UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

INSTRUMENTACIÓN

PRÁCTICA #2

Simulación de un Semáforo

Alumno
VÁSQUEZ CASTAÑEDA CARLOS ANTONIO

Profesor
JOSÉ MANUEL RAMÍREZ ZÁRATE

Grupo 395

Matrícula: 1155057

Febrero 12, 2020

Práctica #2: Simulación de un Semáforo

Carlos A. Vásquez Castañeda 1155057 Grupo 395

Febrero 12, 2020

Introducción

Los sistemas electrónicos con microcontroladores se utilizan en muchos dispositivos hoy en día. Comprenderlos resulta de gran ventaja y realizar un programa que simule uno de estos dispositivos resulta en un beneficio para la comprensión de estos sistemas.

Objetivo

El objetivo de esta práctica es reforzar aquello enseñado por la práctica #1 y volverlo un poco más complicado para entender la lógica de programación detrás de Arduino.

Marco Teórico

Para la elaboración de el semáforo es necesario entender algunas funciones básicas:

- pinMode(pin, mode): configura el modo de salida de cada conexión disponible en el microcontrolador de Arduino.
- digitalWrite(pin, HIGH/LOW): escribe un valor de ALTO o BAJO en la conexión del contacto que se ha elegido, sean 5 volts o 3.3 volts.

Materiales y Equipo

- Arduino UNO
- Arduino IDE
- Proteus 8 Professional
- LEDs (3)
- Resistencia 220 Ω (3)
- Referencias de internet (StackExchange, Arduino.cc)

Desarrollo

Dada la simulación que deseamos realizar es la de un semáforo, tendremos que configurar 3 luces distintas para que actúen como aquellas que tienen un semáforo de verdad. Para esto utilizaremos tres LEDs, uno verde, uno amarillo y otro rojo. Estos los definimos al inicio del código del programa, como se muestra a continuación.

```
1  int v = 13;
2  int a = 12;
3  int r = 11;
```

Una vez asignadas estas variables a los distintas conexiones del microcontrolador, es necesario definirlas como posibles salidas o entradas. Dado el funcionamiento que queremos obtener del controlador, será necesario configurar estas conexiones como salidas.

```
1  void setup () {
2    pinMode(v, OUTPUT);
3    pinMode(a, OUTPUT);
4    pinMode(r, OUTPUT);
5    }
```

Finalmente sigue regular el encendido y apagado de las distintas luces que estarán a nuestra disposición. Para esto, considerando que aún no hemos aprendido estructuras de

código complejas, utilizaremos lo que hemos aprendido en la práctica #1 para realizar el siguiente bloque de código.

```
1  void loop () {
2    digitalWrite(v, 1);
3    delay(5000);
4
5    digitalWrite(v, 0);
6    delay(500);
7    digitalWrite(v, 1);
8    delay(500);
9    digitalWrite(v, 0);
10   delay(500);
11   digitalWrite(v, 1);
12   delay(500);
13   digitalWrite(v, 0);
14   delay(500);
15   digitalWrite(v, 1);
16   delay(500);
17   digitalWrite(v, 0);
18   delay(500);
19   digitalWrite(v, 1);
20   delay(500);
21   digitalWrite(v, 0);
22   delay(500);
23   digitalWrite(v, 1);
24   delay(500);
25   digitalWrite(v, 0);
26 }
```

En el bloque de código anterior enciende el LED de color verde y espera durante 5000

milisegundos o 5 segundos para tomar la siguiente acción. Posteriormente procede a apagar el LED verde, esperar medio segundo, volverlo a encender y esperar otro medio segundo. Todo esto toma lugar cinco veces para simular el parpadeo de un semáforo convencional.

A continuación, inmediatamente después de la última vez que el LED verde se apaga, procederemos a encender el LED amarillo para que no exista ninguna inconsistencia y el semáforo funcione de manera adecuada.

```
1  digitalWrite(a, 1);
2  delay(1000);
3  digitalWrite(a, 0);
```

La función *delay* hace esperar al controlador 1 segundo para tomar la siguiente acción que es apagar el LED amarillo, posteriormente se realiza el siguiente bloque de código para encender el LED rojo, dejarlo encendido durante 2 segundos y posteriormente apagarlo.

```
1  digitalWrite(r, 1);
2  delay(2000);
3  digitalWrite(r, 0);
```

Y dado el bucle infinito, este programa actuará de esta manera las veces que sean necesarias hasta que se desenergize el circuito.

