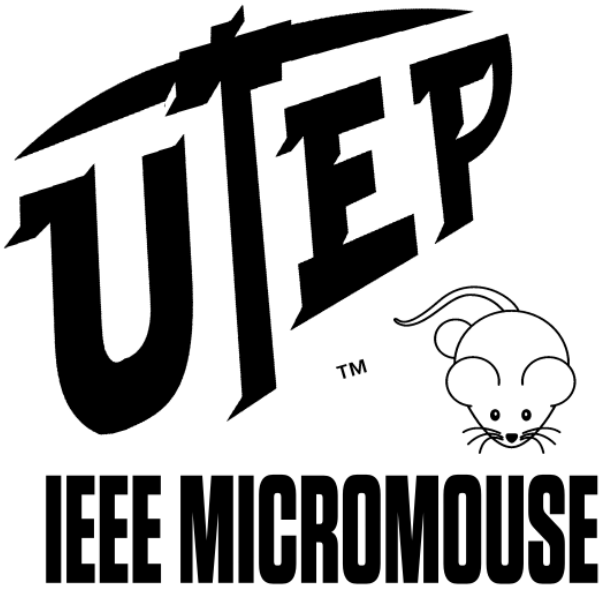

IEEE MICROMOUSE

Workshop Week 2:

Intro to Arduino and motor control

Objective:

Start coding with Arduino and learn the structure of an Arduino sketch, C programming syntax, digital input/output and motor control using the L293D dual H-bridge.

Background Information:

Today you and your teammates will get to code your first Arduino program/sketch! But before that, let's go over what and Arduino is, what can it do and how we are going to use it to solve a maze.

An Arduino is basically a tiny computer that accepts low level commands to control hardware such as DC motors, LEDs, encoders, etc. What sets an Arduino apart from a general-purpose computer is its ability to start a program almost immediately and respond to signal events in a matter of microseconds (sometimes even nanoseconds). This is important to control critical hardware processes with strict timing. A few examples are a computer mouse, computer keyboard, engine control units, home refrigeration, handled gaming devices and medical devices. You will often hear the term "embedded device" or "embedded system" when it comes to Arduinos and other similar platforms, this term is used because these devices are often "hidden" away inside everyday electronics where it might not be obvious that a computer is controlling the whole show!

So, what makes an Arduino so popular? Well, its simplicity to use! Microcontroller/embedded devices usually have a steep learning curve, and Arduino tries to lower the gap for entry level programmers, such as yourself! Now, to solve the maze using the Arduino, we must learn the structure of an Arduino sketch and some C programming. Some of the code might look complicated and intricate, but we will explain it as we go, and after some practice, you will start to see patterns in the software you write. Don't be scared by not knowing or having little practice with C, we will teach the basics and after a few workshops you will get good at C! Let's get started

Arduino Sketch:

First thing you must learn is the basic structure of an Arduino sketch. If you open an Arduino sketch you will see the following two functions.

```
1.  void setup() {
2.    // put your setup code here, to run once:
3.
4.  }
5.
6.  void loop() {
7.    // put your main code here, to run repeatedly:
8.
9.  }
```

Since we are dealing with C programming, we must explain some syntax to you:

1. A function in C is a way to organize code into functional blocks. Using a function allows you to run the same piece of code several times without having to write the same code again. Furthermore, you might think of a function as an extension to a mathematical expression *y = f(x)*, where x is the input and y the output. In C, the input can be a variety of data types and it is not only limited to integers or fractions. Here is the syntax for a function in C that takes 2 integers as its arguments and returns the sum of these 2 integers. To use a function first you must declare what the function does, then you call the function so that it runs the code inside the curly brackets.

```
1.  // A function that returns the sum of 2 integers
2.  // A function has void as its input or output will not accept or return anything
3.  // First define the function does and what is name is
4.  int add(int a, int b)
5.  {
6.    return a + b;
7.  }
8.
9.  void loop()
10. {
11.   //call your addition function and store the value in a variable
12.   int sum = add(1, 2)
13. }
```

2. A comment in C can be expressed in 2 ways

```
1.  // This is a comment. A comment is useful to explain your code
2.
3.  /* This is a multiline comment. Use this format when you must explain your code in detail
4.  and your explanation will span several lines of text
5.  */
6.
```

3. Variables in C express the data type that you are using and virtually every sketch has them. Variables, as the names implies, can change its value at any point in the program. This is true unless you declare the variable as a constant with the keyword *const.* There are several datatypes such as int, float, char, double, long and others. Variables are useful when you need a global value that will change often in your program or when you want to save the result from the output of a function.

```
1.  int integer = 10; /*This is an integer datatype; you declare the datatype then the name of
    the variable and optionally an assignment. In this case we assign the value 10 to a variable
    with the name 'integer'*
2.
3.  float fraction = 3.1415; //This is a float datatype,
4.  char character = 'a'; //This is a character datatype
```

Ok with that out of the way, lets dive into the 2 functions inside your Arduino sketch

```
1.  void setup() {
2.    // put your setup code here, to run once:
3.
4.  }
```

The setup function will be called by the Arduino when you first apply power or hit the reset button. You must define what the setup function does. This function is used to initialize all the pins that are going to be used.

```
1.  void loop() {
2.    // put your main code here, to run repeatedly:
3.
4.  }
```

The loop function will be called continuously by the Arduino ***AFTER*** the setup function has finished executing its instructions. The loop function will execute all the code inside the curly brackets line by line in order. Once it reaches the end, the loop function will go back to the beginning to execute the same code again. This is where you place all your 'useful' code, or the code that runs the main algorithm.

