Contents

main.c

```c
#include <stdint.h>
#include "timer.h"
#include "cmod.h"
#include "oi.h"

#include "driving.h"
#include "irobled.h"
#include "sensing.h"

#include "iroblife.h"

// Delay constants
#define TURN_DELAY_MS   (500)
#define IROB_PERIOD_MS  (100)
#define IROB_CUTOFF_MS  (10)

// Pentagon constants
#define PENTAGON_SIDE_LENGTH    (800)
#define PENTAGON_INNER_ANGLE    (108)
#define PENTAGON_SUPPLEMENT     (180 - PENTAGON_INNER_ANGLE)

// Error for pentagon constants
#define SIDE_LENGTH_ERROR   (5)
#define INNER_ANGLE_ERROR   (8)
#define SUPPLEMENT_ERROR    (5)

// Drive speed
#define DRIVE_SPEED         (50)
// Actually used pentagon constants (with error added)
#define DRIVE_SIDE_LENGTH   (PENTAGON_SIDE_LENGTH + SIDE_LENGTH_ERROR)
#define DRIVE_INNER_ANGLE   (PENTAGON_INNER_ANGLE + INNER_ANGLE_ERROR)
#define DRIVE_SUPPLEMENT    (PENTAGON_SUPPLEMENT + SUPPLEMENT_ERROR)


// Declare global variables
uint8_t suppressButtons = 0;


// Called by irobPeriodic
void iroblifePeriodic(void) {
    // Get bumps and wheel drops sensor packet
    uint8_t bumps_wd = read1ByteSensorPacket(PACKET_BUMPS_AND_WHEEL_DROPS);
    // Variable for holding the new Create LED bits
    uint8_t newLedBits = 0x00;
    if (bumps_wd & MASK_BUMP_LEFT) {
        // The left bumper is active. Light the play LED.
        newLedBits |= PLAY_ROBOT_LED;
    }
    if (bumps_wd & MASK_BUMP_RIGHT) {
        // The right bumper is active. Light the advance LED.
        newLedBits |= ADVANCE_ROBOT_LED;
    }
    // Send the LED bits to the Create
    robotLedSetBits(newLedBits);
```

```
55      if (bumps_wd & MASK_WHEEL_DROP) {
56          // The Create was picked up or is about to go over a ledge. Abort.
57          irobEnd();
58      }
59      // Toggle the command module LEDs
60      byteTx(LEDBothToggle);
61  }
62
63
64  void drivePentagon(int16_t radius) {
65      // Lock Create buttons
66      suppressButtons = 1;
67      // Wait to allow motors to settle
68      delayMsFunc(TURN_DELAY_MS, &irobPeriodic, IROB_PERIOD_MS,
69              IROB_CUTOFF_MS);
70      uint8_t i;
71      for (i = 0; i < 5; i++) { // A pentagon has five sides
72          // Drive along a side
73          driveDistanceTFunc(DRIVE_SPEED, DRIVE_SIDE_LENGTH, &irobPeriodic,
74                  IROB_PERIOD_MS, IROB_CUTOFF_MS);
75          // Wait to allow motors to settle
76          delayMsFunc(TURN_DELAY_MS, &irobPeriodic, IROB_PERIOD_MS,
77                  IROB_CUTOFF_MS);
78          // Turn to face the next side
79          driveAngleTFunc(DRIVE_SPEED, radius, DRIVE_SUPPLEMENT, &irobPeriodic,
80                  IROB_PERIOD_MS, IROB_CUTOFF_MS);
81          // Wait to allow motors to settle
82          delayMsFunc(TURN_DELAY_MS, &irobPeriodic, IROB_PERIOD_MS,
83                  IROB_CUTOFF_MS);
84      }
85      // Unlock Create buttons
86      suppressButtons = 0;
87  }
88
89
90  int main(void) {
91      // Submit to irobPeriodic
92      setIrobPeriodicImpl(&iroblifePeriodic);
93
94      // Initialize the Create
95      irobInit();
96
97      // Initialize global variables
98      suppressButtons = 0;
99
100     // Infinite operation loop
101     for(;;) {
102         // Periodic execution
103         irobPeriodic();
104
105         // Code not executed during delay loops
106         if (!suppressButtons) {
107             // Read the button packet from the Create
108             uint8_t buttons = read1ByteSensorPacket(PACKET_BUTTONS);
109             if (buttons & MASK_BTN_PLAY) {
110                 // The Play button is down. Drive in a Clockwise pentagon.
111                 drivePentagon(RadCW);
```

```
112             } else if (buttons & MASK_BTN_ADVANCE) {
113                 // The Advance button is down. Drive in Counterclockwise pentagon.
114                 // Get into the correct angle
115                 driveAngleTFunc(DRIVE_SPEED, RadCW, DRIVE_INNER_ANGLE, &irobPeriodic,
116                         IROB_PERIOD_MS, IROB_CUTOFF_MS);
117                 // Drive the pentagon
118                 drivePentagon(RadCCW);
119                 // Return to the original angle
120                 driveAngleTFunc(DRIVE_SPEED, RadCCW, DRIVE_INNER_ANGLE, &irobPeriodic,
121                         IROB_PERIOD_MS, IROB_CUTOFF_MS);
122             }
123             // Make sure the create is stopped
124             driveStop();
125         }
126 
127         // Delay for the loop
128         delayMs(IROB_PERIOD_MS);
129     }
130 }
```

