog testbench to thoroughly test the design, name the testbench file keypadencode_tb.v.
4) Use any legal RTL constructs to implement the design.

Do: Run a simulation and view the waveforms –

1) Simulate the design.

Do: Synthesize the RTL code –

1) Synthesize the RTL code (keypadencode.v).

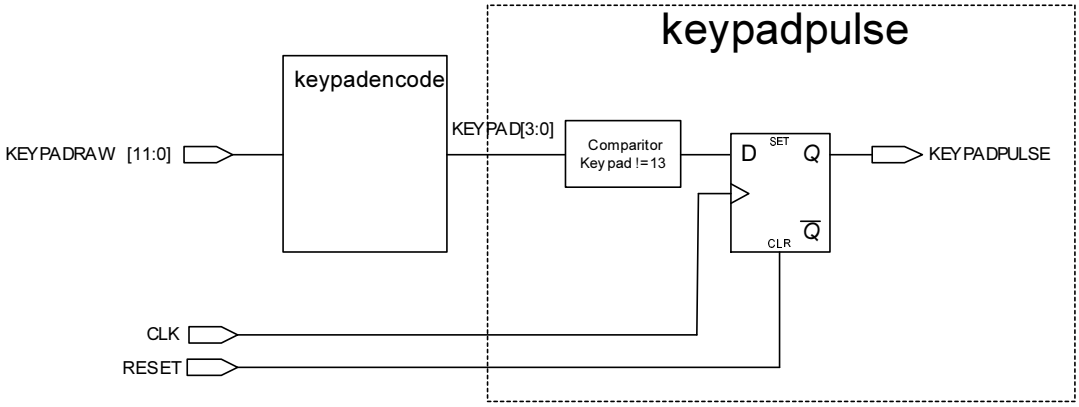
Submit:

1) The keypadencode.v file.
2) The keypadencode_tb.v file.
3) The log file from the synthesis.
4) The log file from the simulation.

Lab 5 – Keypad Pulse Generator

Goal – Generate a one-clock-cycle pulse whenever a key on the keypad is pressed.

Background – If you recall from Lab 4, whenever a key on the keypad is pressed, a BCD value between 0 and 12 is output on KEYPAD. When the keypad is idle, the KEYPAD output has a value of 13. This design takes the KEYPAD output from the "keypadencode" block as an input. The "keypadpulse" block checks the value of KEYPAD. If KEYPAD is != 13 it feeds a logic '1' to the output flop. When KEYPAD = 13, it feeds a logic '0' to the output flop.



Do: Write the Verilog Code –

1) Create the design described above, name the design file keypadpulse.v.
2) Use the port names shown above (inside the keypadpulse dotted box) and call the module keypadpulse.
3) Create a Verilog testbench to thoroughly test the design, name the testbench file keypadpulse_tb.v.
4) Use any legal RTL constructs to implement the design.

Do: Run a simulation and view the waveforms –

1) Simulate the design.

Do: Synthesize the RTL code –

1) Synthesize the RTL code (keypadpulse.v).

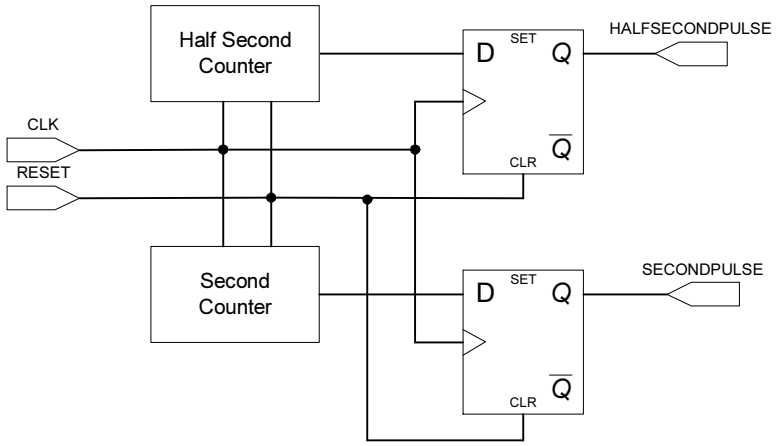
Submit:

1) The keypadpulse.v file.
2) The keypadpulse_tb.v file.
3) The log file from the synthesis.
4) The log file from the simulation.

