```
/*
 * Team 15: "Do It"
 * 2.27.09
 * Arm bumper module
 *
 * Tony Zhang
 * Agustin Ramirez
 * Nina Joshi
 */


#ifndef _ARMS_H_
#define _ARMS_H_

unsigned char ARMS_CheckForArmBumper(void);
unsigned char ARMS_GetArmBumperStatus(void);
void ARMS_Debug(void);

#endif
```

```
/*
 * Team 15: "Do It"
 * 3.2.09
 * Arm bumper module
 *
 * Tony Zhang
 * Agustin Ramirez
 * Nina Joshi
 */


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

// The previous arm bumper event.
static unsigned char lastEvent = NO_EVENT;

/* Returns an arm bumper event if either arm bumper has been hit. */
unsigned char ARMS_CheckForArmBumper() {
	unsigned char event = NO_EVENT;
	if (ADS12_ReadADPin(LEFT_ARM_BUMPER_BIT) < 500) {
	    event = LEFT_ARM_BUMPER_HIT;
	}
	if (ADS12_ReadADPin(RIGHT_ARM_BUMPER_BIT) < 500) {
	    event = RIGHT_ARM_BUMPER_HIT;
	}

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

	return NO_EVENT;
}

/* Returns the current arm bumper status regardless of whether it has changed. */
unsigned char ARMS_GetArmBumperStatus(void) {
	return lastEvent;
}

/* Arm debug */
void ARMS_Debug(void) {
	printf("\nDebugging arm bumpers... Controls:\r\n");
	printf("q: quit\r\n");
	// Break out of loop when user hits 'q'
	for(;;) {
	    // Prints user specified arm bumps.
	    while (!kbhit()) {
		unsigned char event = ARMS_CheckForArmBumper();
```

```
                    if(event == RIGHT_ARM_BUMPER_HIT) {
                            printf("Right arm bumper hit\r\n");
                    }
                    else if(event == LEFT_ARM_BUMPER_HIT) {
                            printf("Left arm bumper hit\r\n");
                    }
            }
            // Clears input character and returns from the function.
            if (getchar() == 'q') {
              lastEvent = NO_EVENT;
              return;
            }
        }
}
```

```
/*
 * Team 15: "Do It"
 * 2.27.09
 * Beacon sensing module
 *
 * Tony Zhang
 * Agustin Ramirez
 * Nina Joshi
 */


#ifndef _BEACON_H_
#define _BEACON_H_

unsigned char BEACON_CheckForBeacon(void);
unsigned char BEACON_GetBeaconStatus(void);
void BEACON_Debug(void);

#endif
```

```
/*
 * Team 15: "Do It"
 * 3.5.09
 * Beacon sensing module
 *
 * Tony Zhang
 * Agustin Ramirez
 * Nina Joshi
 */


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

// The previous beacon event.
static unsigned char lastEvent = NO_EVENT;
// The current beacon event.
static unsigned char thisEvent = NO_EVENT;
// Beacon cutoff threshold. When the beacon has been high for this amount of time, it is on.
static unsigned char beaconCutoff = 0;

/* Returns a BEACON_ON event if the phototransistor signal is between the specified thresholds.
*/
unsigned char BEACON_CheckForBeacon() {
        unsigned char event = NO_EVENT;

        if ((ADS12_ReadADPin(PHOTO_BIT) > BEACON_ON_LOW_THRESHOLD) &&
(ADS12_ReadADPin(PHOTO_BIT) < BEACON_ON_HIGH_THRESHOLD)) {
          event = BEACON_ON;
        }

        if (event != lastEvent) {
            lastEvent = event;
            return event;
        }

  return NO_EVENT;
}

/* Returns the current beacon status regardless of whether it has changed. */
unsigned char BEACON_GetBeaconStatus(void) {
        return lastEvent;
}

/* Beacon debug. */
void BEACON_Debug(void) {
        printf("\nDebugging beacon sensors...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()) {
                        unsigned char event = BEACON_CheckForBeacon();
                        if(event == BEACON_ON) {
                                    printf("Beacon ON\r\n");
                        } else if (event == NO_EVENT) {
                                    //printf("No Event\r\n");
                        }
            }
            // Clears input character and returns from the function.
            if (getchar() == 'q') {
              lastEvent = NO_EVENT;
              return;
            }
        }
}
```

```
/*
 * Team 15: "Do It"
 * 3.5.09
 * Definitions
 *
 * Tony Zhang
 * Agustin Ramirez
 * Nina Joshi
 */


#include <bitdefs.h>

/* Port T bits
 * Channel 1 (right) is T7.