utils/cmod.c

```c
#include "cmod.h"
#include "oi.h"
#include "timer.h"

void initializeCommandModule(void){
  // Disable interrupts. ("Clear interrupt bit")
  cli();

  // One-time setup operations.
  setupIOPins();
  setupTimer();
  setupSerialPort();

  // Enable interrupts. ("Set interrupt bit")
  sei();
}

void setupIOPins(void) {
  // Set I/O pins
  DDRB  = 0x10;
  PORTB = 0xCF;
  DDRC  = 0x00;
  PORTC = 0xFF;
  DDRD  = 0xE6;
  PORTD = 0x7D;
}

void setupSerialPort(void) {
  // Set the transmission speed to 57600 baud, which is what the Create expects,
  // unless we tell it otherwise.
  UBRR0 = 19;

  // Enable both transmit and receive.
  UCSR0B = (_BV(RXCIE0) | _BV(TXEN0) | _BV(RXEN0));
    // UCSR0B = 0x18;

  // Set 8-bit data.
  UCSR0C = (_BV(UCSZ00) | _BV(UCSZ01));
    // UCSR0C = 0x06;
}

void byteTx(uint8_t value) {
  // Transmit one byte to the robot.
  // Wait for the buffer to be empty.
  while(!(UCSR0A & 0x20)) ;

  // Send the byte.
  UDR0 = value;
}

uint8_t byteRx(void) {
  // Receive one byte from the robot.
  // Call setupSerialPort() first.
  // Wait for a byte to arrive in the recieve buffer.
```

```
55    while(!(UCSR0A & 0x80)) ;
56
57    // Return that byte.
58    return UDR0;
59  }
60
61  void baud(uint8_t baud_code) {
62    // Switch the baud rate on both Create and module
63    if(baud_code <= 11)
64    {
65      byteTx(CmdBaud);
66      UCSR0A |= _BV(TXC0);
67      byteTx(baud_code);
68      // Wait until transmit is complete
69      while(!(UCSR0A & _BV(TXC0))) ;
70
71      cli();
72
73      // Switch the baud rate register
74      if(baud_code == Baud115200) {
75        UBRR0 = Ubrr115200;
76      } else if(baud_code == Baud57600) {
77        UBRR0 = Ubrr57600;
78      } else if(baud_code == Baud38400) {
79        UBRR0 = Ubrr38400;
80      } else if(baud_code == Baud28800) {
81        UBRR0 = Ubrr28800;
82      } else if(baud_code == Baud19200) {
83        UBRR0 = Ubrr19200;
84      } else if(baud_code == Baud14400) {
85        UBRR0 = Ubrr14400;
86      } else if(baud_code == Baud9600) {
87        UBRR0 = Ubrr9600;
88      } else if(baud_code == Baud4800) {
89        UBRR0 = Ubrr4800;
90      } else if(baud_code == Baud2400) {
91        UBRR0 = Ubrr2400;
92      } else if(baud_code == Baud1200) {
93        UBRR0 = Ubrr1200;
94      } else if(baud_code == Baud600) {
95        UBRR0 = Ubrr600;
96      } else if(baud_code == Baud300) {
97        UBRR0 = Ubrr300;
98      }
99      sei();
100
101     delayMs(100);
102   }
103 }
```

