```
/*
 * Team Mini
 * 11.10.09
 * Board module
 *
 * Blake English
 * Nina Joshi
 * Peter Miller
 */


#ifndef _BOARD_H_
#define _BOARD_H_

Event_t BOARD_CheckForBoardEvent(void);
void BOARD_ResetGame(unsigned char finalDestination);
void BOARD_EndGame(void);
void BOARD_SetLight(unsigned char finalDestination);
void BOARD_ClearLight(unsigned char finalDestination);
void BOARD_Debug(void);

#endif
```

```
/*
 * Team Mini
 * 11.16.09
 * Board module
 *
 * Blake English
 * Nina Joshi
 * Peter Miller
 */


#include <me218_c32.h>
#include <stdio.h>
#include <ADS12.h>
#include <timers12.h>
#include "defs.h"
#include "events.h"
#include "board.h"

static Event_t lastEvent = NO_EVENT; //The previous event.

/* Prototypes */
static void BOARD_SetDestinationSign(unsigned char finalDestination);
static void BOARD_SetRightAsBank(void);
static void BOARD_SetLeftAsBank(void);
static void BOARD_SetLights(void);
static void BOARD_ClearLights(void);

/* Resets board with appropriate signs and lights. */
void BOARD_ResetGame(unsigned char finalDestination) {
        BOARD_SetDestinationSign(finalDestination); //Sets both signs since they are tied electronically
        BOARD_SetLights(); //Turn both building lights on
        //Photo interrupt circuits are automatically on
}

/* Checks and returns any events that have occurred. Returns a LEFT_SENSED or RIGHT_SENSED.
 * Can only detect one location at a time. */
Event_t BOARD_CheckForBoardEvent(void) {
        Event_t event = NO_EVENT;

        if (ADS12_ReadADPin(LEFT_PHOTO_BIT) > PHOTO_INTERRUPT_THRESHOLD) { //Jumps
to 2.5 V when blocked
            event = LEFT_SENSED;
        } else if (ADS12_ReadADPin(RIGHT_PHOTO_BIT) > PHOTO_INTERRUPT_THRESHOLD) {
//Jumps to 2.5 V when blocked
            event = RIGHT_SENSED;
        }

        if (event != lastEvent) { //Only returns event if it differs from lastEvent
            lastEvent = event;
            return event;
        }

        return NO_EVENT;
}
```

```
/* Clears all building lights (for the end of the game). */
void BOARD_EndGame(void) {
	BOARD_ClearLights();
}

/* Sets the right building as bank and left building as depot. */
static void BOARD_SetRightAsBank(void) {
	SETBIT(SIGN_PORT,SIGN_BIT);
}

/* Sets the left building as bank and right building as depot. */
static void BOARD_SetLeftAsBank(void) {
	CLEARBIT(SIGN_PORT,SIGN_BIT);
}

/* Sets signs according to building final destination parameter specified. */
static void BOARD_SetDestinationSign(unsigned char finalDestination) {
	switch (finalDestination) {
	    case LEFT:
			BOARD_SetLeftAsBank();
			break;
	    case RIGHT:
			BOARD_SetRightAsBank();
			break;
	    default:
			break;
	}
}

/* Sets both building lights on. */
static void BOARD_SetLights(void) {
	SETBIT(LEFT_LIGHT_PORT,LEFT_LIGHT_BIT);
	SETBIT(RIGHT_LIGHT_PORT,RIGHT_LIGHT_BIT);
}

/* Sets a specific building light on as specified by the final destination parameter. */
void BOARD_SetLight(unsigned char finalDestination) {
	switch (finalDestination) {
	    case LEFT:
			SETBIT(LEFT_LIGHT_PORT,LEFT_LIGHT_BIT);
			break;
	    case RIGHT:
			SETBIT(RIGHT_LIGHT_PORT,RIGHT_LIGHT_BIT);
			break;
	    default:
			break;
	}
}

/* Clears both building lights. */
static void BOARD_ClearLights(void) {
	CLEARBIT(LEFT_LIGHT_PORT,LEFT_LIGHT_BIT);
	CLEARBIT(RIGHT_LIGHT_PORT,RIGHT_LIGHT_BIT);
}

/* Clears a specific building light as specified by the final destination parameter. */
```

```
void BOARD_ClearLight(unsigned char finalDestination) {
	switch (finalDestination) {
	    case LEFT:
			CLEARBIT(LEFT_LIGHT_PORT,LEFT_LIGHT_BIT);
			break;
	    case RIGHT:
			CLEARBIT(RIGHT_LIGHT_PORT,RIGHT_LIGHT_BIT);
			break;
	    default:
			break;
	}
}

/* Board debug */
void BOARD_Debug(void) {
	printf("\nDebugging building signs, lights, and location sensors... Controls:\r\n");
	printf("l: left LED, r: right LED, b: both LEDs, c: clear both LEDs,\r\n");
	printf("d: set right as depot, k: set right as bank, q: quit\r\n");
	// Break out of loop when user hits 'q'
	for(;;) {
	    // Prints user specified building sensors.
	    while (!kbhit()) {
			Event_t event = BOARD_CheckForBoardEvent();
			if (event == LEFT_SENSED) {
				printf("Left location sensed\r\n");
			} else if (event == RIGHT_SENSED) {
				printf("Right location sensed\r\n");
			}
	    }
	    // Clears input character and returns from the function.
	    if (kbhit()) {
			// User selects options.
			char ch = getchar();
			switch (ch) {
				case 'l':
					printf("Left LED set\r\n");
					BOARD_SetLight(LEFT);
					break;
				case 'r':
					printf("Right LED set\r\n");
					BOARD_SetLight(RIGHT);
					break;
				case 'c':
					printf("LEDs cleared\r\n");
					BOARD_ClearLights();
					break;
				case 'b':
					printf("Both LEDs set\r\n");
					BOARD_SetLights();
					break;
				case 'd':
					printf("Set right as depot and left to bank\r\n");
					BOARD_SetLeftAsBank();
					break;
				case 'k':
					printf("Set right as bank and left to depot\r\n");
```

```
                                        BOARD_SetRightAsBank();
                                        break;
                                case 'q':
                                        //printf("QUIT\r\n");
                                        return;
                                default:
                                        //printf("ERROR\r\n");
                                        break;
                        }
                }
        }
}
```

```
/*
 * Team Mini
 * 11.18.09
 * Building module
 *
 * Blake English
 * Nina Joshi
 * Peter Miller
 */


#ifndef _DEFS_H_
#define _DEFS_H_

#include <bitdefs.h>
#include <me218_c32.h>

typedef enum {
        WHEEL_CENTERED,
        WHEEL_RIGHT,
        WHEEL_LEFT,
        WHEEL_FWD_BUTTON,
        WHEEL_REV_BUTTON,
        WHEEL_NO_BUTTON,
        WHEEL_BOTH_BUTTONS,
        WHEEL_NO_EVENT
} SteeringWheel_t;

/* Port E bits */ //Input only
#define SAFE_LEVER_BIT 0
#define SAFE_LEVER_DDR DDRE
#define SAFE_LEVER_PORT PORTE

#define SAFE_DIAL_BIT 1
#define SAFE_DIAL_DDR DDRE
#define SAFE_DIAL_PORT PORTE

/* Port T bits */
#define CAR_THROTTLE_FWD_BIT PWMS12_CHAN0 //Digital Output PWM (Tied to T1)
#define CAR_THROTTLE_FWD_DDR DDRT
#define CAR_THROTTLE_FWD_PORT PTT

#define CAR_THROTTLE_REV_BIT PWMS12_CHAN1 //Digital Output PWM (Tied to T0)
#define CAR_THROTTLE_REV_DDR DDRT
#define CAR_THROTTLE_REV_PORT PTT

#define FUEL_GAUGE_BIT PWMS12_CHAN2 //Digital Output PWM

#define IGNITION_LIGHT_BIT 3 //Digital Output
#define IGNITION_LIGHT_DDR DDRT
#define IGNITION_LIGHT_PORT PTT

#define IGNITION_BIT 4 //Digital Input
#define IGNITION_DDR DDRT
#define IGNITION_PORT PTT
```

```
/* Port M bits */
#define OPTO_INPUT_BIT 0 //Opto isolator input Digital Input
#define OPTO_INPUT_DDR DDRM
#define OPTO_INPUT_PORT PTM

#define OPTO_OUTPUT_BIT 1 //Opto isolator output Digital Output
#define OPTO_OUTPUT_DDR DDRM
#define OPTO_OUTPUT_PORT PTM

#define STEERING_WHEEL_FWD_SWITCH 2 //Steering wheel forward switch Digital Input
#define STEERING_WHEEL_FWD_SWITCH_DDR DDRM
#define STEERING_WHEEL_FWD_SWITCH_PORT PTM

#define STEERING_WHEEL_REV_SWITCH 3 //Steering wheel reverse switch Digital Input
#define STEERING_WHEEL_REV_SWITCH_DDR DDRM
#define STEERING_WHEEL_REV_SWITCH_PORT PTM

#define CAR_STEERING_RIGHT_BIT 4 //Car steering right Digital Output
#define CAR_STEERING_RIGHT_DDR DDRM
#define CAR_STEERING_RIGHT_PORT PTM

#define CAR_STEERING_LEFT_BIT 5 //Car steering left Digital Output
#define CAR_STEERING_LEFT_DDR DDRM
#define CAR_STEERING_LEFT_PORT PTM

/* Port AD bits */
#define AD_Init_String "OOOOAAAA" //bits "76543210"

#define RIGHT_PHOTO_BIT 1 //Right building photo interrupt Analog Input
#define RIGHT_PHOTO_DDR DDRAD
#define RIGHT_PHOTO_PORT PTAD

#define LEFT_PHOTO_BIT 0 //Left building photo interrupt Analog Input
#define LEFT_PHOTO_DDR DDRAD
#define LEFT_PHOTO_PORT PTAD

#define SAFE_PHOTO_BIT 2 //Safe photo interrupt Analog Input
#define SAFE_PHOTO_DDR DDRAD
#define SAFE_PHOTO_PORT PTAD

#define STEERING_WHEEL_POT_PIN 3 //Steering wheel potentiometer position Analog Input
#define STEERING_WHEEL_POT_DDR DDRAD
#define STEERING_WHEEL_POT_PORT PTAD

#define WHEEL_VIBRATE_BIT 4 //Wheel vibrate Digital Output
#define WHEEL_VIBRATE_DDR DDRAD
#define WHEEL_VIBRATE_PORT PTAD

#define RIGHT_LIGHT_BIT 5 //Right building light Digital Output
#define RIGHT_LIGHT_DDR DDRAD
#define RIGHT_LIGHT_PORT PTAD

#define LEFT_LIGHT_BIT 6 //Left building light Digital Output
#define LEFT_LIGHT_DDR DDRAD
#define LEFT_LIGHT_PORT PTAD
```

```
#define SIGN_BIT 7 //Board sign servo (controls both) Digital Output
#define SIGN_DDR DDRAD
#define SIGN_PORT PTAD


/* Photo interrupt */
#define PHOTO_INTERRUPT_THRESHOLD 200 //Board photo interrupt
/* Steering */
#define STEERING_POT_LEFT_THRESH 330 //center is about 410
#define STEERING_POT_RIGHT_THRESH 470
#define STEERING_POT_HISTER 50
/* Throttle */
#define THROTTLE_VALUE 85
/* Optoisolator */
//#define OPTO_ON_THRESHOLD_LO 50 //Opto isolator on time (min)
/* PWM */
#define PWMS12_500US 8222
#define PWMS12_1000US 12318
/* Fuel gauge */
#define PWM_FUEL_GAUGE_START_VALUE 100
#define PWM_FUEL_GAUGE_END_VALUE 10
/* Celebration chimes */
#define MAX_CELEBRATION_CHIMES 10


/* Timers */
/* Allow game to last 2 min. */
#define END_TIMER 1
#define END_TIME 45000
/* Optoisolator pulse length. */