Lab 6 – Half and Full Second Pulse Generator

Goal – Using the system clock create output pulses at every half-second and second.

Background – The system clock is a 100 Hz oscillator. Count the number of clock edges and provide a pulse every half and full second.



Do: Write the Verilog Code –

1) Create the design described above, name the design file secondpulse.v.
2) Use the port names shown above and call the module secondpulse.
3) Create a Verilog testbench to thoroughly test the design, name the testbench file secondpulse_tb.v.
4) Use any legal RTL constructs to implement the design.

Do: Run a simulation and view the waveforms –

1) Simulate the design. Verify that your pulses occur exactly every ½ second and every second. **Note** – It is only critical that each pulse occurs ½ second or second from the previous pulse.

Do: Synthesize the RTL code –

1) Synthesize the RTL code (secondpulse.v).

Submit:

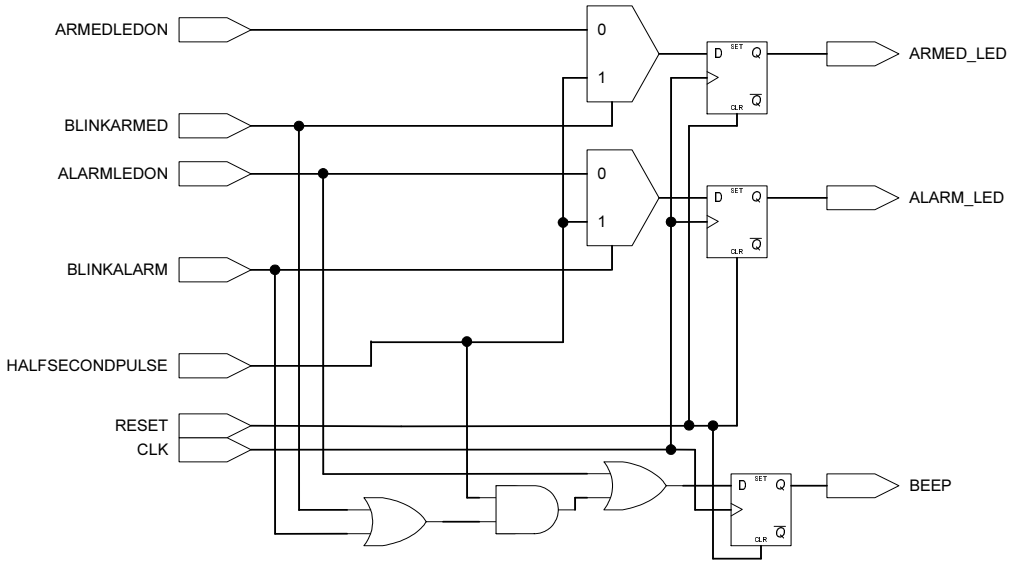
1) The secondpulse.v file.
2) The secondpulse_tb.v file.
3) The log file from the synthesis.
4) The log file from the simulation.

Lab 7 – Alarm and Armed LED, and Beep Control

Goal – Create a control block to enable LED flashing and beeper pulses.

Background – In lab 10 (coming up), you will create a Finite State Machine. The FSM provides the following outputs: ALARMLEDON, ARMEDLEDON, BLINKALARM, and BLINKARMED. This block will use these outputs from the FSM as inputs (as well as the HALFSECONDPULSE output that you created in lab 6) to create the following outputs: ALARM_LED, ARMED_LED, and BEEP. The block will have the following functionality:

1) ARMED_LED will equal HALFSECONDPULSE if ARMEDLEDON and BLINKARMED are both a logic '1'. Otherwise, ARMED_LED will equal ARMEDLEDON.
2) ALARM_LED will equal HALFSECONDPULSE if ALARMLEDON and BLINKALARM are both a logic '1'. Otherwise, ALARM_LED will equal ALARMLEDON.
3) BEEP will equal HALFSECONDPULSE if BLINKALARM or BLINKARMED is a logic '1' and ALARMLEDON is a logic '0'. Otherwise, BEEP will equal a logic '1' when ALARMLEDON is a logic '1'.
4) All outputs will be registered.