utils/cmod.h

```
1  #ifndef INCLUDE_CMOD_H
2  #define INCLUDE_CMOD_H
3
4    #include <avr/io.h>
5    #include <avr/interrupt.h>
6
7    // Setup the I/O pins.
8    void setupIOPins(void);
9
10   // Setup the serial port: Baud rate, transmit/recieve, packet size.
11   void setupSerialPort(void);
12
13   // Contains a collection of commands that allows me to "start" immediately
14   // after calling this command.
15   void initializeCommandModule(void);
16
17   // Send and receive data from the Command Module
18   void byteTx(uint8_t value);
19   uint8_t byteRx(void);
20
21   // Switch the baud rate on both Create and module
22   void baud(uint8_t baud_code);
23 #endif
```

utils/driving.c

```c
#include <stdint.h>
#include "driving.h"
#include "oi.h"
#include "cmod.h"
#include "timer.h"

// Weird constants because squeezing out precision
#define PIe5            314159
#define TENTH_RADIUS    13

// # BASIC COMMANDS #

void drive(int16_t velocity, int16_t radius) {
    // Send the start driving command to the Create
    byteTx(CmdDrive);
    byteTx((uint8_t)((velocity >> 8) & 0x00FF));
    byteTx((uint8_t)(velocity & 0x00FF));
    byteTx((uint8_t)((radius >> 8) & 0x00FF));
    byteTx((uint8_t)(radius & 0x00FF));
}

void driveStop(void) {
    drive(0, RadStraight);
}


// # OPCODE-BASED COMMANDS #

void driveDistanceOp(int16_t velocity, int16_t distance) {
    // Start driving
    drive(velocity, RadStraight);
    // Halt execution of new commands on the Create until reached distance
    byteTx(WaitForDistance);
    byteTx((uint8_t)((distance >> 8) & 0x00FF));
    byteTx((uint8_t)(distance & 0x00FF));
    // Stop the Create
    driveStop();
}

void driveAngleOp(int16_t velocity, int16_t radius, int16_t angle) {
    // Wait for angle opcode compatibility
    if (radius == RadCW) {
        angle = -angle;
    }
    // Start driving
    drive(velocity, radius);
    // Halt execution of new commands on the Create until reached angle
    byteTx(WaitForAngle);
    byteTx((uint8_t)((angle >> 8) & 0x00FF));
    byteTx((uint8_t)(angle & 0x00FF));
    // Stop the Create
    driveStop();
}

```

```
55
56 // # TIMER-BASED COMMANDS #
57
58 void driveDistanceTFunc(int16_t velocity, int16_t distance, void (*func)(void),
59         uint16_t period_ms, uint16_t cutoff_ms) {
60     // Calculate the delay
61     uint32_t time_ms = (1000 * (uint32_t)distance) / (uint32_t)velocity;
62     // Start driving
63     drive(velocity, RadStraight);
64     // Wait delay
65     delayMsFunc(time_ms, func, period_ms, cutoff_ms);
66     // Stop the Create
67     driveStop();
68 }
69
70 void driveAngleTFunc(int16_t velocity, int16_t radius, int16_t angle,
71         void (*func)(void), uint16_t period_ms, uint16_t cutoff_ms) {
72     // Calculate the delay
73     uint32_t time_ms = (PIe5 * TENTH_RADIUS * (uint32_t)angle)
74         / (1800 * (uint32_t)velocity);
75     // Start driving
76     drive(velocity, radius);
77     // Wait delay
78     delayMsFunc(time_ms, func, period_ms, cutoff_ms);
79     // Stop the Create
80     driveStop();
81 }
```

