Custom Logic Design Services
Product Development Division

Chris Post
Product Development Engineer
6111 Massachusetts Avenue
Cambridge, MA 02139

October 10, 2007

Jane Porsche
CEO
Fancy Widgets, Ltd.
8080 Technology Square
Cambridge, MA 02139

Dear Ms. Porsche:

I submit herewith a proposal for a custom car anti-theft system. This system is highly secure as well as easy to use. In addition, it incorporates a special, hidden theft deterrent that is unique to this device.

All questions about the system may be directed to me.

Your consideration of the attached design proposal is greatly appreciated.

Sincerely,

Chris Post

Enclosure: Design Proposal

Car Anti-Theft System

Chris Post
October 10, 2007

Abstract

The car anti-theft system is a comprehensive car alarm system with the addition of a hidden deterrent to car theft. The system implements the usual functionality of a car alarm to detect the status of the doors and ignition and use this information to determine when the audible alarm should be sounded to indicate that an unauthorized user has compromised the car. In addition, the system has a custom component that disconnects power to the fuel pump unless the brake pedal and a special hidden switch are pressed simultaneously and after the ignition switch is activated. This prevents that car from being driven unless the user has knowledge of the hidden aspects of the system.

Table of Contents

List of Figures

1. Overview

The car anti-theft system is designed for the busy car owner who wants to provide maximum protection to his car. The system provides the common measures provided by most car alarms while adding a high degree of automation to the process. The system automatically arms itself when the user exits the car, leaving no chance for the system to be accidentally left unarmed. The system is also easy to use, accounting for common scenarios such as the driver opening the passenger door before going around the car to get into the driver's side. In addition, this system provides s secret deterrent to car thieves by using a proprietary subsystem to control power to the fuel pump. Only a user that knows about the secret subsystem can successfully disable it to provide power to the fuel pump, allowing the car to be driven.

The car anti-theft system has two main status inputs (one each for the driver and passenger doors) and two main outputs (a status light and an audible siren). The driver and passenger door switches let the system know whether each door is open. The status light provides information to the user about the state of the anti-theft system. When it is flashing, the system is armed. In this state, opening either door will put the system into the triggered state, and the status light will be steadily on. After a time delay, the system will begin sounding the audible siren. The siren will continue to sound until a specified time after all the doors are closed, when it will rearm itself, or until the ignition is turned on, where the system will disarm itself. When disarmed, the status light and the siren will both be off. Once the ignition is turned off, the system waits for the driver door to open, then waits for both doors to be closed. Then, it will rearm itself after a time delay.

In order to allow time for the driver to open the passenger door for a guest then walk around the car to open the driver door and turn on the ignition, the delay settings for the driver and passenger doors are individual. By default, the system will wait eight seconds before sounding the audible siren when the driver door is opened, but it will wait fifteen seconds before the siren activates when the passenger door is opened. These time parameters are fully configurable via special input switches. Two switches determine the time parameter to be configured (a two-bit binary value), and four switches determine the time delay value (four bits) to be programmed into the system. In addition to the door delay parameters, there are two other configurable time delays. When the alarm is sounding, the time delay between all the doors closing and the system going into the silent, armed state is configurable. In addition, the time delay between when all the doors close after the occupants leave the car and when the system goes back into the rearmed state is also configurable. Pressing the "reprogram" switch will program the current value of the "time value" switches into the corresponding time parameter as specified by the "time parameter selector" switches.

The status light on the anti-theft system gives the user feedback about the current alarm state. If the light is blinking with a two-second period, the alarm is in the armed state. If the light is steadily on, the alarm has been triggered. This may mean that the system is either waiting to sound the alarm based on the driver or passenger delays, or it is currently sounding the alarm. If the light is off, the system is in the disarmed mode.

The siren on the anti-theft system sounds to ward off an intruder in the car when the alarm has been triggered and the appropriate time delay has expired. In order to make the alarm effective at repelling intruders, the siren implements a special sound alternates between sweeping upwards in frequency through a high-frequency range and sweeping downwards through a low-frequency range. This makes any attempted theft of the car obvious to any bystanders, deterring the person attempting to steal the car.

In addition to the visible and audible parts of the alarm system, the anti-theft system implements a special module to control power to the fuel pump, disabling the car unless it is being operated by someone who knows about the special system. In order for the power for the fuel pump to be activated, the ignition must be on, and then the brake pedal and a special hidden switch must be pressed simultaneously. The fuel pump power is disabled again when the ignition is turned off.

2. Description

2.1 Overall Organization

The car anti-theft system is composed of seven main modules: the debouncer, fuel pump logic, 1Hz clock divider, timer module, time parameters module, anti-theft FSM, and siren generator. The modules are interconnected as shown in Figure 1[1].