Do: Write the Verilog Code –

1) Create the design described above, name the design file ledblink.v.
2) Use the port names shown above and call the module ledblink.
3) Create a Verilog testbench to thoroughly test the design, name the testbench file ledblink_tb.v.
4) Use any legal RTL constructs to implement the design.
5) **Caution** – Watch your port names, they are similar and it is easy to mix them up.

Do: Run a simulation and view the waveforms –

1) Adequate testing of this design block will require the use of the 'secondpulse' block that you created in lab 6.
2) Simulate the design.

Do: Synthesize the RTL code –

1) Synthesize the RTL code (the "ledblink.v" design only).

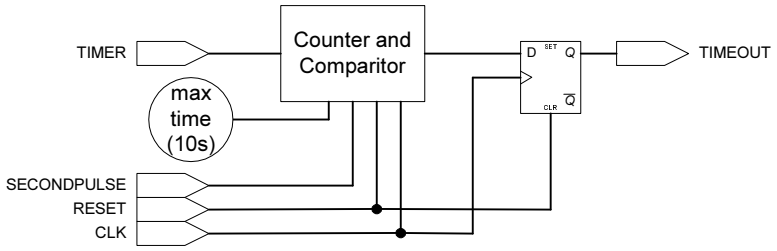
Submit:

1) The ledblink.v file.
2) The ledblink_tb.v file.
3) The log file from the synthesis.
4) The log file from the simulation.

Lab 8 – Timer

Goal – Create a 10 second countdown timer.

Background – As mentioned, you will create a Finite State Machine in lab 10. The Timer block will use an output from the FSM called TIMER, as well as the SECONDPULSE output from the "secondpulse" block. The Timer block will provide a 10 second countdown for the FSM. This is the time the user has to arm or disarm the system. Although the FSM sees this as a countdown timer, we'll actually be building a counter that counts 10 SECONDPULSEs from the "secondpulse" block. Whenever the FSM outputs a logic '1' on the TIMER signal, the Timer block will begin to count to 10. Whenever the RESET signal is a logic '1', or the TIMER signal is a logic '0', the counter will return to zero and wait for the TIMER signal to go to a logic '1' state. When the counter reaches 10, it will output a logic '1' on TIMEOUT for one clock cycle. The 10-second limit will be coded as a parameter.



Do: Write the Verilog Code –

1) Create the design described above, name the design file timer.v.
2) Use the port names shown above and call the module timer.
3) Create a Verilog testbench to thoroughly test the design, name the testbench file timer_tb.v.
4) Use any legal RTL constructs to implement the design.

Do: Run a simulation and view the waveforms –

1) As in lab 7, adequate testing of this design block will require the use of the "secondpulse" block that you created in lab 6.
2) Simulate the design.

Do: Synthesize the RTL code –

1) Synthesize the RTL code ("timer.v").

Submit:

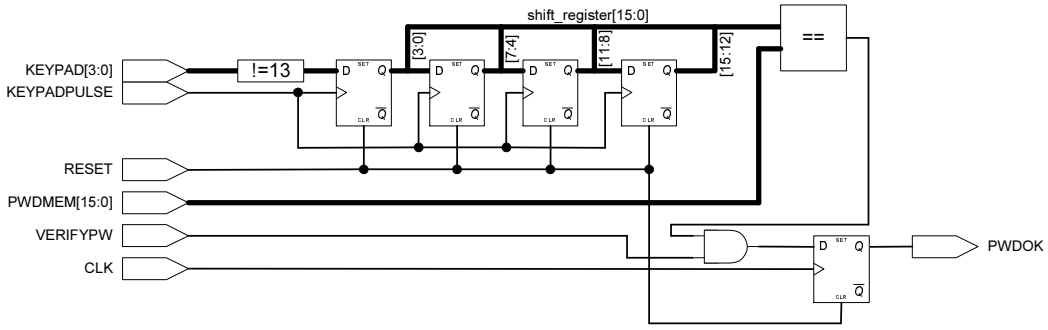
1) The timer.v file.
2) The timer_tb.v file.
3) The log file from the synthesis.
4) The log file from the simulation.

Lab 9 – Password Check

Goal – Take the password information from the ROM and compare it with the password entered on the keypad.

