EEL5666: Intelligent Machines Design Lab Report

"Joe On The Go"

David Dzenitis

Professors Antonio Arroyo and Eric Schwartz

Teaching Assistants Sean Frucht, Devin Hughes, Tim Martin, Ryan Stevens, and Josh Weaver

Table Of Contents

Abstract:

"Joe On The Go" is a full-service, intelligent alarm clock that awakens the user with an alarm and freshly brewed cup of coffee. The software system is essentially a state machine that performs random search, obstacle avoidance, beacon seeking, coffee brewing, and finally, alarm playback. The hardware system consists of two motors, a stabilizing castor, three separate batteries, a primary microcontroller board, two daughter circuit boards, a motor driver board, a power inverter, several sensors, and a piezzo buzzer for alarm playback, and of course a coffee maker.

Executive Summary:

"Joe On The Go" seeks to be the premier wake-up alarm system. Through the use of several sensors, actuators, motors, and sophisticated software, Joe will wake you up with a freshly brewed cup of coffee in the morning. The robots operation begins with random search and obstacle avoidance, using two IR range finders to detect and avoid objects in the robot's path. The robot continues to randomly travel its environment until coming into view of an IR home beacon, which is placed at the user's bedside. Using the robot's own IR beacon, the robot is then able to advance to the user's bedside. Once the range finders again detect something in the robot's path, it is assumed that the robot has reached the user.

Following discovery and navigation to the user, the robot then initiates a brewing cycle that lasts 5 minutes. In the way, the user will be gently woken up with the fresh aroma of coffee prior to being blasted with the 90db buzzer. Once the brewing process is complete, the robot disables the brewing subsystem via a relay, and sounds the buzzer.

Robotic motion is achieved via two, fairly strong DC motors with a castor in the rear for stability. The motors are controlled via a BasicMicro RoboClaw motor driver, which has two channels, one for each motor. The motherboard communicates with the motor driver over a TTL, RS-232 link, which is set-up for one way communication. The speed and direction of each motor is set with a 1-byte command number.

Obstacle avoidance is accomplished using two Sharp GP2D12 IR range sensors connected to the motherboard's analog-to-digital converter. The higher the voltage output of the sensor, the closer an obstacle is to the robot. For purposes of the obstacle avoidance software, any value from 3000 to 4092 (the maximum) is assumed to indicate the presence of a physical obstacle. Once the presence of the obstacle has been established, the robot decides which way to turn based on which sensor has the higher value. If the left sensor value is higher, the robot turns right, navigating the robot further from the obstacle. If the right sensor's value is higher, the robot turns left.

Finding the user is accomplished using a pair of Pololu IR beacons, with one mounted on the robot and one mounted on a small piece of wood placed at the user's bedside. The beacon

mounted on the robot will communicate which direction the home beacon is using a 4-bit bus connected to the motherboard. Each bit corresponds to an cardinal direction: North, South, East, or West. Because the beacon is constantly sensing, the output can be somewhat erratic and must be smoothed for graceful navigation. Each time the software system reads the beacon's output, the result is placed in a ten element array, and the direction most prevalent in that array is taken to be the direction of the home beacon. The robot will then move or turn accordingly to bring it closer to the bedside.

Coffee brewing is accomplished using an 800-watt power inverter from Cobra and a 300-watt coffee maker from Toastess. The brewing subsystem is controlled with a 40-Amp relay connected to one of the custom circuit daughter boards. The brewing process takes approximately 5 minutes, with the status of the brewing cycle reported on the robot's LCD display. Once complete, the alarm sound is achieved using a 90-dB piezzo buzzer from RadioShack.

Introduction:

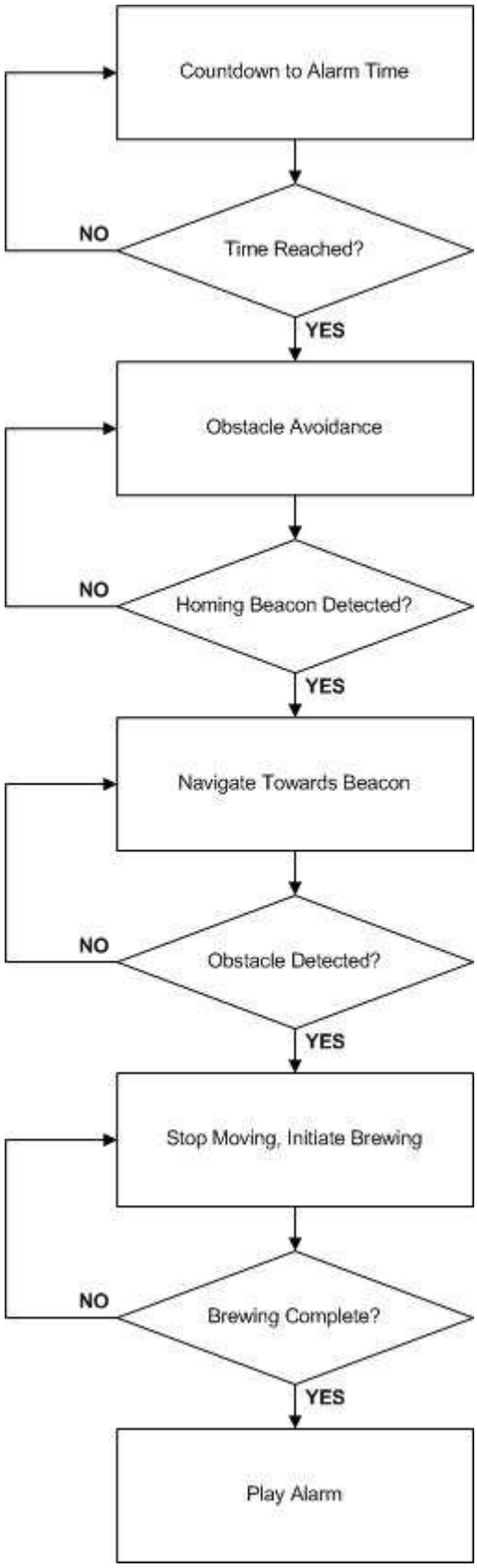
Waking up on time is a task that I can find very challenging on occasion. Further, waking up to a freshly brewed cup of coffee brought to my bedside is something I can only dream of. If past girlfriends are any indication, it is likely that the only way such a thing could occur is if I built a robot to bring the coffee to me. This is where the idea of Coffee-Bot was born.