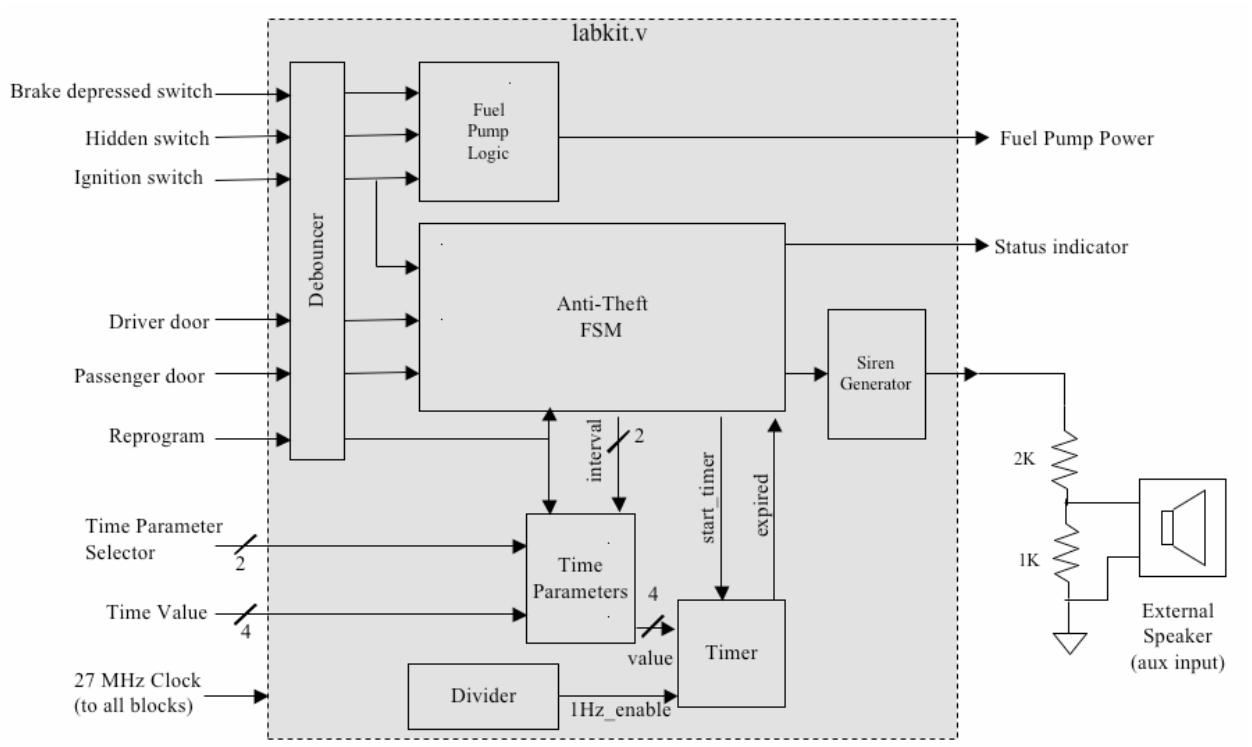

Figure 1 - Block diagram of the anti-theft system

[1] MIT. (2007). *6.111 Course Website.* http://web.mit.edu/6.111/www/f2007/handouts/labs/lab3.html

The debouncer module debounces and synchronizes all the button and switch inputs so that no glitches occur under normal operation.

The fuel pump logic includes the necessary functionality to implement the secret fuel pump control mechanism. It only provides power to the fuel pump when the ignition is turned on, then the brake pedal and hidden switch are activated simultaneously.

The 1Hz clock divider takes the 27MHz system clock and divides it to a 1Hz pulse for use in the timer module. The timer module gets a value in seconds from the time parameters module and counts down when the anti-theft FSM signals on the start_timer control line. The time parameters module stores the default time values and also reads the time parameter selector, time value, and reprogram input switches to allow the user to program different time values into the module. This module also receives a 2-bit interval value from the anti-theft FSM which determines which time delay value to send to the timer module.

The anti-theft FSM is a state machine that, based on the inputs it receives, controls the timer, status light, and siren generator. The siren generator takes an enable signal from the anti-theft FSM and produces a square wave for audio output that sweeps up and down respectively through high and low frequency ranges. All modules are attached to a global reset signal to restore every unit to the default internal values and states on power-up or user reset.

2.2 Debouncer

Since the output from mechanical pushbuttons and switches produce very rapid voltage fluctuations when switched, they can accidentally transmit multiple presses in digital logic levels during a very short time period when pressed. Thus, it is important to make sure that the signal from every mechanical input only shows one change in logic level when the state of the button changes.

The debouncer module accomplishes this by ensuring that only changes in state that last longer than 0.01 seconds are transmitted to the other modules in the system. In order to do this, the debouncer has an internal counter that counts up 270 000 (0.01s with a 27MHz clock) if the incoming signal does not change state. If the incoming signal changes state, the counter will reset to zero. The outgoing signal will only reflect the change in the incoming signal if the counter reaches 270 000.

2.3 Fuel Pump Logic

The fuel pump logic implements the secret theft deterrent of controlling the power to the fuel pump to only allow power when the correct sequence of inputs is received. It works by using a simple 3-state finite state machine. When the reset signal is received, the FSM is in the reset state (the default state), and power to the fuel pump is disabled. If the ignition is turned on, the FSM goes into the ignition state, where it will wait for both the brake pedal and the hidden switch to be activated at the same time before making a transition to the active state. In the ignition state, the power to the pump is still disabled. Once in the active state, power to the pump is enabled, and the car can run. Also, once in the active state, the FSM ignored the brake pedal and hidden switch inputs, so power will be latched on solely by the ignition switch. In either the ignition or active states, the FSM will return to the reset state if the ignition is turned off. The power to the pump is controlled using combinational logic to assert the pump power signal when the state variable reflects that the FSM is in the active state.

2.4 1Hz Clock Divider

The timer module uses seconds to count its timing values, so there must be a source for a 1Hz clock in the anti-theft system. The 1Hz clock divider takes the 27MHz system clock and divides it down to a 1Hz pulse for use in the timer module. To do this, the divider uses an internal counter that is incremented on every rising edge of the system clock. The counter counts from 0 to (n - 1), where n is the number of clock cycles of the system clock that occur for every pulse on the output clock. Since the anti-theft system has a 27MHz system clock, the parameter n is set to 270 000 000. The output pulse is controlled by combinational logic which makes the output active when the counter reaches (n - 1). This only occurs for one clock cycle, so the 1Hz signal is already output as a pulse.

The clock divider's reset signal is asserted when there is a global reset, but is also connected to the start_timer signal from the anti-theft FSM via an OR gate, so that when either the global reset or start_timer signal is asserted, the internal counter restarts at 0. This is so that the timer module receives its first 1Hz pulse exactly a second from when the start_timer signal is asserted, ensuring accurate timing.

2.5 Timer Module

The timer module counts down the number of seconds received in 4 bits on the value line when the start_timer signal is asserted by the anti-theft FSM. When the countdown is finished, the timer asserts the expired signal. Internally, the timer keeps track of its operations using a 4-bit counter and a single bit timing flag. On reset, the counter is reset to zero, and the timing flag is de-asserted. When the start_timer signal from the anti-theft FSM is asserted, the timer latches the current input from the value line into the counter and asserts the timing flag. While the timing flag is asserted, the timer will decrement the counter every time a pulse is received from the 1Hz clock divider. When the counter reaches zero, the timing flag is de-asserted. The expired signal is asserted using combinational logic when the count reaches 0, and it is output as a pulse to the anti-theft FSM.

Since the time value is received from the time parameters module, and is selected in that module by the anti-theft FSM, there is an extra clock cycle delay between when start_timer is asserted and when a valid time value reaches the timer module if both the interval and start_timer signals were sent from the anti-theft FSM in the same clock cycle. Thus, the anti-theft FSM must compensate for this delay by setting up the interval signal a clock cycle before it asserts the start_timer signal. (See section 2.7 for further detail on this operation.) Furthermore, when the start_timer signal is asserted, the timer will restart timing even if it is already counting down. This allows the anti-theft FSM to correctly reset the timer module in the case that the anti-theft FSM makes a transition, based on a non-timer input, between two states that both use the timer module.