Background – In reality, a password for a home alarm system would not be very secure if it is stored in a ROM at the manufacturer. But, for now we will use this as our design method (Grad students will have a chance to change this later). We'll assume the ROM has a stored password value of 1-2-3-4 (each password integer represents 4 bits of a 16-bit ROM. The ROM binary value = 0001001000110100). Use a parameter to code this value into your testbench.

The block will store the KEYPAD 4-bit BCD output from the keypadencode block. The value will be stored (shifted) into the register when the KEYPADPULSE signal from the keypadpulse block has a rising edge, and KEYPAD is not equal to 13. The block will continuously compare the value of the shift register to the value stored in the ROM. When the ROM value matches the value stored in the shift register, and the VERIFYPW signal from the FSM is a logic '1', the block will output a logic '1' to PWDOK port for one clock cycle.



Do: Write the Verilog Code –

1) Create the design described above, name the design file passwordchk.v.
2) Use the port names shown above and call the module timer.
3) Create a Verilog testbench to thoroughly test the design, name the testbench file passwordchk_tb.v.
4) Use any legal RTL constructs to implement the design.

Do: Run a simulation and view the waveforms –

1) Testing of this design block will require the use of the "keypadencode" and "keypadpulse" blocks that you created in previous lab exercises.
2) Simulate the design.

Do: Synthesize the RTL code –

1) Synthesize the RTL code (passwordchk.v).

Submit:

5) The passwordchk.v file.
6) The passwordchk_tb.v file.
7) The log file from the synthesis.
8) The log file from the simulation.