#define OPTO_TIMER 2
#define OPTO_TIME 75
/* Blinking light timer */
#define BLINK_TIMER 3
#define BLINK_TIME 75
/* Failure buzzer timer */
#define BUZZER_TIMER 0
#define BUZZER_TIME 3000
/* Celebration chime timer */
#define CELEBRATION_TIMER 4
#define CELEBRATION_TIME 3000

/* Convenience */
#define FALSE 0
#define TRUE 1

#define LOW 0
#define HIGH 1

#define LEFT 0
#define RIGHT 1

#define SETBIT(ADDRESS,BIT) (ADDRESS |= (1<<BIT)) //set bit (high)      //output
#define CLEARBIT(ADDRESS,BIT) (ADDRESS &= ~(1<<BIT)) //clear bit (low)  //input
#define CHECKBIT(ADDRESS,BIT) (ADDRESS & (1<<BIT)) //test single bit in I/O location
#endif
```

```
/*
 * Team Mini
 * 11.13.09
 * Driving module
 *
 * Blake English
 * Nina Joshi
 * Peter Miller

 */


#ifndef _DRIVING_H_
#define _DRIVING_H_

Event_t DRIVING_CheckForDrivingEvent(void);
void DRIVING_InitFuelGauge(void);
void DRIVING_SetFuelGauge(void);
void DRIVING_SetFuelGaugeFull(void);
void DRIVING_SetIgnitionLight(void);
void DRIVING_ClearIgnitionLight(void);
void DRIVING_DriveCar(void);
void DRIVING_SetWheelVibrate(void);
void DRIVING_ClearWheelVibrate(void);
void STEERING_initCarControlPins(void);
void STEERING_initSteeringWheelSwitchPins(void);
void STEERING_stopCarMotor(void);
void STEERING_centerCarSteering(void);
void DRIVING_Debug(void);

#endif
```

```
/*
 * Team Mini
 * 11.14.09
 * Driving module
 *
 * Blake English
 * Nina Joshi
 * Peter Miller
 */


#include <me218_c32.h>
#include <stdio.h>
#include <ADS12.h>
#include <PWMS12.h>
#include <timers12.h>
#include "defs.h"
#include "events.h"
#include "driving.h"

static Event_t lastEvent = NO_EVENT; //The previous event.
static unsigned int gasGaugeTimerTick;
static unsigned int gasGaugeTime;
static unsigned int gasGaugeLastTime;

/* Prototypes */
static void STEERING_leftCarSteering(void);
static void STEERING_rightCarSteering(void);
static void STEERING_fwdCarMotor(void);
static void STEERING_revCarMotor(void);
static short STEERING_getSteeringWheelPotValue(void);
static SteeringWheel_t STEERING_getSteeringWheelPosition(void);
static SteeringWheel_t STEERING_getSteeringWheelThrottleSwitchStates(void);

/* Checks and returns any events that have occurred. Returns an IGNITION_ON if button pressed. */
Event_t DRIVING_CheckForDrivingEvent(void) {
        Event_t event = NO_EVENT;

        if(CHECKBIT(IGNITION_PORT,IGNITION_BIT) == 0) {
          event = IGNITION_ON;
        }

        if (event != lastEvent) { //Only returns event if it differs from lastEvent
            lastEvent = event;
            return event;
        }

        return NO_EVENT;
}

/* Finds a step that divides the end timer value by the limits of the servo PWM values.
 * This step is used later to calculate the servo PWM value based on the current time.
 * For example, 50-100% PWM values corresponding to 1-450000 timer values. */
void DRIVING_InitFuelGauge(void) {
        gasGaugeLastTime = TMRS12_GetTime();
        gasGaugeTime = 0;
```

```
	gasGaugeTimerTick = END_TIME / ( PWM_FUEL_GAUGE_START_VALUE -
PWM_FUEL_GAUGE_END_VALUE );
}

/* Sets the fuel gauge servo voltage to the appropriate value for the time. */
void DRIVING_SetFuelGauge(void) {
	unsigned int curTime;
	unsigned int TimeStep;
	unsigned char dutycycle;

	curTime = TMRS12_GetTime();
	TimeStep = curTime - gasGaugeLastTime;
	gasGaugeTime += TimeStep;

	dutycycle = PWM_FUEL_GAUGE_START_VALUE-(gasGaugeTime/gasGaugeTimerTick);

	if (dutycycle < 1) {
	    dutycycle = 1;
	}

	PWMS12_SetDuty(dutycycle, FUEL_GAUGE_BIT);
	gasGaugeLastTime = curTime;
}

/* Set the fuel gauge to full, when the time is 0. */
void DRIVING_SetFuelGaugeFull(void) {
	PWMS12_SetDuty(PWM_FUEL_GAUGE_START_VALUE, FUEL_GAUGE_BIT);
}

/* Set the ignition light on. */
void DRIVING_SetIgnitionLight(void) {
	SETBIT(IGNITION_LIGHT_PORT,IGNITION_LIGHT_BIT);
}

/* Clear the ignition light. */
void DRIVING_ClearIgnitionLight(void) {
	CLEARBIT(IGNITION_LIGHT_PORT,IGNITION_LIGHT_BIT);
}

/* Figures out which driving state we are in, and calls the corresponding function to handle the event. */
void DRIVING_DriveCar(void) {
	switch(STEERING_getSteeringWheelThrottleSwitchStates()) { //Throttle control
	    case WHEEL_REV_BUTTON:
			printf("REV\r\n");
			STEERING_revCarMotor();
			break;
	    case WHEEL_FWD_BUTTON:
			printf("FWD\r\n");
			STEERING_fwdCarMotor();
			break;
	    case WHEEL_BOTH_BUTTONS:
			//printf("BOTH\r\n");
			STEERING_stopCarMotor();
			break;
	    case WHEEL_NO_BUTTON:
			//printf("STOP\r\n");
```

```
                STEERING_stopCarMotor();
                break;
        default:
                break;
    }

    switch(STEERING_getSteeringWheelPosition()) { //Steering direction control
        case WHEEL_RIGHT:
                printf("wheel right\r\n");
                STEERING_rightCarSteering();
                break;
        case WHEEL_LEFT:
                printf("wheel left\r\n");
                STEERING_leftCarSteering();
                break;
        case WHEEL_CENTERED:
                printf("wheel centered\r\n");
                STEERING_centerCarSteering();
                break;
        default:
                break;
    }
}

/* Sets the steering wheel vibrate motor on. */
void DRIVING_SetWheelVibrate(void) {
    SETBIT(WHEEL_VIBRATE_PORT,WHEEL_VIBRATE_BIT);
}

/* Clears the steering wheel vibrate motor. */
void DRIVING_ClearWheelVibrate(void) {
    CLEARBIT(WHEEL_VIBRATE_PORT,WHEEL_VIBRATE_BIT);
}

/* Initializes steering wheel input pins DDR. */
void STEERING_initSteeringWheelSwitchPins(void) {
    STEERING_WHEEL_FWD_SWITCH_DDR = CLEARBIT(
STEERING_WHEEL_FWD_SWITCH_DDR, STEERING_WHEEL_FWD_SWITCH); //input
    STEERING_WHEEL_REV_SWITCH_DDR = CLEARBIT(
STEERING_WHEEL_FWD_SWITCH_DDR, STEERING_WHEEL_REV_SWITCH);  //input
}

/* Initializes car control output pins DDR. */
void STEERING_initCarControlPins(void) {
    CAR_STEERING_LEFT_DDR = SETBIT(CAR_STEERING_LEFT_DDR,
CAR_STEERING_LEFT_BIT); //output
    CAR_STEERING_RIGHT_DDR = SETBIT(CAR_STEERING_RIGHT_DDR,
CAR_STEERING_RIGHT_BIT); //output
    WHEEL_VIBRATE_DDR = SETBIT(WHEEL_VIBRATE_DDR,WHEEL_VIBRATE_BIT);
//output
}

/* Stops car motor. */
void STEERING_stopCarMotor(void) {
    PWMS12_SetDuty(0, CAR_THROTTLE_FWD_BIT);
    PWMS12_SetDuty(0, CAR_THROTTLE_REV_BIT);
```

```
}

/* Sets forward car motor on. */
static void STEERING_fwdCarMotor(void) {
	PWMS12_SetDuty(THROTTLE_VALUE, CAR_THROTTLE_FWD_BIT);
	PWMS12_SetDuty(0, CAR_THROTTLE_REV_BIT);
}

/* Sets reverse car motor on. */
static void STEERING_revCarMotor(void) {
	PWMS12_SetDuty(0, CAR_THROTTLE_FWD_BIT);
	PWMS12_SetDuty(THROTTLE_VALUE, CAR_THROTTLE_REV_BIT);
}

/* Sets car steering to center. */
void STEERING_centerCarSteering(void) {
	CAR_STEERING_LEFT_PORT =
CLEARBIT(CAR_STEERING_LEFT_PORT,CAR_STEERING_LEFT_BIT);
	CAR_STEERING_RIGHT_PORT =
CLEARBIT(CAR_STEERING_RIGHT_PORT,CAR_STEERING_RIGHT_BIT);
}

/* Sets car steering for turning left. */
static void STEERING_leftCarSteering(void) {
	CAR_STEERING_LEFT_PORT =
SETBIT(CAR_STEERING_LEFT_PORT,CAR_STEERING_LEFT_BIT);
	CAR_STEERING_RIGHT_PORT =
CLEARBIT(CAR_STEERING_RIGHT_PORT,CAR_STEERING_RIGHT_BIT);
}

/* Sets car steering for turning right. */
static void STEERING_rightCarSteering(void) {
	CAR_STEERING_LEFT_PORT =
CLEARBIT(CAR_STEERING_LEFT_PORT,CAR_STEERING_LEFT_BIT);
	CAR_STEERING_RIGHT_PORT =
SETBIT(CAR_STEERING_RIGHT_PORT,CAR_STEERING_RIGHT_BIT);
}

/* Figures out current steering wheel position: centered, left, or right. */
static SteeringWheel_t STEERING_getSteeringWheelPosition(void) {
	SteeringWheel_t wheelPosition = WHEEL_CENTERED;
	short SteeringPotValue;

	SteeringPotValue = STEERING_getSteeringWheelPotValue();

	if(SteeringPotValue > STEERING_POT_RIGHT_THRESH) {
	    wheelPosition = WHEEL_RIGHT;
	} else if (SteeringPotValue < STEERING_POT_LEFT_THRESH) {
	    wheelPosition = WHEEL_LEFT;
	} else if (SteeringPotValue < STEERING_POT_RIGHT_THRESH && SteeringPotValue >
STEERING_POT_LEFT_THRESH) {
	    wheelPosition = WHEEL_CENTERED;
	}

	return wheelPosition;
}
```

```
/* Reads steering wheel analog input. */
static short STEERING_getSteeringWheelPotValue(void) {
	return ADS12_ReadADPin(STEERING_WHEEL_POT_PIN);
}

/* Figures out current throttle switch state: forward, reverse, both, none. */
static SteeringWheel_t STEERING_getSteeringWheelThrottleSwitchStates(void) {
	SteeringWheel_t switchState = WHEEL_NO_BUTTON;
	char switchFWD;
	char switchREV;

	switchFWD = CHECKBIT(STEERING_WHEEL_FWD_SWITCH_PORT,
STEERING_WHEEL_FWD_SWITCH);
	switchREV = CHECKBIT(STEERING_WHEEL_REV_SWITCH_PORT,
STEERING_WHEEL_REV_SWITCH);

	if (switchFWD == 0 && switchREV == 0) {
	    switchState = WHEEL_BOTH_BUTTONS;
	} else if (switchFWD != 0 && switchREV != 0) {
	    switchState = WHEEL_NO_BUTTON;
	} else if (switchFWD == 0 && switchREV != 0) {
	    switchState = WHEEL_FWD_BUTTON;
	} else if (switchFWD != 0 && switchREV == 0) {
	    switchState = WHEEL_REV_BUTTON;
	}

	return switchState;
}

/* Driving debug */
void DRIVING_Debug(void) {
	printf("\nDebugging driving... Controls:\r\n");
	printf("h: ignition light on, l:ignition light off, s: wheel vibrate on,\r\n");
	printf("c: wheel vibrate off, f: car forward, b:car reverse, t: car left,\r\n");
	printf("r: car right, q: quit\r\n");
	// Break out of loop when user hits 'q'
	for(;;) {
	    // Prints user specified building sensors.