2.6 Time Parameters Module

The time parameters module stores the 4 different time delay values used in the anti-theft system and passes the value for the appropriate parameter selected by the anti-theft FSM to the timer module. All the time parameters can also be reprogrammed via the user input switches. The timer parameters, their associated parameter number, and their default values are shown in Table 1[2].

Interval Name	Symbol	Parameter Number	Default Time (sec)	Time Value
Arming delay	t_arm_delay	00	6	0110
Countdown, driver's door	t_driver_delay	01	8	1000
Countdown, passenger door	t_passenger_delay	10	15	1111
Siren on time	t_alarm_on	11	10	1010

Table 1 - Timing parameters

On every system clock cycle, the time parameters module will output the appropriate time value based on the parameter number received from the anti-theft FSM on the interval lines. The time parameters module will also check for the reprogram signal from the user, which will latch the time value from the time value selector switched into memory for the correct interval based on the value of the time parameter selector switches. When there is a global reset, the time parameters module restores all the time delay intervals to their default values as specified in Table 1. Since there are only 4 time parameters of 4 bits each, the space required to store the values is very small, so the values are stored in register memory to improve the latency for accessing the values from memory.

2.7 Anti-Theft FSM

The anti-theft FSM is the core controller for the alarm system. It takes inputs from the ignition, driver, passenger, and reprogram switches. It also controls the time parameters module, timer, siren generator, and status indicator. The main state machine consists of 11 states, as shown in Figure 2. The armed state is the default state when a global reset signal is asserted. Also, any time the reprogram signal is asserted, the FSM will return to the armed state. The outputs of the FSM have flags in each state specifying whether the timer is enabled, which interval to select from the timer parameters module, the mode for the status light, and whether

[2] Adapted from: MIT. (2007). *6.111 Course Website.* http://web.mit.edu/6.111/www/f2007/handouts/labs/lab3.html

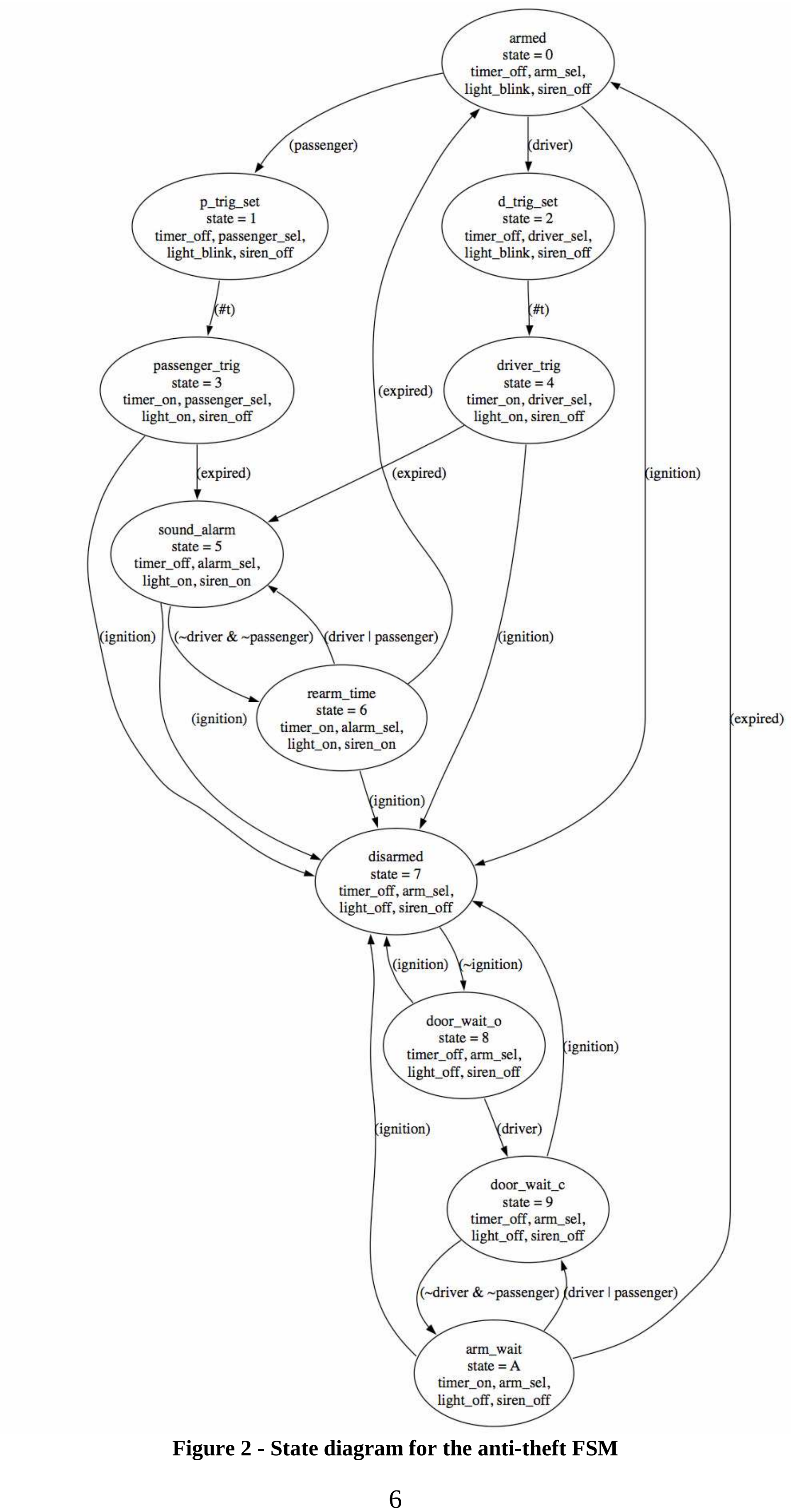

Figure 2 - State diagram for the anti-theft FSM

the siren is enabled. The timer signal is converted to a pulse before being sent to the timer module to ensure synchronicity with the clock.

In order to simplify the process of specifying the outputs for each state, a single 6-bit register "outputs" was created to hold the status of all the outputs. The individual bits of this signal are wired to the appropriate individual signals to the other modules. The timer_off and timer_on flags specify whether the start_timer signal is being de-asserted or asserted, respectively. The time delay parameter can be one of four flags (arm_sel, driver_sel, passenger_sel, and alarm_sel) to specify which time delay interval is being selected by the time parameters module. The light status is controlled using the 3 flags, light_off, light_on, and light_blink. These 2 bits for the light status are sent to a light controller sub module, which controls the actual status light output. The siren_off or siren_on flags control the enable signal sent to the siren generator.