Just to give you a taste on what this loop will look like once we finish the micromouse, below is the pseudo code for the complete micromouse algorithm inside the loop function

```
1.  /*Micromouse algorithm to solve an unknown maze*/
2.  void loop()
3.  {
4.    //while mouse is not on the center cell
5.              //scan for walls
6.              //add walls to the maze
7.              //run path finding algorithm (flood fill)
8.              //Find best neighbor (neighbor with lowest value)
9.              //Turn towards best neighbor
10.             //Move forward
11.   //Found center of the maze!
12. }
13.
```

Arduino functions for digital input/outputs

Next, we must learn how to control digital hardware. To be more specific, we must know how to setup digital input/outputs, read digital inputs, and write to digital outputs. On your micromouse board, you will find a button next to the Arduino. Let's set it up as a digital input, read its value and store it in a variable.

```
1.  #include “micromouse.h”
2.
3.  void setup() {
4.    // put your setup code here, to run once:
5.    pinMode(BUTTON, INPUT);       //Set button pin as input
6.  }
7.
8.  void loop() {
9.    // put your main code here, to run repeatedly:
10.   int value = digitalRead(BUTTON);
11. }
```

Ok cool, we can read a digital input, but what can we do with this? Let's experiment with a digital output. Let's set up a digital output that will light up the onboard LED on the Arduino.

```
1.  #include "micromouse.h"
2.
3.  void setup() {
4.    // put your setup code here, to run once:
5.    pinMode(USER_LED, OUTPUT);
6.    pinMode(BUTTON, INPUT);
7.  }
8.
9.  void loop() {
10.   // put your main code here, to run repeatedly:
11.   int value = digitalRead(BUTTON);
12. }
13.
```

Then let's use a conditional statements to know whether the button was pressed or not. If it was pressed light up the LED, if it was not, then turn it off.

```
1.  #include "micromouse.h"
2.
3.  void setup() {
4.    // put your setup code here, to run once:
5.    pinMode(USER_LED, OUTPUT);
6.    pinMode(BUTTON, INPUT);
7.  }
8.
9.  void loop() {
10.   // put your main code here, to run repeatedly:
11.   int value = digitalRead(BUTTON);
12.
13.   if(value == HIGH)
14.   {
15.     digitalWrite(USER_LED, HIGH);
16.   }
17.   else
18.   {
19.     digitalWrite(USER_LED, LOW);
20.   }
21. }
22.
```

The only new syntax that you see on the previous code is the ***if*** and ***else*** statement, so let's go over it.

5. The if statement checks whether a statement is true or false. If it is true, then code inside the curly brackets will execute, if **IT IS NOT** true, then the code inside the else's statement curly brackets will execute. In the above example we are checking whether the variable 'value' is HIGH (button was pressed) or else (button was not pressed).

Motor control and AnalogWrite

Now we are ready to do more advanced stuff! We are going to control DC motors using 5 pins from the Arduino. These signals will control the speed and direction of the motors individually. We know that we must setup these pins as outputs, so let's do that inside the setup function

```
1.  #include "micromouse.h"
2.  void setup() {
3.    /*Setup motor pins*/
4.    pinMode(ENABLE, OUTPUT);
5.    pinMode(MOTOR_0_1A, OUTPUT);
6.    pinMode(MOTOR_0_2A, OUTPUT);
7.    pinMode(MOTOR_1_1A, OUTPUT);
8.    pinMode(MOTOR_1_2A, OUTPUT);
9.  }
10.
11. void loop() {
12. }
```

To control the motor direction and velocity, we are going to use the following truth table for the L293D chip.

Table 3. Bidirectional DC Motor Control

EN	1A	2A	FUNCTION(1)
H	L	H	Turn right
H	H	L	Turn left

(1) L = low, H = high, X = don't care

Table 3. Bidirectional DC Motor Control (continued)

EN	1A	2A	FUNCTION(1)
H	L	L	Fast motor stop
H	H	H	Fast motor stop
L	X	X	Free-running motor stop

