

Contents

lib2a.c	2
lib2a.h	3
lib2b.c	4
lib2b.h	6
proj2a.c	7
proj2b.c	8
utils/cmod.c	9
utils/cmod.h	11
utils/driving.c	12
utils/driving.h	14
utils/irobled.c	16
utils/irobled.h	18
utils/iroblib.c	19
utils/iroblib.h	21
utils/iroblife.c	22
utils/iroblife.h	24
utils/irobserial.c	25
utils/irobserial.h	27
utils/oi.h	28
utils/sensing.c	32
utils/sensing.h	34
utils/timer.c	36
utils/timer.h	38

lib2a.c

```
1  #include "lib2a.h"
2  #include "irobserial.h"
3  #include "sensing.h"
4  #include "oi.h"
5  #include "cmmod.h"
6
7  // Called by irobPeriodic
8  void irobIlifePeriodic(void) {
9      // Switch output to USB
10     setSerialDestination(SERIAL_USB);
11     // Output sensor values
12     irobprintf("Charging State: %d\n", getSensorUInt8(SenChargeState));
13     irobprintf("Voltage: %d\n", getSensorUInt16(SenVolt1));
14     irobprintf("Current: %d\n", getSensorInt16(SenCurr1));
15     irobprintf("Battery Temperature: %d\n", getSensorInt8(SenTemp));
16     irobprintf("Battery Charge: %d\n", getSensorUInt16(SenCharge1));
17     irobprintf("Battery Capacity: %d\n", getSensorUInt16(SenCap1));
18     irobprintf("Wall Signal: %d\n", getSensorUInt16(SenWallSig1));
19     irobprintf("Cliff Left Signal: %d\n", getSensorUInt16(SenCliffLSig1));
20     irobprintf("Cliff Front Left Signal: %d\n", getSensorUInt16(SenCliffFLSig1));
21     irobprintf("Cliff Front Right Signal: %d\n", getSensorUInt16(SenCliffFRSig1));
22     irobprintf("Cliff Right Signal: %d\n", getSensorUInt16(SenCliffRSig1));
23     // Spacing
24     byteTx('\n');
25     // Switch output back to Create for updating sensor values
26     setSerialDestination(SERIAL_CREATE);
27 }
```

lib2a.h

```
1  #ifndef LIB2A_H
2  #define LIB2A_H
3
4  //! Called by irobPeriodic
5  void iroblifePeriodic(void);
6
7  #endif
```