As the state diagram reflects, the time delay interval parameter must be setup a clock cycle before the timer_on flag is set so that the time parameters module can send the appropriate time value to the timer module. In the anti-theft FSM, there are states where the timer is enabled. When the FSM reaches rearm_time (state 6), it activates the timer with the alarm_sel time parameter value. There is only one state from which the rearm_time can be reached (sound_alarm, state 5), so the alarm_sel parameter can be sent to the time parameters module in plenty of time for the timer to get the correct time delay value. Likewise, arm_wait (state A) activates the timer with the arm_sel parameter. This state is only reached via door_wait_c (state 9), so the arm_sel parameter can be set during door_wait_c.

With the passenger_trig (state 3) and driver_trig (state 4) states, however, there must be two extra states in order to set the correct time parameter values a clock cycle early, since either of these paths can be triggered from the armed (state 0) state. Thus, p_trig_set (state 1) and d_trig_set (state 2) setup the correct time delay parameters, then immediately make the transition to the passenger_trig or driver_trig states, respectively.

Since the FSM uses a 2-bit signal to control the status of the light, there is a sub module that takes this signal and controls the actual status light output. Within this module, there is a dedicated 1Hz clock divider to send a pulse which makes the light turn on or off every second when the light is in the light_blink mode. This divider is reset when the light control module enters the light_blink mode to ensure that the first transition is a second after the light mode changes state. In addition, the light control sub module also accounts for the light_off and light_on modes.

2.8 Siren Generator

The siren generator in the anti-theft system employs a special sweeping audio effect where it starts at a middle frequency, ascends to a high frequency, then drops down to the middle frequency to descend to a low frequency before restarting the cycle. This is accomplished using two variable clock dividers. The first clock divider (the siren clock) generates pulses on which the siren generator causes transitions in the audio output. This clock has a variable divisor input which is controlled by the siren generator. The second clock (the sweep clock) is a variable clock divider where the divisor is wired to a numerical constant in the main siren generator module. This clock sends a pulse every time the divisor of the siren clock is to be incremented or decremented by 1. Thus, it controls the speed of the frequency sweeping action of the siren.

The siren generator module uses a single bit variable to hold the state of the audio generation. When asc = 1, the generator is in the ascending mode; it is in the descending mode otherwise. On every cycle of the system clock, the siren generator will reset all internal values if a global reset is asserted, or on the rising edge of the enable signal. It also causes a transition on the siren audio output if the siren clock outputs a pulse during that clock cycle. Initially, the siren generator is in the ascending mode, so it will sweep up in frequency, decrementing the divisor sent to the siren clock every time there is a pulse from the sweep clock, until the current frequency hits the high frequency. Then, the asc variable is deasserted, and the middle frequency divisor is sent to the siren clock. The siren clock is also reset every time the siren generator makes a jump in frequency. When the asc variable is de-asserted, the siren generator will sweep down in frequency, incrementing the divisor for the siren clock, until it hits the lower frequency. It will then assert the asc variable, sent the middle frequency divisor to the siren clock, and restart the cycle.

Since the variable clock divider must be able to accept a changing divisor without being reset in order to ensure a smoothly sweeping audio output, it is a specially modified version of a regular clock divider. Normally a clock divider would output a pulse exactly when the current internal counter of the divider is one less than the divisor. In the case of a variable divisor, however, the divisor could drop to a value lower than the current count, which would cause the clock divider to not output a pulse until it is reset again. In the variable divider implementation in the siren generator, however, the variable clock divider checks for the case where the counter is greater than or equal to one less than the divisor value, ensuring that accurate pulses will always be output regardless of the status of a changing divisor.

3. Conclusion

The car anti-theft system implements all the necessary security for securing a car against theft. The system automatically arms itself after a preset delay when the driver exits the car and all the doors are closed, preventing the situation where the user accidentally leaves the system unarmed. There is also a hidden theft deterrent mechanism that controls power to the fuel pump. This special feature only allows the fuel pump power to be enabled after the ignition is turned on and then the brake pedal and a special hidden switch are activated at the same time. Thus, if an unauthorized user disables the alarm system, there is still a hidden system preventing him from driving the car away. In addition to being secure, the system is also easy to use. It accounts for the case where the driver wants to open the passenger door to let the passenger in before going around to the driver's side of the car by allowing a longer delay before the alarm sounds on the passenger door. Thus, this system is the ideal balance of security and efficiency for a car anti-theft system.

4. References

MIT. (2007). *6.111 Course Website.* http://web.mit.edu/6.111/www/f2007/handouts/labs/lab3.html (9 October 2007).