 * Channel 0 (left) is T6.
 */
#define LEFT_WHEEL_CHANNEL_BIT PWMS12_CHAN1
#define RIGHT_WHEEL_CHANNEL_BIT PWMS12_CHAN0
#define RELEASE_MOTOR_ENABLE_BIT BIT5HI
#define LEFT_WHEEL_DIRECTION_BIT BIT7HI
#define RIGHT_WHEEL_DIRECTION_BIT BIT6HI
/* Port AD bits */
#define LEFT_ARM_BUMPER_BIT 0
#define RIGHT_ARM_BUMPER_BIT 1
#define PHOTO_BIT 2
#define SWITCH_BIT BIT3HI

/* Beacon */
#define BEACON_ON_LOW_THRESHOLD 300
#define BEACON_ON_HIGH_THRESHOLD 800

/* Timers */
/* Allow game to last 2 min. */
#define END_TIMER 6
#define END_TIME 30000
/* Kickspin at full speed for 800 ms. */
#define KICKSPIN_TIMER 1
#define KICKSPIN_TIME 200
/* Reverse motors to break spin for 120 ms. */
#define REVERSE_TIMER 7
#define REVERSE_TIME 30
/* Wait 400 ms before releasing arms. */
#define ARM_TIMER 2
#define ARM_TIME 100
/* Arm releasing motor should run for 600 ms. */
#define RELEASE_TIMER 3
#define RELEASE_TIME 150
/* 400 ms time delay before switching to moving left. */
#define RIGHT_DELAY_TIMER 5
#define RIGHT_DELAY_TIME 100
/* 400 ms time delay before switching to moving right. */
#define LEFT_DELAY_TIMER 4
#define LEFT_DELAY_TIME 100

/* Convenience */
```

```
#define false 0
#define true 1
```

```
/*
 * Team 15: "Do It"
 * 2.26.09
 * Event detection module
 *
 * Tony Zhang
 * Agustin Ramirez
 * Nina Joshi
 */


#ifndef _EVENTS_H_
#define _EVENTS_H_

#define NO_EVENT                0
#define SWITCH_ON                       1
#define BEACON_ON                   2
#define RIGHT_BODY_BUMPER_HIT           3
#define RIGHT_ARM_BUMPER_HIT        4
#define LEFT_ARM_BUMPER_HIT             5
#define END_TIMER_EXPIRED         6
#define KICKSPIN_TIMER_EXPIRED      7
#define REVERSE_TIMER_EXPIRED       8
#define ARM_TIMER_EXPIRED           9
#define RELEASE_TIMER_EXPIRED     10
#define RIGHT_DELAY_TIMER_EXPIRED   11
#define LEFT_DELAY_TIMER_EXPIRED    12


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

unsigned char EVENTS_CheckForEvents(void);
void EVENTS_Debug(void);

#endif
```

```
/*
 * Team 15: "Do It"
 * 3.2.09
 * Event detection module
 *
 * Tony Zhang
 * Agustin Ramirez
 * Nina Joshi
 */


#include <me218_c32.h>
#include <stdio.h>
#include "events.h"
#include "defs.h"
#include "timers12.h"
#include "arms.h"
#include "beacon.h"
#include "switch.h"

/* Checks and returns any events that have occured . */
unsigned char EVENTS_CheckForEvents(void) {
	unsigned char event = NO_EVENT;

	// Check switch change
	event = SWITCH_CheckForSwitch();
	if (event != NO_EVENT)
	    return event;

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

	// Check arm bumpers
	event = ARMS_CheckForArmBumper();
	if (event != NO_EVENT)
	    return event;

	// Check beacon sensors
	event = BEACON_CheckForBeacon();

	return event;
}

/* Checks all timers and if any has expired, clear the timer and return the corresponding event. */
static unsigned char CheckForTimerExpired() {
	if(TMRS12_IsTimerExpired(END_TIMER)) {
	    TMRS12_ClearTimerExpired(END_TIMER);
	    return END_TIMER_EXPIRED;
	}
	if(TMRS12_IsTimerExpired(KICKSPIN_TIMER)) {
	    TMRS12_ClearTimerExpired(KICKSPIN_TIMER);
	    return KICKSPIN_TIMER_EXPIRED;
	}
	if(TMRS12_IsTimerExpired(REVERSE_TIMER)) {
```

```
            TMRS12_ClearTimerExpired(REVERSE_TIMER);
            return REVERSE_TIMER_EXPIRED;
        }
        if(TMRS12_IsTimerExpired(ARM_TIMER)) {
            TMRS12_ClearTimerExpired(ARM_TIMER);
            return ARM_TIMER_EXPIRED;
        }
        if(TMRS12_IsTimerExpired(RELEASE_TIMER)) {
            TMRS12_ClearTimerExpired(RELEASE_TIMER);
            return RELEASE_TIMER_EXPIRED;
        }
        if(TMRS12_IsTimerExpired(RIGHT_DELAY_TIMER)) {
            TMRS12_ClearTimerExpired(RIGHT_DELAY_TIMER);
            return RIGHT_DELAY_TIMER_EXPIRED;
        }
        if(TMRS12_IsTimerExpired(LEFT_DELAY_TIMER)) {
            TMRS12_ClearTimerExpired(LEFT_DELAY_TIMER);
            return LEFT_DELAY_TIMER_EXPIRED;
        }

        return NO_EVENT;
}

/* Events debug. */
void EVENTS_Debug(void) {
        printf("\nDebugging events...Controls:\r\n");
        printf("e: end timer, k: kickspin timer, v: reverse timer, a: arm timer,\r\n");