lib2b.c

```
1  #include "lib2b.h"
2  #include "sensing.h"
3  #include "oi.h"
4  #include "driving.h"
5
6  ///! Previous IR sensor value
7  uint8_t irPrevious = 0;
8
9  ///! True iff the robot should not move at all
10 uint8_t cannotRotateOrAdvance(void) {
11     return getSensorUint8(SenBumpDrop) & MASK_WHEEL_DROP;
12 }
13
14 ///! True iff the robot should not drive forward
15 uint8_t cannotAdvance(void) {
16     return getSensorUint8(SenBumpDrop) || !(getSensorUint16(SenCliffLSig1) &&
17         getSensorUint16(SenCliffFLSig1) && getSensorUint16(SenCliffFRSig1)
18         && getSensorUint16(SenCliffRSig1));
19 }
20
21 ///! Begin or continue driving forward if it is allowed, otherwise stop
22 void driveForwardIfAllowable(void) {
23     if (cannotAdvance()) {
24         // Shouldn't be moving forward: just stop
25         driveStop();
26     } else {
27         // Forward being pressed and able to move forward: drive!
28         drive(SPEED, RadStraight);
29     }
30 }
31
32 ///! Begin or continue turning
33 void turnContinuous(int16_t radius) {
34     drive(SPEED, radius);
35 }
36
37 ///! Called by overTurn periodically while turning.
38 ///! Updates sensors and stops if unsafe to continue.
39 void doWhileTurning(void) {
40     // Get most recent sensor values
41     updateSensors();
42     if (cannotRotateOrAdvance()) {
43         // Shouldn't be moving: just stop
44         driveStop();
45     }
46     // Keep going
47 }
48
49 ///! Turn an extra 30 degrees. Does not return until fully turned.
50 ///! Calls doWhileTurning periodically for sensor updating and safety.
51 void overTurn(int16_t radius) {
52     driveAngleTFunc(SPEED, radius, OVER_TURN_ANGLE,
53         &doWhileTurning, UPDATE_SENSOR_DELAY_PERIOD,
54         UPDATE_SENSOR_DELAY_CUTOFF);
```

```
55 }
56
57 /// Called by irobPeriodic
58 void iroblifePeriodic(void) {
59     // Get most recent sensor values
60     updateSensors();
61     // Get IR sensor value
62     uint8_t ir = getSensorUint8(SenIRChar);
63     if (cannotRotateOrAdvance()) {
64         // Shouldn't be moving: just stop
65         driveStop();
66     } else {
67         switch (ir) {
68             case IR_FORWARD:
69                 // Drive forward if allowable
70                 driveForwardIfAllowable();
71                 break;
72             case IR_LEFT:
73                 // Turn left
74                 turnContinuous(RadCCW);
75                 break;
76             case IR_RIGHT:
77                 // Turn right
78                 turnContinuous(RadCW);
79                 break;
80             default:
81                 // Movement button isn't being pressed
82                 switch (irPrevious) {
83                     case IR_LEFT:
84                         // Turn an extra 30 degrees
85                         overTurn(RadCCW);
86                         break;
87                     case IR_RIGHT:
88                         // Turn an extra 30 degrees
89                         overTurn(RadCW);
90                         break;
91                     default:
92                         // No buttons pressed and no over-turning left: stop
93                         driveStop();
94                 }
95                 break;
96         }
97     }
98     // Bookkeeping
99     irPrevious = ir;
100 }
```

lib2b.h

```
1  #ifndef LIB2B_H
2  #define LIB2B_H
3
4  #include <stdint.h>
5
6  // Driving constants
7  #define SPEED          (500)
8  #define OVER_TURN_ANGLE (30)
9
10 //! True iff the robot should not move at all
11 uint8_t cannotRotateOrAdvance(void);
12
13 //! True iff the robot should not drive forward
14 uint8_t cannotAdvance(void);
15
16 //! Begin or continue driving forward if it is allowed, otherwise stop
17 void driveForwardIfAllowable(void);
18
19 //! Begin or continue turning
20 void turnContinuous(int16_t radius);
21
22 //! Called by overTurn periodically while turning.
23 //! Updates sensors and stops if unsafe to continue.
24 void doWhileTurning(void);
25
26 //! Turn an extra 30 degrees. Does not return until fully turned.
27 //! Calls doWhileTurning periodically for sensor updating and safety.
28 void overTurn(int16_t radius);
29
30 //! Called by irobPeriodic
31 void iroblifePeriodic(void);
32
33 #endif
```

proj2a.c

```
1  #include "iroblife.h"
2  #include "sensing.h"
3
4  #include "lib2a.h"
5
6  // Delay constant
7  #define IROB_PERIOD_MS  (1000)
8
9  int main(void) {
10     // Submit to irobPeriodic
11     setIrobPeriodicImpl(&iroblifePeriodic);
12
13     // Initialize the Create
14     irobInit();
15
16     // Infinite operation loop
17     for(;;) {
18         // Periodic execution
19         irobPeriodic();
20
21         // Delay for the loop; one second
22         delayAndUpdateSensors(IROB_PERIOD_MS);
23     }
24 }
```

proj2b.c

```
1  #include "timer.h"
2  #include "sensing.h"
3  #include "iroblife.h"
4
5  #include "lib2b.h"
6
7  int main(void) {
8      // Submit to irobPeriodic
9      setIrobPeriodicImpl(&iroblifePeriodic);
10
11     // Initialize the Create
12     irobInit();
13
14     // Infinite operation loop
15     for(;;) {
16         // Periodic execution
17         irobPeriodic();
18
19         // Delay to avoid overflows
20         delayMs(UPDATE_SENSOR_DELAY_PERIOD);
21         // Wait if sensors coming in so next loop has clean start
22         waitForSensors();
23     }
24 }
```