5. Appendices

Appendix A – Top Level Module (labkit.v)

```
///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
///////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
///////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2006-Mar-08: Corrected default assignments to "vga_out_red", "vga_out_green"
//              and "vga_out_blue". (Was 10'h0, now 8'h0.)
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
//              "disp_data_out", "analyzer[2-3]_clock" and
//              "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//              actually populated on the boards. (The boards support up to
//              256Mb devices, with 25 address lines.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
//
```

```
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
//              value. (Previous versions of this file declared this port to
//              be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
//              actually populated on the boards. (The boards support up to
//              72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
//
///////////////////////////////////////////////////////////////////////////////

module labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
          ac97_bit_clock,

          vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
          vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
          vga_out_vsync,

          tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
          tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
          tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

          tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
          tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
          tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
          tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

          ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
          ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

          ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
          ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

          clock_feedback_out, clock_feedback_in,

          flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
          flash_reset_b, flash_sts, flash_byte_b,

          rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

          mouse_clock, mouse_data, keyboard_clock, keyboard_data,

          clock_27mhz, clock1, clock2,

          disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
          disp_reset_b, disp_data_in,

          button0, button1, button2, button3, button_enter, button_right,
          button_left, button_down, button_up,

          switch,

          led,

          user1, user2, user3, user4,

          daughtercard,

          systemace_data, systemace_address, systemace_ce_b,
          systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

          analyzer1_data, analyzer1_clock,
          analyzer2_data, analyzer2_clock,
          analyzer3_data, analyzer3_clock,
          analyzer4_data, analyzer4_clock);

   output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
   input  ac97_bit_clock, ac97_sdata_in;

   output [7:0] vga_out_red, vga_out_green, vga_out_blue;
   output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
     vga_out_hsync, vga_out_vsync;

   output [9:0] tv_out_ycrcb;
   output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
     tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
     tv_out_subcar_reset;

   input  [19:0] tv_in_ycrcb;
   input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
     tv_in_hff, tv_in_aff;
   output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
     tv_in_reset_b, tv_in_clock;
   inout  tv_in_i2c_data;

   inout  [35:0] ram0_data;
   output [18:0] ram0_address;
   output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
   output [3:0] ram0_bwe_b;

   inout  [35:0] ram1_data;
   output [18:0] ram1_address;
   output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
   output [3:0] ram1_bwe_b;

   input  clock_feedback_in;
   output clock_feedback_out;

   inout  [15:0] flash_data;
   output [23:0] flash_address;
   output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
   input  flash_sts;

   output rs232_txd, rs232_rts;
   input  rs232_rxd, rs232_cts;

   input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

   input  clock_27mhz, clock1, clock2;

   output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
   input  disp_data_in;
   output  disp_data_out;
```

```

   input  button0, button1, button2, button3, button_enter, button_right,
     button_left, button_down, button_up;
   input  [7:0] switch;
   output [7:0] led;

   inout [31:0] user1, user2, user3, user4;

   inout [43:0] daughtercard;

   inout  [15:0] systemace_data;
   output [6:0]  systemace_address;
   output systemace_ce_b, systemace_we_b, systemace_oe_b;
   input  systemace_irq, systemace_mpbrdy;

   output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
          analyzer4_data;
   output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

   ////////////////////////////////////////////////////////////////////////////
   //
   // I/O Assignments
   //
   ////////////////////////////////////////////////////////////////////////////

   // Audio Input and Output
   assign beep= 1'b0;
   assign audio_reset_b = 1'b0;
   assign ac97_synch = 1'b0;
   assign ac97_sdata_out = 1'b0;
   // ac97_sdata_in is an input

   // VGA Output
   assign vga_out_red = 8'h0;
   assign vga_out_green = 8'h0;
   assign vga_out_blue = 8'h0;
   assign vga_out_sync_b = 1'b1;
   assign vga_out_blank_b = 1'b1;
   assign vga_out_pixel_clock = 1'b0;
   assign vga_out_hsync = 1'b0;
   assign vga_out_vsync = 1'b0;

   // Video Output
   assign tv_out_ycrcb = 10'h0;
   assign tv_out_reset_b = 1'b0;
   assign tv_out_clock = 1'b0;
   assign tv_out_i2c_clock = 1'b0;
   assign tv_out_i2c_data = 1'b0;
   assign tv_out_pal_ntsc = 1'b0;
   assign tv_out_hsync_b = 1'b1;
   assign tv_out_vsync_b = 1'b1;
   assign tv_out_blank_b = 1'b1;
   assign tv_out_subcar_reset = 1'b0;

   // Video Input
   assign tv_in_i2c_clock = 1'b0;
   assign tv_in_fifo_read = 1'b0;
```

```
      assign tv_in_fifo_clock = 1'b0;
      assign tv_in_iso = 1'b0;
      assign tv_in_reset_b = 1'b0;
      assign tv_in_clock = 1'b0;
      assign tv_in_i2c_data = 1'bZ;
      // tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
      // tv_in_aef, tv_in_hff, and tv_in_aff are inputs

      // SRAMs
      assign ram0_data = 36'hZ;
      assign ram0_address = 19'h0;
      assign ram0_adv_ld = 1'b0;
      assign ram0_clk = 1'b0;
      assign ram0_cen_b = 1'b1;
      assign ram0_ce_b = 1'b1;
      assign ram0_oe_b = 1'b1;
      assign ram0_we_b = 1'b1;
      assign ram0_bwe_b = 4'hF;
      assign ram1_data = 36'hZ;
      assign ram1_address = 19'h0;
      assign ram1_adv_ld = 1'b0;
      assign ram1_clk = 1'b0;
      assign ram1_cen_b = 1'b1;
      assign ram1_ce_b = 1'b1;
      assign ram1_oe_b = 1'b1;
      assign ram1_we_b = 1'b1;
      assign ram1_bwe_b = 4'hF;
      assign clock_feedback_out = 1'b0;
      // clock_feedback_in is an input

      // Flash ROM
      assign flash_data = 16'hZ;
      assign flash_address = 24'h0;
      assign flash_ce_b = 1'b1;
      assign flash_oe_b = 1'b1;
      assign flash_we_b = 1'b1;
      assign flash_reset_b = 1'b0;
      assign flash_byte_b = 1'b1;
      // flash_sts is an input

      // RS-232 Interface
      assign rs232_txd = 1'b1;
      assign rs232_rts = 1'b1;
      // rs232_rxd and rs232_cts are inputs

      // PS/2 Ports
      // mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs

      // LED Displays

      // Commenting out default values so debug information can be displayed
   // assign disp_blank = 1'b1;
   // assign disp_clock = 1'b0;
   // assign disp_rs = 1'b0;
   // assign disp_ce_b = 1'b1;
   // assign disp_reset_b = 1'b0;
   // assign disp_data_out = 1'b0;

      // disp_data_in is an input

      // Buttons, Switches, and Individual LEDs

      // Changed so led[1:0] can be used
   // assign led = 8'hFF;
      assign led[7:2] = 6'b111111;

      // button0, button1, button2, button3, button_enter, button_right,
      // button_left, button_down, button_up, and switches are inputs

      // User I/Os
      // Changed so user[0] can be used
   // assign user1 = 32'hZ;
      assign user1[31:1] = 31'hZ;
      assign user2 = 32'hZ;
      assign user3 = 32'hZ;
      assign user4 = 32'hZ;

      // Daughtercard Connectors
      assign daughtercard = 44'hZ;

      // SystemACE Microprocessor Port
      assign systemace_data = 16'hZ;
      assign systemace_address = 7'h0;
      assign systemace_ce_b = 1'b1;
      assign systemace_we_b = 1'b1;
      assign systemace_oe_b = 1'b1;
      // systemace_irq and systemace_mpbrdy are inputs

      // Logic Analyzer
      // Commenting out so signals can be routed to LA
   // assign analyzer1_data = 16'h0;
   // assign analyzer1_clock - 1'b1;
      assign analyzer2_data = 16'h0;
      assign analyzer2_clock = 1'b1;
      assign analyzer3_data = 16'h0;
      assign analyzer3_clock = 1'b1;
      assign analyzer4_data = 16'h0;
      assign analyzer4_clock = 1'b1;

       // Setup a reset signal, taken from lecture handouts
       // Using button_up as reset button, since the others are taken
       wire power_on_reset;
       SRL16 reset_sr(.D(1'b0), .CLK(clock_27mhz), .Q(power_on_reset),
           .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
       defparam reset_sr.INIT = 16'hFFFF;
       wire reset, user_reset;
       debounce db_reset(power_on_reset, clock_27mhz, ~button_up, user_reset);
       assign reset = user_reset | power_on_reset;

       // Declare module interconnectoin signals

       wire hidden, brake, driver, passenger, ignition, reprogram;
       wire [1:0] parm_sel;
       wire [3:0] time_val;

       wire pump, light, siren;

       wire [1:0] interval;
       wire start_timer, expired;
       wire [3:0] fsm_state;
       wire [3:0] value;
       wire clk_one_hz;
       wire [3:0] timer_count;

       // Assign LA pods
       assign analyzer1_data = {clk_one_hz, start_timer, expired, value[3:0],
   interval[1:0], reprogram, passenger, driver, ignition, hidden, brake, light};
       assign analyzer1_clock = clock_27mhz;

       // Debeounce all the button/switch inputs
       debounce db_hidden(reset, clock_27mhz, ~button0, hidden);
       debounce db_brake(reset, clock_27mhz, ~button1, brake);
       debounce db_driver(reset, clock_27mhz, ~button2, driver);
       debounce db_passenger(reset, clock_27mhz, ~button3, passenger);
       debounce db_ignition(reset, clock_27mhz, switch[7], ignition);
       debounce db_reprogram(reset, clock_27mhz, ~button_enter, reprogram);
       debounce db_parm_sel0(reset, clock_27mhz, switch[4], parm_sel[0]);
       debounce db_parm_sel1(reset, clock_27mhz, switch[5], parm_sel[1]);
       debounce db_time_val0(reset, clock_27mhz, switch[0], time_val[0]);
       debounce db_time_val1(reset, clock_27mhz, switch[1], time_val[1]);
       debounce db_time_val2(reset, clock_27mhz, switch[2], time_val[2]);
       debounce db_time_val3(reset, clock_27mhz, switch[3], time_val[3]);

       // Assign the output lights
       assign led[0] = ~light;
       assign led[1] = ~pump;

       // Instantiate the Anti-Theft FSM module
       mainFSM control_fsm(.reset(reset), .clk(clock_27mhz), .ignition(ignition),
            .driver(driver), .passenger(passenger), .reprogram(reprogram),
            .interval(interval), .start_timer(start_timer), .expired(expired),
            .light(light), .siren(siren), .state(fsm_state));

      // Instantiate the Time Parameters Module
      time_parms time_parameters(.reset(reset), .clk(clock_27mhz),
            .reprogram(reprogram), .parm_sel(parm_sel), .time_val(time_val),
            .interval(interval), .value(value));

      // Instantiate the Clock Divider Module
      clkdiv one_hz_div(.reset(reset | start_timer), .clkin(clock_27mhz),
            .clkout(clk_one_hz));

      // Instantiate the Timer module
      timer timer_module(.reset(reset), .clk(clock_27mhz), .clk_one_hz(clk_one_hz),
            .value(value), .start_time(start_timer), .expired(expired),
            .count(timer_count));

      // Instantiate the Fuel Pump Logic
      fuelpump pump_logic(.reset(reset), .clk(clock_27mhz), .ignition(ignition),
            .brake(brake), .hidden(hidden), .pump(pump));

      // Instantiate a controller for the VFD display for debug information
      // Main FSM state and Timer countdown value are displayed
      display_16hex hexout(.reset(reset), .clock_27mhz(clock_27mhz),
            .data({56'h00000000000000, timer_count, fsm_state}),
            .disp_blank(disp_blank), .disp_clock(disp_clock), .disp_rs(disp_rs),
            .disp_ce_b(disp_ce_b), .disp_reset_b(disp_reset_b),
            .disp_data_out(disp_data_out));

      // Instantiate the Siren Generator Module
      siren siren_gen(.reset(reset), .clk(clock_27mhz), .siren_en(siren),
            .siren_out(user1[0]));

   endmodule
```