        printf("s: release motor timer, r: right delay timer, l: left delay 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 'k':
                                        printf("Setting kickspin timer...\r\n");
                                        TMRS12_InitTimer(KICKSPIN_TIMER, KICKSPIN_TIME);
                                        break;
                                case 'v':
                                        printf("Setting reverse timer...\r\n");
                                        TMRS12_InitTimer(REVERSE_TIMER, REVERSE_TIME);
                                        break;
                                case 'a':
                                        printf("Setting arm timer...\r\n");
                                        TMRS12_InitTimer(ARM_TIMER, ARM_TIME);
                                        break;
                                case 's':
                                        printf("Setting release motor timer...\r\n");
                                        TMRS12_InitTimer(RELEASE_TIMER, RELEASE_TIME);
                                        break;
                                case 'r':
                                        printf("Setting right delay timer...\r\n");
```

```
                                        TMRS12_InitTimer(RIGHT_DELAY_TIMER,
RIGHT_DELAY_TIME);
                                        break;
                                case 'l':
                                        printf("Setting release motor timer...\r\n");
                                        TMRS12_InitTimer(LEFT_DELAY_TIMER,
LEFT_DELAY_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 SWITCH_ON:
                                printf("Switch on\r\n");
                                break;
                        case BEACON_ON:
                                printf("Beacon on\r\n");
                                break;
                        case RIGHT_BODY_BUMPER_HIT:
                                printf("Right body bumper hit\r\n");
                                break;
                        case RIGHT_ARM_BUMPER_HIT:
                                printf("Right arm bumper hit\r\n");
                                break;
                        case LEFT_ARM_BUMPER_HIT:
                                printf("Left arm bumper hit\r\n");
                                break;
                        case END_TIMER_EXPIRED:
                                printf("End timer expired\r\n");
                                break;
                        case KICKSPIN_TIMER_EXPIRED:
                                printf("Kickspin timer expired\r\n");
                                break;
                        case REVERSE_TIMER_EXPIRED:
                                printf("Reverse timer expired\r\n");
                                break;
                        case ARM_TIMER_EXPIRED:
                                printf("Arm timer expired\r\n");
                                break;
                        case RELEASE_TIMER_EXPIRED:
                                printf("Release timer expired\r\n");
                                break;
                        case RIGHT_DELAY_TIMER_EXPIRED:
                                printf("Right delay timer expired\r\n");
                                break;
                        case LEFT_DELAY_TIMER_EXPIRED:
                                printf("Left delay timer expired\r\n");
                                break;
                        default:
```

```
                    printf("Error: undefined event!\r\n");
                    break;
                }
            }
        }
```

```
/*
 * Team 15: "Do It"
 * 3.5.09
 * Main program
 *
 * Tony Zhang
 * Agustin Ramirez
 * Nina Joshi
 */


#include <me218_c32.h>
#include <stdio.h>
#include <PWMS12.h>
#include <ADS12.h>
#include <timers12.h>
#include "defs.h"
#include "wheels.h"
#include "arms.h"
#include "events.h"
#include "beacon.h"
#include "release.h"
#include "switch.h"
#include "state.h"


static void MAIN_Debug(void);

void main (void) {
        // Initialize timers and wheels.