Diagrama de Flujo

Observando el proceso mediante bloques de diagrama de flujo luciría similar a esto:

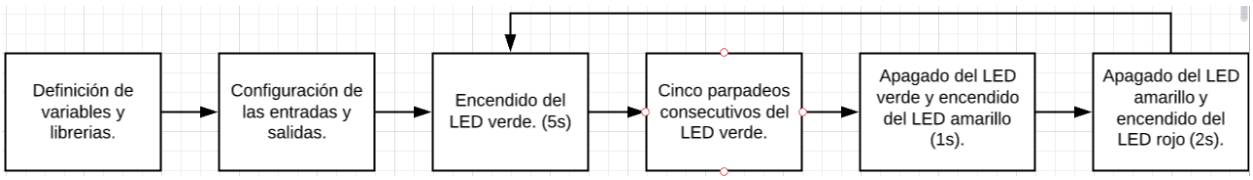

Figure 1: Proceso del programa elaborado anteriormente.

Como es posible observar, la complejidad del programa ha aumentado un poco, sin embargo sigue siendo bastante lineal y el microcontrolador no toma decisiones en base a otros

datos. Podríamos decir que sólo sigue una serie de instrucciones indefinidamente y es por eso que nuestro diagrama de flujo se muestra tan simple.

Código

A continuaciön se muestra el código en su totalidad.

```
1 int v = 13;
2 int a = 12;
3 int r = 11;
4
5 void setup () {
6   pinMode(v, OUTPUT);
7   pinMode(a, OUTPUT);
8   pinMode(r, OUTPUT);
9 }
10
11
12 void loop () {
13
14   digitalWrite(v, 1);
15   delay(5000);
16
17   digitalWrite(v, 0);
18   delay(500);
19   digitalWrite(v, 1);
20   delay(500);
21   digitalWrite(v, 0);
22   delay(500);
23   digitalWrite(v, 1);
24   delay(500);
```

```
25   digitalWrite(v, 0);
26   delay(500);
27   digitalWrite(v, 1);
28   delay(500);
29   digitalWrite(v, 0);
30   delay(500);
31   digitalWrite(v, 1);
32   delay(500);
33   digitalWrite(v, 0);
34   delay(500);
35   digitalWrite(v, 1);
36   delay(500);
37   digitalWrite(v, 0);
38
39   digitalWrite(a, 1);
40   delay(1000);
41   digitalWrite(a, 0);
42   digitalWrite(r, 1);
43   delay(2000);
44   digitalWrite(r, 0);
45
46 }
```

Diagrama Esquemático

El diagrama esquemático es bastante parecido al de la práctica #1, sólo se repite la misma conexión hecha en la conexión #13, por lo tanto a partir de la conexión 11, 12 y 13 se conectará a cada una, una resistencia de 220 Ω y un LED de distinto color en serie. Cada una de estas conexiones irá a la referencia para completar el circuito eléctrico.

A continuación se muestra el diagrama esquemático:

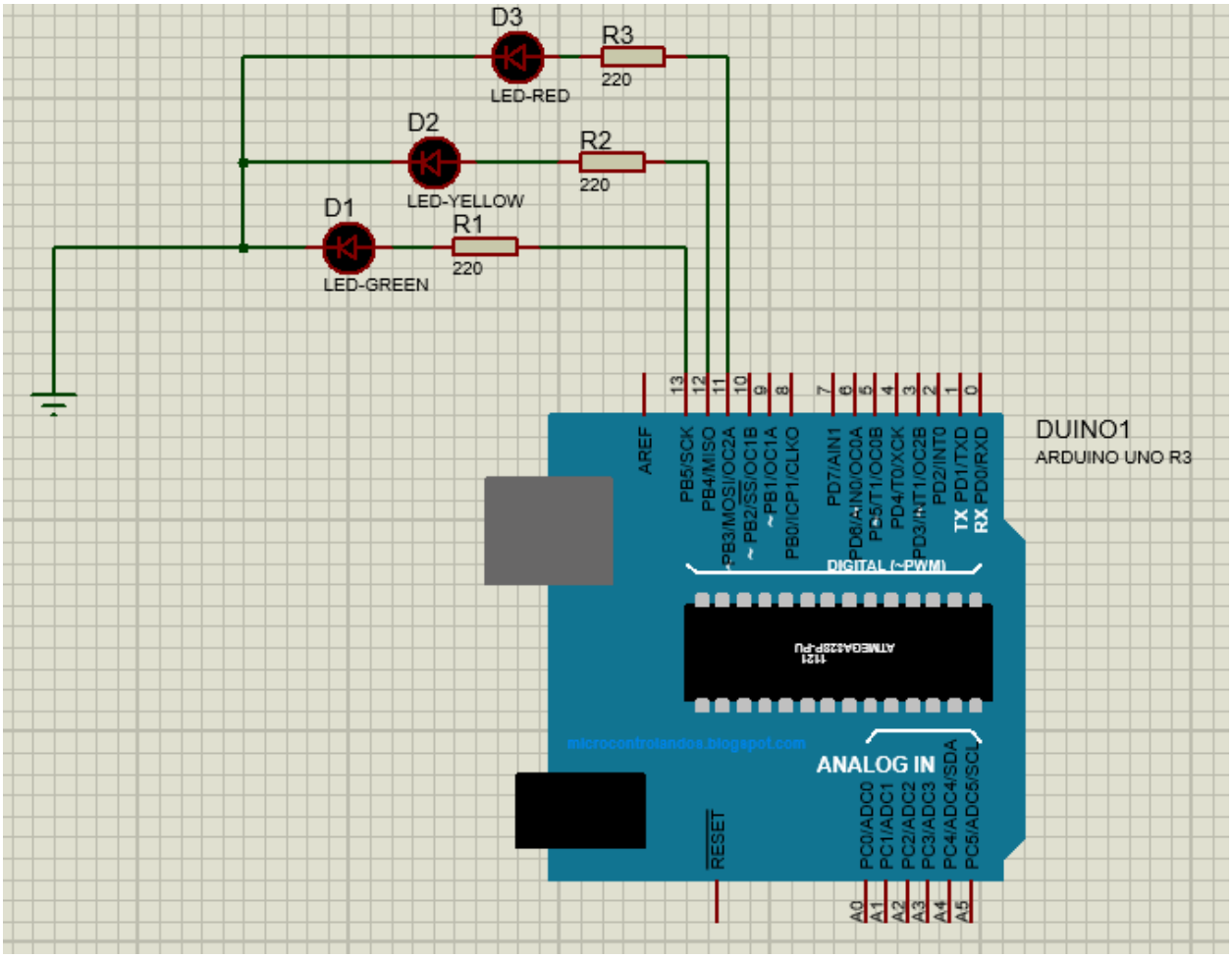

Figure 2: Diagrama de las conexiones necesarias y componentes utilizados.

Conclusión

A través de esta práctica es sencillo entender la lógica detrás del lenguaje de prorgamación Arduino, reforzando lo visto en la práctica anterior. Sin embargo, después de ver los resultados y el código, es posible concluir que es muy poco eficiente y es posible optimizarlo mucho más mediante estructuras de código más complejas como los *if statements* y los bucles.

Anexos

Hexdump

```
:100000000C945C000C9479000C9479000C947900A9
```

```
:100010000C9479000C9479000C9479000C9479007C
:100020000C9479000C9479000C9479000C9479006C
:100030000C9479000C9479000C9479000C9479005C
:100040000C94B6010C9479000C9479000C9479000E
:100050000C9479000C9479000C9479000C9479003C
:100060000C9479000C947900000000070002010054
:1000700000030406000000000000000000102040864
:10008000102040800102040810200102040810 2002
:1000900004040404040404020202020203032E
:1000A000030303030000000250028002B000000CC
:1000B0000000240027002A0011241FBECFEFD8E043
:1000C000DEBFCDBF11E0A0E0B1E0EAE2F5E002C0A2
:1000D00005900D92A630B107D9F721E0A6E0B1E076
:1000E00001C01D92AF30B207E1F70E9486020C9466
:1000F00093020C94000061E0809104010E944B0186
:1001000061E0809102010E944B0161E08091000159
:100110000C944B01CF93DF93C4E0D1E061E0888180
:100120000E94870168E873E180E090E00E94250268
:1001300060E088810E94870164EF71E080E090E0D8
:100140000E94250261E088810E94870164EF71E0CE
:1001500080E090E00E94250260E088810E94870193
:1001600064EF71E080E090E00E94250261E0888108
:100170000E94870164EF71E080E090E00E94250218
:1001800060E088810E94870164EF71E080E090E088
:100190000E94250261E088810E94870164EF71E07E
:1001A00080E090E00E94250260E088810E94870143
:1001B00064EF71E080E090E00E94250261E08881B8
:1001C0000E94870164EF71E080E090E00E942502C8
:1001D00060E088810E94870164EF71E080E090E038
:1001E0000E94250261E088810E94870164EF71E02E
:1001F00080E090E00E94250260E088810E948701F3
:10020000C2E0D1E061E088810E94870168EE73E07E
:1002100080E090E00E94250260E088810E948701D2
:10022000C0E0D1E061E088810E94870160ED77E065
:1002300080E090E00E94250260E08881DF91CF910C
:100240000C948701833081F028F4813099F082305A
:10025000A1F008958630A9F08730B9F08430D1F448
:1002600080918 0008F7D03C0809180008F77809384
:1002700080000895 84B58F7702C084B58F7D84BDDA
:1002800008958091B0008F7703C08091B0008F7D7A
:100290008093B0000895CF93DF9390E0FC01E45881
:1002A000FF4F2491FC01E057FF4F8491882361F1B7
:1002B00090E0880F991FFC01E255FF4FC591D49142
:1002C000FC01EC55FF4FA591B491611109C09FB796
:1002D000F894888120958223 8883EC912E230BC08B
```

```
:1002E000623061F49FB7F8948881322F3095832370
:1002F0008883EC912E2B2C939FBF06C08FB7F89468
:10030000E8812E2B28838FBFDF91CF9108951F9313
:10031000CF93DF93282F30E0F901E859FF4F849104
:10032000F901E458FF4FD491F901E057FF4FC49110
:10033000CC23C1F0162F81110E942201EC2FF0E096
:10034000EE0FFF1FEC55FF4FA591B4919FB7F894A6
:10035000111104C08C91D095D82302C0EC91DE2BF2
:10036000DC939FBFDF91CF911F9108951F920F9251
:100370000FB60F9211242F933F938F939F93AF93B8
:10038000BF9380910701909108801A0910901B0915C
:100390000A013091060123E0230F2D3720F4019646
:1003A000A11DB11D05C026E8230F0296A11DB11D98
:1003B000209306018093070190930801A0930901FF
:1003C000B0930A0180910B0190910C01A0910D0155
:1003D000B0910E010196A11DB11D80930B01909368
:1003E0000C01A0930D01B0930E01BF91AF919F91AD
:1003F0008F913F912F910F900FBE0F901F901895E6
:100400003FB7F89480910B0190910C01A0910D01E0
:10041000B0910E0126B5A89B05C02F3F19F001969B
:10042000A11DB11D3FBFBA2FA92F982F8827820F7A
:10043000911DA11DB11DBC01CD0142E0660F771FCA
:10044000881F991F4A95D1F70895CF92DF92EF92B6
:10045000FF92CF93DF936B017C010E940002EB01BE
:10046000C114D104E104F10479F00E9400026C1B74
:100470007D0B683E7340A0F381E0C81AD108E10803
:10048000F108C851DC4FECCFDF91CF91FF90EF9096
:10049000DF90CF900895789484B5826084BD84B550
:1004A000816084BD85B5826085BD85B5816085BD6F
:1004B000EEE6F0E0808181608083E1E8F0E0108288
:1004C000808182608083808181608083E0E8F0E0C9
:1004D000808081816080833E1EBF0E0808184608083B3
:1004E000E0EBF0E0808181608083EAE7F0E08081EA
:1004F000846080838081826080838081816080834A
:10050000808180688083109 2C10008950E944B0210
:100510000E947B00C0E0D0E00E948A002097E1F3B7
:0A0520000E940000F9CFF894FFCF0D
:06052A000B000C000D00A7
:00000001FF
```