FORMAN CHRISTIAN COLLEGE
(A CHARTERED UNIVERSITY)

CSCS 306 – Embedded Systems
Fall 22
Lab - 05

Adeel Akmal – 231465420
Muhammad Sameed Gilani - 231488347

INTRODUCTION:

- **delayMicroseconds(us)**
  Parameters →us: the number of microseconds to pause. Allowed data types: unsigned int.
  Returns → Nothing

Pauses the program for the amount of time (in microseconds) specified by the parameter. There are a thousand microseconds in a millisecond and a million microseconds in a second. Currently, the largest value that will produce an accurate delay is 16383; larger values can produce an extremely short delay. This could change in future Arduino releases. For delays longer than a few thousand microseconds, you should use `delay()` instead.

- **pulseIn(pin, value)**

  Parameters → pin: the number of the Arduino pin on which you want to read the pulse. Allowed data types: int. value: type of pulse to read: either HIGH or LOW. Allowed data types: int. timeout (optional): the number of microseconds to wait for the pulse to start; default is one second. Allowed data types: unsigned long.

  Returns → The length of the pulse (in microseconds) or 0 if no pulse started before the timeout. Data type: unsigned long.

Reads a pulse (either `HIGH` or `LOW`) on a pin. For example, if `value` is `HIGH`, `pulseIn()` waits for the pin to go from `LOW` to `HIGH`, starts timing, then waits for the pin to go `LOW` and stops timing. Returns the length of the pulse in microseconds or gives up and returns 0 if no complete pulse was received within the timeout.

The timing of this function has been determined empirically and will probably show errors in longer pulses. Works on pulses from 10 microseconds to 3 minutes in length.

- **map(value, fromLow, fromHigh, toLow, toHigh)**

  Parameters → `value`: the number to map.

  `fromLow`: the lower bound of the value's current range.

  `fromHigh`: the upper bound of the value's current range.

  `toLow`: the lower bound of the value's target range.

  `toHigh`: the upper bound of the value's target range.

  Returns → The mapped value. Data type: long.

Re-maps a number from one range to another. That is, a value of **fromLow** would get mapped to **toLow**, a value of **fromHigh** to **toHigh**, values in-between to values in-between, etc.

Does not constrain values to within the range, because out-of-range values are sometimes intended and useful. The `constrain()` function may be used either before or after this function, if limits to the ranges are desired.

Note that the "lower bounds" of either range may be larger or smaller than the "upper bounds" so the `map()` function may be used to reverse a range of numbers, for example

```
y = map(x, 1, 50, 50, 1);
```

The function also handles negative numbers well, so that this example

```
y = map(x, 1, 50, 50, -100);
```

is also valid and works well.

The map() function uses integer math so will not generate fractions, when the math might indicate that it should do so. Fractional remainders are truncated, and are not rounded or averaged.

- **analogWrite(pin, value)**
  Parameters →pin: the Arduino pin to write to. Allowed data types: int. value: the duty cycle: between 0 (always off) and 255 (always on). Allowed data types: int.
- **Returns** → Nothing

rites an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to analogWrite(), the pin will generate a steady rectangular wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or digitalWrite()) on the same pin.

- **millis()**
  Parameters → Nothing
  Returns →Number of milliseconds passed since the program started. Data type: unsigned long.

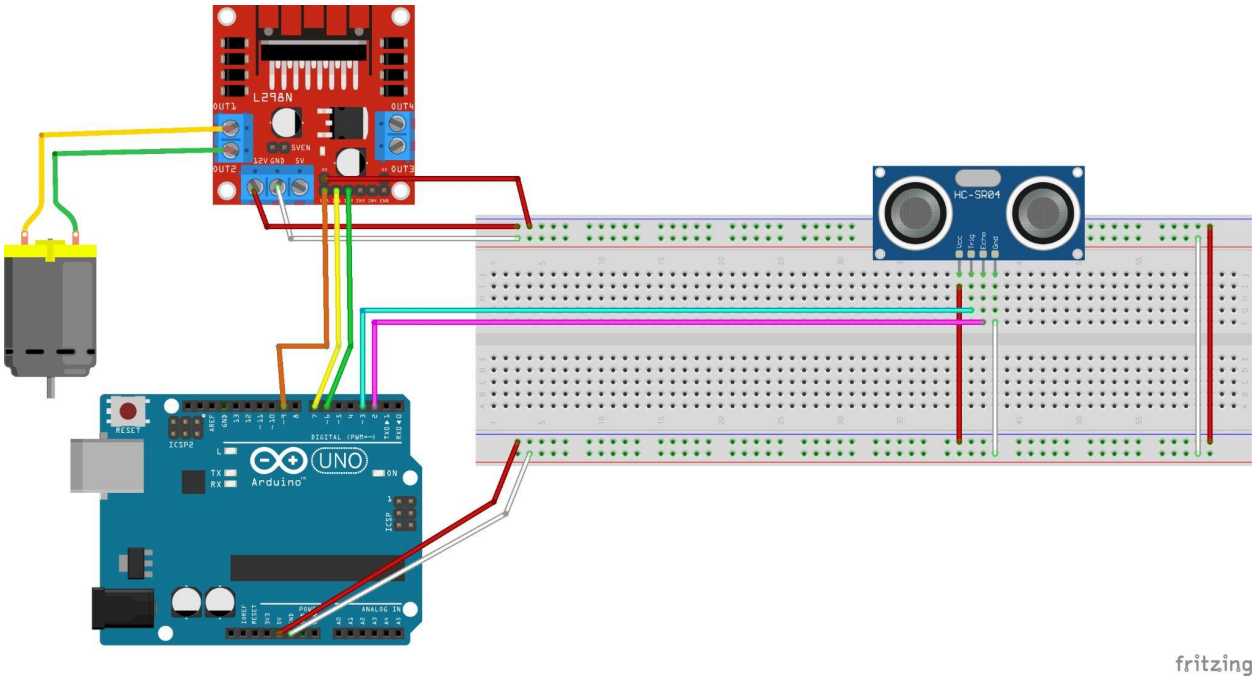
Returns the number of milliseconds passed since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.