In the table you can see that applying a HIGH to Enable, LOW to 1A and HIGH to 2A will turn the motor right. The motor will turn left if Enable is HIGH, 1A is HIGH and 2A is LOW. Finally, the motor will stop if either Enable is LOW, or if both 1A and 2A are LOW. Using this table, we can make the mouse move forward, backwards, left or right by simply changing the order of the signal values. To make this more interesting, let's also add user input with the button.

```
1.  #include "micromouse.h"
2.
3.  const int speed = 64;
4.
5.  void setup() {
6.    /*Setup motor pins*/
7.    pinMode(ENABLE, OUTPUT);
8.    pinMode(MOTOR_0_1A, OUTPUT);
9.    pinMode(MOTOR_0_2A, OUTPUT);
10.   pinMode(MOTOR_1_1A, OUTPUT);
11.   pinMode(MOTOR_1_2A, OUTPUT);
12.
13.   /*Setup button pin*/
14.   pinMode(BUTTON, INPUT);
15. }
16.
17. void loop() {
18.   /*Wait for user input*/
19.   while(digitalRead(BUTTON) != HIGH)
20.   {
21.
22.   }
23.   delay(1000);
24.
```

```
25.   /*Forward*****************************/
26.   analogWrite(MOTOR_0_1A, 0);
27.   analogWrite(MOTOR_0_2A, speed);
28.
29.   analogWrite(MOTOR_1_1A, speed);
30.   analogWrite(MOTOR_1_2A, 0);
31. }
```

On the previous code, there are 3 things that stand out. First, we have

```
1.  while(digitalRead(BUTTON) != HIGH)
2.  {
3.
4.  }
```

this introduces a new C syntax instruction, the ***while*** loop statement. This instruction will execute the code inside the curly brackets ***UNTIL*** the condition is met. In this case we are constantly asking if the user has pressed the button or to put it simply, do nothing WHILE the BUTTON **IS NOT** HIGH.

Another interesting piece of code is line 3.

```
1.  const int speed = 64;
```

Here, we are declaring a variable named speed that will be visible to all portions of the Arduino sketch. Notice that we are declaring this variable as ***const*** since we are not planning to change this variable once the program starts.

Declaring global variables like this can be useful when several portions of your program use the same value. If the value changes in the future, all references to that variable will change as well. So, for the example above, you only must change the global 'speed' variable to modify the speed in which the mouse moves.

Lastly, on line 23, we have the function

```
1.  delay(1000)
```

This will simply delay execution of the next instruction by 1000 milliseconds or 1 second

Exercise:

Now is your turn to write some code and start practicing with C. Using the code template below implement the following algorithm at the WRITE YOUR OWN CODE comment

```
/*This micromouse workshop will teach you:
 * 1. Arduino sketch structure
 * 2. Basic C syntax
 * 3. Digital input output
 * 4. Motor driver truth table
 * 5. Driving forward/ backwards, turning left/right
 *
 *
 * L293D dual h-bridge datasheet https://www.ti.com/lit/ds/symlink/l293.pdf
 *
 * Bidirectional motor control page 11-12
 * ENABLE     |     MOTORx_1A     |     MOTORx_2A     |     FUNCTION
 * ------------------------------------------------------------------------------
 *  HIGH      |       LOW         |       HIGH        |     Turn right
 *  HIGH      |       HIGH        |       LOW         |     Turn left
 *  HIGH      |       LOW         |       LOW         |     Fast motor stop
 *  LOW       |       X           |       X           |     Free-running motor stop
 *
 *  MOTORx_1A and MOTORx_2A can be replaced with PWM signals to control the velocity of the motor
 */

#include "micromouse.h"

const int speed = 64;

void setup() {
  /*Setup motor pins*/
  pinMode(ENABLE, OUTPUT);
  pinMode(MOTOR_0_1A, OUTPUT);
  pinMode(MOTOR_0_2A, OUTPUT);
  pinMode(MOTOR_1_1A, OUTPUT);
  pinMode(MOTOR_1_2A, OUTPUT);

  /*Setup button pin*/
  pinMode(BUTTON, INPUT);
}

void loop() {
  /*Wait for user input*/
  while(digitalRead(BUTTON) != HIGH);
  /*Delay 1000 milliseconds*/
  delay(1000);

  /*Forward*/
  analogWrite(MOTOR_0_1A, 0);
  analogWrite(MOTOR_0_2A, speed);

  analogWrite(MOTOR_1_1A, speed);
  analogWrite(MOTOR_1_2A, 0);

  /*Delay 1000 milliseconds*/
  delay(1000);

  /*Stop*/
  analogWrite(MOTOR_0_1A, 0);
  analogWrite(MOTOR_0_2A, 0);
```

```
57.
58.   analogWrite(MOTOR_1_1A, 0);
59.   analogWrite(MOTOR_1_2A, 0);
60.
61.   /******WRITE YOUR OWN CODE*******/
62.   /*Wait for user input*/
63.
64.   /*Delay 1000 milliseconds*/
65.
66.   /*Backward*/
67.
68.   /*Delay 1000 milliseconds
69.
70.   /*Stop*/
71.
72.   /*Wait for user input*/
73.
74.   /*Delay 1000 milliseconds*/
75.
76.   /*Turn right*/
77.
78.   /*Delay 1000 milliseconds
79.
80.   /*Stop*/
81.
82.   /*Wait for user input*/
83.
84.   /*Delay 1000 milliseconds*/
85.
86.   /*Turn left*/
87.
88.   /*Delay 1000 milliseconds
89.
90.   /*Stop*/
91. }
```