Appendix B – Debouncer (debounce.v)

```
// Switch Debounce Module
// use your system clock for the clock input
// to produce a synchronous, debounced output
module debounce (reset, clock, noisy, clean);
   parameter DELAY = 270000;   // .01 sec with a 27Mhz clock
   input reset, clock, noisy;
   output clean;

   reg [18:0] count;
   reg new, clean;

   always @(posedge clock)
     if (reset)
       begin
     count <= 0;
     new <= noisy;
     clean <= noisy;
       end
     else if (noisy != new)
       begin
     new <= noisy;
     count <= 0;
       end
     else if (count == DELAY)
       clean <= new;
     else
       count <= count+1;

endmodule
```

Appendix C – Level to Pulse Converter (levpulse.v)

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
```

```
// Create Date:    20:31:58 10/04/2007
// Design Name:
// Module Name:    levpulse
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module levpulse(clk, level, pulse);
   input clk, level;
   output pulse;
   reg qnot;

   // Convert a level to a pulse
   always @ (posedge clk) begin
         qnot <= ~level;
   end

   assign pulse = (qnot & level);

endmodule
```

Appendix D – Fuel Pump Logic (fuelpump.v)

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    21:21:07 10/04/2007
// Design Name:
// Module Name:    fuelpump
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module fuelpump(reset, clk, ignition, brake, hidden, pump);
   input reset, clk, ignition, brake, hidden;
   output pump;
   reg [1:0] state;

   // Define the states
   parameter s_reset = 2'b00;
   parameter s_ignition = 2'b01;
   parameter s_active = 2'b10;

   // Create a simple FSM
   always @ (posedge clk) begin
      if (reset) state <= s_reset;
         case (state)
               s_reset           : state <= ignition ? s_ignition : state;
               s_ignition  : state <= ~ignition ? s_reset : ((brake & hidden) ? s_active : state);
               s_active          : state <= (reset | ~ignition) ? s_reset : state;
               default           : state <= s_reset;
         endcase
   end

   // Turn the pump power on if we're in the active state
   assign pump = (state == s_active);

endmodule
```