        TMRS12_Init(TMRS12_RATE_4MS);
        WHEELS_Init();

        // 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());
        }
}

/* 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");
        for(;;) {
            WHEELS_Debug();
            BEACON_Debug();
            ARMS_Debug();
            RELEASE_Debug();
            EVENTS_Debug();
            SWITCH_Debug();
        }
}
```

```
/*
 * Team 15: "Do It"
 * 2.27.09
 * Arm release module
 *
 * Tony Zhang
 * Agustin Ramirez
 * Nina Joshi
 */


#ifndef _RELEASE_H_
#define _RELEASE_H_

void RELEASE_ReleaseArms(void);
void RELEASE_StopRelease(void);
void RELEASE_Debug(void);

#endif
```

```
/*
 * Team 15: "Do It"
 * 2.27.09
 * Arm release module
 *
 * Tony Zhang
 * Agustin Ramirez
 * Nina Joshi
 */


#include <me218_c32.h>
#include <stdio.h>
#include <timers12.h>
#include "defs.h"
#include "release.h"

/* Starts the motor by setting the enable bit high. */
void RELEASE_ReleaseArms(void) {
	DDRT |= RELEASE_MOTOR_ENABLE_BIT;
	PTT |= RELEASE_MOTOR_ENABLE_BIT;
}

/* Stops the motor by clearing the enable bit. */
void RELEASE_StopRelease(void) {
	PTT &= ~RELEASE_MOTOR_ENABLE_BIT;
}

/* Release debug. */
void RELEASE_Debug(void) {
	// Break out of loop when user hits 'q'
	for(;;) {
	    printf("\nDebugging arm release... Controls:\r\n");
	    printf("press any key to release arms, q: quit\r\n");
	    // Releases arms when user hits a key.
	    if(getchar() == 'q') break;
	    //Start arm release motor, start release timer, and when timer expires, stop the motor.
	    printf("RELEASING...");
	    RELEASE_ReleaseArms();
	    TMRS12_InitTimer(RELEASE_TIMER, RELEASE_TIME);
	    while(!TMRS12_IsTimerExpired(RELEASE_TIMER)) ;
	    RELEASE_StopRelease();
	    printf("Stopped release motor\r\n");
	}
}
```

```
/*
 * Team 15: "Do It"
 * 2.28.09
 * State machine module
 *
 * Tony Zhang
 * Agustin Ramirez
 * Nina Joshi
 */


#ifndef _STATE_H_
#define _STATE_H_

void STATE_InitStateMachine(void);
void STATE_RunStateMachine(unsigned char event);

#endif
```

```
/*
 * Team 15: "Do It"
 * 3.5.09
 * State machine module
 *
 * Tony Zhang
 * Agustin Ramirez
 * Nina Joshi
 */


#include <me218_c32.h>
#include <stdio.h>
#include <timers12.h>
#include "state.h"
#include "defs.h"
#include "events.h"
#include "beacon.h"
#include "wheels.h"
#include "release.h"
#include "switch.h"

#define WAITING_FOR_SWITCH       0
#define KICKSPIN                 1
#define SEARCHING_FOR_BEACON     2
#define ORIENT                   3
#define WAITING_TO_MOVE_EAST     4
#define MOVING_EAST              5
#define WAITING_TO_MOVE_WEST     6
#define MOVING_WEST              7

static unsigned char state;
static void WaitingForSwitchState(unsigned char event);
static void KickspinState(unsigned char event);
static void SearchingForBeaconState(unsigned char event);
static void OrientState(unsigned char event);
static void WaitingToMoveEastState(unsigned char event);
static void MovingEastState(unsigned char event);
static void WaitingToMoveWestState(unsigned char event);
static void MovingWestState(unsigned char event);

/* Initialize state machine by setting initial state and wheel configuration. */
void STATE_InitStateMachine() {
        // Make sure the wheels are stopped.