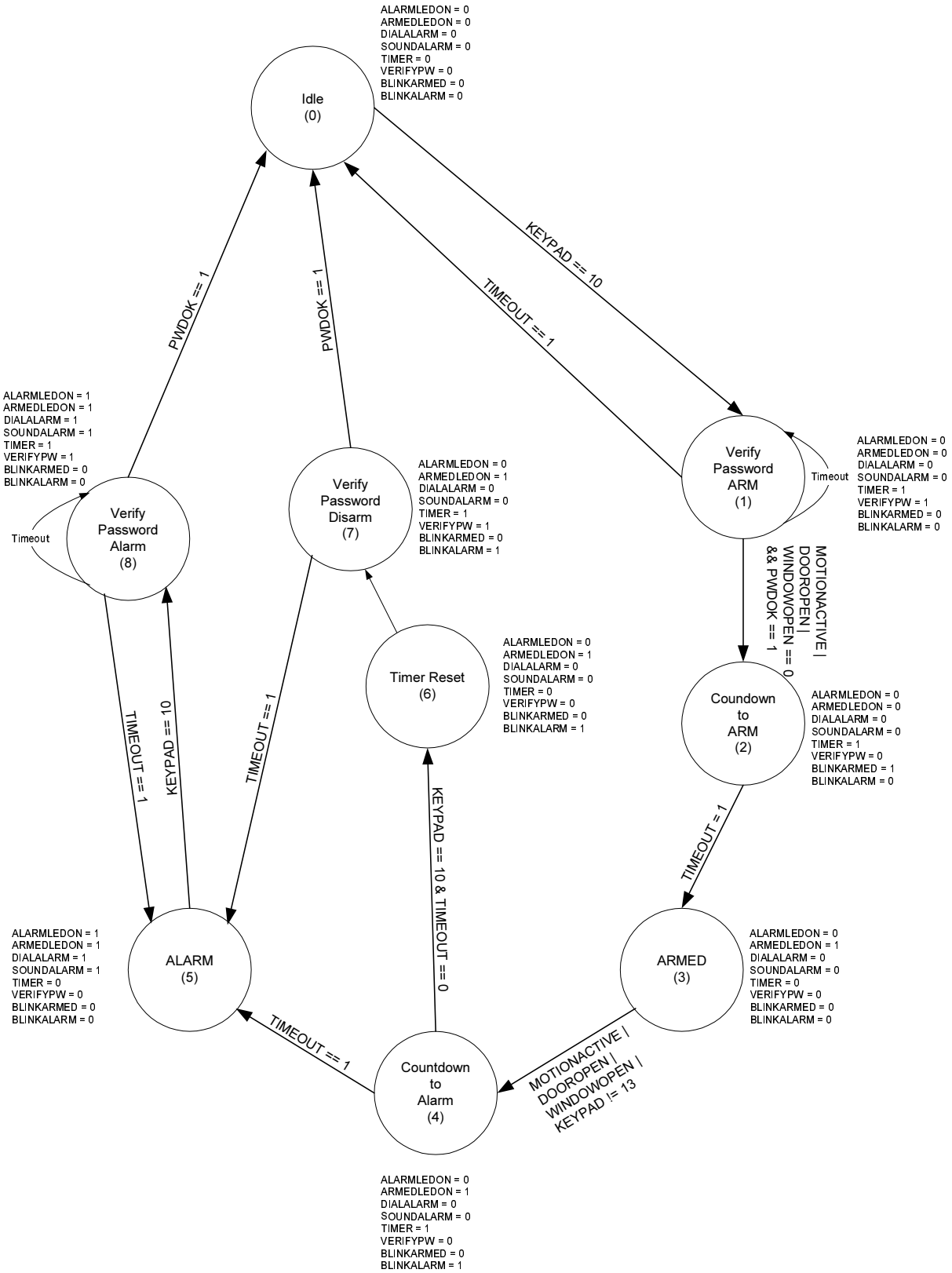
Lab 10 – Finite State Machine

Goal – Create an 8-state Finite State Machine (FSM) that will control the alarm system.

Background – The FSM controls all of the functions of the alarm system. It has the following states:

0. IDLE
1. VERIFY_PW_ARM
2. COUNTDOWN_ARM
3. ARMED
4. COUNTDOWN_ALARM
5. ALARM
6. TIMER_RESET
7. VERIFY_PW_DISARM
8. VERIFY_PW_ALARM

The state machine will be dependent on both inputs and the current state to determine the next state of the machine. The outputs of the state machine are dependent on the current state of the FSM only (i.e. Moore FSM). The functionality of the block is defined in the state diagram on the next page.



Do: Write the Verilog RTL Code –

1) Create the FSM using module name alarmfsm and file name alarmfsm.v file.
2) Use the correct methods for Verilog FSM RTL Code..

Do: Syntax Check –

1) Since you are going to bring all of the design blocks together in the next exercise, test the FSM in the next lab exercise. However, it may be a good idea to check your syntax and make sure it compiles correctly with the Verilog simulator.

Do: Synthesize the RTL code –

1) Synthesize the RTL code (alarmfsm.v).

Submit:

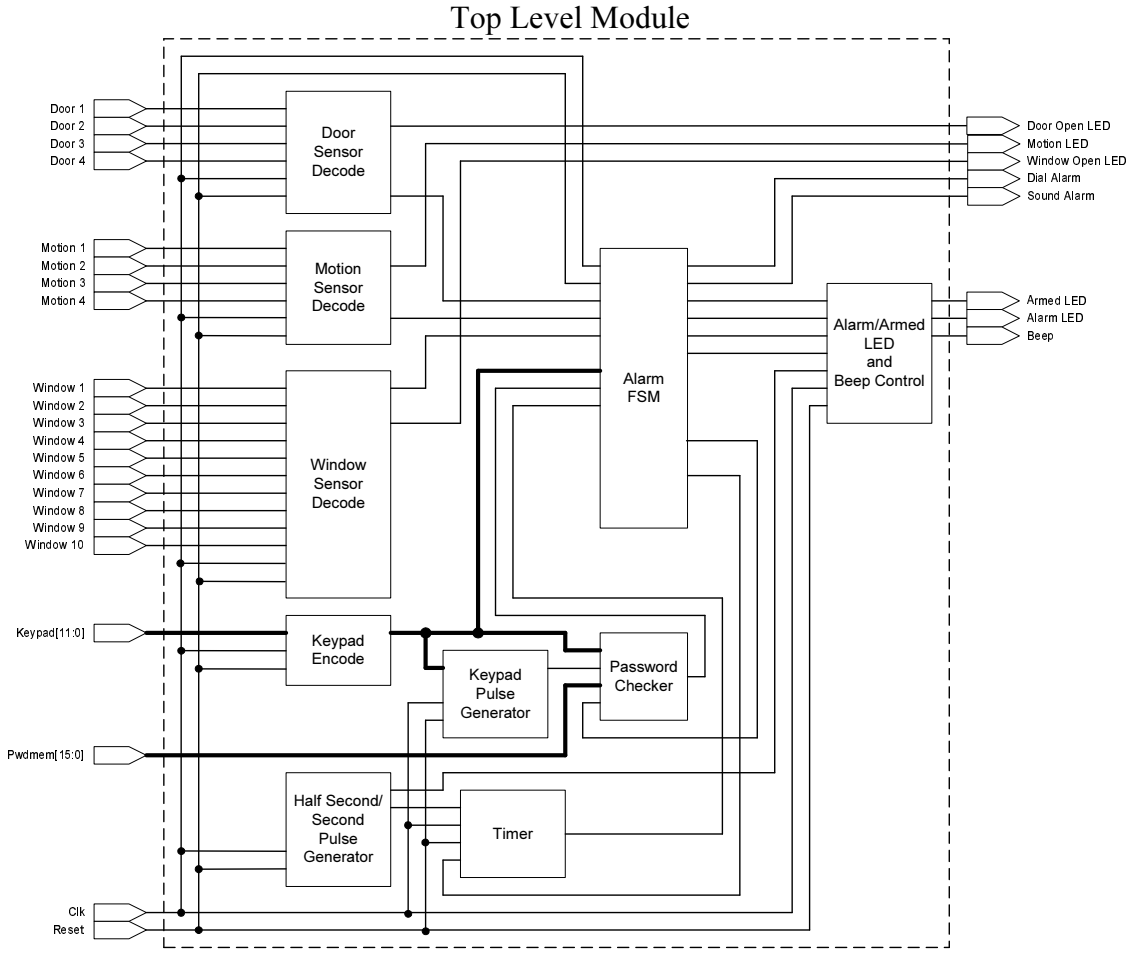
1) Submit the synthesis log file and the "alarmfsm.v" file.

Lab 11 – Bringing it all together

Goal – Bring all of the design pieces together to create the top-level design. Write system tasks to test the design.

Background – Congratulations! You've finished all of the design blocks in your design hierarchy. Now it is time to bring it all together in one module, test the completed design, and synthesize the design. Bringing the design together is accomplished by creating a top-level module with all of the external ports defined, all of the lower-level design modules instantiated, and the wire interconnect between the modules (see diagram below).

To test the design, you'll write a set of tasks to simulate the window, door, and motion detectors, as well as the pressing of buttons on the keypad.



Do: Write the Verilog RTL Code –

1) Write the necessary code to create the top level design. Call the design module top, and the design file top.v.
2) Write a set of tasks to provide stimulus to inputs of the top level design. Write the testbench as a set of task calls. Call the testbench file top_tb.v. Place the tasks in a separate file and use the `include statement to include the tasks in the testbench.

Do: Run a simulation and view the waveforms –

1) Simulate the design.

Do: Synthesize the RTL code –

1) Synthesis will require bringing all of the lower level design files (RTL files) into the project and synthesizing the design.

Submit:

1) The top.v file.
2) The top_tb.v file.
3) The tasks.v file.
4) The log file from the synthesis.
5) The log file from the simulation.

Lab 12 – Design Enhancement
(Grad Students Only)

Goal – Rework the design to implement an EEPROM to store a user-defined password.

Background – Enhance the design by implementing the functionality to support an EEPROM instead of a ROM for password storage. An EEPROM will allow a new password to be stored by the user.

EEPROM specifications – When EEPWRITE is low, the EEPROM will output the data stored in the EEPROM core on the EEPDATAOUT port. When WRITE is high, the EEPROM will take the data on EEPDATAIN and write it to the EEPROM core. The EEPWRITE signal has to be a logic '1' for 500ms for a valid write operation.

The EEPROM Verilog file is provided (see the class web page).

The module name is ***eeprom1x16***. The ports (in order) are:

EEPDATAOUT[15:0]
EEPDATAIN[15:0]
EEPWRITE

Design Requirement –

1) The user will be required to enter their old password before creating a new one.

Design Hint–

1) Key 11 on the keypad (the "#" key) is unused. It could be used as a "set password" button.

Do:

1) Modify, test, and synthesize the design.

Submit:

1) All files modified or new files created in modifying the original design.
2) The synthesis log file.