	    while (!kbhit()) {
			Event_t event = DRIVING_CheckForDrivingEvent();
			if (event == IGNITION_ON) {
				printf("Ignition on\r\n");
			}
			DRIVING_DriveCar();
	    }
	    // Clears input character and returns from the function.
	    if (kbhit()) {
			// User selects options.
			char ch = getchar();
			switch (ch) {
				case 'h':
					printf("Ignition light on\r\n");
					DRIVING_SetIgnitionLight();
					break;
				case 'l':
```

```
                    printf("Ignition light off\r\n");
                    DRIVING_ClearIgnitionLight();
                    break;
                case 's':
                    printf("Wheel vibrate on\r\n");
                    DRIVING_SetWheelVibrate();
                    break;
                case 'c':
                    printf("Wheel vibrate off\r\n");
                    DRIVING_ClearWheelVibrate();
                    break;
                case 'f':
                    printf("Car forward motor\r\n");
                    STEERING_fwdCarMotor();
                    break;
                case 'b':
                    printf("Car right steering\r\n");
                    STEERING_revCarMotor();
                    break;
                case 't':
                    printf("Car left steering\r\n");
                    STEERING_leftCarSteering();
                    break;
                case 'r':
                    printf("Car right steering\r\n");
                    STEERING_rightCarSteering();
                    break;
                case 'q':
                    //printf("QUIT\r\n");
                    return;
                default:
                    //printf("ERROR\r\n");
                    break;
            }
        }
    }
}
```

```
/*
 * Team Mini
 * 11.18.09
 * Event detection module
 *
 * Blake English
 * Nina Joshi
 * Peter Miller
 */


#ifndef _EVENTS_H_
#define _EVENTS_H_

typedef enum {
	NO_EVENT,
	OPTOISOLATOR_ON,
	FORCE_START_HIT,
	IGNITION_ON,
	FINAL_DESTINATION_REACHED,
	END_TIMER_EXPIRED,
	OPTO_TIMER_EXPIRED,
	SAFE_CLEARED,
	LEFT_SENSED,
	RIGHT_SENSED,
	SET_UP_TIMER_EXPIRED,
	BLINK_TIMER_EXPIRED,
	BUZZER_TIMER_EXPIRED,
	CELEBRATION_TIMER_EXPIRED
} Event_t;

/* Helper function to check if any timers have expired since no separate routine for that. */
static Event_t CheckForTimerExpired(void);

Event_t EVENTS_CheckForEvents(void);
void EVENTS_Debug(void);

#endif
```

```
/*
 * Team Mini
 * 11.18.09
 * Event detection module
 *
 * Blake English
 * Nina Joshi
 * Peter Miller
 */


#include <me218_c32.h>
#include <stdio.h>
#include "events.h"
#include "defs.h"
#include "timers12.h"
#include "opto.h"
#include "driving.h"
#include "board.h"
#include "safe.h"

/* Checks and returns any events that have occurred. */
Event_t EVENTS_CheckForEvents(void) {
	Event_t event = NO_EVENT;

	// Check if a timer expired
	event = CheckForTimerExpired();
	if (event != NO_EVENT)
	    return event;

	//opto isolator event: optoisolator, force start
	event = OPTO_CheckForStartEvent();
	if (event != NO_EVENT)
	    return event;

	//driving event: ignition
	event = DRIVING_CheckForDrivingEvent();
	if (event != NO_EVENT)
	    return event;

	//board event: right or left sensed
	event = BOARD_CheckForBoardEvent();
	if (event != NO_EVENT)
	    return event;

	//safe event: safe cleared
	event = SAFE_CheckForSafeEvent();
	if (event != NO_EVENT)
	    return event;

	return event;
}

/* Checks all timers and if any has expired, clear the timer and return the corresponding event. */
static Event_t CheckForTimerExpired() {
	if(TMRS12_IsTimerExpired(END_TIMER)) {
```

```
            TMRS12_ClearTimerExpired(END_TIMER);
            return END_TIMER_EXPIRED;
        }
        if(TMRS12_IsTimerExpired(OPTO_TIMER)) {
            TMRS12_ClearTimerExpired(OPTO_TIMER);
            return OPTO_TIMER_EXPIRED;
        }
        if(TMRS12_IsTimerExpired(BUZZER_TIMER)) {
            TMRS12_ClearTimerExpired(BUZZER_TIMER);
            return BUZZER_TIMER_EXPIRED;
        }
        if(TMRS12_IsTimerExpired(BLINK_TIMER)) {
            TMRS12_ClearTimerExpired(BLINK_TIMER);
            return BLINK_TIMER_EXPIRED;
        }
        if(TMRS12_IsTimerExpired(CELEBRATION_TIMER)) {
            TMRS12_ClearTimerExpired(CELEBRATION_TIMER);
            return CELEBRATION_TIMER_EXPIRED;
        }
        return NO_EVENT;
}

/* Events debug. */
void EVENTS_Debug(void) {
        printf("\nDebugging events...Controls:\r\n");
        printf("e: end timer, o: opto timer, z: buzzer timer, b: blink timer, c: celebration timer, q: quit \r\n");
        // Break out of loop when user hits 'q'
        for(;;) {
            if (kbhit()) {
                        // User selects options.
                        char ch = getchar();
                        switch (ch) {
                                    case 'e':
                                                printf("Setting end timer...\r\n");
                                                TMRS12_InitTimer(END_TIMER, END_TIME);
                                                break;
                                    case 'o':
                                                printf("Setting opto timer...\r\n");
                                                TMRS12_InitTimer(OPTO_TIMER, OPTO_TIME);
                                                break;
                                    case 'z':
                                                printf("Setting buzzer timer...\r\n");
                                                TMRS12_InitTimer(BUZZER_TIMER, BUZZER_TIME);
                                                break;
                                    case 'b':
                                                printf("Setting blinking timer...\r\n");
                                                TMRS12_InitTimer(BLINK_TIMER, BLINK_TIME);
                                                break;
                                    case 'c':
                                                printf("Setting celebration timer...\r\n");
                                                TMRS12_InitTimer(CELEBRATION_TIMER,
CELEBRATION_TIME);
                                                break;
                                    case 'q':
                                                return;
                                    default:
```

```
                                        break;
                        }
                }
                // Print out event that has occurred.
                switch(EVENTS_CheckForEvents()) {
                        case NO_EVENT:
                                //printf("No event\r\n");
                                break;
                        case OPTOISOLATOR_ON:
                                printf("Optoisolator on\r\n");
                                break;
                        //Will never be hit since the switch is electronically hooked up to the opto input.
                        //This is here to allow for future flexibility.
                        case FORCE_START_HIT:
                                printf("Force start hit\r\n");
                                break;
                        case IGNITION_ON:
                                printf("Ignition on\r\n");
                                break;
                        case END_TIMER_EXPIRED:
                                printf("End timer expired\r\n");
                                break;
                        case OPTO_TIMER_EXPIRED:
                                printf("Opto timer expired\r\n");
                                break;
                        case SAFE_CLEARED:
                                printf("Safe Cleared\r\n");
                                break;
                        case LEFT_SENSED:
                                printf("Left location sensed\r\n");
                                break;
                        case RIGHT_SENSED:
                                printf("Right location sensed\r\n");
                                break;
                        case BUZZER_TIMER_EXPIRED:
                                printf("Buzzer timer expired\r\n");
                                break;
                        case BLINK_TIMER_EXPIRED:
                                printf("Blinking timer expired\r\n");
                                break;
                        case CELEBRATION_TIMER_EXPIRED:
                                printf("Celebration timer expired\r\n");
                                break;
                        default:
                                printf("Error: undefined event!\r\n");
                                break;
                }
        }
}
```

```
/*
 * Team Mini
 * 11.18.09
 * Main program
 *
 * Blake English
 * Nina Joshi
 * Peter Miller
 */


#include <me218_c32.h>
#include <stdio.h>
#include <PWMS12.h>
#include <ADS12.h>
#include <timers12.h>
#include "defs.h"
#include "events.h"
#include "state.h"
#include "board.h"
#include "opto.h"
#include "driving.h"
#include "safe.h"

/* Prototypes */
static void MAIN_Debug(void);
static void MAIN_Init(void);

void main (void) {
        TMRS12_Init(TMRS12_RATE_1MS); //Initialize timers
        PWMS12_Init(); //Initialize routine for the PWMS functions
        MAIN_Init(); //Initialize ports

        //Uncomment the following line to debug and not run the state machine.