utils/cmod.c

```
1  #include "cmod.h"
2  #include "oi.h"
3  #include "timer.h"
4
5  void initializeCommandModule(void){
6      // Disable interrupts. ("Clear interrupt bit")
7      cli();
8
9      // One-time setup operations.
10     setupIOPins();
11     setupTimer();
12     setupSerialPort();
13
14     // Enable interrupts. ("Set interrupt bit")
15     sei();
16 }
17
18 void setupIOPins(void) {
19     // Set I/O pins
20     DDRB = 0x10;
21     PORTB = 0xCF;
22     DDRC = 0x00;
23     PORTC = 0xFF;
24     DDRD = 0xE6;
25     PORTD = 0x7D;
26 }
27
28 void setupSerialPort(void) {
29     // Set the transmission speed to 57600 baud, which is what the Create expects,
30     // unless we tell it otherwise.
31     UBRR0 = 19;
32
33     // Enable both transmit and receive.
34     UCSROB = (_BV(RXCIE0) | _BV(TXEN0) | _BV(RXEN0));
35     // UCSROB = 0x18;
36
37     // Set 8-bit data.
38     UCSROC = (_BV(UCSZ00) | _BV(UCSZ01));
39     // UCSROC = 0x06;
40 }
41
42 void byteTx(uint8_t value) {
43     // Transmit one byte to the robot.
44     // Wait for the buffer to be empty.
45     while(!(UCSROA & 0x20)) ;
46
47     // Send the byte.
48     UDR0 = value;
49 }
50
51 uint8_t byteRx(void) {
52     // Receive one byte from the robot.
53     // Call setupSerialPort() first.
54     // Wait for a byte to arrive in the receive buffer.
```

```
55  while(!(UCSROA & 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      UCSROA |= _BV(TXC0);
67      byteTx(baud_code);
68      // Wait until transmit is complete
69      while(!(UCSROA & _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

```
1  #include <stdint.h>
2  #include "driving.h"
3  #include "oi.h"
4  #include "cmmod.h"
5  #include "timer.h"
6
7  // Weird constants because squeezing out precision
8  #define PIe5          314159
9  #define TENTH_RADIUS  13
10
11 // # BASIC COMMANDS #
12
13 void drive(int16_t velocity, int16_t radius) {
14     // Send the start driving command to the Create
15     byteTx(CmdDrive);
16     byteTx((uint8_t)((velocity >> 8) & 0x00FF));
17     byteTx((uint8_t)(velocity & 0x00FF));
18     byteTx((uint8_t)((radius >> 8) & 0x00FF));
19     byteTx((uint8_t)(radius & 0x00FF));
20 }
21
22 void driveStop(void) {
23     drive(0, RadStraight);
24 }
25
26
27 // # OPCODE-BASED COMMANDS #
28
29 void driveDistanceOp(int16_t velocity, int16_t distance) {
30     // Start driving
31     drive(velocity, RadStraight);
32     // Halt execution of new commands on the Create until reached distance
33     byteTx(WaitForDistance);
34     byteTx((uint8_t)((distance >> 8) & 0x00FF));
35     byteTx((uint8_t)(distance & 0x00FF));
36     // Stop the Create
37     driveStop();
38 }
39
40 void driveAngleOp(int16_t velocity, int16_t radius, int16_t angle) {
41     // Wait for angle opcode compatibility
42     if (radius == RadCW) {
43         angle = -angle;
44     }
45     // Start driving
46     drive(velocity, radius);
47     // Halt execution of new commands on the Create until reached angle
48     byteTx(WaitForAngle);
49     byteTx((uint8_t)((angle >> 8) & 0x00FF));
50     byteTx((uint8_t)(angle & 0x00FF));
51     // Stop the Create
52     driveStop();
53 }
54
```