utils/driving.h

```
1  #ifndef DRIVING_H
2  #define DRIVING_H
3
4  #include <stdint.h>
5
6  // # BASIC COMMANDS #
7
8  //! Drive at a certain speed in a certain direction.
9  /*!
10  *  Returns immediately.
11  *
12  *  Directions: straight, clockwise, counterclockwise.
13  *
14  *  \param velocity     The speed in mm/s.
15  *  \param radius       Either RadStraight, RadCW, or RadCCW (see oi.h).
16  */
17 void drive(int16_t velocity, int16_t radius);
18
19 //! Stop the robot.
20 void driveStop(void);
21
22
23 // # OPCODE-BASED COMMANDS #
24
25 //! Drive a certain distance at a certain speed.
26 /*!
27  *  Drive a certain distance using the Create wait for distance opcode.
28  *
29  *  \param velocity     The speed in mm/s.
30  *  \param distance     The distance to travel in mm.
31  */
32 void driveDistanceOp(int16_t velocity, int16_t distance);
33
34 //! Rotate a certain angle at a certain speed.
35 /*!
36  *  Drive a certain angle using the Create wait for angle opcode.
37  *
38  *  \param velocity     The speed in mm/s.
39  *  \param radius       Either RadCW or RadCCW (see oi.h).
40  *  \param angle        The angle to rotate in degrees.
41  */
42 void driveAngleOp(int16_t velocity, int16_t radius, int16_t angle);
43
44
45 // # TIMER-BASED COMMANDS #
46
47 //! Drive a certain distance at a certain speed.
48 /*!
49  *  Drive a certain distance using a timer.
50  *
51  *  \param velocity     The speed in mm/s.
52  *  \param distance     The distance to travel in mm.
53  *  \param func         The function to execute periodically.
54  *  \param period_ms    The interval to execute the function.
```

```
55   *  \param cutoff_ms    The number of milliseconds before the end to stop
56   *                      attempting to start the function.
57   */
58  void driveDistanceTFunc(int16_t velocity, int16_t distance, void (*func)(void),
59          uint16_t period_ms, uint16_t cutoff_ms);
60
61  //! Drive a certain angle at a certain speed.
62  /*!
63   *  Drive a certain angle using a timer.
64   *
65   *  \param velocity     The speed in mm/s.
66   *  \param radius       Either RadCW or RadCCW (see oi.h).
67   *  \param angle        The angle to rotate in degrees.
68   *  \param func         The function to execute periodically.
69   *  \param period_ms    The interval to execute the function.
70   *  \param cutoff_ms    The number of milliseconds before the end to stop
71   *                      attempting to start the function.
72   */
73  void driveAngleTFunc(int16_t velocity, int16_t radius, int16_t angle,
74          void (*func)(void), uint16_t period_ms, uint16_t cutoff_ms);
75
76  #endif
```

utils/irobled.c

```c
#include <stdint.h>
#include "irobled.h"
#include "cmod.h"
#include "oi.h"

// The current state of the leds.
struct {
    uint8_t bits;
    uint8_t color;
    uint8_t intensity;
} iroblibState;

void irobledCmd(uint8_t bits, uint8_t color, uint8_t intensity) {
    // Modify the state
    iroblibState.bits = bits;
    iroblibState.color = color;
    iroblibState.intensity = intensity;
    // Update
    irobledUpdate();
}

void irobledUpdate(void) {
    // Send the led command using the current state
    byteTx(CmdLeds);
    byteTx(iroblibState.bits);
    byteTx(iroblibState.color);
    byteTx(iroblibState.intensity);
}

void irobledInit(void) {
    irobledCmd(NEITHER_ROBOT_LED, POWER_LED_RED, 0xFF);
}

void powerLedSet(uint8_t color, uint8_t intensity) {
    irobledCmd(iroblibState.bits, color, intensity);
}

void robotLedSetBits(uint8_t bits) {
    iroblibState.bits = bits;
    irobledUpdate();
}

void robotLedOn(uint8_t led) {
    iroblibState.bits |= led;
    irobledUpdate();
}

void robotLedOff(uint8_t led) {
    iroblibState.bits &= ~led;
    irobledUpdate();
}

void robotLedToggle(uint8_t led) {
    iroblibState.bits ^= led;
```

```
55      irobledUpdate();
56  }
```

utils/irobled.h

```
1  #ifndef IROBLED_H
2  #define IROBLED_H
3
4  #include <stdint.h>
5
6  // Colors for the power led.
7  #define POWER_LED_GREEN   (0x00)
8  #define POWER_LED_RED     (0xFF)
9
10 // Bits for the other leds.
11 #define NEITHER_ROBOT_LED (0x00)
12 #define PLAY_ROBOT_LED    (0x02)
13 #define ADVANCE_ROBOT_LED (0x08)
14 #define BOTH_ROBOT_LED    (0x0A)
15
16 //! Send an led command to the Create.
17 void irobledCmd(uint8_t bits, uint8_t color, uint8_t intensity);
18 //! Update the leds. Probably won't have to use.
19 void irobledUpdate(void);
20 //! Initialize the leds to red for power and off for the others.
21 void irobledInit(void);
22
23 //! Set the color and intensity of the power led.
24 void powerLedSet(uint8_t color, uint8_t intensity);
25
26 // Functions for modifying one or both of the other leds.
27 void robotLedSetBits(uint8_t bits);
28 void robotLedOn(uint8_t led);
29 void robotLedOff(uint8_t led);
30 void robotLedToggle(uint8_t led);
31
32 #endif
```