        //MAIN_Debug();

        //Initialize the state machine and loop forever to run it.
        STATE_InitStateMachine();
        for(;;) {
            STATE_RunStateMachine(EVENTS_CheckForEvents());
        }
}

/* Initialize port data directions for outputs and inputs, initialize pwms, and set opto output low. */
static void MAIN_Init(void) {
        SETBIT(OPTO_OUTPUT_DDR,OPTO_OUTPUT_BIT); //output
        CLEARBIT(OPTO_INPUT_DDR,OPTO_INPUT_BIT); //input
        SETBIT(LEFT_LIGHT_DDR,LEFT_LIGHT_BIT); //output
        SETBIT(RIGHT_LIGHT_DDR,RIGHT_LIGHT_BIT); //output
        ADS12_Init(AD_Init_String); //ad port
        CLEARBIT(IGNITION_DDR,IGNITION_BIT); //input
        SETBIT(IGNITION_LIGHT_DDR,IGNITION_LIGHT_BIT); //output
        STEERING_initCarControlPins();
        STEERING_initSteeringWheelSwitchPins();

        //No need to configure Port T 0 and 1 since PWMS does this. Direction set as an output.
```

```
        PWMS12_SetPeriod(PWMS12_500US, PWMS12_GRP0); //t0 and t1
        PWMS12_SetPeriod(PWMS12_500US, PWMS12_GRP1); //t2
        PWMS12_SetDuty(0, CAR_THROTTLE_FWD_BIT); //car throttle forward off
        PWMS12_SetDuty(0, CAR_THROTTLE_REV_BIT); //car throttle reverse off

        CLEARBIT(OPTO_OUTPUT_PORT,OPTO_OUTPUT_BIT); //Set opto output to low at the start
}

/* Will always loop as it runs debugging tests for each module. These debugging
 * tests are ONLY for debugging and have blocking code. */
static void MAIN_Debug(void) {
        printf("\nDebugging...\r\n");
        printf("\nNote that debugging is case sensitive.\r\n");
        for(;;) {
            DRIVING_Debug();
            BOARD_Debug();
            OPTO_Debug();
            EVENTS_Debug();
            //SAFE_Debug();
        }
}
```

```
/*
 * Team Mini
 * 11.11.09
 * Optoisolator module
 *
 * Blake English
 * Nina Joshi
 * Peter Miller
 */


#ifndef _OPTO_H_
#define _OPTO_H_

Event_t OPTO_CheckForStartEvent(void);
void OPTO_PulseOutputHi(void);
void OPTO_PulseOutputLo(void);
void OPTO_Debug(void);

#endif
```

```
/*
 * Team Mini
 * 11.16.09
 * Opto module
 *
 * Blake English
 * Nina Joshi
 * Peter Miller
 */


#include <me218_c32.h>
#include <stdio.h>
#include <ADS12.h>
#include <timers12.h>
#include "defs.h"
#include "events.h"
#include "opto.h"

Event_t lastEvent = NO_EVENT; //The previous event.
Event_t thisEvent = NO_EVENT; //The current event.

/* Checks and returns any events that have occurred. Returns a OPTOISOLATOR_ON event
 * if the phototransistor signal is not zero. */
Event_t OPTO_CheckForStartEvent(void) {
        Event_t event = NO_EVENT;

        if (CHECKBIT(OPTO_INPUT_PORT,OPTO_INPUT_BIT) != 0) { //Does not check if within time
constraints: 50ms to 100ms
            event = OPTOISOLATOR_ON;
        }

        if (event != lastEvent) { //Only returns event if it differs from lastEvent
            lastEvent = event;
            return event;
        }

        return NO_EVENT;
}

/* Sets the opto ouptut to high. */
void OPTO_PulseOutputHi(void) {
        SETBIT(OPTO_OUTPUT_PORT,OPTO_OUTPUT_BIT);
}

/* Clears the opto output to low. */
void OPTO_PulseOutputLo(void) {
        CLEARBIT(OPTO_OUTPUT_PORT,OPTO_OUTPUT_BIT);
}

/* Opto debug. */
void OPTO_Debug(void) {
        printf("\nDebugging optoisolator...Controls:\r\n");
        printf("q: quit\r\n");
        // Break out of loop when user hits 'q'
        for(;;) {
```

```
		// Prints beacon value until user quits.
		while (!kbhit()) {
			Event_t event = OPTO_CheckForStartEvent();
			if(event == OPTOISOLATOR_ON) {
				printf("Optoisolator ON\r\n");
			}
		}
		// Clears input character and returns from the function.
		if (kbhit()) {
			// User selects options.
			char ch = getchar();
			switch (ch) {
				case 'q':
					//printf("QUIT\r\n");
					return;
				default:
					//printf("ERROR\r\n");
					break;
			}
		}
	}
}
```

```
/*****************************************************************************
 Module
      SafeEvent.h

 Description
      This is the header file for Safe module.

 Notes


 History
 When          Who What/Why
 -------------- --- --------
 10/30/2009     pm  File started
 11/17/2009     pm  File completed
*****************************************************************************/

#ifndef SAFEEVENT_H
#define SAFEEVENT_H

#include <Bin_Const.h>
#include "events.h"

typedef enum  { SAFE_NO_EVENT,
            RESET_SAFE,
            CRACK_FIRST_NUM,
            CRACK_SECOND_NUM,
            CRACK_THIRD_NUM,
            SAFE_CRACKED,
            OPEN_SAFE,
            ENTER_SAFE,
            CLOSE_SAFE,
            END_GAME,

            TIMER_EXPIRED,

            TICK,
            TARGET_REACHED,
            SAFE_OPENED,
            SAFE_CLOSED,
            SAFE_ENTERED,

            OVERSHOOT,
            NEXT_STAGE,

} SafeEvent_t;


Event_t    SAFE_CheckForSafeEvent(void);
void       SAFE_CelebrateWin(void);
void       SAFE_Service_CloseSafeModular(void);
void       SAFE_Init(void);
void       SAFE_Main(void);
```

```
#endif   /* SAFEEVENT_H */

/*------------------------------ End of file ------------------------------*/
```

```
/*****************************************************************************
 Info:
          Peter Miller
          ME218, Project
          Due 11.18.09

 Module
  SafeModule.c

 Description
  This module contains all necessary functions to receive and respond to user
  inputs at the safe module for TeamMini's Italian Job Flicker.

 Notes


 History
 When          Who What/Why
 -------------- --- --------
 11/06/09      pm  Pseudocode for SafeMain()
 11/12/09      pm  All SafeMain() functions completed and working
 11/17/09      pm  Additional celebration, init, and reset routines completed

*****************************************************************************/


/*---------------- Module Includes --------------------------*/
#include <stdio.h>
#include <timers12.h>
#include <me218_c32.h>
#include <timers12.h>
#include <ADS12.h>
#include "safe.h"
#include "defs.h"
#include "events.h"


/*---------------- Module Defines ---------------------------*/
//#define TEST_SHIFT_REGISTER
//#define TEST_SWITCH
//#define TEST_TIMER

#define TIMER_RATE              TMRS12_RATE_1MS

#define OFF                     '0'
#define ON                      '1'

#define REGISTERS               8
#define REGISTER_CONTROL_PINS     PTT
#define SERIAL_DATA_IN_HI       BIT5HI    // SER, Pin 14 TO T5
#define SerialDataInLO          BIT5LO
#define SERIAL_DATA_PAUSE       2
#define SERIAL_CLOCK_HI         BIT6HI    // SRCLK, Pin 11 TO T6
#define SERIAL_CLOCK_LO         BIT6LO
#define SERIAL_CLOCK_PAUSE      2
#define REGISTER_CLOCK_HI       BIT7HI    // RCLK, Pin 12 TO T7
```

```
#define REGISTER_CLOCK_LO         BIT7LO
#define REGISTER_CLOCK_PAUSE      2

#define SPIN_CW_LED               7        // Q7, Pin 7
#define SPIN_CCW_LED              6        // Q6, Pin 6
#define FIRST_NUM_IND_LED         5        // Q5, Pin 5
#define SECOND_NUM_IND_LED        4        // Q4, Pin 4
#define SAFE_LOCKED               3        // Q3, Pin 3
#define SAFE_UNLOCKED             2        // Q2, Pin 2
#define TICKER_SOLENOID           0        // Q1, Pin 1
#define LOCK_SOLENOID             1        // Q0, Pin 15

#define LOCK_SOLENOID_ENGAGED     ON
#define LOCK_SOLENOID_DISENGAGED  OFF

#define FLASH_TIMER               7
#define OVERSHOOT_TIMER           6
#define OVERSHOOT_FLASH_TIMER     5
#define PING_TIMER        7

#define MAX_DIAL_POSITIONS        15
#define MIN_DIAL_POSITIONS        5
#define MAX_OVERSHOOT_TICKS       0
#define OVERSHOOT_FLASHES         10

#define SAFE_DIAL_SWITCH          PTM       // SWITCH TO M3
#define SAFE_DIAL_SWITCH_HI       BIT3HI
#define SAFE_DIAL_SWITCH_LO       BIT3LO

#define SAFE_OPEN_SWITCH          PTM       // SWITCH TO M4
#define SAFE_OPEN_SWITCH_HI       BIT4HI
#define SAFE_OPEN_SWITCH_LO       BIT4LO

//#define SAFE_ENTERED_INTERUPT     PTM       // SWITCH TO M5
//#define SAFE_ENTERED_INTERUPT_HI  BIT5HI
//#define SAFE_ENTERED_INTERUPT_LO  BIT5LO


/*---------------- Constants ----------------------------------*/
const unsigned int flashTime            = 600;    // ms
const unsigned int overshootTime        = 500;   // ms
const unsigned int overshootFlashTime   = 125;
const unsigned int celebrationTime      = 150;

/*---------------- Variables ----------------------------------*/
static SafeEvent_t  currentStage;

static char         registerStates[9]   = {OFF,OFF,OFF,OFF,OFF,OFF,OFF,OFF,'\0'};

static int          DialTargetTicks     = 0;

static int          UserTicks           = 0;



/*---------------- Function Prototypes ----------------------*/
```

```
void       pauseRegister(int);
void       updateRegister(void);
void       Service_SafeModuleReset(void);
void       Service_GenerateTargetTicks(void);
SafeEvent_t getCurrentStage(void);
SafeEvent_t Event_FlashTimerExpired(void);
void       Service_MonitorLamps(void);
SafeEvent_t Event_DialTick(void);
void       Service_UpdateUserTicks(void);
SafeEvent_t Event_PlayerReachedTarget(void);
void       Service_PrepareForOvershoot(void);
SafeEvent_t Event_OverShootTimerExpired(void);
void       Service_UpdateStage(void);
void       Service_RespondToOvershoot(void);
void       Service_DisengageLockSolenoid(void);
SafeEvent_t Event_SafeDoorState(void);
SafeEvent_t Event_SafeEntered(void);
void       SAFE_Service_CloseSafeModule(void);
Event_t    SAFE_CheckForSafeEvent(void);
void       SAFE_Init(void);
void       SAFE_Main(void);



/*---------------- Module Code ------------------------------*/


/*************************************************
 * Function: updateRegister()
 * Creator:   Peter Miller              Date: 11/7/2009
 *
 * This function updates the shift register based on list of values
 * that are stored in an array.  The array contains the desired pin
 * state for all shift register outputs; these desired states are
 * updated throughout the code.