        // Set initial state.
        WHEELS_Stop();
        state = WAITING_FOR_SWITCH;
}

/* Figure out which state we are in, and call the corresponding function to handle the event. */
void STATE_RunStateMachine(unsigned char event) {
        switch (state) {
            case WAITING_FOR_SWITCH:
              printf("Waiting \r\n");
                      WaitingForSwitchState(event);
                      printf("\r\n %d", event);
```

```
                break;
            case KICKSPIN:
              printf("Kickspin \r\n");
                KickspinState(event);
                printf("\r\n %d", event);
                break;
            case SEARCHING_FOR_BEACON:
              printf("Searching for beacon \r\n");
                SearchingForBeaconState(event);
                printf("\r\n %d", event);
                break;
            case ORIENT:
              printf("Orienting \r\n");
                OrientState(event);
                break;
            case WAITING_TO_MOVE_EAST:
              printf("Waiting to East \r\n");
                WaitingToMoveEastState(event);
                break;
            case MOVING_EAST:
            printf("East \r\n");
                MovingEastState(event);
                break;
            case WAITING_TO_MOVE_WEST:
              printf("Waiting to West \r\n");
                WaitingToMoveWestState(event);
                break;
            case MOVING_WEST:
              printf("West \r\n");
                MovingWestState(event);
                break;
            default:
                printf("ERROR: Undefined state\n");
                break;
        }
}

/* If switch goes on, kickspin wheels and set kickspin timer.*/
static void WaitingForSwitchState(unsigned char event) {
        switch(event) {
            case SWITCH_ON:
                TMRS12_InitTimer(END_TIMER, END_TIME);
                WHEELS_Kickspin();
                TMRS12_InitTimer(KICKSPIN_TIMER, KICKSPIN_TIME);
                state = KICKSPIN;
                break;
            case END_TIMER_EXPIRED:
                WHEELS_Stop();
                state = WAITING_FOR_SWITCH;
                break;
            default:
                break;
        }
}

/* When kickspin timer expires, continue with a normal spin and search for the beacon. */
```

```
static void KickspinState(unsigned char event) {
	switch(event) {
	   case KICKSPIN_TIMER_EXPIRED:
			WHEELS_Spin();
			state = SEARCHING_FOR_BEACON;
			break;
	   case END_TIMER_EXPIRED:
			WHEELS_Stop();
			state = WAITING_FOR_SWITCH;
			break;
	   default:
			break;
	}
}

/* When beacon is sensed as on, break our motors and when breaking completes, we move onto
orienting. */
static void SearchingForBeaconState(unsigned char event) {
	switch(event) {
	   case BEACON_ON:
			WHEELS_SpinBreak();
			TMRS12_InitTimer(REVERSE_TIMER, REVERSE_TIME);
			break;
	   case REVERSE_TIMER_EXPIRED:
			WHEELS_Orient();
			TMRS12_InitTimer(ARM_TIMER, ARM_TIME);
			state = ORIENT;
			break;
	   case END_TIMER_EXPIRED:
			WHEELS_Stop();
			state = WAITING_FOR_SWITCH;
			break;
	   default:
			break;
	}
}

/* When arm release timer expires, we begin releasing our arms. When the release timer expires,
 * begin moving the bot east.*/
static void OrientState(unsigned char event) {