Appendix E – 1Hz Clock Divider (clkdiv.v)

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    20:49:37 10/03/2007
// Design Name:
// Module Name:    clkdiv
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module clkdiv(reset, clkin, clkout);
   input reset, clkin;
   output clkout;
   reg [25:0] count;

   // Define when the output pulses for every n input clock cycles
   parameter every_n = 27000000;

   // Count up until we need an output pulse, then do it again
   always @ (posedge clkin) begin
         count <= ((count == (every_n - 1)) | reset) ? 0 : count + 1;
   end

   // Send an output pulse if the counter is done
   assign clkout = (count == (every_n - 1));

endmodule
```

Appendix F – Timer Module (timer.v)

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    18:08:51 10/04/2007
// Design Name:
// Module Name:    timer
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module timer(reset, clk, clk_one_hz, value, start_time, expired, count);
   input reset;
   input clk;
   input clk_one_hz;
   input [3:0] value;
   input start_time;
   output expired;
   output [3:0] count;
   reg [3:0] count;

   wire exp_lev;
   reg timing;

   always @ (posedge clk) begin
         // Make sure the timer is stopped on reset
         if (reset) begin
               count <= 0;
               timing <= 0;
         end

         // When start_time asserted, latch the time value and start counting
down
         if (start_time) begin
               timing <= 1;
```

```
                count <= value;
          end

          // If the counter reaches 0, stop timing
          if ((count == 0) & timing) begin
                timing <= 0;
          end

          // Decrement the counter on the 1Hz clock
          if (timing & clk_one_hz) begin
                count <= count - 1;
          end
    end

    // If the counter is done, send the expired signal
    assign exp_lev = ((count == 0) & timing);

    // Make sure the expired signal outputs as a pulse
    levpulse exp_out (.clk(clk), .level(exp_lev), .pulse(expired));

endmodule
```

Appendix G – Timer Parameters Module (timer_parms.v)

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    23:27:12 10/03/2007
// Design Name:
// Module Name:    time_parms
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module time_parms(reset, clk, reprogram, parm_sel, time_val, interval, value);
   input reset, clk, reprogram;
   input [1:0] parm_sel;
   input [3:0] time_val;
   input [1:0] interval;
   output [3:0] value;
   reg [3:0] value;
   reg [3:0] t_arm_delay;
   reg [3:0] t_driver_delay;
   reg [3:0] t_passenger_delay;
   reg [3:0] t_alarm_on;
```

```

    // Define each time parameter
    parameter t_arm_sel = 2'b00;
    parameter t_driver_sel = 2'b01;
    parameter t_passenger_sel = 2'b10;
    parameter t_alarm_sel = 2'b11;

    always @ (posedge clk) begin
          // Restore all the values to their defaults on reset
          if (reset) begin
                t_arm_delay <= 4'd6;
                t_driver_delay <= 4'd8;
                t_passenger_delay <= 4'd15;
                t_alarm_on <= 4'd10;
          end

          // On reprogram, store the time_val from the switches in the right
          // parameter
          if (reprogram) begin
                case (parm_sel)
                      t_arm_sel         : t_arm_delay <= time_val;
                      t_driver_sel      : t_driver_delay <= time_val;
                      t_passenger_sel: t_passenger_delay <= time_val;
                      t_alarm_sel       : t_alarm_on <= time_val;
                      default                 : value <= value;
                endcase
          end

          // Output the correct value to the timer module based on the interval
          // input
          case (interval)
                t_arm_sel         : value <= t_arm_delay;
                t_driver_sel      : value <= t_driver_delay;
                t_passenger_sel: value <= t_passenger_delay;
                t_alarm_sel       : value <= t_alarm_on;
                default                 : value <= 2'b00;
          endcase
    end

endmodule
```

Appendix H – Anti-Theft FSM (mainFSM.v)

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:53:04 10/04/2007
// Design Name:
// Module Name:    mainFSM
// Project Name:
// Target Devices:
// Tool versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module mainFSM(reset, clk, ignition, driver, passenger, reprogram, interval,
                           start_timer, expired, light, siren, state);
   input reset, clk, ignition, driver, passenger, reprogram;
   output [1:0] interval;
   output start_timer;
   input expired;
   output light, siren;

   output [3:0] state;
   reg [3:0] state;
   wire [1:0] lightstate;
   reg [5:0] outputs;
   wire start_timer_lev;

   // Create a single variable for holding status of the outputs.
   assign {start_timer_lev, interval[1:0], lightstate[1:0], siren} = outputs[5:0];

   // Define flags for the outputs

   // Flags for the interval output to select which time parameter to use
   parameter arm_sel = 2'b00;
   parameter driver_sel = 2'b01;
   parameter passenger_sel = 2'b10;
   parameter alarm_sel = 2'b11;

   // Flags for the status light
   parameter light_off = 2'b00;
   parameter light_on = 2'b01;
   parameter light_blink = 2'b10;

   // Flags for the timer
   parameter timer_off = 1'b0;
   parameter timer_on = 1'b1;

   // Flags for the siren
   parameter siren_off = 1'b0;
   parameter siren_on = 1'b1;

   // Define state names for the main FSM
   parameter armed = 4'b0000;
   parameter p_trig_set = 4'b0001;
   parameter d_trig_set = 4'b0010;
   parameter passenger_trig = 4'b0011;
   parameter driver_trig = 4'b0100;
   parameter sound_alarm = 4'b0101;
   parameter rearm_time = 4'b0110;
   parameter disarmed = 4'b0111;
   parameter door_wait_o = 4'b1000;
```

```
    parameter door_wait_c = 4'b1001;
    parameter arm_wait = 4'b1010;

    // Create the FSM
    always @ (posedge clk) begin
        // For every state, go back to armed if there is a reset or reprogram
        if (reset | reprogram) state <= armed;
        case (state)
            armed: begin
                outputs <= {timer_off, arm_sel, light_blink, siren_off};
                if (ignition) state <= disarmed;
                else if (driver) state <= d_trig_set;
                else if (passenger) state <= p_trig_set;
            end
            p_trig_set: begin
                outputs <= {timer_off, passenger_sel, light_blink, siren_off};
                state <= passenger_trig;
            end
            d_trig_set: begin
                outputs <= {timer_off, driver_sel, light_blink, siren_off};
                state <= driver_trig;
            end
            passenger_trig: begin
                outputs <= {timer_on, passenger_sel, light_on, siren_off};
                if (ignition) state <= disarmed;
                else if (expired) state <= sound_alarm;
            end
            driver_trig: begin
                outputs <= {timer_on, driver_sel, light_on, siren_off};
                if (ignition) state <= disarmed;
                else if (expired) state <= sound_alarm;
            end
            sound_alarm: begin
                outputs <= {timer_off, alarm_sel, light_on, siren_on};
                if (ignition) state <= disarmed;
                else if (~driver & ~passenger) state <= rearm_time;
            end
            rearm_time: begin
                outputs <= {timer_on, alarm_sel, light_on, siren_on};
                if (ignition) state <= disarmed;
                else if (driver | passenger) state <= sound_alarm;
                else if (expired) state <= armed;
            end
            disarmed: begin
                outputs <= {timer_off, arm_sel, light_off, siren_off};
                if (~ignition) state <= door_wait_o;
            end
            door_wait_o: begin
                outputs <= {timer_off, arm_sel, light_off, siren_off};
                if (ignition) state <= disarmed;
                else if (driver) state <= door_wait_c;
            end
            door_wait_c: begin
                outputs <= {timer_off, arm_sel, light_off, siren_off};
                if (ignition) state <= disarmed;
                else if (~driver & ~passenger) state <= arm_wait;
```

```
               end
               arm_wait: begin
                     outputs <= {timer_on, arm_sel, light_off, siren_off};
                     if (ignition) state <= disarmed;
                     else if (driver | passenger) state <= door_wait_c;
                     else if (expired) state <= armed;
               end
               default: state <= armed;
         endcase
   end

   // Convert the start_timer flag to a pulse for output
   levpulse start_timer_ltp (.clk(clk), .level(start_timer_lev), .pulse(start_timer));

   // Instantiate the light control unit
   lightcontrol light_ctl(.reset(reset), .clk(clk), .lightstate(lightstate),
         .light(light));

endmodule
```