 *
 *************************************************************/

void updateRegister(void){
  int reg = 0;

  for (reg = 0; reg < REGISTERS; reg++){                    // Begin loop to shift out data
    if (registerStates[reg] == OFF){                        //   1) Place MSB of NewOutputs on SerialDataOut port
line
      REGISTER_CONTROL_PINS = REGISTER_CONTROL_PINS & SerialDataInLO;
    } else {
      REGISTER_CONTROL_PINS = REGISTER_CONTROL_PINS | SERIAL_DATA_IN_HI;
    }
    pauseRegister(SERIAL_DATA_PAUSE);                       //   2) Pause...


    REGISTER_CONTROL_PINS = REGISTER_CONTROL_PINS | SERIAL_CLOCK_HI;      //   3) Pulse
SerialClock high then low to clock data into the ‘595
    pauseRegister(SERIAL_CLOCK_PAUSE);
```

```
   REGISTER_CONTROL_PINS = REGISTER_CONTROL_PINS & SERIAL_CLOCK_LO;      //   4) Shift
NewOutputs 1 bit to the left
  }

  pauseRegister(SERIAL_CLOCK_PAUSE);


  REGISTER_CONTROL_PINS = REGISTER_CONTROL_PINS | REGISTER_CLOCK_HI;      // Set
RegisterClock high then low to latch '595 outputs
  pauseRegister(REGISTER_CLOCK_PAUSE);
  REGISTER_CONTROL_PINS = REGISTER_CONTROL_PINS & REGISTER_CLOCK_LO;

}




/***********************************************
 * Function: pauseRegister()
 * Creator:   Peter Miller                Date: 11/7/2009
 *
 * This function creates pauses in the ns range for use when the
 * register is updated.
 *
 ************************************************************/

void pauseRegister(int pauseLength){
  int pause = 0;
  int num = 0;

  for (pause = 0; pause < pauseLength; pause++){
    num++;
  }
}




/***********************************************
 * Function: Service_SafeModuleReset()
 * Creator:   Peter Miller                Date: 11/7/2009
 *
 * This function is a reset function.
 *
 * NOTE: This function has been modified since its original state;
 * its original functions has been dispered to other lower-level
 * functions.
 *
 ************************************************************/

void Service_SafeModuleReset(void){
  Service_GenerateTargetTicks();                        // Get a new number of target dial ticks and reset the current
UserTicks value.
}
```

```
/*************************************************
 * Function: Service_GenerateTargetTicks()
 * Creator:   Peter Miller              Date: 11/7/2009
 *
 * This function generates a random number of target dial ticks and resets
 * the current value of user ticks.
 *
 *************************************************************/

void Service_GenerateTargetTicks(void){
  unsigned int currentTime   = TMRS12_GetTime();                                    // Get the current time...
  DialTargetTicks   = (currentTime % (MAX_DIAL_POSITIONS - MIN_DIAL_POSITIONS)) +
MIN_DIAL_POSITIONS;   // Modulo the current time by the number of possible dial
                                                                                    // positions to generate a random number of target
dial ticks.

  UserTicks = 0;                                                                    // Reset the current value of user ticks.

  //DialTargetTicks   = 1;                                                          // Comment this to set the number of dial
ticks to a set value
  printf("\r\nTargetTicks = %i",DialTargetTicks);                                   // Print the new target tick value.
}




/*************************************************
 * Function: getCurrentStage()
 * Creator:   Peter Miller              Date: 11/7/2009
 *
 * This function returns the current stage of the game.
 *
 *************************************************************/

SafeEvent_t getCurrentStage(void){
  return currentStage;
}




/*************************************************
 * Function: Event_FlashTimerExpired()
 * Creator:   Peter Miller              Date: 11/7/2009
 *
 * This function returns a SaveEvent_t for the state of the Flash Timer.
 *
 *************************************************************/

SafeEvent_t Event_FlashTimerExpired(void){
  SafeEvent_t Event = SAFE_NO_EVENT;                        // The default return value is NO_EVENT

  if (TMRS12_IsTimerExpired(FLASH_TIMER) == TMRS12_EXPIRED){      // If the flash timer has expired...
    TMRS12_ClearTimerExpired(FLASH_TIMER);                    //   1) Clear the timer
```

```
   Event = TIMER_EXPIRED;                          //   2) Return a TIMER_EXPIRED value
  }

  return Event;
}



/***********************************************
 * Function: Service_MonitorLamps()
 * Creator:   Peter Miller              Date: 11/7/2009
 *
 * This function monitors the lamps depending on the current game stage.
 * Each stage has two possible paired lamp states associated with it; the
 * paired states are used to flash the appropriate lamps.  This function
 * only called when the flash timer expires, so it must reset the flash timer
 * every time it is called.
 *
 ************************************************************/

void Service_MonitorLamps(void){
  TMRS12_InitTimer(FLASH_TIMER, flashTime);              // Initiate the flash timer since it has just expired

  if (currentStage == RESET_SAFE){                       // For example, when the safe is in the RESET_SAFE
stage...
    registerStates[SPIN_CW_LED]       = ON;              // Only the SPIN_CW and...
    registerStates[SPIN_CCW_LED]      = OFF;
    registerStates[FIRST_NUM_IND_LED] = OFF;
    registerStates[SECOND_NUM_IND_LED]= OFF;
    registerStates[SAFE_LOCKED]       = ON;              // SAFE_LOCKED_LAMP should be on.
    registerStates[SAFE_UNLOCKED]     = OFF;
    registerStates[TICKER_SOLENOID]   = OFF;
    registerStates[LOCK_SOLENOID]     = LOCK_SOLENOID_ENGAGED;  // The safe should be locked.
  }


  else if (currentStage == CRACK_FIRST_NUM){             // In this state, we want to flip the state of the
CRACK_FIRST_NUM lamp, so
    if (registerStates[SPIN_CW_LED] == ON){              // Check the state of this lamp and invert its state.
      registerStates[SPIN_CW_LED]        = OFF;
      registerStates[SPIN_CCW_LED]       = OFF;
      registerStates[FIRST_NUM_IND_LED]  = OFF;
      registerStates[SECOND_NUM_IND_LED] = OFF;
      registerStates[SAFE_LOCKED]        = ON;
      registerStates[SAFE_UNLOCKED]      = OFF;

    }
    else {
      registerStates[SPIN_CW_LED]        = ON;
      registerStates[SPIN_CCW_LED]       = OFF;
      registerStates[FIRST_NUM_IND_LED]  = ON;
      registerStates[SECOND_NUM_IND_LED] = OFF;
      registerStates[SAFE_LOCKED]        = ON;
      registerStates[SAFE_UNLOCKED]      = OFF;
    }
```

```
  }


  else if (currentStage == CRACK_SECOND_NUM){
    if (registerStates[SPIN_CCW_LED] == ON){
      registerStates[SPIN_CW_LED]         = OFF;
      registerStates[SPIN_CCW_LED]        = OFF;
      registerStates[FIRST_NUM_IND_LED]   = ON;
      registerStates[SECOND_NUM_IND_LED]  = OFF;
      registerStates[SAFE_LOCKED]         = ON;
      registerStates[SAFE_UNLOCKED]       = OFF;
    }
    else {
      registerStates[SPIN_CW_LED]         = OFF;
      registerStates[SPIN_CCW_LED]        = ON;
      registerStates[FIRST_NUM_IND_LED]   = ON;
      registerStates[SECOND_NUM_IND_LED]  = ON;
      registerStates[SAFE_LOCKED]         = ON;
      registerStates[SAFE_UNLOCKED]       = OFF;
    }
  }


  else if (currentStage == CRACK_THIRD_NUM){
    if (registerStates[SPIN_CW_LED] == ON){
      registerStates[SPIN_CW_LED]         = OFF;
      registerStates[SPIN_CCW_LED]        = OFF;
      registerStates[FIRST_NUM_IND_LED]   = ON;
      registerStates[SECOND_NUM_IND_LED]  = ON;
      registerStates[SAFE_LOCKED]         = OFF;
      registerStates[SAFE_UNLOCKED]       = OFF;
    }
    else {
      registerStates[SPIN_CW_LED]         = ON;
      registerStates[SPIN_CCW_LED]        = OFF;
      registerStates[FIRST_NUM_IND_LED]   = ON;
      registerStates[SECOND_NUM_IND_LED]  = ON;
      registerStates[SAFE_LOCKED]         = ON;
      registerStates[SAFE_UNLOCKED]       = OFF;
    }
  }

  else if ((currentStage == SAFE_CRACKED) || (currentStage == OPEN_SAFE) || (currentStage ==
CLOSE_SAFE)){
    registerStates[SPIN_CW_LED]         = OFF;
    registerStates[SPIN_CCW_LED]        = OFF;
    registerStates[FIRST_NUM_IND_LED]   = OFF;
    registerStates[SECOND_NUM_IND_LED]  = OFF;
    registerStates[SAFE_LOCKED]         = OFF;
    registerStates[SAFE_UNLOCKED]       = ON;
  }

  else if (currentStage == END_GAME){
    registerStates[SPIN_CW_LED]         = OFF;
    registerStates[SPIN_CCW_LED]        = OFF;
    registerStates[FIRST_NUM_IND_LED]   = OFF;
```

```
    registerStates[SECOND_NUM_IND_LED]  = OFF;
    registerStates[SAFE_LOCKED]         = ON;
    registerStates[SAFE_UNLOCKED]       = OFF;
  }

  updateRegister();
}



/*****************************************************
 * Function: Event_DialTick()
 * Creator:   Peter Miller                Date: 11/7/2009
 *
 * This function returns 'TICK' iff the state of the safe switch has
 * changed to HI and an adequate amount of time has passed since the
 * last ON state of the switch.