	switch(event) {
	   case ARM_TIMER_EXPIRED:
			RELEASE_ReleaseArms();
			TMRS12_InitTimer(RELEASE_TIMER, RELEASE_TIME);
			break;
	   case RELEASE_TIMER_EXPIRED:
			RELEASE_StopRelease();
			WHEELS_MoveRight();
			state = MOVING_EAST;
			break;
	   case END_TIMER_EXPIRED:
			WHEELS_Stop();
			state = WAITING_FOR_SWITCH;
			break;
	   default:
			break;
```

```
	}
}

/* When right arm bumper is hit, switch bot direction to waiting to move west. */
static void MovingEastState(unsigned char event) {
	switch(event) {
	   case RIGHT_ARM_BUMPER_HIT:
			WHEELS_SlowMoveRight();
			TMRS12_InitTimer(RIGHT_DELAY_TIMER, RIGHT_DELAY_TIME);
			state = WAITING_TO_MOVE_WEST;
			break;
	   case END_TIMER_EXPIRED:
			WHEELS_Stop();
			state = WAITING_FOR_SWITCH;
	     break;
	   default:
			break;
	}
}

/* When right arm bumper delay timer expires, switch bot direction to moving west. */
static void WaitingToMoveWestState(unsigned char event) {
	switch(event) {
	   case RIGHT_DELAY_TIMER_EXPIRED:
			WHEELS_MoveLeft();
			state = MOVING_WEST;
			break;
	   case END_TIMER_EXPIRED:
			WHEELS_Stop();
			state = WAITING_FOR_SWITCH;
			break;
	   default:
			break;
	}
}

/* When left arm bumper is hit, switch bot direction to waiting to move east. */
static void MovingWestState(unsigned char event) {
	switch(event) {
	   case LEFT_ARM_BUMPER_HIT:
			WHEELS_SlowMoveLeft();
			TMRS12_InitTimer(LEFT_DELAY_TIMER, LEFT_DELAY_TIME);
			state = WAITING_TO_MOVE_EAST;
			break;
	   case END_TIMER_EXPIRED:
			WHEELS_Stop();
			state = WAITING_FOR_SWITCH;
			break;
	   default:
			break;
	}
}

/* When left arm bumper delay timer expires, switch bot direction to moving east. */
static void WaitingToMoveEastState(unsigned char event) {
	switch(event) {
```

```
            case LEFT_DELAY_TIMER_EXPIRED:
                    WHEELS_MoveRight();
                    state = MOVING_EAST;
                    break;
            case END_TIMER_EXPIRED:
                    WHEELS_Stop();
                    state = WAITING_FOR_SWITCH;
                    break;
            default:
                    break;
        }
}
```

```
/*
 * Team 15: "Do It"
 * 2.28.09
 * Switch module
 *
 * Tony Zhang
 * Agustin Ramirez
 * Nina Joshi
 */


#ifndef _SWITCH_H_
#define _SWITCH_H_

unsigned char SWITCH_CheckForSwitch(void);
void SWITCH_Debug(void);

#endif
```

```
/*
 * Team 15: "Do It"
 * 3.2.09
 * Switch module
 *
 * Tony Zhang
 * Agustin Ramirez
 * Nina Joshi
 */


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

// The previous switch event.
static unsigned char lastEvent = NO_EVENT;

/* Checks for switch event. */
unsigned char SWITCH_CheckForSwitch() {
	unsigned char event = NO_EVENT;

	if ((PTIAD & SWITCH_BIT) == 0){
	    event = SWITCH_ON;
	    return event;
	}

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

/* Switch debug. */
void SWITCH_Debug(void) {
	printf("\nDebugging switch...Controls:\r\n");
	printf("q: quit\r\n");
	// Break out of loop when user hits 'q'
	for(;;) {
	    // Prints switch value until user quits.
	    unsigned char event = SWITCH_CheckForSwitch();
	    if(event == SWITCH_ON) {
			printf("Switch on\r\n");
	    } else {
			printf("ERROR: unrecognizable switch\r\n");
	    }
	    // Clears input character and returns from the function.
	    if (getchar() == 'q') {
			lastEvent = NO_EVENT;
			return;
	    }
```

```
        }
    }
```

```
/*
 * Team 15: "Do It"
 * 3.5.09
 * Wheel module
 *
 * Tony Zhang
 * Agustin Ramirez
 * Nina Joshi
 */


#ifndef _WHEELS_H_
#define _WHEELS_H_

void WHEELS_Init(void);
void WHEELS_Kickspin(void);
void WHEELS_Spin(void);
void WHEELS_SpinBreak(void);
void WHEELS_Orient(void);
void WHEELS_MoveLeft(void);
void WHEELS_MoveRight(void);
void WHEELS_MediumMoveLeft(void);
void WHEELS_MediumMoveRight(void);
void WHEELS_SlowMoveLeft(void);
void WHEELS_SlowMoveRight(void);
void WHEELS_Stop(void);
void WHEELS_Debug(void);

#endif
```

```
/*
 * Team 15: "Do It"
 * 3.5.09
 * Wheel module
 *
 * Tony Zhang
 * Agustin Ramirez
 * Nina Joshi
 */


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

#define BACKWARD 1                      //Bot moving right if both are backward and on south wall
#define FORWARD 0                       //Bot moving left if both are forward and on south wall
#define MAX_SPEED 100
#define FAST 100
#define MEDIUM 70
#define SLOW 60
#define SPIN_SPEED 58
#define RIGHT_WHEEL_BIAS_LEFT .84
#define RIGHT_WHEEL_BIAS_RIGHT .855
#define LEFT_WHEEL_BIAS 1.4
#define Left_WHEEL_ORIENT_BIAS .9

static void SetLeftWheelSpeed(uchar speed);
static void SetRightWheelSpeed(uchar speed);
static void SetWheelDirections(uchar leftDir, uchar rightDir);

/* Initialize ports and PWM to control wheels. */
void WHEELS_Init(void) {
	PWMS12_Init(); //Initialize routine for the PWMS functions
	ADS12_Init("OOIIIAAA"); //Initiate A/D library

	//No need to configure Port T 0 and 1 since PWMS does this. Direction set as an output.
	DDRT |= LEFT_WHEEL_DIRECTION_BIT; //Motor 1 direction set as output
	DDRT |= RIGHT_WHEEL_DIRECTION_BIT; //Motor 2 direction set as output

	PWMS12_SetDuty(0, LEFT_WHEEL_CHANNEL_BIT); //Motor 1 off
	PWMS12_SetDuty(0, RIGHT_WHEEL_CHANNEL_BIT); //Motor 2 off

	PTT |= LEFT_WHEEL_DIRECTION_BIT; //Motor 1 direction set
	PTT |= RIGHT_WHEEL_DIRECTION_BIT; //Motor 2 direction set
}

/* Set wheels to max speed during a spin and spin directions to overcome friction. */
void WHEELS_Kickspin(void) {
	SetWheelDirections(FORWARD, BACKWARD); //CCW
	SetLeftWheelSpeed(MAX_SPEED);
	SetRightWheelSpeed(MAX_SPEED);
```

```
}

/* Set the wheels to normal spin speed. Assumes directions already set by WHEELS_Kickspin. */
void WHEELS_Spin(void) {
	SetLeftWheelSpeed(SPIN_SPEED*LEFT_WHEEL_BIAS); //CCW
	SetRightWheelSpeed(SPIN_SPEED);
}

/* Reverse wheels from spinning to quickly break spin. */
void WHEELS_SpinBreak(void) {
	SetWheelDirections(BACKWARD, FORWARD);
	SetLeftWheelSpeed(MAX_SPEED); //CW
	SetRightWheelSpeed(MAX_SPEED);
}

/* Sets wheels to perform a gentle turn to orient itself with its back against the south wall. */
void WHEELS_Orient(void) {
	SetWheelDirections(BACKWARD, BACKWARD);
	SetLeftWheelSpeed(MAX_SPEED*Left_WHEEL_ORIENT_BIAS);
	SetRightWheelSpeed(MAX_SPEED);
}

/* Moves right at maximum speed. */
void WHEELS_MoveRight(void) {
	SetWheelDirections(BACKWARD, BACKWARD);
	SetLeftWheelSpeed(MAX_SPEED);
	SetRightWheelSpeed(MAX_SPEED*RIGHT_WHEEL_BIAS_RIGHT);
}

/* Moves left at maximum speed. */
void WHEELS_MoveLeft(void) {
	SetWheelDirections(FORWARD, FORWARD);
	SetLeftWheelSpeed(MAX_SPEED);
	SetRightWheelSpeed(MAX_SPEED*RIGHT_WHEEL_BIAS_LEFT);
}

/* Moves right at medium speed. */
void WHEELS_MediumMoveRight(void) {
	SetWheelDirections(BACKWARD, BACKWARD);
	SetLeftWheelSpeed(MEDIUM);
	SetRightWheelSpeed(MEDIUM*RIGHT_WHEEL_BIAS_RIGHT);
}

/* Moves left at medium speed. */
void WHEELS_MediumMoveLeft(void) {
	SetWheelDirections(FORWARD, FORWARD);
	SetLeftWheelSpeed(MEDIUM);
	SetRightWheelSpeed(MEDIUM*RIGHT_WHEEL_BIAS_LEFT);
}

/* Moves right at slow speed. */
void WHEELS_SlowMoveRight(void) {
	SetWheelDirections(BACKWARD, BACKWARD);
	SetLeftWheelSpeed(SLOW);
	SetRightWheelSpeed(SLOW*RIGHT_WHEEL_BIAS_RIGHT);
}
```

```
/* Moves left at slow speed. */
void WHEELS_SlowMoveLeft(void) {
	SetWheelDirections(FORWARD, FORWARD);
	SetLeftWheelSpeed(SLOW);
	SetRightWheelSpeed(SLOW*RIGHT_WHEEL_BIAS_LEFT);
}

/* Stops wheel movement. */
void WHEELS_Stop(void) {
	SetLeftWheelSpeed (0);
	SetRightWheelSpeed (0);
}

/* Sets left wheel speed to a specified value. */
static void SetLeftWheelSpeed(uchar speed) {
	PWMS12_SetDuty(speed, LEFT_WHEEL_CHANNEL_BIT); //Motor 1
}

/* Sets right wheel speed to a specified value. */
static void SetRightWheelSpeed(uchar speed) {
	PWMS12_SetDuty(speed, RIGHT_WHEEL_CHANNEL_BIT); //Motor 2
}

/* Sets wheel directions to either forward or backward as specified. */
static void SetWheelDirections (uchar leftDir, uchar rightDir) {
	if (leftDir == FORWARD) {
	   PTT |= LEFT_WHEEL_DIRECTION_BIT; //Set left wheel bit if moving forward
	}
	else {
	   PTT &= ~LEFT_WHEEL_DIRECTION_BIT; //Clear left wheel bit if moving backward
	}
	if (rightDir == FORWARD) {
	   PTT |= RIGHT_WHEEL_DIRECTION_BIT; //Set right wheel bit if moving forward
	}
	else {
	   PTT &= ~RIGHT_WHEEL_DIRECTION_BIT; //Clear right wheel bit if moving backward
	}
}

/* To debug, hit 'r' to move right, 'l' to move left, 's' to spin, or 'q' to quit. */
void WHEELS_Debug(void) {
	printf("\nDebugging wheels... Controls:\r\n");
	printf("r: move right, l: move left, s: spin, q: quit\r\n");
	// Break out of loop when user hits 'q'
	for(;;) {
	   // User selects options
	   char ch = getchar();
	   switch(ch) {
			case 'l':
				printf("Move left...\r\n");
				WHEELS_MoveLeft();
				break;
			case 'r':
				printf("Move right...\r\n");
				WHEELS_MoveRight();
```

```
                        break;
                case 's':
                        printf("Spinning...\r\n");
                        //Kickspin motors, start kickspin timer, and wait until it expires to continue
with normal wheel spinning.
                        WHEELS_Kickspin();
                        TMRS12_InitTimer(KICKSPIN_TIMER, KICKSPIN_TIME);
                        while(!TMRS12_IsTimerExpired(KICKSPIN_TIMER));
                        WHEELS_Spin();
                        break;
                case 'q':
                        // Stops the wheels and returns from the function.
                        WHEELS_Stop();
                        return;
            }
        }
}
```