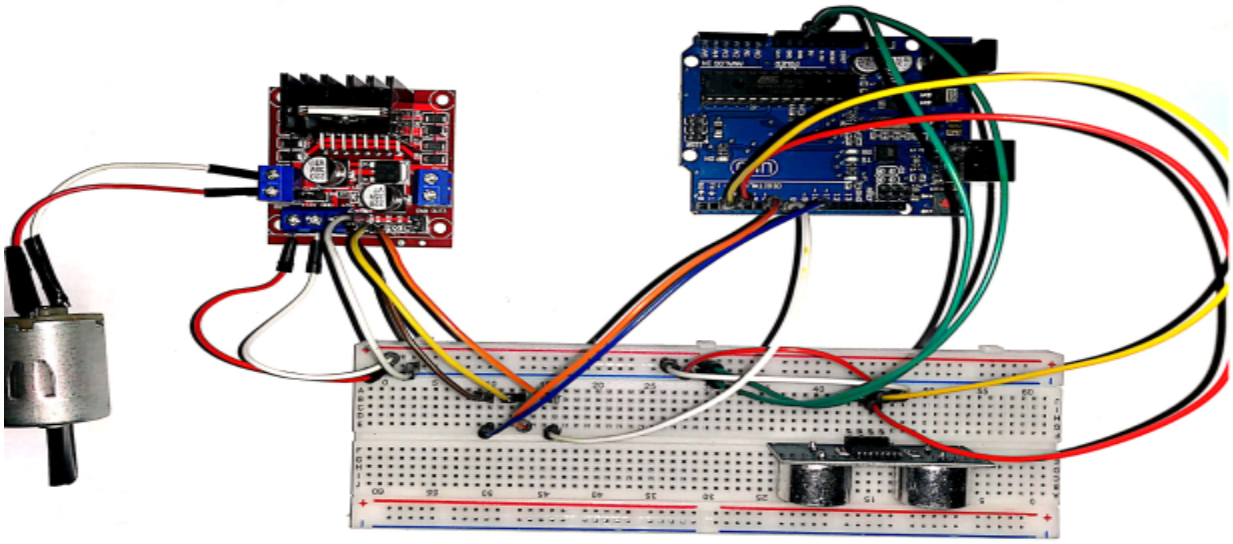
LOGIC:

- Initialized all variables and set necessary pins to their modern using port registers.
- Use pulseIn() and appropriate delays to use the ultrasonic sensor and calculate the distance reading from it.
- Set the speed of the motor using map(), according to the current distance reading.
- If the distance is > 20 then speed is 255
  If the distance is <5 then speed is 0
- Use millis() as a non time blocking delay, which after every 3 seconds, prints the current distance from the ultrasonic sensor.

* There is a flag set up, which in every alternate reading of the Ultrasonic sensor sets the current reading (Which comes out to 0) to the previous reading. It basically removes the 0 readings from the sensor, which causes issues with the distance reading we print.

CIRCUIT:





CODE:

```
int echoPin = 2;

int pwm = 9;
long pwm_speed;

long startTime = 0;
int distKeep;
bool flag = false;

long duration; // variable for the duration of sound wave travel
int distance; // variable for the distance measurement

void setup(){

  DDRD = B11101000; // Turn pin 2,6,7 to Output and 3 to input
  DDRB = B00000010; // Turn pin 9 to output
```

```
  PORTD = B01000000; // Turn pin 6 high and pin 7 low

  Serial.begin(9600);
}

void loop(){

  PORTD = B01000000; // pin 3 low
  delayMicroseconds(2);

  PORTD = B01001000; // pin 3 high

  delayMicroseconds(10);
  PORTD = B01000000; // pin 3 low

  duration = pulseIn(echoPin, HIGH);

  distance = duration * 0.034 / 2;  // Sonar distance reading

  if(flag == false){    // flag setup to fix millis() getting
the distance 0
    distKeep = distance;
  }
  flag = !flag;


  pwm_speed = map(distance,5,20,0,255); // Set speed according
to the sonars distance

  if(distance > 20){
      pwm_speed = 255;
  }

  if(distance < 5){
      pwm_speed = 0;
  }

  analogWrite(pwm,pwm_speed);
```

```
  if(distance == 0){
    distance = distKeep;
  }

  if(millis() - startTime >= 3000){
    startTime = millis();

    Serial.print("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");
  }

}
```

OUTPUT: