Laboratorio de Principios de Mecatrónica

11 de febrero de 2022

Práctica #1

Microcontroladores

Grupo: 1

L002

Estudiante:

- García Cortez Alejandro
- Hermida Flores Manuel Joaquín
- Chicatti Avendaño Josué Doménico

Profesor:

Benito Granados–Rojas

Índice

1. Introducción

Esta práctica tiene como objetivo ser la introducción al laboratorio de Principios de Mecatrónica. A lo largo del curso estaremos usando el Arduino MEGA 2560 y en esta práctica desarrollamos 3 actividades, no solo para conocer más sobre el microcontrolador, sino para también comparar las distintas formas en que se puede programar.

La primer actividad consistió en correr un programa de ejemplo, que simplemente encendía y apagaba un LED. Luego se modificó para que parpadée solo cuando se presiona un botón. En esta actividad programamos al mismo tiempo que implementamos físicamente un circuito sencillo.

La siguiente actividad consistió en realizar el programa de parpadeo, solo que usando un lenguaje de bajo nivel, específicamente ensamblador. El propósito general de esta actividad fue ver las diferencias entre alto y bajo nivel, así como identificar los diferentes comandos mnemónicos y registros.

Finalmente, la última actividad consistió en programar, a alto nivel, la secuencia de dos semáforos. Cada semáforo consistió de 3 LEDs, que se encendían y apagaban en una secuencia específica.

2. Experimentos y Simulaciones

2.1. Blink (Encendido y apagado de un LED)

Este primer ejercicio tuvo tres pequeñas subpartes.

Primero, simplemente ejecutamos el IDE de Arduino e importamos el código Blink que viene listo para correr y que hace parpadear el LED, incluído en el Arduino, una vez por segundo (1 Hz). Esto también sirvió para verificar que todo estuviera bien con el Arduino y la conexión a la PC.

Segundo, realizamos una conexión con una resistencia pull-up de 10k para incorporar un botón como entrada y otra resistencia para limitar la corriente hacia un LED externo que se usó como salida. En el código sólo tuvimos que indicar qué pines digitales íbamos a usar:

```
const int buttonPin = 7;
const int ledPin = 13;

void setup() {
    pinMode(ledPin, OUTPUT);
    pinMode(buttonPin, INPUT);
}
...
```

Tercero, cambiamos el código de la segunda parte para que cuando el botón se esté presionando, el LED parpadée a una menor frecuencia: una vez cada dos segundos (0.5Hz).

```
    ...
    void loop() {
  if(digitalRead(buttonPin)==HIGH)
  {
    digitalWrite(ledPin, HIGH);
    delay(2000);
    digitalWrite(ledPin, LOW);
    delay(2000);
  }
  else{
    digitalWrite(ledPin, HIGH);
    delay(1000);
    digitalWrite(ledPin, LOW);
    delay(1000);
  }
}
```

2.2. Blink ASM

El siguiente ejercicio de la práctica consistió en repetir el programa de *Blink*, solo que ahora programando en lenguaje ensamblador.

El programa comienza en el *setup*. Asignando un valor a DDRB, declaramos que pines del puerto B funcionarán como entradas y salidas. En este caso asignamos el pin 7 como un output y del 0-6 se asignan como inputs, aunque no se usen. El pin 7 del puerto B es equivalente al pinD13.

A continuación comenzamos con el programa en *loop*. Primero establecemos la rutina 'inicio', que comienza poniendo en 1, el bit 7 del registro 5. Este registro es el mismo que el del port B, al asignar 1 en el bit 7, efectivamente estamos prendiendo el LED. A continuación llama a la rutina 'tiempo'. Después de ejecutarla usa el comando CBI para asignar 0 en el bit 7 del registro 5, efectivamente apagando el LED. Después de esperar otro momento, el porgrama vuelve a comenzar.

La subrutina tiempo consiste en una serie de subrutinas que se encargan de 'quemar' ciclos de reloj. Para esto nos apoyamos de 3 registros: r22, r21 y r20, a los cuales se les carga un valor con LDI (Load Immediate). Por otro lado, se van declarando las subrutinas LOOP_1, LOOP_2, LOOP_3, las cuales están anidadas. El programa comienza entonces a contar regresivamente. Empieza decrementando (DEC) el valor del registro 20 hasta que este llega a 0, entonces salta a LOOP_2, que decrementa el valor del registro 21, pero vuelve a ejecutar LOOP_1. Siguiendo al terminar este programa, se han quemado un poco más de $255 * 255 * 45 = 2926125$ ciclos, lo cual

equivale a poco menos de un segundo en la realidad.

A partir de esta forma de contar el tiempo, y de repetir las instrucciones en *loop*, se logra que el LED parpadée. A continuación se presenta el código comentado.

```
void setup()
{
  DDRB = DDRB | B10000000; // Data Direction Register B: Inputs 0-6, Output 7
  // Con este comando configuramos el registro del puerto B como una entrada
}
void loop()
{
  asm (
    "inicio: \n\t"          // Rutina "inicio"
    "sbi 0x05,0x07 \n\t"    // SBI asigna un 1 al bit 7 del registro 5
    // EL registro 0X05 corresponde a PORTB
    "call tiempo \n\t"      // Llamamos a la subrutina tiempo
    "cbi 0x05,0x07 \n\t"    // SBI asigna un 0 al bit 7 del registro 5
    // EL registro 0X05 corresponde a PORTB
    "call tiempo \n\t"      // Llamamos a la subrutina tiempo
    "jmp main \n\t"         // Saltamos a main, volvemos a comenzar.

    "tiempo: \n\t"      //subrutina "tiempo"
    "LDI r22, 45 \n\t"  // Carga 45 al registro 22?
        "LOOP_3: \n\t"      // Subsubrutina loop3
        "LDI r21, 255 \n\t" // Carga 255 al registro r21
            "LOOP_2: \n\t"      // Subsubrutina loop2
            "LDI r20, 255 \n\t" // Carga 255 al registro 20
                "LOOP_1: \n\t"      // Subsubrutina loop1
                "DEC r20 \n\t"      // Resta 1 al registro 20
                "BRNE LOOP_1 \n\t"  // = 0 continua, =/= 0 ejecuta LOOP_1
            "DEC r21 \n\t"      // Resta 1 al registro 21
            "BRNE LOOP_2 \n\t"  // = 0 continua, =/= 0 ejecuta LOOP_2
        "DEC r22 \n\t"      // Resta 1 al registro 22
        "BRNE LOOP_3 \n\t"  // = 0 continua, =/= 0 ejecuta LOOP_3
    "ret \n\t"          // Return
  );
}
```

2.3. Semáforo digital arduino

El último ejercicio consistió en implementar dos semáforos digitales. Cada semáforo se construyó con 3 LEDs (verde, amarillo y rojo), cada uno con su respectiva resistencia. La conexión en la protoboard se hizo de la misma manera que el primer ejercicio. El puerto digital se conecta a la pata positiva del LED, la pata negativa

se une con la resistencia y la resistencia va a tierra. Esta misma conexión se realiza para cada LED como se observa en la Figura 1.

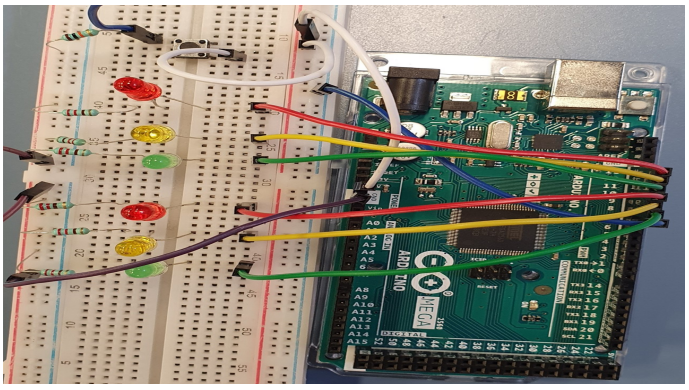

Figura 1: Conexión en protoboard

Posteriormente se escribió el código en el IDE de Arduino. El código debe generar el comportamiento en los semáforos dado por la siguiente tabla:

Cuadro 1: Tabla de estados de semáforos

Estado semáforo 1			Estado semáforo 2			Duración (s)
rojo	amarillo	verde	rojo	amarillo	verde	
HIGH	LOW	LOW	LOW	LOW	HIGH	5
HIGH	LOW	LOW	LOW	HIGH	LOW	1
LOW	LOW	HIGH	HIGH	LOW	LOW	5
LOW	HIGH	LOW	HIGH	LOW	LOW	1

El código se divide en tres partes principales:

1. La primera parte del código consiste en la declaración de variables que van a tomar el valor de cada uno de los puertos digitales conectados a los LEDs. De esta manera podemos manejar cada LED con un nombre adecuado.

```
const int r1 = 13;
// se repite para cada LED restantes:
// a1, v1, r2, a2, v2
...
```

2. La segunda sección es el void setup(), que es la primera función que se ejecuta y prepara el programa. Por esa razón, ahí definimos que nuestros pines se comportarán como salidas (OUTPUTS), ya que estamos trabajando con los LEDs.

```
void setup() {
    pinMode(r1, OUTPUT);
    // se repite para las demás variables
    ...
}
```

3. Finalmente tenemos el void loop(), nuestra función principal que será ejecutada mientras la placa esté encendida. En esta sección prendemos y apagamos los LEDs usando digitalWrite(LED, HIGH/LOW), y los mantenemos en ese estado con los delays. De cierta manera podemos ver cada estado dividido por el delay.

```
void loop() {
    digitalWrite(r2, LOW);
    digitalWrite(a1, LOW);
    digitalWrite(r1, HIGH);
    digitalWrite(v2, HIGH);
    delay(5000);
    digitalWrite(v2, LOW);
    digitalWrite(a2, HIGH);
    delay(1000);
    digitalWrite(r1, LOW);
    digitalWrite(a2, LOW);
    digitalWrite(v1, HIGH);
    digitalWrite(r2, HIGH);
    delay(5000);
    digitalWrite(v1, LOW);
    digitalWrite(a1, HIGH);
    delay(1000);
}
```

Hay que considerar que void loop() seguirá ejecutándose indefinidamente. Debido a ese comportamiento, las luces que quedarón prendidas al final de ella deben apagarse en las primeras líneas de la función. El código completo se encuentra en el repositorio cuyo link se encuentra al final del documento.

3. Conclusiones

Adicionalmente a lo ya explicado en cada sección sólo queda agregar que al realizar la práctica identificamos algunos de los vínculos con los temas de la clase de teoría. Se cumplieron los objetivos de la práctica mediante el uso de un Arduino y su IDE.

Especialmente, reflexionar sobre el código de alto y bajo nivel evidencía la diferencia en las curvas de aprendizaje de ambos lenguajes. Además, el lenguaje de alto nivel permite realizar un proceso en menos líneas de código y utiliza un lenguaje más comprensible.

4. Enlaces externos

`https://github.com/ManoHF/lab-mecatronica`

Referencias

[1] "AVR GPIO Programming Tutorial - Atmega328p - AVR - 8-bit - Arduino," *Arnab Kumar Das*, Dec. 29, 2018. https://www.arnabkumardas.com/arduino-tutorial/gpio-programming/ (accessed Feb. 10, 2022).