utils/iroblib.c

```c
#include "iroblib.h"
#include "oi.h"
#include "cmod.h"
#include "timer.h"

// Define songs to be played later
void defineSongs(void) {
  // Reset song
  byteTx(CmdSong);
  byteTx(RESET_SONG);
  byteTx(4);
  byteTx(60);
  byteTx(6);
  byteTx(72);
  byteTx(6);
  byteTx(84);
  byteTx(6);
  byteTx(96);
  byteTx(6);

  // Start song
  byteTx(CmdSong);
  byteTx(START_SONG);
  byteTx(6);
  byteTx(69);
  byteTx(18);
  byteTx(72);
  byteTx(12);
  byteTx(74);
  byteTx(12);
  byteTx(72);
  byteTx(12);
  byteTx(69);
  byteTx(12);
  byteTx(77);
  byteTx(24);
}

// Ensure that the robot is On.
void powerOnRobot(void) {
  // If Create's power is off, turn it on
  if(!RobotIsOn) {
    while(!RobotIsOn) {
      RobotPwrToggleLow;
      delayMs(500);  // Delay in this state
      RobotPwrToggleHigh;  // Low to high transition to toggle power
      delayMs(100);  // Delay in this state
      RobotPwrToggleLow;
    }
    delayMs(3500);  // Delay for startup
  }

  // Flush the buffer
  while( (UCSR0A & 0x80) && UDR0);
```

```
55  }
56
57  // Ensure that the robot is OFF.
58  void powerOffRobot(void) {
59    // If Create's power is on, turn it off
60    if(RobotIsOn) {
61      while(RobotIsOn) {
62        RobotPwrToggleLow;
63        delayMs(500);  // Delay in this state
64        RobotPwrToggleHigh;  // Low to high transition to toggle power
65        delayMs(100);  // Delay in this state
66        RobotPwrToggleLow;
67      }
68    }
69  }
```

utils/iroblib.h

```
1  #ifndef INCLUDE_IROBLIB_H
2  #define INCLUDE_IROBLIB_H
3
4  #include <avr/io.h>
5  #include <avr/interrupt.h>
6
7  // Constants
8  #define RESET_SONG 0
9  #define START_SONG 1
10
11 void defineSongs(void);
12   // Songs
13   // Indicator that the robot is Powered on and has reset.
14
15 void powerOnRobot(void);
16 void powerOffRobot(void);
17   // Power the create On/Off.
18 #endif
```

utils/iroblife.c

```c
#include <stdlib.h>
#include <stdint.h>
#include "iroblife.h"

#include "timer.h"
#include "cmod.h"
#include "iroblib.h"
#include "oi.h"

#include "irobled.h"
#include "driving.h"

void irobPeriodicImplNull(void) {
}

void (*irobPeriodicImpl)(void) = &irobPeriodicImplNull;

void setIrobPeriodicImpl(void (*func)(void)) {
    irobPeriodicImpl = func;
}

void irobInit(void) {
    // Set up Create and module
    initializeCommandModule();

    // Is the Robot on
    powerOnRobot();
    // Start the create
    byteTx(CmdStart);
    // Set the baud rate for the Create and Command Module
    baud(Baud57600);
    // Define some songs so that we know the robot is on.
    defineSongs();
    // Deprecated form of safe mode. I use it because it will
    // turn of all LEDs, so it's essentially a reset.
    byteTx(CmdControl);
    // We are operating in FULL mode.
    byteTx(CmdFull);

    // Make sure the robot stops.
    // As a precaution for the robot and your grade.
    driveStop();

    // Play the reset song and wait while it plays.
    byteTx(CmdPlay);
    byteTx(RESET_SONG);
    delayMs(750);

    // Turn the power button on to red.
    irobledInit();
}

void irobPeriodic(void) {
    // Call the user's periodic function
```

```
55      irobPeriodicImpl();
56      // Exit if the black button on the command module is pressed.
57      if(UserButtonPressed) {
58          irobEnd();
59      }
60  }
61
62  void irobEnd(void) {
63      // Stop the Create
64      driveStop();
65      // Power off the Create
66      powerOffRobot();
67      // Exit the program
68      exit(1);
69  }
```

utils/iroblife.h

```
1  #ifndef IROBLIFE_H
2  #define IROBLIFE_H
3
4  /*
5   *  The irobPeriodic function in this library calls a function given to
6   *  setIrobPeriodicImpl. The default value does nothing, but you can give
7   *  it another function as a hook for periodically executed code.
8   */
9
10 //! Default periodic function. Does nothing.
11 void irobPeriodicImplNull(void);
12 //! Set the function that irobPeriodic calls.
13 void setIrobPeriodicImpl(void (*func)(void));
14
15 //! Initialize the Create. Call this at the beginning of your main.
16 void irobInit(void);
17 //! Periodic operations. Call this in your main loop.
18 //! Calls the function last given to setIrobPeriodicImpl.
19 void irobPeriodic(void);
20 //! Stops and shuts down the Create, then exits. Call this to end the program.
21 void irobEnd(void);
22
23 #endif
```

utils/oi.h

```
1  /* oi.h
2   *
3   * Definitions for the Open Interface
4   */
5
6  #ifndef OI_H
7  #define OI_H
8
9  // Command values
10 #define CmdStart        128
11 #define CmdBaud         129
12 #define CmdControl      130
13 #define CmdSafe         131
14 #define CmdFull         132
15 #define CmdSpot         134
16 #define CmdClean        135
17 #define CmdDemo         136
18 #define CmdDrive        137
19 #define CmdMotors       138
20 #define CmdLeds         139
21 #define CmdSong         140
22 #define CmdPlay         141
23 #define CmdSensors      142
24 #define CmdDock         143
25 #define CmdPWMMotors    144
26 #define CmdDriveWheels  145
27 #define CmdOutputs      147
28 #define CmdSensorList   149
29 #define CmdIRChar       151
30 #define WaitForDistance 156
31 #define WaitForAngle    157
32
33
34 // Sensor byte indices - offsets in packets 0, 5 and 6
35 #define SenBumpDrop     0
36 #define SenWall         1
37 #define SenCliffL       2
38 #define SenCliffFL      3
39 #define SenCliffFR      4
40 #define SenCliffR       5
41 #define SenVWall        6
42 #define SenOverC        7
43 #define SenIRChar       10
44 #define SenButton       11
45 #define SenDist1        12
46 #define SenDist0        13
47 #define SenAng1         14
48 #define SenAng0         15
49 #define SenChargeState  16
50 #define SenVolt1        17
51 #define SenVolt0        18
52 #define SenCurr1        19
53 #define SenCurr0        20
54 #define SenTemp         21
```

```
55  #define SenCharge1      22
56  #define SenCharge0      23
57  #define SenCap1         24
58  #define SenCap0         25
59  #define SenWallSig1     26
60  #define SenWallSig0     27
61  #define SenCliffLSig1   28
62  #define SenCliffLSig0   29
63  #define SenCliffFLSig1  30
64  #define SenCliffFLSig0  31
65  #define SenCliffFRSig1  32
66  #define SenCliffFRSig0  33
67  #define SenCliffRSig1   34
68  #define SenCliffRSig0   35
69  #define SenInputs       36
70  #define SenAInput1      37
71  #define SenAInput0      38
72  #define SenChAvailable  39
73  #define SenOIMode       40
74  #define SenOISong       41
75  #define SenOISongPlay   42
76  #define SenStreamPckts  43
77  #define SenVel1         44
78  #define SenVel0         45
79  #define SenRad1         46
80  #define SenRad0         47
81  #define SenVelR1        48
82  #define SenVelR0        49
83  #define SenVelL1        50
84  #define SenVelL0        51
85
86
87  // Sensor packet sizes
88  #define Sen0Size        26
89  #define Sen1Size        10
90  #define Sen2Size        6
91  #define Sen3Size        10
92  #define Sen4Size        14
93  #define Sen5Size        12
94  #define Sen6Size        52
95
96  // Sensor bit masks
97  #define WheelDropFront  0x10
98  #define WheelDropLeft   0x08
99  #define WheelDropRight  0x04
100 #define BumpLeft        0x02
101 #define BumpRight       0x01
102 #define BumpBoth        0x03
103 #define BumpEither      0x03
104 #define WheelDropAll    0x1C
105 #define ButtonAdvance   0x04
106 #define ButtonPlay      0x01
107
108
109 // LED Bit Masks
110 #define LEDAdvance       0x08
111 #define LEDPlay         0x02
```

```
112 #define LEDsBoth        0x0A
113
114 // OI Modes
115 #define OIPassive       1
116 #define OISafe          2
117 #define OIFull          3
118
119
120 // Baud codes
121 #define Baud300         0
122 #define Baud600         1
123 #define Baud1200        2
124 #define Baud2400        3
125 #define Baud4800        4
126 #define Baud9600        5
127 #define Baud14400       6
128 #define Baud19200       7
129 #define Baud28800       8
130 #define Baud38400       9
131 #define Baud57600       10
132 #define Baud115200      11
133
134
135 // Drive radius special cases
136 #define RadStraight     32768
137 #define RadCCW          1
138 #define RadCW           -1
139
140
141
142 // Baud UBRRx values
143 #define Ubrr300         3839
144 #define Ubrr600         1919
145 #define Ubrr1200        959
146 #define Ubrr2400        479
147 #define Ubrr4800        239
148 #define Ubrr9600        119
149 #define Ubrr14400       79
150 #define Ubrr19200       59
151 #define Ubrr28800       39
152 #define Ubrr38400       29
153 #define Ubrr57600       19
154 #define Ubrr115200      9
155
156
157 // Command Module button and LEDs
158 #define UserButton        0x10
159 #define UserButtonPressed (!(PIND & UserButton))
160
161 #define LED1              0x20
162 #define LED1Off           (PORTD |= LED1)
163 #define LED1On            (PORTD &= ~LED1)
164 #define LED1Toggle        (PORTD ^= LED1)
165
166 #define LED2              0x40
167 #define LED2Off           (PORTD |= LED2)
168 #define LED2On            (PORTD &= ~LED2)
```

```
169 #define LED2Toggle        (PORTD ^= LED2)
170
171 #define LEDBoth           0x60
172 #define LEDBothOff        (PORTD |= LEDBoth)
173 #define LEDBothOn         (PORTD &= ~LEDBoth)
174 #define LEDBothToggle     (PORTD ^= LEDBoth)
175
176
177 // Create Port
178 #define RobotPwrToggle      0x80
179 #define RobotPwrToggleHigh (PORTD |= 0x80)
180 #define RobotPwrToggleLow  (PORTD &= ~0x80)
181
182 #define RobotPowerSense    0x20
183 #define RobotIsOn          (PINB & RobotPowerSense)
184 #define RobotIsOff         !(PINB & RobotPowerSense)
185
186 // Command Module ePorts
187 #define LD2Over         0x04
188 #define LD0Over         0x02
189 #define LD1Over         0x01
190
191 #endif
```

utils/sensing.c

```c
#include <stdint.h>
#include "sensing.h"
#include "cmod.h"
#include "oi.h"

uint8_t read1ByteSensorPacket(uint8_t packetId) {
    // Send the packet ID
    byteTx(CmdSensors);
    byteTx(packetId);
    // Read the packet byte
    return byteRx();
}
```

utils/sensing.h

```
#ifndef SENSING_H
#define SENSING_H

#include <stdint.h>

#define PACKET_BUMPS_AND_WHEEL_DROPS    (7)
#define MASK_WHEEL_DROP_CASTER          (1 << 4)
#define MASK_WHEEL_DROP_LEFT            (1 << 3)
#define MASK_WHEEL_DROP_RIGHT           (1 << 2)
#define MASK_WHEEL_DROP                 (0x1C)
#define MASK_BUMP_LEFT                  (1 << 1)
#define MASK_BUMP_RIGHT                 (1 << 0)
#define MASK_BUMP                       (0x03)

#define PACKET_BUTTONS                  (18)
#define MASK_BTN_ADVANCE                (1 << 2)
#define MASK_BTN_PLAY                   (1 << 0)

//! Read in a 1-byte sensor packet.
/*!
 *  What is a sensor packet? A byte (or bytes) containing data from a set of
 *  sensors, often shifted and ORed together. See the Create Open Interface
 *  documentation for more.
 *
 *  Currently Available Sensor Packets  (v = read1ByteSensorPacket(packetId)):
 *      Bumps and Wheel Drops   (packetId = PACKET_BUMPS_AND_WHEEL_DROPS):
 *          Caster Drop         (v & MASK_WHEEL_DROP_CASTER)
 *          Left Wheel Drop     (v & MASK_WHEEL_DROP_LEFT)
 *          Right Wheel Drop    (v & MASK_WHEEL_DROP_RIGHT)
 *          Any Wheel Drop      (v & MASK_WHEEL_DROP)
 *          Left Bumper         (v & MASK_BUMP_LEFT)
 *          Right Bumper        (v & MASK_BUMP_RIGHT)
 *          Either Bumper       (v & MASK_BUMPER)
 *      Create Buttons          (packetId = PACKET_BUTTONS):
 *          Advance Button      (v & MASK_BTN_ADVANCE)
 *          Play Button         (v & MASK_BTN_PLAY)
 *
 *  \param packetId     The ID of the packet to retrieve, as defined by the
 *                      Create Open Interface.
 */
uint8_t read1ByteSensorPacket(uint8_t packetId);

#endif
```