Appendix I – Light Control Unit (lightcontrol.v)

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:14:19 10/07/2007
// Design Name:
// Module Name:    lightcontrol
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module lightcontrol(reset, clk, lightstate, light);
   input reset, clk;
   input [1:0] lightstate;
   output light;
   reg light;

   wire blink_reset, blink;
   reg blink_reset_lev;

   // Define flags for the light state
   parameter light_off = 2'b00;
   parameter light_on = 2'b01;
   parameter light_blink = 2'b10;

   // Convert the blink_reset signal to a pulse
   levpulse blink_reset_ltp (.clk(clk), .level(blink_reset_lev),
.pulse(blink_reset));
   // instantiate a 1Hz clock for blinking
   clkdiv light_div (.reset(blink_reset), .clkin(clk), .clkout(blink));

   // Control the light based on the input light state
   always @ (posedge clk) begin
         if (reset) light <= 1'b0;
         case (lightstate)
               light_off: begin
                     light <= 1'b0;
                     blink_reset_lev <= 1'b0;
               end
               light_on: begin
                     light <= 1'b1;
                     blink_reset_lev <= 1'b0;
               end
               light_blink: begin
                     blink_reset_lev <= 1'b1;
                     light <= (blink) ? ~light : light;
               end
               default: light <= 1'b0;
         endcase
   end

endmodule
```

Appendix J – Siren Generator (siren.v)

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    23:20:06 10/07/2007
// Design Name:
// Module Name:    siren
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module siren(reset, clk, siren_en, siren_out);
   input reset, clk, siren_en;
   output siren_out;
```

```
    reg [15:0] every_n;
    wire en_p, siren_pulse, sweep_clk;
    reg asc, siren, div_reset;

    // Define the sweeping endpoint frequencies
    parameter asc_l = 27000000/(2*500);
    parameter asc_h = 27000000/(2*667);
    parameter desc_l = 27000000/(2*400);
    parameter desc_h = 27000000/(2*500);
    // Define the sweeping speed
    parameter sweep_n_par = 27000000/6750;

    wire [15:0] sweep_n;
    assign sweep_n = sweep_n_par;

    // Also get the enable signal as a pulse
    levpulse en_rst(.clk(clk), .level(siren_en), .pulse(en_p));

    // Instantiate a variable clock to generate the siren audio frequency
    varclkdiv n_div(.reset(div_reset), .clk(clk), .every_n(every_n),
          .clkout(siren_pulse));

    // Instantiate a clock for sweeping the audio frequency
    varclkdiv sweep_div(.reset(reset), .clk(clk), .every_n(sweep_n),
          .clkout(sweep_clk));

    always @ (posedge clk) begin
          // Twiddle the audio output on the audio clock
          if (siren_pulse) siren <= ~siren;

          // Reset all values on a reset
          if (reset | en_p) begin
                every_n <= asc_l;
                asc <= 1'b1;
                div_reset <= 1'b1;
                siren <= 1'b0;
          end

          // If we've hit the high end of the ascending range, go to the descending
          // range
          else if (asc & (every_n <= asc_h)) begin
                every_n <= desc_h;
                asc <= 1'b0;
                div_reset <= 1'b1;
          end
          // If we've hit the low end of the descending range, go to the ascending
          // range
          else if (~asc & (every_n >= desc_l)) begin
                every_n <= asc_l;
                asc <= 1'b1;
                div_reset <= 1'b1;
          end
          // If we're ascending, sweep the frequency up
          else if (asc) begin
```

```
               every_n <= (sweep_clk) ? (every_n - 1) : every_n;
               div_reset <= 1'b0;
         end
         // If we're descending, sweep the frequency down
         else if (~asc) begin
               every_n <= (sweep_clk) ? (every_n + 1) : every_n;
               div_reset <= 1'b0;
         end
   end

   // Make sure audio is only output when the enable signal is asserted
   assign siren_out = (siren_en) ? siren : 1'b0;

endmodule
```

Appendix K – Variable Clock Divider (varclkdiv.v)

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    23:53:23 10/07/2007
// Design Name:
// Module Name:    varclkdiv
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module varclkdiv(reset, clk, every_n, clkout);
   input reset, clk;
   input [15:0] every_n;
   output clkout;
   reg clkout;

   reg [15:0] count;

   // Create a variable clock divider, making sure it can handle having it's
   // dividing value changed without being reset
   always @ (posedge clk) begin
         count <= ((count >= (every_n - 1)) | reset) ? 0 : count + 1;
         clkout <= (count >= (every_n - 1));
   end

endmodule
```