Joe-on-the-go seeks to accomplish its various tasks through the use of a mobile platform, coupled with a time-telling alarm subsystem, target location subsystem, and finally, a coffee brewing subsystem. The mobile platform consists of two motorized wheels and a castor, mounted under a 12" square piece of wood. The time-telling alarm system is maintained in both hardware and software using a real-time clock chip, along with the cpu's onboard timers, with an alarm time configurable via an LED screen and keypad. The target location system is built using two IR beacons, one mounted to the robot and one stationary at the user's bedside. The coffee brewing system is responsible for actually producing the coffee.

Integrated System:

The system as a whole is built upon the alarm, brewing, and navigation subsystems working in concert. The system's flow of operations is as follows:

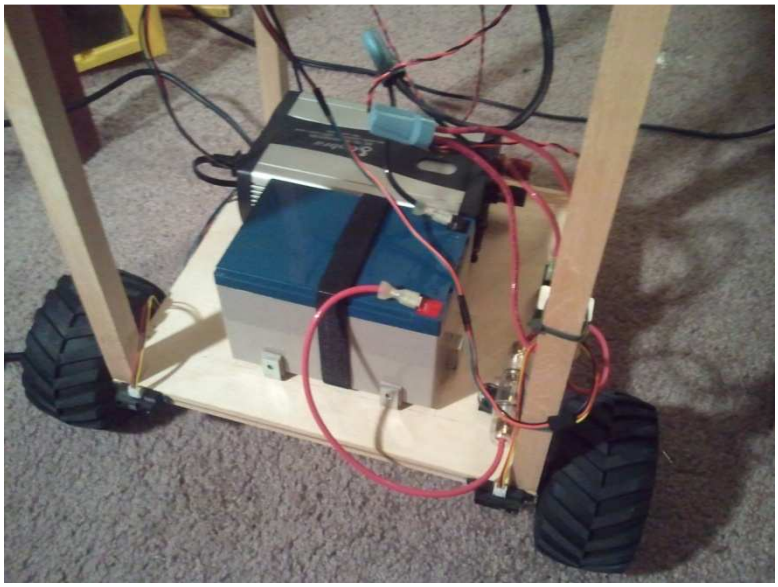
1. Wait for wake-up time, which is currently performed using a countdown system. A real-time clock system is currently in development.
2. Perform obstacle avoidance while attempting to locate target beacon.
3. Once home beacon has been detected, navigate towards it.
4. Once sufficiently close to target beacon, initiate brewing cycle
5. Once brewing cycle has completed, disable brewing subsystem.
6. Initiate alarm playback.


Countdown to Alarm Time
Time Reached?
NO
YES
Obstacle Avoidance
Homing Beacon Detected?
NO
YES
Navigate Towards Beacon
Obstacle Detected?
NO
YES
Stop Moving, Initiate Brewing
Brewing Complete?
NO
YES
Play Alarm

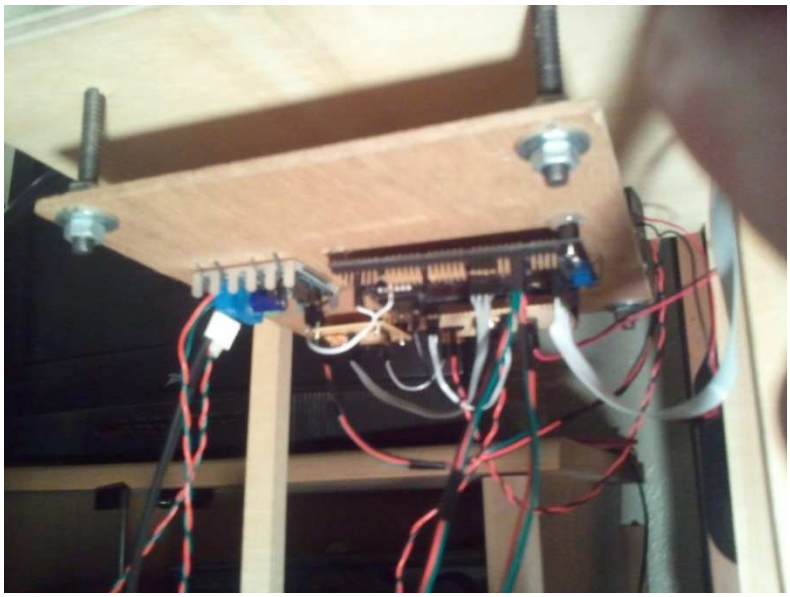
Mobile Platform:

The mobile platform is to consist of two 12-inch squares, each separated by 4 18-inch square dowels, which are 3/4" wide. The two motors and stabilizing castor are mounted below the base square. On top of the base square are the two IR range finders, the large, 12-volt battery, the power inverter, 40-amp relay, and 30-amp fuse. The four different boards used to control the robot are mounted on a single piece of wood that hangs beneath the upper platform via four hang bolts (headless). On top of the upper platform are the IR beacon and coffee maker.

Top View of Robot



Bottom View of Robot



View of Removable Electronics Board

Actuation:

There are two primary types of actuation used in this system. The first is the navigation system, which will drive the wheel motors and turn the robot using its castor as necessary to avoid obstacles. The second actuation system is responsible for the brewing process and playing of the alarm.

The motor system's purpose is to provide mobility to the robot and allow it to move around. The motors used are two, 12-volt, 152rpm gear head motors from Lynxmotion. These motors were chosen for their relatively high stall torque of 231.5 oz.-in. Each weighs 7.28 oz. and has an outside diameter of 37mm.

Image of Motor Used

The two motors are controlled using BasicMicro's RoboClaw motor driver. The motor driver has two channels allowing it to control both motors almost simultaneously. While the RoboClaw has several operation modes, the one employed on this robot is the simple RS-232 system, which is

one-way. 1 byte with an additional stop bit are sent to the motor driver with to indicate what action the driver should perform. A number between 1 and 128 are used to control the first motor, while a number between 129 and 255 are used to control the second motor. Sending zero will shutdown both motors at once, though each motor could be shut down individually if desired.

BasicMicro's RoboClaw Motor Controller

In order to not have to worry about keeping track of which motor is operating in forward vs. which motor needs to be reversed, I connected one motor in reverse so that the value sent to each motor would be similar to achieve forward motion. Below is a table of the values used in my software for controlling the motors:

Operation:	Channel 1:	Channel 2:
Stop (Individual Motor)	0x40	0xC0
Full Speed Reverse	0x01	0x80
Normal Reverse	0x21	0xA0
Slow Reverse	0x31	0xB0
Full Speed Forward	0x7F	0xFF
Fast Forward	0x60	0xE0
Normal Forward	0x50	0xD1
Slow Forward	0x4C	0xCC

Table of Opcodes for Various Motor Speeds

Turning the robot was accomplished by sending one slow forward and reverse opcode for each motor. The robot turned when either an obstacle was detected or when the IR beacon was sensed. For obstacle avoidance, the approach was simply to turn the robot roughly 90 degrees away from the side of the robot that is closer to the obstacle. Thus, if the left IR range finder had the higher number, the robot would turn right.

For navigating towards the IR beacon, a slightly more complex algorithm was used. Because the IR beacon mounted on the robot is constantly sending data, the robot's behavior can be somewhat erratic if no smoothing is applied to the data. Thus, every time the beacon's data was

read, the value was placed in a circular buffer of ten elements. Afterwards, the buffer was scanned and the direction reading showing up the most times was taken as the direction the robot needed to travel. This provided for very graceful movement as the robot was driving up to the home beacon.

The brewing system was the second method of actuation used on the robot. In order to control when the inverter and coffee maker would operate, their power was either supplied or cut-off by a 40-amp automotive relay. The microprocessor board would switch a transistor between cut-off and saturation modes which in turn controlled the relay.

Another transistor was also used to control when the buzzer would sound. Detailed schematics of this system are provided in Appexdix B.

Sensors:

There are several different types of sensors that must be used in this project:

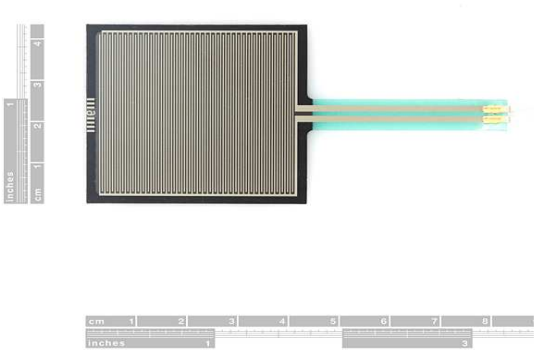
- IR Beacon Transceivers for target location detection. I am currently considering using the Pololu IR Beacon Transceiver Pair (http://www.pololu.com/catalog/product/702).
- IR-range finders mounted down low for obstacle avoidance.
- A Force-Sensitive Resistor to detect the presence or absence of the coffee cup.

The IR beacons were sourced from Pololu.com in Las Vegas, Nevada. They operate at 9-volts and provide logic output of 5-volts, which were reduced down to 3.3-volts using a voltage divider comprised of a 1kOhm and 2kOhm resistor in series. Initially, the beacon's output was read via a GPIO port, but after researching other people who have successfully integrated the beacons into their robots, I decided to read each of the directional signals via the ADC. This allowed me to determine which direction had the strongest signal of the directions read. It also important to note that the beacon uses inverted logic signals, with a "1" corresponding to 0-volts and a "0" corresponding to 5-volts.

The Pololu IR Beacon

Each time the beacon was read, the direction with the strongest signal was placed into a 10 element circular buffer along with the directions from the previous 9 readings. Whichever direction is most prevalent in the buffer is taken as the direction of the home beacon.

The IR range finders are Sharp GP2D12 IR Sensors and were sourced from Lynxmotion in Illinois. They output a voltage corresponding to the proximity of an obstacle in front of the sensor, which is then read through the ADC. A value of 3000 or higher (4092 max.) is interpreted by the software to indicate the presence of an obstacle.



Sparkfun SEN-09376 Force Sensitive Resistor

The force sensitive resistor is used to detect the presence of the coffee cup on the robot. It is connected in series with a 2kOhm resistor to create a variable voltage divider, which is then read through the ADC. Unfortunately, due to time constraints, this sensor was not yet integrated into the robot for Demo Day.

Behaviors:

The robot exhibits three primary behaviors as illustrated in the flowchart earlier. The first behavior is obstacle avoidance while searching for a line of sight with the home IR beacon. So long as the home IR beacon is not in sight, the robot will continue to move around its environment, avoiding obstacles, until line-of-sight has been achieved. When an obstace is encountered, the robot will turn away from it. Thus, if the obstacle is closer to the left side of the robot, the robot will turn right until it's path is clear, and vice-versa.

Once line-of-sight with the IR beacon is achieved, the robot assumes a clear path to the beacon. It will first orientate itself to be facing home. It will then proceed towards the beacon. However, the robot's heading with respect to home is constantly monitored and the robot will re-orientate itself as necessary during the course of navigating towards the beacon. Once an obstacle is sensed, it is assumed that the obstacle is the user's bedside.

Finally, the robot initiates the brewing cycle, which is accomplished by first activating the relay to supply power to the inverter and coffee maker. After extensive timing experiments, it has been determined that it takes roughly five minutes for the coffee maker to brew a cup of coffee. Thus, once power is supplied to the brewing subsystem, the processor simply waits for five minutes, updating the lcd display with the time remaining each minute.

After the coffee has completed brewing, the relay is deactivated, and the buzzer is activated.

Experimental Layout and Results:

First and foremost, prior to any actual experimentation of the robot system as a whole, I tested each separate component to determine how it worked and set preliminary sensor thresholds, etc. This way when I was actually testing the robot I would at least be close to correct operation before perfecting the software further.

The experimental layout I used during development of the robot was the breezeway outside my apartment using a couple boxes to serve as the obstacles. The robot was developed in an iterative fashion, where each system was developed and tested before moving on to the next system. The first task that I developed and corrected was obstacle avoidance. Following that, I added the beacon sensing and seeking capabilities. Finally, I included the coffee brewing and alarm system in testing.

The test of each system separately proved to be very valuable in getting correct operation out of my robot. Through testing obstacle avoidance I was able to better calibrate what value from the IR range finder should be considered an obstacle. While I started with 3500, I eventually dropped it to 3000 to give the robot enough room to turn without hitting the obstacle in the process.

Also, while I started with the beacons initially running through an input port as GPIO, I eventually switched to using the ADC because it allowed me to better use the beacon's data and determine which direction to travel in. One pitfall that I encountered while developing the beacon sensing is that I needed to add delay between each time the robot determined what action to take. Prior to this, my robot moved only in a circle because it was constantly turning and never getting a chance to realize it was line up with "home" correctly. It was through this sort of experimentation that I was able to get the robot operating correctly.

Conclusion:

In conclusion, I'm fairly satisfied with the final result in that the robot performs it tasks well and does not appear fidgety. However, I would have liked a little more time to include the force resistor and include a real-time clock for actual time keeping. As far as technical caveats are concerned, I would say the design and implementation of my custom circuitry was by far the most difficult and I wish I had more knowledge about circuits to prevent an incident where my custom circuit caught fire! However, I learned a great deal through the process and in a way that I'm likely to never forget.

Looking to the future, I'm currently integrating the force resistor and real-time clock. I would like to eventually enhance the alarm to be some form of music as opposed to the simple piezzo buzzer. I'm also considering building a custom charger that could charge the three batteries simultaneously without my having to remove them from the robot.

Documentation:

Atmel:

Doc 8067: Atmel XMega 128 A1 Microprocessor Preliminary
Doc 8077: Atmel Xmega 128 A1 Manual
Doc 8049: Using The XMega USART
Doc 8308: Getting Started Writing C Code For XMega
Doc 8308: XMega Basics
Doc8050: Using the XMega I/O Pins and External Interrupts
Doc 8043: XMega Interrupts and the Programmable Multi-Level Interrupt Controller

Basic Micro:

B0099: Robo Claw 2 Channel 5A Motor Controller Data Sheet

Interlink Electronics:

94-0004: Interlink Electronics FSR™ Force Sensing Resistors™ Integration Guide
94-00009: FSR-406 Datasheet

Pololu IR Beacons:

Hazlett, Christopher. "Beacon Locating Robot - Powered by Arduino and IR Transceiver." http://www.robotishappy.com/2009/12/beacon-locating-robot-powered-by-arduino-and-ir-transceiver/. Retrieved 4/18/2010.

Powersonic:

PS-12120 Rechargeable Sealed Lead-Acid Battery Datasheet

Pridgen-Vermeer Robotics:

Pridgen, Mike. "Pridgen Vermeer Robotics XMega 128 Manual." Jan. 2010

Maxim:

19-5339: DS3234 Extremely Accurate SPI Bus with Integrated Crystal and SRAM Datasheet

Xiamen Amotec Display Co. Ltd:

ADM1602K-NSR-FBS: Specifications of LCD Module

Appendix A: Program Code

```
#define SENSOR_OBSTACLE_THRESHOLD 3000
#define BEACON_DETECT_THRESHOLD 500

#define NUM_BEACON_READINGS 10

#define NO_HEADING 0
#define NORTH_HEADING 1
#define EAST_HEADING 2
#define SOUTH_HEADING 3
#define WEST_HEADING 4

#define PERIPHERALS_OFF 0x00
#define RELAY_ON 0x01
#define BUZZER_ON 0x02

void DetectObstacles(void);

void DetectBeacon(void);

void TurnLeft(void);

void TurnRight(void);

void MoveForward(void);

void StopMoving(void);

void BrewCoffee(void);
```

```c
#include <avr/io.h>
#include <stdlib.h>
#include <stdbool.h>
#include "PVR.h"
#include "usart.h"
#include "main.h"

bool BeaconDetected;
bool ObstacleDetected;

int  RightObstacleSensor;
int  LeftObstacleSensor;
int  CurrentBeaconReading;

uint8_t CurrentBeaconHeading;

uint8_t  BeaconHeadings[NUM_BEACON_READINGS];


int main(void)
{
    xmegaInit();                            //setup XMega
    delayInit();                            //setup delay functions
    ADCAInit();                             //setup PORTA analong readings
    lcdInit();                              //setup LCD on PORTK
    PORTQ_DIR |= 0x01;                      //set Q0 (LED) as output
    PORTJ_DIR |= 0xFF;
    PORTJ_OUT = 0x00;
    PORTH_DIR &= 0xF0;                      //set lower nibble of port H to input



    RS232Init();
    RS232Send(DRIVE_FULL_STOP);

    lcdGoto(0,0);
    lcdString("Seeking base ");
    lcdGoto(1,0);
    lcdString("station...");

    BeaconDetected = false;
    ObstacleDetected = false;

    RightObstacleSensor = 0;
    LeftObstacleSensor = 0;
    CurrentBeaconHeading = NO_HEADING;

    CurrentBeaconReading = 0;
    for(int x = 0; x < NUM_BEACON_READINGS; x++)
```

```
    {
        BeaconHeadings[x] = NO_HEADING;
    }


    /*****************************************************
     *                                                   *
     * Phase 1:  Obstacle avoidance, seeking beacon      *
     *                                                   *
     *****************************************************/
    while(!BeaconDetected)
    {
        DetectBeacon();
        DetectObstacles();

        if(ObstacleDetected)
        {
            StopMoving();
            delay_ms(1000);

            if(RightObstacleSensor > LeftObstacleSensor)
            {
                TurnLeft();
                delay_ms(2000);
            }
            else
            {
                TurnRight();
                delay_ms(2000);
            }
        }
        else
        {
            MoveForward();
        }


    }

    /*****************************************************
     *                                                   *
     * PHASE 2:  Navigate to beacon                      *
     *                                                   *
     *****************************************************/
    ObstacleDetected = false;
    while(!ObstacleDetected)
    {
        DetectObstacles();
        DetectBeacon();
```

```
        switch(CurrentBeaconHeading)
        {
            case NORTH_HEADING:
                MoveForward();
                break;
            case EAST_HEADING:
                TurnRight();
                delay_ms(1500);
                StopMoving();
                DetectBeacon();
                delay_ms(500);
                break;
            case WEST_HEADING:
                TurnLeft();
                delay_ms(1500);
                StopMoving();
                DetectBeacon();
                delay_ms(500);
                break;
            case SOUTH_HEADING:
                TurnRight();
                delay_ms(3200); //roughly how long it takes to make a u-turn
                StopMoving();
                DetectBeacon();
                delay_ms(500);
                break;
            default:
                MoveForward();
                break;
        }

        DetectBeacon();
    }

    //Obstacle detected, we're at the bedside!! Start brewing...
    StopMoving();
    lcdGoto(0,0);
    lcdString("Brewing Coffee..");
    lcdGoto(1,0);
    lcdString("                ");

    BrewCoffee();

    PORTJ_OUT = BUZZER_ON;
    delay_ms(60000);

    return 0;
```

```
/*
    PORTJ_OUT = 0x00;

//  PORTJ_OUT = 0xFF;

//  delay_ms(10000);

//  PORTJ_OUT = 0x00;

    int rangeFinder = 0;
    char rangeString[5];
    unsigned char led_flag = 0;

    for(int x = 10; x >= 0; x--)
    {
        lcdGoto(1,0);
        itoa(x,rangeString, 10);
        lcdString(rangeString);
        delay_ms(1000);
    }

    //initial brewing:
    PORTJ_OUT = 0x04;
    //delay_ms(7 * 60 * 1000);
    delay_ms(5000);
    //PORTJ_OUT = 0x0000;
    //PORTJ_OUT = 0x0110;
    //delay_ms(10000);
*/
}

void DetectObstacles(void)
{
    int rightAvgAccum = 0;
    int leftAvgAccum = 0;

    //note: 8 iterations chosen to prevent overflow of 16-bit signed int
    for(int x = 0; x < 8; x++)
    {
        rightAvgAccum += ADCA0();
        leftAvgAccum += ADCA1();
    }

    RightObstacleSensor = rightAvgAccum / 8;
    LeftObstacleSensor  = leftAvgAccum / 8;

    if( (RightObstacleSensor > SENSOR_OBSTACLE_THRESHOLD) || (LeftObstacleSensor >
SENSOR_OBSTACLE_THRESHOLD) )
    {
```

```
        ObstacleDetected = true;
    }
    else
    {
        ObstacleDetected = false;
    }

    return;

}

void DetectBeacon(void)
{
    //read port H 10 times to find heading if any...
    int nVal = 0;
    int eVal = 0;
    int sVal = 0;
    int wVal = 0;

    //read ADC
    nVal = ADCA4();
    eVal = ADCA5();
    sVal = ADCA6();
    wVal = ADCA7();

    //determine minimum of values read:
    int minValue = BEACON_DETECT_THRESHOLD;
    uint8_t winningDirection = NO_HEADING;

    if(nVal < minValue)
    {
        minValue = nVal;
        winningDirection = NORTH_HEADING;
    }

    if(eVal < minValue)
    {
        minValue = eVal;
        winningDirection = EAST_HEADING;
    }

    if(sVal < minValue)
    {
        minValue = sVal;
        winningDirection = SOUTH_HEADING;
    }

    if(wVal < minValue)
    {
```

```
        minValue = wVal;
        winningDirection = WEST_HEADING;
    }


    //add winning direction to array:
    BeaconHeadings[CurrentBeaconReading] = winningDirection;
    CurrentBeaconReading++;
    if(CurrentBeaconReading >= NUM_BEACON_READINGS)
    {
        CurrentBeaconReading = 0;
    }

    //scan array and determine overall "official heading"
    short votes[5];
    for(int x = 0; x < 5; x++){
        votes[x] = 0;
    }

    for(int x = 0; x < NUM_BEACON_READINGS; x++){
        votes[BeaconHeadings[x]] += 1;
    }

    short mostVotes = 0;
    uint8_t currentWinner = NO_HEADING;

    if(votes[NO_HEADING] > mostVotes){
        mostVotes = votes[NO_HEADING];
        currentWinner = NO_HEADING;
    }

    if(votes[NORTH_HEADING] > mostVotes){
        mostVotes = votes[NORTH_HEADING];
        currentWinner = NORTH_HEADING;
    }

    if(votes[EAST_HEADING] > mostVotes){
        mostVotes = votes[EAST_HEADING];
        currentWinner = EAST_HEADING;
    }

    if(votes[SOUTH_HEADING] > mostVotes){
        mostVotes = votes[SOUTH_HEADING];
        currentWinner = SOUTH_HEADING;
    }

    if(votes[WEST_HEADING] > mostVotes){
        mostVotes = votes[WEST_HEADING];
        currentWinner = WEST_HEADING;
```

```
    }

    CurrentBeaconHeading = currentWinner;
    if(currentWinner != NO_HEADING)
    {
        BeaconDetected = true;
    }
    else
    {
        BeaconDetected = false;
    }

    lcdGoto(1,0);
    char tempStr[5];
    itoa(CurrentBeaconHeading, tempStr, 10);
    lcdString(tempStr);

    return;
}

//right motor = ch1, left motor = ch2
//right goes backward, left goes forward
void TurnRight(void)
{
    RS232Send(CH1_REVERSE_SLOW);
    RS232Send(CH2_FORWARD_SLOW);

    return;
}


void TurnLeft(void)
{
    RS232Send(CH2_REVERSE_SLOW);
    RS232Send(CH1_FORWARD_SLOW);

    return;
}

void MoveForward(void)
{
    RS232Send(CH1_FORWARD_NORMAL);
    RS232Send(CH2_FORWARD_NORMAL);

    return;
}

void StopMoving(void)
{
```

```
    RS232Send(DRIVE_FULL_STOP);
}

void BrewCoffee(void)
{
    //PORTJ_DIR |= 0xFF;
    PORTJ_OUT = RELAY_ON;
    lcdGoto(0,0);
    lcdString("Brewing coffee..");

    //loop that waits 5 minutes, updating display with status
    for(int x = 5; x > 0; x--)
    {
        lcdGoto(1,0);
        char minLeft[2];
        itoa(x, minLeft, 10);
        lcdString(minLeft);
        lcdGoto(1,1);
        lcdString(" min. remaining");
        delay_ms(60000);
    }

    PORTJ_OUT = PERIPHERALS_OFF;

    return;
}
```

```
#ifndef  __usart_h__
#define  __usart_h__

#include <avr/io.h>
#include <stdbool.h>
#include "usart.h"

#define BAUD 9200L
#define BSCALE_VALUE 0
#define BSEL_VALUE 207  //for baud rate of 9600
#define USART USARTF0
#define USART_PORT PORTF

#define DRIVE_FULL_STOP 0x00
#define CH1_STOP 0x40
#define CH2_STOP 0xC0

#define CH1_FULL_REVERSE 0x01
#define CH1_REVERSE_NORMAL 0x21
#define CH1_REVERSE_SLOW 0x31

#define CH1_FULL_FORWARD 0x7F
#define CH1_FORWARD_FAST 0x60
#define CH1_FORWARD_NORMAL 0x50
#define CH1_FORWARD_SLOW 0x4C

#define CH2_FULL_REVERSE 0x80
#define CH2_REVERSE_NORMAL 0xA0
#define CH2_REVERSE_SLOW 0xB0

#define CH2_FULL_FORWARD 0xFF
#define CH2_FORWARD_FAST 0xE0
#define CH2_FORWARD_NORMAL 0xD1
#define CH2_FORWARD_SLOW 0xCC

void RS232Init(void);

void RS232Send(char);

#endif
```

```c
#include <avr/io.h>
#include <stdbool.h>
#include "usart.h"

/****************
 * RS-232 USART *
 ****************/

void RS232Init(void)
{
    USART_PORT.DIRSET = PIN3_bm;    //sets pin 3 as output (bm = bit mask)
    USART_PORT.DIRCLR = PIN2_bm;    //sets pin 2 as input (even though not used)

    //USARTF0: 8 data bits, no parity, 1 stop bit
    USART.CTRLC = (uint8_t) USART_CHSIZE_8BIT_gc | USART_PMODE_DISABLED_gc | false;

    // set baud //(USARTC0.BAUDCTRLB & 0xF0) |
    #ifndef F_CPU
    #define F_CPU 32000000L
    #endif

    USART.BAUDCTRLB = (((F_CPU /BAUD)>>4)-1)>>8;
    USART.BAUDCTRLA = ((F_CPU /BAUD)>>4)-1;

    USART.CTRLB |= USART_RXEN_bm;
    USART.CTRLB |= USART_TXEN_bm;

    return;
}

void RS232Send(char data)
{
    //wait for any previous data to be sent:
    while( (USART.STATUS & USART_DREIF_bm) == 0 ) {}
    USART.DATA = data;

    return;
}
```

```
#ifndef  __PVR_h__
#define  __PVR_h__

#include <avr/io.h>
#include <avr/interrupt.h>
#include "PVR.h"

#define LCD         PORTK_OUT
#define LCDDDR      PORTK_DIR

volatile int delaycnt;

void xmegaInit(void);

void delayInit(void);

void delay_ms(int cnt);

void delay_us(int cnt);

void lcdDataWork(unsigned char c);

void lcdData(unsigned char c);

void lcdCharWork(unsigned char c);

void lcdChar(unsigned char c);

void lcdString(unsigned char ca[]);

void lcdInt(int value);

void lcdGoto(int row, int col);

void lcdInit(void);

void ServoCInit(void);

void ServoDInit(void);

void ServoC0(int value);

void ServoC1(int value);

void ServoC2(int value);

void ServoC3(int value);

void ServoC4(int value);
```

```
void ServoC5(int value);

void ServoD0(int value);

void ServoD1(int value);

void ServoD2(int value);

void ServoD3(int value);

void ServoD4(int value);

void ServoD5(int value);

void ADCAInit(void);

int ADCA0(void);

int ADCA1(void);

int ADCA2(void);

int ADCA3(void);

int ADCA4(void);

int ADCA5(void);

int ADCA6(void);

int ADCA7(void);


#endif
```

```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include "PVR.h"

/*********
 * Xmega *
 *********/

void xmegaInit(void)
{
    CCP = 0xD8;
    CLK_PSCTRL = 0x00;
    PORTQ_DIR = 0x01;
    //setup oscilllator
    OSC_CTRL = 0x02;                        //enable 32MHz internal clock
    while ((OSC_STATUS & 0x02) == 0);       //wait for oscillator to be ready
    CCP = 0xD8;                             //write signature to CCP
    CLK_CTRL = 0x01;                        //select internal 32MHz RC oscillator
}

/*********
 * Delay *
 *********/

void delayInit(void)
{
    TCF1_CTRLA = 0x01;                      //set clock/1
    TCF1_CTRLB = 0x31;                      //enable COMA and COMB, set to FRQ
    TCF1_INTCTRLB = 0x00;                   //turn off interrupts for COMA and COMB
    SREG |= CPU_I_bm;                       //enable all interrupts
    PMIC_CTRL |= 0x01;                      //enable all low priority interrupts
}

void delay_ms(int cnt)
{
    delaycnt = 0;                           //set count value
    TCF1_CCA = 32000;                       //set COMA to be 1ms delay
    TCF1_CNT = 0;                           //reset counter
    TCF1_INTCTRLB = 0x01;                   //enable low priority interrupt for delay
    while (cnt != delaycnt);                //delay
    TCF1_INTCTRLB = 0x00;                   //disable interrupts
}

void delay_us(int cnt)
{
    delaycnt = 0;                           //set counter
    TCF1_CCA = 32;                          //set COMA to be 1us delay
    TCF1_CNT = 0;                           //reset counter
    TCF1_INTCTRLB = 0x01;                   //enable low priority interrupt for delay
```

```
    while (cnt != delaycnt);                //delay
    TCF1_INTCTRLB = 0x00;                   //disable interrupts
}

SIGNAL(TCF1_CCB_vect)
{
    delaycnt++;
}

SIGNAL(TCF1_CCA_vect)
{
    delaycnt++;
}

/*******
 * LCD *
 *******/

#define LCD         PORTK_OUT
#define LCDDDR      PORTK_DIR

void lcdDataWork(unsigned char c)
{
    c &= 0xF0;                              //keep data bits, clear the rest
    c |= 0x08;                              //set E high
    LCD = c;                                //write to LCD
    delay_ms(2);                            //delay
    c ^= 0x08;                              //set E low
    LCD = c;                                //write to LCD
    delay_ms(2);                            //delay
    c |= 0x08;                              //set E high
    LCD = c;                                //write to LCD
    delay_ms(2);                            //delay
}

void lcdData(unsigned char c)
{
    unsigned char cHi = c & 0xF0;           //give cHi the high 4 bits of c
    unsigned char cLo = c & 0x0F;           //give cLo the low 4 bits of c
    cLo = cLo * 0x10;                       //shift cLo left 4 bits
    lcdDataWork(cHi);
    lcdDataWork(cLo);
}

void lcdCharWork(unsigned char c)
{
    c &= 0xF0;                              //keep data bits, clear the rest
    c |= 0x0A;                              //set E and RS high
    LCD = c;                                //write to LCD
```

```
    delay_ms(2);                        //delay
    c ^= 0x08;                          //set E low
    LCD = c;                            //write to LCD
    delay_ms(2);                        //delay
    c |= 0x08;                          //set E high
    LCD = c;                            //write to LCD
    delay_ms(2);                        //delay
}

void lcdChar(unsigned char c)
{
    unsigned char cHi = c & 0xF0;       //give cHi the high 4 bits of c
    unsigned char cLo = c & 0x0F;       //give cLo the low 4 bits of c
    cLo = cLo * 0x10;                   //shift cLo left 4 bits
    lcdCharWork(cHi);
    lcdCharWork(cLo);
}


void lcdString(unsigned char ca[])
{
    int i = 0;
    while (ca[i] != '\0')
    {
        lcdChar(ca[i++]);
    }
}

void lcdInt(int value)
{
    int temp_val;
    int x = 10000;
    int leftZeros=5;

    if (value<0)
    {
         lcdChar('-');
         value *= -1;
    }

    while (value / x == 0)
    {
        x/=10;
        leftZeros--;
    }


    while ((value > 0) || (leftZeros>0))
    {
```

```
        temp_val = value / x;
        value -= temp_val * x;
        lcdChar(temp_val+ 0x30);
        x /= 10;
        leftZeros--;
    }

    while (leftZeros>0)
    {
        lcdChar(0+ 0x30);
        leftZeros--;
    }

    return;
}

void lcdGoto(int row, int col)
{
    unsigned char pos;
    if ((col >= 0 && col <= 19) && (row >= 0 && row <= 3))
    {
        pos = col;
        if (row == 1)
            pos += 0x40;
        else if (row == 2)
            pos += 0x14;
        else if (row == 3)
            pos += 0x54;
        lcdData(0x80 + pos);
    }
}

void lcdInit(void)
{
    delayInit();                    //set up the delay functions
    LCDDDR = 0xFF;                  //set LCD port to outputs.
    delay_ms(20);                   //wait to ensure LCD powered up
    lcdDataWork(0x30);              //put in 4 bit mode, part 1
    delay_ms(10);                   //wait for lcd to finish
    lcdDataWork(0x30);              //put in 4 bit mode, part 2
    delay_ms(2);                    //wait for lcd to finish
    lcdData(0x32);                  //put in 4 bit mode, part 3
    lcdData(0x2C);                  //enable 2 line mode
    lcdData(0x0C);                  //turn everything on
    lcdData(0x01);                  //clear LCD
}

/*********
 * Servo *
```

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

void ServoCInit(void)
{
    TCC0_CTRLA = 0x05;              //set TCC0_CLK to CLK/64
    TCC0_CTRLB = 0xF3;              //Enable OC A, B, C, and D.  Set to Single Slope
PWM
                                    //OCnX = 1 from Bottom to CCx and 0 from CCx to Top
    TCC0_PER = 10000;               //20ms / (1/(32MHz/64)) = 10000.  PER = Top
    TCC1_CTRLA = 0x05;              //set TCC1_CLK to CLK/64
    TCC1_CTRLB = 0x33;              //Enable OC A and B.  Set to Single Slope PWM
                                    //OCnX = 1 from Bottom to CCx and 0 from CCx to Top
    TCC1_PER = 10000;               //20ms / (1/(32MHz/64)) = 10000.  PER = Top
    PORTC_DIR = 0x3F;               //set PORTC5:0 to output
    TCC0_CCA = 0;                   //PWMC0 off
    TCC0_CCB = 0;                   //PWMC1 off
    TCC0_CCC = 0;                   //PWMC2 off
    TCC0_CCD = 0;                   //PWMC3 off
    TCC1_CCA = 0;                   //PWMC4 off
    TCC1_CCB = 0;                   //PWMC5 off
}

void ServoDInit(void)
{
    TCD0_CTRLA = 0x05;              //set TCC0_CLK to CLK/64
    TCD0_CTRLB = 0xF3;              //Enable OC A, B, C, and D.  Set to Single Slope
PWM
                                    //OCnX = 1 from Bottom to CCx and 0 from CCx to Top
    TCD0_PER = 10000;               //20ms / (1/(32MHz/64)) = 10000.  PER = Top
    TCD1_CTRLA = 0x05;              //set TCC1_CLK to CLK/64
    TCD1_CTRLB = 0x33;              //Enable OC A and B.  Set to Single Slope PWM
                                    //OCnX = 1 from Bottom to CCx and 0 from CCx to Top
    TCD1_PER = 10000;               //20ms / (1/(32MHz/64)) = 10000.  PER = Top
    PORTD_DIR = 0x3F;               //set PORTC5:0 to output
    TCD0_CCA = 0;                   //PWMC0 off
    TCD0_CCB = 0;                   //PWMC1 off
    TCD0_CCC = 0;                   //PWMC2 off
    TCD0_CCD = 0;                   //PWMC3 off
    TCD1_CCA = 0;                   //PWMC4 off
    TCD1_CCB = 0;                   //PWMC5 off
}

void ServoC0(int value)
{
    if (value > 100)                //cap at +/- 100
        value = 100;                //  -100 => 1ms
    else if (value < -100)          //  0    => 1.5ms
        value = -100;               //  100  => 2ms
    value *= 5;                     //multiply value by 2.5
```

```
    value /= 2;                     //  new range +/- 250
    TCC0_CCA = (750 + value);       //Generate PWM.
}

void ServoC1(int value)
{
    if (value > 100)                //cap at +/- 100
        value = 100;                //  -100 => 1ms
    else if (value < -100)          //  0    => 1.5ms
        value = -100;               //  100  => 2ms
    value *= 5;                     //multiply value by 2.5
    value /= 2;                     //  new range +/- 250
    TCC0_CCB = (750 + value);       //Generate PWM.
}

void ServoC2(int value)
{
    if (value > 100)                //cap at +/- 100
        value = 100;                //  -100 => 1ms
    else if (value < -100)          //  0    => 1.5ms
        value = -100;               //  100  => 2ms
    value *= 5;                     //multiply value by 2.5
    value /= 2;                     //  new range +/- 250
    TCC0_CCC = (750 + value);       //Generate PWM.
}

void ServoC3(int value)
{
    if (value > 100)                //cap at +/- 100
        value = 100;                //  -100 => 1ms
    else if (value < -100)          //  0    => 1.5ms
        value = -100;               //  100  => 2ms
    value *= 5;                     //multiply value by 2.5
    value /= 2;                     //  new range +/- 250
    TCC0_CCD = (750 + value);       //Generate PWM.
}

void ServoC4(int value)
{
    if (value > 100)                //cap at +/- 100
        value = 100;                //  -100 => 1ms
    else if (value < -100)          //  0    => 1.5ms
        value = -100;               //  100  => 2ms
    value *= 5;                     //multiply value by 2.5
    value /= 2;                     //  new range +/- 250
    TCC1_CCA = (750 + value);       //Generate PWM.
}

void ServoC5(int value)
```

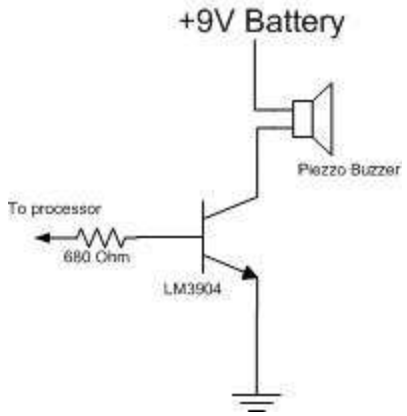
```
{
    if (value > 100)                //cap at +/- 100
        value = 100;                //  -100 => 1ms
    else if (value < -100)          //  0    => 1.5ms
        value = -100;               //  100  => 2ms
    value *= 5;                     //multiply value by 2.5
    value /= 2;                     //  new range +/- 250
    TCC1_CCB = (750 + value);       //Generate PWM.
}

void ServoD0(int value)
{
    if (value > 100)                //cap at +/- 100
        value = 100;                //  -100 => 1ms
    else if (value < -100)          //  0    => 1.5ms
        value = -100;               //  100  => 2ms
    value *= 5;                     //multiply value by 2.5
    value /= 2;                     //  new range +/- 250
    TCD0_CCA = (750 + value);       //Generate PWM.
}

void ServoD1(int value)
{
    if (value > 100)                //cap at +/- 100
        value = 100;                //  -100 => 1ms
    else if (value < -100)          //  0    => 1.5ms
        value = -100;               //  100  => 2ms
    value *= 5;                     //multiply value by 2.5
    value /= 2;                     //  new range +/- 250
    TCD0_CCB = (750 + value);       //Generate PWM.
}

void ServoD2(int value)
{
    if (value > 100)                //cap at +/- 100
        value = 100;                //  -100 => 1ms
    else if (value < -100)          //  0    => 1.5ms
        value = -100;               //  100  => 2ms
    value *= 5;                     //multiply value by 2.5
    value /= 2;                     //  new range +/- 250
    TCD0_CCC = (750 + value);       //Generate PWM.
}

void ServoD3(int value)
{
    if (value > 100)                //cap at +/- 100
        value = 100;                //  -100 => 1ms
    else if (value < -100)          //  0    => 1.5ms
        value = -100;               //  100  => 2ms
```

```
    value *= 5;                     //multiply value by 2.5
    value /= 2;                     //  new range +/- 250
    TCD0_CCD = (750 + value);       //Generate PWM.
}

void ServoD4(int value)
{
    if (value > 100)                //cap at +/- 100
        value = 100;                //  -100 => 1ms
    else if (value < -100)          //  0    => 1.5ms
        value = -100;               //  100  => 2ms
    value *= 5;                     //multiply value by 2.5
    value /= 2;                     //  new range +/- 250
    TCD1_CCA = (750 + value);       //Generate PWM.
}

void ServoD5(int value)
{
    if (value > 100)                //cap at +/- 100
        value = 100;                //  -100 => 1ms
    else if (value < -100)          //  0    => 1.5ms
        value = -100;               //  100  => 2ms
    value *= 5;                     //multiply value by 2.5
    value /= 2;                     //  new range +/- 250
    TCD1_CCB = (750 + value);       //Generate PWM.
}

/********
 * ADCA *
 ********/

void ADCAInit(void)
{
    delayInit();
    ADCA_CTRLB = 0x00;              //12bit, right adjusted
    ADCA_REFCTRL = 0x10;            //set to Vref = Vcc/1.6 = 2.0V (approx)
    ADCA_CH0_CTRL = 0x01;           //set to single-ended
    ADCA_CH0_INTCTRL = 0x00;        //set flag at conversion complete.  Disable
interrupt
    ADCA_CH0_MUXCTRL = 0x08;        //set to Channel 1
    ADCA_PRESCALER = 0x03;          //set the speed to slow for higher accuracy
    ADCA_CTRLA |= 0x01;             //Enable ADCA
}

int ADCA0(void)
{
    ADCA_CH0_MUXCTRL = 0x00;        //Set to Pin 0
    ADCA_CTRLA |= 0x04;             //Start Conversion on ADCA Channel 0
    while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01); //wait for conversion to complete
```

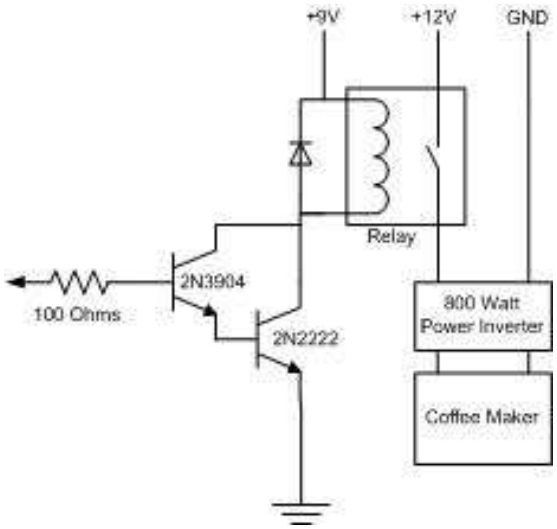
```
    delay_ms(5);
    int value = ADCA_CH0_RES;       //grab result
    return value;                   //return result
}

int ADCA1(void)
{
    ADCA_CH0_MUXCTRL = 0x08;        //Set to Pin 1
    ADCA_CTRLA |= 0x04;             //Start Conversion on ADCA Channel 0
    while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01); //wait for conversion to complete
    delay_ms(5);
    int value = ADCA_CH0_RES;       //grab result
    return value;                   //return result
}

int ADCA2(void)
{
    ADCA_CH0_MUXCTRL = 0x10;        //Set to Pin 2
    ADCA_CTRLA |= 0x04;             //Start Conversion on ADCA Channel 0
    while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01); //wait for conversion to complete
    delay_ms(5);
    int value = ADCA_CH0_RES;       //grab result
    return value;                   //return result
}

int ADCA3(void)
{
    ADCA_CH0_MUXCTRL = 0x18;        //Set to Pin 3
    ADCA_CTRLA |= 0x04;             //Start Conversion on ADCA Channel 0
    while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01); //wait for conversion to complete
    delay_ms(5);
    int value = ADCA_CH0_RES;       //grab result
    return value;                   //return result
}

int ADCA4(void)
{
    ADCA_CH0_MUXCTRL = 0x20;        //Set to Pin 4
    ADCA_CTRLA |= 0x04;             //Start Conversion on ADCA Channel 0
    while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01); //wait for conversion to complete
    delay_ms(5);
    int value = ADCA_CH0_RES;       //grab result
    return value;                   //return result
}

int ADCA5(void)
{
    ADCA_CH0_MUXCTRL = 0x28;        //Set to Pin 5
    ADCA_CTRLA |= 0x04;             //Start Conversion on ADCA Channel 0
```

```
    while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01); //wait for conversion to complete
    delay_ms(5);
    int value = ADCA_CH0_RES;       //grab result
    return value;                   //return result
}

int ADCA6(void)
{
    ADCA_CH0_MUXCTRL = 0x30;        //Set to Pin 6
    ADCA_CTRLA |= 0x04;             //Start Conversion on ADCA Channel 0
    while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01); //wait for conversion to complete
    delay_ms(5);
    int value = ADCA_CH0_RES;       //grab result
    return value;                   //return result
}

int ADCA7(void)
{
    ADCA_CH0_MUXCTRL = 0x38;        //Set to Pin 7
    ADCA_CTRLA |= 0x04;             //Start Conversion on ADCA Channel 0
    while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01); //wait for conversion to complete
    delay_ms(5);
    int value = ADCA_CH0_RES;       //grab result
    return value;                   //return result
}
```

Appendix B: Circuit Schematics



Schematic For Circuit Connected To Processor's I/O Pin to Control Buzzer



Schematic of Darlington Pair Circuit Used to Control Relay and Enable/Disable Inverter and Coffee Maker