utils/timer.c

```c
#include <stdint.h>
#include "timer.h"    // Declaration made available here


// Timer variables defined here
volatile uint32_t delayTimerCount = 0;   // Definition checked against declaration
volatile uint8_t  delayTimerRunning = 0; // Definition checked against declaration


ISR(USART_RX_vect) {  //SIGNAL(SIG_USART_RECV)
    // Serial receive interrupt to store sensor values

    // CSCE 274 students, I have only ever used this method
    // when retrieving/storing a large amount of sensor data.
    // You DO NOT need it for this assignment. If i feel it
    // becomes relevant, I will show you how/when to use it.
}

//SIGNAL(SIG_OUTPUT_COMPARE1A)
ISR(TIMER1_COMPA_vect) {
    // Interrupt handler called every 1ms.
    // Decrement the counter variable, to allow delayMs to keep time.
    if(delayTimerCount != 0) {
        delayTimerCount--;
    } else {
        delayTimerRunning = 0;
    }
}

void setupTimer(void) {
    // Set up the timer 1 interupt to be called every 1ms.
    // It's probably best to treat this as a black box.
    // Basic idea: Except for the 71, these are special codes, for which details
    // appear in the ATMega168 data sheet. The 71 is a computed value, based on
    // the processor speed and the amount of "scaling" of the timer, that gives
    // us the 1ms time interval.
    TCCR1A = 0x00;
    // TCCR1B = 0x0C;
    TCCR1B = (_BV(WGM12) | _BV(CS12));
    OCR1A = 71;
    // TIMSK1 = 0x02;
    TIMSK1 = _BV(OCIE1A);
}

// Delay for the specified time in ms without updating sensor values
void delayMs(uint32_t time_ms) {
    delayTimerRunning = 1;
    delayTimerCount = time_ms;
    while(delayTimerRunning) ;
}

void delayMsFunc(uint32_t time_ms, void (*func)(void), uint16_t period_ms,
        uint16_t cutoff_ms) {
    // Initialize the conditions for the delay loop
```

```
55      uint32_t lastExec = time_ms;
56      uint32_t nextExec = lastExec - period_ms;
57      // Start the timer
58      delayTimerRunning = 1;
59      delayTimerCount = time_ms;
60      // Wait until the timer runs out (delayTimerCount decrements every ms)
61      while(delayTimerRunning) {
62          // If it's before the cutoff and time for the next execution
63          if (delayTimerCount > cutoff_ms && delayTimerCount <= nextExec) {
64              // Execute the function
65              lastExec = delayTimerCount;
66              nextExec = lastExec - period_ms;
67              func();
68          }
69      }
70  }
```

utils/timer.h

```
1  #ifndef INCLUDE_TIMER_H
2  #define INCLUDE_TIMER_H
3
4  #include <avr/io.h>
5  #include <avr/interrupt.h>
6
7  // Interrupts.
8  ISR(TIMER1_COMPA_vect);
9
10 // Timer functions
11 void setupTimer(void);
12 void delayMs(uint32_t time_ms);
13
14 // Declaration of timer variables
15 extern volatile uint32_t delayTimerCount;
16 extern volatile uint8_t  delayTimerRunning;
17
18 //! Wait milliseconds, execute a function periodically.
19 /*!
20  *  Executes a function at an interval until a cutoff has passed, returning
21  *  after a total number of milliseconds have passed.
22  *
23  *  \param time_ms      The total number of seconds to wait.
24  *  \param func         The function to execute periodically.
25  *  \param period_ms    The interval to execute the function.
26  *  \param cutoff_ms    The number of milliseconds before the end to stop
27  *                      attempting to start the function.
28  */
29 void delayMsFunc(uint32_t time_ms, void (*func)(void), uint16_t period_ms,
30         uint16_t cutoff_ms);
31
32 #endif
```