 *
 *****************************************************************/

SafeEvent_t Event_DialTick(void){
  static unsigned int lastTickTime  = 0;
  static unsigned int lastTickValue = 0;

  unsigned int currentTickValue;
  unsigned int currentTime;

  SafeEvent_t  Event = SAFE_NO_EVENT;                       // The default event state is SAFE_NO_EVENT

  currentTickValue  = CHECKBIT(SAFE_DIAL_PORT,SAFE_DIAL_BIT);     // Test the current state of the safe
dial switch
  currentTime       = TMRS12_GetTime();                     // Get and store the current time

  if (currentTime - lastTickTime >= 100){                   // If at least 1ms has passed since the last test switch ON
state...
    if (currentTickValue != lastTickValue){                 // ...and the last switch state was LO...
      if (currentTickValue != 0){
        Event = TICK;                                   // ...then a TICK event has occurred.
      }
      lastTickTime = currentTime;                           // Update the last time value.
      lastTickValue = currentTickValue;                     // Update the last tick state.
    }
  }

  return Event;                                         // Return the event
}



/*****************************************************
 * Function: Service_UpdateUserTicks()
 * Creator:   Peter Miller                Date: 11/7/2009
 *
 * This function updates the number of user ticks.
```

```
 *
 ************************************************************/

void Service_UpdateUserTicks(void){
  UserTicks++;
}



/************************************************
 * Function: Event_PlayerReachedTarget()
 * Creator:   Peter Miller                Date: 11/8/2009
 *
 * This function returns a SaveEvent_t dependent on whether the user has reached
 * the correct number of safe ticks.
 *
 ************************************************************/

SafeEvent_t Event_PlayerReachedTarget(void){
  SafeEvent_t Event = SAFE_NO_EVENT;                              // The default return value is NO_EVENT

  if ((UserTicks == DialTargetTicks) || (UserTicks > MAX_DIAL_POSITIONS)){  // If the user has reached the
target ticks...
    Event = TARGET_REACHED;                                       // ...update the return value to
TARGET_REACHED
  }

  return Event;
}



/************************************************
 * Function: Service_PrepareForOvershoot()
 * Creator:   Peter Miller                Date: 11/9/2009
 *
 * This function pings the ticker solenoid and updates the lamps; it also
 * starts a timer that is used to test for a dial tick overshoot.  This
 * function is only called when the user reaches the target tick value.
 *
 ************************************************************/

void Service_PrepareForOvershoot(void){
  UserTicks = 0;                                  // Since the user has reached the target tick value, the number of user
ticks must be reset

  registerStates[TICKER_SOLENOID] = ON;                    // Set the ticker solenoid to turn on in the register pin
array
  updateRegister();                                // Update the register since this actually turns the solenoid ON

  TMRS12_InitTimer(OVERSHOOT_TIMER, overshootTime);          // Initiate an overshoot timer
}
```

```
/*****************************************************
 * Function: Event_OverShootTimerExpired()
 * Creator:   Peter Miller              Date: 11/9/2009
 *
 * This function checks for the overshoot timer to expire and checks
 * that the maximum number of overshoot ticks has not been surpassed.
 * It returns a safe event based on these two results.
 *
 *****************************************************************/

SafeEvent_t Event_OverShootTimerExpired(void){
  SafeEvent_t Event = SAFE_NO_EVENT;                        // The default return value is NO_EVENT

  if (TMRS12_IsTimerExpired(OVERSHOOT_TIMER) == TMRS12_EXPIRED){  // If the overshoot timer
expires...
    TMRS12_ClearTimerExpired(OVERSHOOT_TIMER);                  //   1) clear the overshoot timer

    registerStates[TICKER_SOLENOID] = OFF;                      //   2) turn off the ticker solenoid
    updateRegister();                                   //   3) update the shift register

    if (UserTicks > MAX_OVERSHOOT_TICKS){                       //   4) if the number of user ticks exceeds the max
overshoot ticks...
      Event = OVERSHOOT;                                //      ...set the return value to correspond to 'overshoot'
      printf("\r\n User Ticks %i",UserTicks);
      printf("\r\n Max OverShoot Ticks %i",MAX_OVERSHOOT_TICKS);
    }

    else{
      Event = NEXT_STAGE;                               //      ...otherwise, set the return value to correspond to 'stage
cleared'
    }

  }

  return Event;

}



/*****************************************************
 * Function: Service_UpdateStage()
 * Creator:   Peter Miller              Date: 11/9/2009
 *
 * This function updates the current stage.  This safe module follows a
 * linear progression, so this function updates the current stage value
 * to the next defined stage.
 *
 *****************************************************************/

void Service_UpdateStage(void){
  if (currentStage == RESET_SAFE){              // For example, if the current stage is RESET_SAFE and this
function is called...
    currentStage = CRACK_FIRST_NUM;             // ...the current is updated to CRACK_FIRST_NUM
    printf("\r\nWaiting To Crack First Number");
```

```
  }
  else if (currentStage == CRACK_FIRST_NUM){
    currentStage = CRACK_SECOND_NUM;
    printf("\r\nWaiting To Crack Second Number");
  }
  else if(currentStage == CRACK_SECOND_NUM){
    currentStage = CRACK_THIRD_NUM;
    printf("\r\nWaiting To Crack Third Number");
  }
  else if(currentStage == CRACK_THIRD_NUM){
    currentStage = SAFE_CRACKED;
    printf("\r\nWaiting To Safe Cracked");
  }
  else if(currentStage == SAFE_CRACKED){
    currentStage = OPEN_SAFE;
    printf("\r\nWaiting To Open Safe");
  }
  else if(currentStage == OPEN_SAFE){
    currentStage = ENTER_SAFE;
    printf("\r\nWaiting To Enter Safe");
  }
  else if(currentStage == ENTER_SAFE){
    currentStage = CLOSE_SAFE;
    printf("\r\nWaiting To Close Safe");
  }
  else if(currentStage == CLOSE_SAFE){
    currentStage = END_GAME;
    printf("\r\nWaiting To End Game");
  }
  else if(currentStage == END_GAME){
    currentStage = RESET_SAFE;
    printf("\r\nWaiting To Reset");
  }
}



/************************************************
 * Function: Service_RespondToOvershoot()
 * Creator:   Peter Miller              Date: 11/9/2009
 *
 * This function responds to an overshoot.  It flashes all of the safe lights three times
 * while not permitting any additional user input during this time.  When the overshoot
 * value is surpassed, the safe is reset and the player must restart at the first safe
 * stage.
 *
 * NOTE: This is intentional blocking code.
 *
 ************************************************************/

void Service_RespondToOvershoot(void){
  int i;
  currentStage = RESET_SAFE;                    // Reset the current stage

  // Blink all lights 3 times; use a timer to properly space the ON/OFF states
  for (i = 0; i < OVERSHOOT_FLASHES; i++){
```

```
   registerStates[SPIN_CW_LED]        = OFF;
   registerStates[SPIN_CCW_LED]       = OFF;
   registerStates[FIRST_NUM_IND_LED]  = OFF;
   registerStates[SECOND_NUM_IND_LED] = OFF;
   registerStates[SAFE_LOCKED]        = OFF;
   registerStates[SAFE_UNLOCKED]      = OFF;

   updateRegister();

   TMRS12_InitTimer(OVERSHOOT_FLASH_TIMER, overshootFlashTime);

   while (TMRS12_IsTimerExpired(OVERSHOOT_FLASH_TIMER) != TMRS12_EXPIRED){}

   registerStates[SPIN_CW_LED]        = ON;
   registerStates[SPIN_CCW_LED]       = ON;
   registerStates[FIRST_NUM_IND_LED]  = ON;
   registerStates[SECOND_NUM_IND_LED] = ON;
   registerStates[SAFE_LOCKED]        = ON;
   registerStates[SAFE_UNLOCKED]      = ON;

   updateRegister();

   TMRS12_InitTimer(OVERSHOOT_FLASH_TIMER, overshootFlashTime);

   while (TMRS12_IsTimerExpired(OVERSHOOT_FLASH_TIMER) != TMRS12_EXPIRED){}
  }

  TMRS12_ClearTimerExpired(OVERSHOOT_TIMER);          // Reset the overshoot timer

  Service_SafeModuleReset();                          // Reset the safe
}



/***********************************************
 * Function: Service_DisengageLockSolenoid()
 * Creator:   Peter Miller              Date: 11/9/2009
 *
 * This function unlocks the safe by disengaging the lock solenoid.
 *
 ************************************************************/

void Service_DisengageLockSolenoid(void){
  registerStates[LOCK_SOLENOID] = LOCK_SOLENOID_DISENGAGED; // Set the lock solenoid to be
disengaged
  updateRegister();                                   // Update the shift register that controlls the lamps and solenoids
}



/***********************************************
 * Function: Event_Safe_Opened()
 * Creator:   Peter Miller              Date: 11/9/2009
 *
 * This function checks the state of the switch on the safe door.
 *
```

```
 ************************************************************/

SafeEvent_t Event_SafeDoorState(void){
  SafeEvent_t Event = SAFE_CLOSED;                    // The default return value is SAFE_CLOSED

  if (!(CHECKBIT(SAFE_LEVER_PORT,SAFE_LEVER_BIT))){        // If the safe door has been opened (the safe
switch is open)...
    Event = SAFE_OPENED;                              // ...update the return value SAFE_OPENED
  }

  return Event;
}



/*************************************************
 * Function: Event_Safe_Opened()
 * Creator:   Peter Miller                Date: 11/9/2009
 *
 * This function checks that the user has broken the IR beam that runs
 * across the safe opening.
 *
 ************************************************************/

SafeEvent_t Event_SafeEntered(void){                          // The default return value is SAFE_CLOSED
  SafeEvent_t event = SAFE_NO_EVENT;

  if (ADS12_ReadADPin(SAFE_PHOTO_BIT) > PHOTO_INTERRUPT_THRESHOLD) {    // If the safe has been
opened (the beam is broken)...
                    event = SAFE_ENTERED;                                   // ...update the return value
SAFE_ENTERED
            }

  return event;
}



/*************************************************
 * Function: SAFE_Service_CloseSafeModule()
 * Creator:   Peter Miller                Date: 11/9/2009
 *
 * This function closes the safe module by turning all of the lamps off and
 * unlocking the safe door.  It also initializes a celebration timer.
 *
 ************************************************************/

void SAFE_Service_CloseSafeModule(void){
  TMRS12_InitTimer(PING_TIMER, celebrationTime);              // Initialize the celebration timer

  registerStates[SPIN_CW_LED]       = OFF;                  // Turn everything OFF
  registerStates[SPIN_CCW_LED]      = OFF;
  registerStates[FIRST_NUM_IND_LED] = OFF;
  registerStates[SECOND_NUM_IND_LED]= OFF;
  registerStates[SAFE_LOCKED]       = OFF;
  registerStates[SAFE_UNLOCKED]     = OFF;
```

```
  registerStates[TICKER_SOLENOID]   = OFF;
  registerStates[LOCK_SOLENOID]     = LOCK_SOLENOID_DISENGAGED;

  updateRegister();
}



/***********************************************
 * Function: SAFE_CheckForSafeEvent
 * Creator:   Peter Miller              Date:
 *
 * This function returns a value based on whether or not the safe has been
 * cracked.