```
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

```
1  #include <stdint.h>
2  #include "irobled.h"
3  #include "cmod.h"
4  #include "oi.h"
5
6  // The current state of the leds.
7  struct {
8      uint8_t bits;
9      uint8_t color;
10     uint8_t intensity;
11 } iroblibState;
12
13 void irobledCmd(uint8_t bits, uint8_t color, uint8_t intensity) {
14     // Modify the state
15     iroblibState.bits = bits;
16     iroblibState.color = color;
17     iroblibState.intensity = intensity;
18     // Update
19     irobledUpdate();
20 }
21
22 void irobledUpdate(void) {
23     // Send the led command using the current state
24     byteTx(CmdLeds);
25     byteTx(iroblibState.bits);
26     byteTx(iroblibState.color);
27     byteTx(iroblibState.intensity);
28 }
29
30 void irobledInit(void) {
31     irobledCmd(NEITHER_ROBOT_LED, POWER_LED_ORANGE, 0xFF);
32 }
33
34 void powerLedSet(uint8_t color, uint8_t intensity) {
35     irobledCmd(iroblibState.bits, color, intensity);
36 }
37
38 void robotLedSetBits(uint8_t bits) {
39     iroblibState.bits = bits;
40     irobledUpdate();
41 }
42
43 void robotLedOn(uint8_t led) {
44     iroblibState.bits |= led;
45     irobledUpdate();
46 }
47
48 void robotLedOff(uint8_t led) {
49     iroblibState.bits &= ~led;
50     irobledUpdate();
51 }
52
53 void robotLedToggle(uint8_t led) {
54     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_ORANGE  (0x40)
9  #define POWER_LED_RED     (0xFF)
10
11 // Bits for the other leds.
12 #define NEITHER_ROBOT_LED (0x00)
13 #define PLAY_ROBOT_LED    (0x02)
14 #define ADVANCE_ROBOT_LED (0x08)
15 #define BOTH_ROBOT_LED    (0x0A)
16
17 //! Send an led command to the Create.
18 void irobledCmd(uint8_t bits, uint8_t color, uint8_t intensity);
19 //! Update the leds. Probably won't have to use.
20 void irobledUpdate(void);
21 //! Initialize the leds to red for power and off for the others.
22 void irobledInit(void);
23
24 //! Set the color and intensity of the power led.
25 void powerLedSet(uint8_t color, uint8_t intensity);
26
27 // Functions for modifying one or both of the other leds.
28 void robotLedSetBits(uint8_t bits);
29 void robotLedOn(uint8_t led);
30 void robotLedOff(uint8_t led);
31 void robotLedToggle(uint8_t led);
32
33 #endif
```

utils/iroblib.c

```
1  #include "iroblib.h"
2  #include "oi.h"
3  #include "cmod.h"
4  #include "timer.h"
5
6  // Define songs to be played later
7  void defineSongs(void) {
8      // Reset song
9      byteTx(CmdSong);
10     byteTx(RESET_SONG);
11     byteTx(4);
12     byteTx(60);
13     byteTx(6);
14     byteTx(72);
15     byteTx(6);
16     byteTx(84);
17     byteTx(6);
18     byteTx(96);
19     byteTx(6);
20
21     // Start song
22     byteTx(CmdSong);
23     byteTx(START_SONG);
24     byteTx(6);
25     byteTx(69);
26     byteTx(18);
27     byteTx(72);
28     byteTx(12);
29     byteTx(74);
30     byteTx(12);
31     byteTx(72);
32     byteTx(12);
33     byteTx(69);
34     byteTx(12);
35     byteTx(77);
36     byteTx(24);
37 }
38
39 // Ensure that the robot is On.
40 void powerOnRobot(void) {
41     // If Create's power is off, turn it on
42     if(!RobotIsOn) {
43         while(!RobotIsOn) {
44             RobotPwrToggleLow;
45             delayMs(500); // Delay in this state
46             RobotPwrToggleHigh; // Low to high transition to toggle power
47             delayMs(100); // Delay in this state
48             RobotPwrToggleLow;
49         }
50         delayMs(3500); // Delay for startup
51     }
52
53     // Flush the buffer
54     while( (UCSROA & 0x80) && UDRO);
```

```
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

```
1  #include <stdlib.h>
2  #include <stdint.h>
3  #include "iroblife.h"
4
5  #include "timer.h"
6  #include "cmod.h"
7  #include "iroblib.h"
8  #include "oi.h"
9
10 #include "sensing.h"
11 #include "irobled.h"
12 #include "driving.h"
13 #include "irobserial.h"
14
15 void irobPeriodicImplNull(void) {
16 }
17
18 void (*irobPeriodicImpl)(void) = &irobPeriodicImplNull;
19
20 void setIrobPeriodicImpl(void (*func)(void)) {
21     irobPeriodicImpl = func;
22 }
23
24 void irobInit(void) {
25     // Set up Create and module
26     initializeCommandModule();
27     // Set Create as default serial destination
28     setSerialDestination(SERIAL_CREATE);
29
30     // Is the Robot on
31     powerOnRobot();
32     // Start the create
33     byteTx(CmdStart);
34     // Set the baud rate for the Create and Command Module
35     baud(Baud57600);
36     // Define some songs so that we know the robot is on.
37     defineSongs();
38     // Deprecated form of safe mode. I use it because it will
39     // turn off all LEDs, so it's essentially a reset.
40     byteTx(CmdControl);
41     // We are operating in FULL mode.
42     byteTx(CmdFull);
43
44     // Make sure the robot stops.
45     // As a precaution for the robot and your grade.
46     driveStop();
47
48     // Play the reset song and wait while it plays.
49     byteTx(CmdPlay);
50     byteTx(RESET_SONG);
51     delayMs(750);
52
53     // Turn the power button on to red.
54     irobledInit();
```

```
55 }
56
57 void irobPeriodic(void) {
58     // Call the user's periodic function
59     irobPeriodicImpl();
60     // Exit if the black button on the command module is pressed.
61     if(UserButtonPressed) {
62         irobEnd();
63     }
64 }
65
66 void irobEnd(void) {
67     // Stop the Create
68     driveStop();
69     // Power off the Create
70     powerOffRobot();
71     // Exit the program
72     exit(1);
73 }
```

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/irobserial.c

```
1  #include <stdint.h>
2  #include <stdarg.h>
3  #include <stdio.h>
4  #include "irobserial.h"
5  #include "cmod.h"
6  #include "oi.h"
7  #include "timer.h"
8
9  uint8_t serialDestination = SERIAL_SWITCHING;
10
11 void setSerialDestination(uint8_t dest) {
12     serialDestination = SERIAL_SWITCHING;
13     // Which serial port should byteTx and byteRx talk to?
14     // Ensure any pending bytes have been sent. Without this, the last byte
15     // sent before calling this might seem to disappear.
16     delayMs(10);
17     // Configure the port.
18     if (dest == SERIAL_CREATE) {
19         PORTB &= ~0x10 ;
20     } else {
21         PORTB |= 0x10 ;
22     }
23     // Wait a bit to let things get back to normal. According to the docs, this
24     // should be at least 10 times the amount of time needed to send one byte.
25     // This is less than 1 millisecond. We are using a much longer delay to be
26     // super extra sure.
27     delayMs(10);
28     serialDestination = dest;
29 }
30
31 uint8_t getSerialDestination(void) {
32     return serialDestination;
33 }
34
35 void irobprint(char* str) {
36     char c;
37     // Null-terminated string
38     while ((c = *(str++)) != '\0') {
39         // Print each byte
40         byteTx(c);
41     }
42 }
43
44 char printfBuffer[PRINTF_BUFFER_SIZE];
45
46 void irobprintf(const char* format, ...) {
47     char* fp = &printfBuffer[0];
48     va_list ap;
49     va_start(ap, format);
50     // Format the string
51     vsnprintf(fp, PRINTF_BUFFER_SIZE, format, ap);
52     va_end(ap);
53     // Print the string
54     irobprint(fp);
```

```
55 }
56
57 void irobnprintf(uint16_t size, const char* format, ...) {
58     // Create a buffer
59     char formatted[size];
60     char* fp = &formatted[0];
61     va_list ap;
62     va_start(ap, format);
63     // Format the string
64     vsnprintf(fp, size, format, ap);
65     va_end(ap);
66     // Print the string
67     irobprint(fp);
68 }
```

utils/irobserial.h

```
1  #ifndef IROBSERIAL_H
2  #define IROBSERIAL_H
3
4  #include <stdint.h>
5  #include <stdarg.h>
6
7  #define SERIAL_CREATE      (1)
8  #define SERIAL_USB        (2)
9  #define SERIAL_SWITCHING   (0xFF)
10
11 #define PRINTF_BUFFER_SIZE  (0xFF)
12
13 ///! Set the serial output (CREATE or USB)
14 ///! Takes some time.
15 void setSerialDestination(uint8_t dest);
16
17 ///! Get the serial output (CREATE or USB)
18 uint8_t getSerialDestination(void);
19
20 ///! Print a string
21 void irobprint(char* str);
22
23 ///! Print a formatted string (Max length: 255 bytes)
24 void irobprintf(const char* format, ...);
25
26 ///! Print a formatted string (for strings longer than 255 bytes)
27 void irobnprintf(uint16_t size, const char* format, ...);
28
29 #endif
```

utils/ui.h

```
1  /* ui.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 SenStreamPkts    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 Ubr300          3839
144 #define Ubr600          1919
145 #define Ubr1200         959
146 #define Ubr2400         479
147 #define Ubr4800         239
148 #define Ubr9600         119
149 #define Ubr14400        79
150 #define Ubr19200        59
151 #define Ubr28800        39
152 #define Ubr38400        29
153 #define Ubr57600        19
154 #define Ubr115200       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

```
1  #include <stdint.h>
2  #include "sensing.h"
3  #include "cmod.h"
4  #include "timer.h"
5  #include "oi.h"
6  #include "irobserial.h"
7
8  volatile uint8_t usartActive = 0;
9  volatile uint8_t sensorIndex = 0;
10 volatile uint8_t sensorBuffer[Sen6Size];
11 volatile uint8_t sensors[Sen6Size];
12
13 void requestPacket(uint8_t packetId) {
14     byteTx(CmdSensors);
15     byteTx(packetId);
16 }
17
18 uint8_t read1ByteSensorPacket(uint8_t packetId) {
19     // Send the packet ID
20     requestPacket(packetId);
21     // Read the packet byte
22     return byteRx();
23 }
24
25 ISR(USART_RX_vect) {
26     // Cache the retrieved byte
27     uint8_t tmpUDRO;
28     tmpUDRO = UDR0;
29     // Don't do anything if we're not looking
30     if (usartActive) {
31         if (getSerialDestination() == SERIAL_CREATE) {
32             // New sensor data from the create
33             sensorBuffer[sensorIndex++] = tmpUDRO;
34         } else {
35             // Probably input from the computer, loop old values around
36             sensorBuffer[sensorIndex] = sensors[sensorIndex];
37             sensorIndex++;
38         }
39         if (sensorIndex >= Sen6Size) {
40             // Reached end of sensor packet
41             usartActive = 0;
42         }
43     }
44 }
45
46 void updateSensors(void) {
47     // Don't do anything if sensors are still coming in
48     if (!usartActive) {
49         uint8_t i;
50         for (i = 0; i < Sen6Size; i++) {
51             // Copy in the sensor buffer so the most recent data is available
52             sensors[i] = sensorBuffer[i];
53         }
54         // Bookkeeping
```



```
55     sensorIndex = 0;
56     usartActive = 1;
57     // Request all sensor data
58     requestPacket(PACKET_ALL);
59 }
60 }
61
62 void waitForSensors(void) {
63     // Sensors data are coming in if usartActive is true
64     while(usartActive);
65 }
66
67 void delayAndUpdateSensors(uint32_t time_ms) {
68     // Update sensors while waiting
69     delayMsFunc(time_ms, &updateSensors, 1, UPDATE_SENSOR_DELAY_CUTOFF);
70 }
71
72 uint8_t getSensorUInt8(uint8_t index) {
73     // Already in the right format
74     return sensors[index];
75 }
76
77 int8_t getSensorInt8(uint8_t index) {
78     uint8_t x = getSensorUInt8(index);
79     // Convert to signed; not implementation-dependent, and optimizes away
80     return x < (1 << 7) ? x : x - (1 << 8);
81 }
82
83 uint16_t getSensorUInt16(uint8_t index1) {
84     // Combine msB and lsB
85     return (sensors[index1] << 8) | sensors[index1 + 1];
86 }
87
88 int16_t getSensorInt16(uint8_t index1) {
89     uint16_t x = getSensorUInt16(index1);
90     // Convert to signed; more opaque hex values b/c avr complains for 1 << 16
91     return x < 0x8000 ? x : x - 0x10000;
92 }
```

utils/sensing.h

```

1  #ifndef SENSING_H
2  #define SENSING_H
3
4  #include <stdint.h>
5
6  #define UPDATE_SENSOR_DELAY_PERIOD      (1)
7  #define UPDATE_SENSOR_DELAY_CUTOFF      (10)
8
9
10 #define PACKET BUMPS_AND_WHEEL_DROPS    (7)
11 #define MASK_WHEEL_DROP_CASTER           (1 << 4)
12 #define MASK_WHEEL_DROP_LEFT            (1 << 3)
13 #define MASK_WHEEL_DROP_RIGHT           (1 << 2)
14 #define MASK_WHEEL_DROP                  (0x1C)
15 #define MASK_BUMP_LEFT                   (1 << 1)
16 #define MASK_BUMP_RIGHT                  (1 << 0)
17 #define MASK_BUMP                        (0x03)
18
19 #define PACKET_BUTTONS                    (18)
20 #define MASK_BTN_ADVANCE                  (1 << 2)
21 #define MASK_BTN_PLAY                     (1 << 0)
22
23 #define IR_LEFT                           (129)
24 #define IR_FORWARD                        (130)
25 #define IR_RIGHT                          (131)
26
27 #define PACKET_ALL                        (6)
28
29 ///! Request a sensor packet. \see read1ByteSensorPacket(uint8_t)
30 /*!
31  * \deprecated {
32  *     This uses the old, non-USART-based way of retrieving sensor data.
33  * }
34 */
35 void requestPacket(uint8_t packetId);
36
37 ///! Read in a 1-byte sensor packet.
38 /*!
39  * \deprecated {
40  *     This uses the old, non-USART-based way of retrieving sensor data.
41  * }
42  *
43  * What is a sensor packet? A byte (or bytes) containing data from a set of
44  * sensors, often shifted and ORed together. See the Create Open Interface
45  * documentation for more.
46  *
47  * Currently Available Sensor Packets (v = read1ByteSensorPacket(packetId)):
48  *     Bumps and Wheel Drops (packetId = PACKET BUMPS_AND_WHEEL_DROPS):
49  *         Caster Drop (v & MASK_WHEEL_DROP_CASTER)
50  *         Left Wheel Drop (v & MASK_WHEEL_DROP_LEFT)
51  *         Right Wheel Drop (v & MASK_WHEEL_DROP_RIGHT)
52  *         Any Wheel Drop (v & MASK_WHEEL_DROP)
53  *         Left Bumper (v & MASK_BUMP_LEFT)
54  *         Right Bumper (v & MASK_BUMP_RIGHT)

```

```
55 *          Either Bumper          (v & MASK BUMPER)
56 *          Create Buttons          (packetId = PACKET_BUTTONS):
57 *          Advance Button          (v & MASK_BTN_ADVANCE)
58 *          Play Button             (v & MASK_BTN_PLAY)
59 *
60 *  \param packetId    The ID of the packet to retrieve, as defined by the
61 *                    Create Open Interface.
62 */
63 uint8_t read1ByteSensorPacket(uint8_t packetId);
64
65 ///! Request all packets (will be retrieved by USART)
66 void updateSensors(void);
67
68 ///! Wait for all packets to be recieved by USART
69 void waitForSensors(void);
70
71 ///! delayMs that updates sensors
72 void delayAndUpdateSensors(uint32_t time_ms);
73
74 ///! Get an unsigned 1-byte sensor value
75 uint8_t getSensorUInt8(uint8_t index);
76
77 ///! Get a signed 1-byte sensor value
78 int8_t getSensorInt8(uint8_t index);
79
80 ///! Get an unsigned 2-byte sensor value, indexed by the more significant
81 ///! (lower index) byte
82 uint16_t getSensorUInt16(uint8_t index1);
83
84 ///! Get a signed 2-byte sensor value, indexed by the more significant
85 ///! (lower index) byte
86 int16_t getSensorInt16(uint8_t index1);
87
88 #endif
```

utils/timer.c

```
1  #include <stdint.h>
2  #include "timer.h"    // Declaration made available here
3
4
5  // Timer variables defined here
6  volatile uint32_t delayTimerCount = 0;    // Definition checked against declaration
7  volatile uint8_t  delayTimerRunning = 0;  // Definition checked against declaration
8
9
10 // Chris -- moved to sensing.c
11 /*ISR(USART_RX_vect) { //SIGNAL(SIG_USART_RECV)
12     // Serial receive interrupt to store sensor values
13
14     // CSCE 274 students, I have only ever used this method
15     // when retrieving/storing a large amount of sensor data.
16     // You DO NOT need it for this assignment. If i feel it
17     // becomes relevant, I will show you how/when to use it.
18 }*/
19
20 //SIGNAL(SIG_OUTPUT_COMPARE1A)
21 ISR(TIMER1_COMPA_vect) {
22     // Interrupt handler called every 1ms.
23     // Decrement the counter variable, to allow delayMs to keep time.
24     if(delayTimerCount != 0) {
25         delayTimerCount--;
26     } else {
27         delayTimerRunning = 0;
28     }
29 }
30
31 void setupTimer(void) {
32     // Set up the timer 1 interrupt to be called every 1ms.
33     // It's probably best to treat this as a black box.
34     // Basic idea: Except for the 71, these are special codes, for which details
35     // appear in the ATmega168 data sheet. The 71 is a computed value, based on
36     // the processor speed and the amount of "scaling" of the timer, that gives
37     // us the 1ms time interval.
38     TCCR1A = 0x00;
39     // TCCR1B = 0x0C;
40     TCCR1B = (_BV(WGM12) | _BV(CS12));
41     OCR1A = 71;
42     // TIMSK1 = 0x02;
43     TIMSK1 = _BV(OCIE1A);
44 }
45
46 // Delay for the specified time in ms without updating sensor values
47 void delayMs(uint32_t time_ms) {
48     delayTimerRunning = 1;
49     delayTimerCount = time_ms;
50     while(delayTimerRunning) ;
51 }
52
53 void delayMsFunc(uint32_t time_ms, void (*func)(void), uint16_t period_ms,
54                 uint16_t cutoff_ms) {
```

```
55     // Initialize the conditions for the delay loop
56     uint32_t lastExec = time_ms;
57     uint32_t nextExec = lastExec - period_ms;
58     // Start the timer
59     delayTimerRunning = 1;
60     delayTimerCount = time_ms;
61     // Wait until the timer runs out (delayTimerCount decrements every ms)
62     while(delayTimerRunning) {
63         // If it's before the cutoff and time for the next execution
64         if (delayTimerCount > cutoff_ms && delayTimerCount <= nextExec) {
65             // Execute the function
66             lastExec = delayTimerCount;
67             nextExec = lastExec - period_ms;
68             func();
69         }
70     }
71 }
```

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
```