 *
 ************************************************************/
Event_t SAFE_CheckForSafeEvent(void){
  if (currentStage == END_GAME){                    // If the safe has been cracked, return the same.
     return SAFE_CLEARED;
    }
    else{
     return NO_EVENT;                               // Otherwise, return NO_EVENT
    }
}




/***********************************************
 * Function: SAFE_Init
 * Creator:   Peter Miller              Date:
 *
 * This function initializes the safe module by setting I/O and
 * setting the safe lamps and solenoids to their default / initial
 * desired values.
 *
 ************************************************************/
void SAFE_Init(void){
  currentStage = RESET_SAFE;

  DDRT = DDRT | SERIAL_DATA_IN_HI;                  // Output, T5
  DDRT = DDRT | SERIAL_CLOCK_HI;                    // Output, T6
  DDRT = DDRT | REGISTER_CLOCK_HI;                  // Output, T7

  CLEARBIT(SAFE_LEVER_DDR,SAFE_LEVER_BIT);          // Intput, E0
  CLEARBIT(SAFE_DIAL_DDR,SAFE_DIAL_BIT);            // Intput, E1

  registerStates[SPIN_CW_LED]       = OFF;
  registerStates[SPIN_CCW_LED]      = OFF;
  registerStates[FIRST_NUM_IND_LED] = OFF;
  registerStates[SECOND_NUM_IND_LED]= OFF;
  registerStates[SAFE_LOCKED]       = ON;
  registerStates[SAFE_UNLOCKED]     = OFF;
  registerStates[TICKER_SOLENOID]   = OFF;
  registerStates[LOCK_SOLENOID]     = LOCK_SOLENOID_ENGAGED;
```

```
  updateRegister();
}




/*************************************************
 * Function: SAFE_CelebrateWin
 * Creator:   Peter Miller                Date:
 *
 * This function pings the ticker solenoid during the celebration sequence
 *
 *************************************************************/
void SAFE_CelebrateWin(void){
  static int sequence = 0;
  printf("CELEBRATE!");

  if (TMRS12_IsTimerExpired(PING_TIMER) == TMRS12_EXPIRED){     // If the celebration timer expires...
    if (sequence == 0){                                  //    ...if the current sequence is '0'
      registerStates[TICKER_SOLENOID]     = ON;               //       1) turn the ticker solenoid ON
      registerStates[SAFE_LOCKED]         = ON;               //       2) turn the SAFE_LOCKED lamp ON
      sequence = 1;
    }

    else {                                               //    ...otherwise...
      registerStates[TICKER_SOLENOID]     = OFF;              //       1) turn the ticker solenoid OFF
      registerStates[SAFE_LOCKED]         = OFF;              //       2) turn the SAFE_LOCKED lamp OFF
      sequence = 0;

    }

    updateRegister();                                    //  Update the shift register
    TMRS12_InitTimer(PING_TIMER, celebrationTime);             //  Initialze the celebration timer

  }
}




 /*************************************************
 * Function:  Test Harness for Stages
 * Creator:   Peter Miller                Date:
 *
 * This is the main() function Safe Module.
 *
 * NOTE: This function was originally the test harness for this module.
 *
 *************************************************************/
void SAFE_Main(void){
  SafeEvent_t CurrentEvent;
  SafeEvent_t Stage;
```

```
  /*DDRT = DDRT | SERIAL_DATA_IN_HI;                  // Output, T5
  DDRT = DDRT | SERIAL_CLOCK_HI;                    // Output, T6
  DDRT = DDRT | REGISTER_CLOCK_HI;                  // Output, T7

  DDRM = DDRM & SAFE_DIAL_SWITCH_LO;                // Input, M3
  DDRM = DDRM & SAFE_OPEN_SWITCH_LO;                // Input, M4
  DDRM = DDRM & SAFE_ENTERED_INTERUPT_LO;           // Input, M5


  TMRS12_Init(TIMER_RATE);
  currentStage = RESET_SAFE;


  while(1){*/
    Stage = getCurrentStage();

  //  THE FOLLOWING TWO EVENTS ARE ALWAYS CHECKED; THE REFER TO EVENTS THAT MUST
BE CHECKED REGARDLESS OF THE STAGE OF THE GAME
    if (Event_FlashTimerExpired() == TIMER_EXPIRED){
      Service_MonitorLamps();
    }

    if (Event_DialTick() == TICK){
      Service_UpdateUserTicks();
      printf("\r\nIncrement Tick Counter");
    }


  //  THE FOLLOWING EVENTS ARE CHECKED BASED ON THE CURRENT STAGE OF THE GAME:
    if (Stage == RESET_SAFE){
      Service_SafeModuleReset();
      Service_MonitorLamps();
      Service_UpdateStage();
      printf("\r\nRESET");
    }


    else if ((Stage == CRACK_FIRST_NUM) || (Stage == CRACK_SECOND_NUM) || (Stage ==
CRACK_THIRD_NUM)){
      CurrentEvent = Event_OverShootTimerExpired();

      if (Event_PlayerReachedTarget() == TARGET_REACHED){
        printf("\r\nTick Target Reached");
        Service_PrepareForOvershoot();
      }
      if (CurrentEvent == OVERSHOOT){
        Service_RespondToOvershoot();
        printf("\r\nPlayer Overshoots");
      }
      else if(CurrentEvent == NEXT_STAGE){
        Service_UpdateStage();
        Service_MonitorLamps();
        Service_GenerateTargetTicks();
        printf("\r\nPlayer Clears Stage");
      }
    }
```

```
    else if (Stage == SAFE_CRACKED){
      Service_DisengageLockSolenoid();
      Service_UpdateStage();
      printf("\r\nSafe cracked");
    }

    else if (Stage == OPEN_SAFE){
      if (Event_SafeDoorState() == SAFE_OPENED){
         Service_UpdateStage();
         printf("\r\nSafe opened");
      }
    }

    else if (Stage == ENTER_SAFE){
      if (Event_SafeEntered() == SAFE_ENTERED){
         Service_UpdateStage();
         printf("\r\nSafe Entered");
      }
    }

    else if (Stage == CLOSE_SAFE){
      if (Event_SafeDoorState() == SAFE_CLOSED){
         SAFE_Service_CloseSafeModule();
         //SAFE_CelebrateWin();
         Service_UpdateStage();
         printf("\r\nSafe Closed");
      }
    }

    else if (Stage == END_GAME){
      SAFE_CelebrateWin();
    }


  //}
}
```

```
/*************************************************
 * Function:  Test Harness for Timer
 * Creator:   Peter Miller                Date:
 *
 * This function is the test harness for the timers.
 *
 *************************************************************/
#ifdef TEST_TIMER

void main(void){
  TMRS12_Init(TIMER_RATE);
```

```
  TMRS12_InitTimer(FLASH_TIMER, 100);
  printf("\r\nStart");

  char i = 0;
  while(1){


    if (TMRS12_IsTimerExpired(FLASH_TIMER) == TMRS12_EXPIRED){
      printf("\r\nEnd");
      break;
    }

    else if (TMRS12_IsTimerExpired(FLASH_TIMER) == TMRS12_NOT_EXPIRED){
      printf("\r\nNot yet...");
    }

    else {
      printf("\r\nHELP!");
    }
  }



}
#endif




 /*********************************************
 * Function:  Test Harness for Switch
 * Creator:   Peter Miller              Date:
 *
 * This function is the test harness for the switches.
 *
 *************************************************************/
#ifdef TEST_SWITCH

void main(void){
  unsigned int currentTickValue;

  DDRT = DDRT | SERIAL_DATA_IN_HI;                  // Output
  DDRT = DDRT | SERIAL_CLOCK_HI;                   // Output
  DDRT = DDRT | REGISTER_CLOCK_HI;                   // Output

  DDRM = DDRM & SAFE_DIAL_SWITCH_LO;                  // Input



  while(1){
     currentTickValue  = SAFE_DIAL_SWITCH & SAFE_DIAL_SWITCH_HI;
     printf("\r\n State = %i",currentTickValue);
  }
}
```

```
#endif



/*************************************************
 * Function:  Test Harness (for shift register)
 * Creator:   Peter Miller                Date: 11/7/2009
 *
 * This function is the test harness for the shift register.
 *
 *************************************************************/
#ifdef TEST_SHIFT_REGISTER

void main(void){
  int i = 0;

  DDRT = DDRT | SERIAL_DATA_IN_HI;
  DDRT = DDRT | SERIAL_CLOCK_HI;
  DDRT = DDRT | REGISTER_CLOCK_HI;

//  while (1){
    for (i = 0; i < 300; i++){
      pauseRegister(20000);
    }

    registerStates[1]   = OFF;
    registerStates[2]   = OFF;
    registerStates[3]   = OFF;
    registerStates[4]   = OFF;
    registerStates[5]   = OFF;
    registerStates[6]   = OFF;
    registerStates[7]   = OFF;
    registerStates[8]   = OFF;

    updateRegister();

    printf("\r\nState OFF");

    for (i = 0; i < 300; i++){
      pauseRegister(20000);
    }

    registerStates[1]   = ON;
    registerStates[2]   = OFF;
    registerStates[3]   = OFF;
    registerStates[4]   = OFF;
    registerStates[5]   = OFF;
    registerStates[6]   = OFF;
    registerStates[7]   = OFF;
    registerStates[8]   = OFF;

    updateRegister();

    printf("\r\nState 1");
```

```
for (i = 0; i < 300; i++){
  pauseRegister(20000);
}

registerStates[1]   = ON;
registerStates[2]   = ON;
registerStates[3]   = OFF;
registerStates[4]   = OFF;
registerStates[5]   = OFF;
registerStates[6]   = OFF;
registerStates[7]   = OFF;
registerStates[8]   = OFF;

updateRegister();

printf("\r\nState 12");

for (i = 0; i < 300; i++){
  pauseRegister(20000);
}


registerStates[1]   = ON;
registerStates[2]   = ON;
registerStates[3]   = ON;
registerStates[4]   = OFF;
registerStates[5]   = OFF;
registerStates[6]   = OFF;
registerStates[7]   = OFF;
registerStates[8]   = OFF;

updateRegister();

printf("\r\nState 123");

for (i = 0; i < 300; i++){
  pauseRegister(20000);
}


registerStates[1]   = ON;
registerStates[2]   = ON;
registerStates[3]   = ON;
registerStates[4]   = ON;
registerStates[5]   = OFF;
registerStates[6]   = OFF;
registerStates[7]   = OFF;
registerStates[8]   = OFF;

updateRegister();

printf("\r\nState 1234");

for (i = 0; i < 300; i++){
```

```
  pauseRegister(20000);
}


registerStates[1]   = ON;
registerStates[2]   = ON;
registerStates[3]   = ON;
registerStates[4]   = ON;
registerStates[5]   = ON;
registerStates[6]   = OFF;
registerStates[7]   = OFF;
registerStates[8]   = OFF;

updateRegister();

printf("\r\nState 12345");

for (i = 0; i < 300; i++){
  pauseRegister(20000);
}


registerStates[1]   = ON;
registerStates[2]   = ON;
registerStates[3]   = ON;
registerStates[4]   = ON;
registerStates[5]   = ON;
registerStates[6]   = ON;
registerStates[7]   = OFF;
registerStates[8]   = OFF;

updateRegister();

printf("\r\nState 123456");

for (i = 0; i < 300; i++){
  pauseRegister(20000);
}


registerStates[1]   = ON;
registerStates[2]   = ON;
registerStates[3]   = ON;
registerStates[4]   = ON;
registerStates[5]   = ON;
registerStates[6]   = ON;
registerStates[7]   = ON;
registerStates[8]   = OFF;

updateRegister();

printf("\r\nState 1234567");

for (i = 0; i < 300; i++){
  pauseRegister(20000);
}
```

```
    registerStates[1]   = ON;
    registerStates[2]   = ON;
    registerStates[3]   = ON;
    registerStates[4]   = ON;
    registerStates[5]   = ON;
    registerStates[6]   = ON;
    registerStates[7]   = ON;
    registerStates[8]   = ON;

    updateRegister();

    printf("\r\nState 12345678");

    for (i = 0; i < 300; i++){
      pauseRegister(20000);
    }
 // }
}


#endif

/*------------------------------ End of file ------------------------------*/
```

```
/*
 * Team Mini
 * 11.9.09
 * State machine module
 *
 * Blake English
 * Nina Joshi
 * Peter Miller
 */


#ifndef _STATE_H_
#define _STATE_H_

void STATE_InitStateMachine(void);
void STATE_RunStateMachine(Event_t event);

#endif
```

```
/*
 * Team Mini
 * 11.18.09
 * State machine module
 *
 * Blake English
 * Nina Joshi
 * Peter Miller
 */


#include <me218_c32.h>
#include <stdio.h>
#include <timers12.h>
#include "state.h"
#include "defs.h"
#include "events.h"
#include "board.h"
#include "opto.h"
#include "driving.h"
#include "safe.h"

/* States */
#define WAITING_FOR_SWITCH                  0
#define WAITING_FOR_IGNITION                1
#define WAITING_TO_REACH_DESTINATION        2
#define DEALING_WITH_SAFE                   3
#define PULSING_OPTOISOLATOR                4

static unsigned char state;
static unsigned char finalDestination;

/* Prototypes */
static void WaitingForSwitchState(Event_t event);
static void WaitingForIgnitionState(Event_t event);
static void WaitingToReachDestinationState(Event_t event);
static void DealingWithSafeState(Event_t event);
static void PulsingOptoisolatorState(Event_t event);

/* Initialize state machine by setting initial state and final destination. */
void STATE_InitStateMachine() {
        finalDestination = RIGHT;
        state = WAITING_FOR_SWITCH;
}

/* Figure out which state we are in, and call the corresponding function to handle the event. */
void STATE_RunStateMachine(Event_t event) {
        switch (state) {
            case WAITING_FOR_SWITCH:
                    //printf("Waiting for switch \r\n");
                    WaitingForSwitchState(event);
                    break;
            case WAITING_FOR_IGNITION:
                    printf("Waiting for ignition \r\n");
                    WaitingForIgnitionState(event);
                    break;
```

```
            case WAITING_TO_REACH_DESTINATION:
                        printf("Waiting to reach destination \r\n");
                        WaitingToReachDestinationState(event);
                        break;
            case DEALING_WITH_SAFE:
                        //printf("Dealing with safe \r\n");
                        DealingWithSafeState(event);
                        break;
            case PULSING_OPTOISOLATOR:
                        printf("Pulsing optoisolator state \r\n");
                        PulsingOptoisolatorState(event);
                        break;
            default:
                        printf("ERROR: Undefined state\n");
                        break;
        }
}

/* If optoisolator input is on or force start is hit, prepare to begin the game.*/
static void WaitingForSwitchState(Event_t event) {
        switch(event) {
            case OPTOISOLATOR_ON:
            case FORCE_START_HIT:
                        CLEARBIT(OPTO_OUTPUT_PORT,OPTO_OUTPUT_BIT);
                        SAFE_Init();
                        DRIVING_SetFuelGaugeFull();
                        BOARD_ResetGame(finalDestination);
                        DRIVING_SetIgnitionLight();
                        TMRS12_InitTimer(BLINK_TIMER, BLINK_TIME);
                        state = WAITING_FOR_IGNITION;
                        break;
            default:
                        break;
        }
}

/* If ignition button is hit, begin driving state. Until then, blink ignition indicator light and
 * continually adjust fuel gauge. */
static void WaitingForIgnitionState(Event_t event) {
        static unsigned char lightState = 1;
        switch (event) {
            case IGNITION_ON:
                        DRIVING_ClearIgnitionLight();
                        TMRS12_InitTimer(END_TIMER, END_TIME); //start game timer
                        DRIVING_InitFuelGauge();
                        DRIVING_DriveCar();
                        DRIVING_SetWheelVibrate();
                        TMRS12_InitTimer(BLINK_TIMER, BLINK_TIME);
                        state = WAITING_TO_REACH_DESTINATION;
                        break;
            case BLINK_TIMER_EXPIRED: //alternate state of ignition light (so it blinks)
                        TMRS12_InitTimer(BLINK_TIMER, BLINK_TIME);
                        if (lightState == 1) {
                                    DRIVING_SetIgnitionLight();
                                    lightState = 0;
                        } else {
```

```
                DRIVING_ClearIgnitionLight();
                lightState = 1;
            }
            break;
        default:
            //printf("\r\n %d", finalDestination);
            DRIVING_SetFuelGauge();
            break;
    }
}

/* If the car is sensed in the final destination building, stop all driving and begin dealing with the safe.
 * If end timer expires, stop all driving, pulse opto output high, and begin failure sequence. Otherwise,
 * blink final destination building lights, allow driving, and adjust fuel gauge. */
static void WaitingToReachDestinationState(Event_t event) {
    static unsigned char lightState = 1;
    switch(event) {
        case RIGHT_SENSED:
            if (finalDestination == RIGHT) {
                STEERING_stopCarMotor();
                STEERING_centerCarSteering();
                DRIVING_ClearWheelVibrate();
                BOARD_SetLight(finalDestination);
                state = DEALING_WITH_SAFE;
                finalDestination = LEFT;
            }
            break;
        case LEFT_SENSED:
            if (finalDestination == LEFT) {
                STEERING_stopCarMotor();
                STEERING_centerCarSteering();
                DRIVING_ClearWheelVibrate();
                BOARD_SetLight(finalDestination);
                state = DEALING_WITH_SAFE;
                finalDestination = RIGHT;
            }
            break;
      case BLINK_TIMER_EXPIRED:
            TMRS12_InitTimer(BLINK_TIMER, BLINK_TIME);
            if (lightState == 1) {
                BOARD_SetLight(finalDestination);
                lightState = 0;
            } else {
                BOARD_ClearLight(finalDestination);
                lightState = 1;
            }
            break;
        case END_TIMER_EXPIRED:
            TMRS12_StopTimer(BLINK_TIMER);
            TMRS12_ClearTimerExpired(BLINK_TIMER);
            STEERING_stopCarMotor();
            STEERING_centerCarSteering();
            DRIVING_ClearWheelVibrate();

            TMRS12_ClearTimerExpired(END_TIMER);
            SAFE_Service_CloseSafeModular();
```

```
                TMRS12_InitTimer(BUZZER_TIMER, BUZZER_TIME);
                DRIVING_SetIgnitionLight(); //Set buzzer on
                TMRS12_InitTimer(OPTO_TIMER, OPTO_TIME);
                OPTO_PulseOutputHi();
                state = PULSING_OPTOISOLATOR;
                break;
        default:
                DRIVING_DriveCar();
                DRIVING_SetFuelGauge();
                break;
    }
}

/* If safe is cleared, reset safe, begin celebration sequence, and pulse opto ouptut high.
 * If end timer expires, reset safe, turn buzzer on, pulse output high, and begin failure sequence.
 * Otherwise, deal with the safe and adjust the fuel gauge. */
static void DealingWithSafeState(Event_t event) {
    switch(event) {
        case SAFE_CLEARED:
                SAFE_Service_CloseSafeModular();
                TMRS12_InitTimer(CELEBRATION_TIMER, CELEBRATION_TIME);
                TMRS12_InitTimer(OPTO_TIMER, OPTO_TIME);
                OPTO_PulseOutputHi();
                state = PULSING_OPTOISOLATOR;
                break;
        case END_TIMER_EXPIRED:
                TMRS12_ClearTimerExpired(END_TIMER);
                SAFE_Service_CloseSafeModular();
                TMRS12_InitTimer(BUZZER_TIMER, BUZZER_TIME);
                DRIVING_SetIgnitionLight(); //Set Buzzer on
                TMRS12_InitTimer(OPTO_TIMER, OPTO_TIME);
                OPTO_PulseOutputHi();
                state = PULSING_OPTOISOLATOR;
                break;
        default:
                SAFE_Main();
                DRIVING_SetFuelGauge();
                break;
    }
}

/* If the celebration timer expires, return to waiting for switch state. If buzzer timer expires,
 * first turn the buzzer off and then return to waiting for switch state. If opto timer expires,
 * clear opto output to low and clear board lights. Otherwise, if the celebration timer is active,
 * continue to celebrate the win by chiming the safe solenoid. */
static void PulsingOptoisolatorState(Event_t event) {
    switch(event) {
        case OPTO_TIMER_EXPIRED:
                TMRS12_ClearTimerExpired(OPTO_TIMER);
                OPTO_PulseOutputLo();
                BOARD_EndGame(); //clear lights and keep opto interuptors and servo signs
                break;
        case CELEBRATION_TIMER_EXPIRED:
                state = WAITING_FOR_SWITCH;
                break;
        case BUZZER_TIMER_EXPIRED:
```

```
                DRIVING_ClearIgnitionLight(); //Turn Buzzer off
                state = WAITING_FOR_SWITCH;
                break;
        default:
                if(TMRS12_IsTimerActive(CELEBRATION_TIMER) == TMRS12_ACTIVE){
                        SAFE_CelebrateWin();
                }
                break;
    }
}
```