KITS ARDUINO PARA APOYAR EL APRENDIZAJE Y ENSEÑANZA EN LA PROGRAMACIÓN DE MICROCONTROLADORES

DOCUMENTO DE PROTOCOLO DE PRUEBAS DE GUÍAS DIDÁCTICAS

AUTORES:
Brayan Camilo Lozano Polanía
Daniel Naranjo Imbachi

PRESENTADO A:
Juan Carlos Giraldo Carvajal , Ph.D.
Arturo Fajardo Jaimes, Ph.D.

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE ELECTRÓNICA
BOGOTÁ D.C.
2023

1. INTRODUCCIÓN

En este documento se realizará el protocolo de pruebas correspondiente al software de las guías didácticas que se usan para el proceso de enseñanza – aprendizaje de programación de microcontroladores, las cuales hacen uso del kit de aprendizaje, de tal forma que se puede verificar si la lógica y la codificación están correctas, para después poderla verificar con el hardware prestablecido.

2. OBJETIVOS

Objetivo General:
El objetivo principal es verificar por medio de un protocolo de pruebas los procesos de diseño de códigos utilizados en las guías de aprendizaje que se utilizan junto con el kit de aprendizaje.

Objetivos Específicos:

- Verificar las entradas y salida que intervienen en el programa diseñado mediante un diagrama de caja negra.
- Verificar la FSM (si se implementa) del programa y determinar sus posibles estados.
- Verificar los flujos generales y alternos que se generan en el programa diseñado.
- Verificar si el código diseñado compila en el programa especificado.

3. GLOSARIO DE TÉRMINOS

- FSM: Máquina de estado finito.
- Compilar: Traducir con un compilador un programa en lenguaje de alto nivel a lenguaje de la máquina.

4. EXPLICACIÓN DEL PROTOCOLO APLICADO

Para iniciar se explica el protocolo de pruebas desarrollado, que consiste en 5 partes (6 si cuenta con FSM), que se presentan a continuación:

- **Diagrama de Caja Negra:**
  Se dibuja la caja negra en el que se colocan las entradas y salidas que participan en el sistema.
- **Máquina de estados finito:**
  Se dibuja la FSM correspondiente al programa, en donde se expliquen cada uno de los estados y sus cambios.
- **Diagrama y grafo de flujo general:**
  Se dibuja tanto el diagrama de flujo y el grafo de flujo para después poder hacer el listado de los posibles caminos que se pueden llegar a ejecutar.
- **Listado de los posibles flujos alternos y del programa principal:**
  Se listan los posibles caminos que se deben tener en cuenta cuando se ejecute el programa.
- **Valores de entrada y salida al ejecutar los caminos establecidos:**
  Se colocan los valores de entrada y de salida esperados en el momento de ejecutar los caminos que se listaron en el paso anterior.
- **Compilación del programa:**
  Se realiza la digitación del código deseado, y se compila para verificar si quedó bien escrito.

5. PROGRAMAS A REVISAR

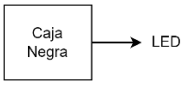
Inicialmente se listan cada uno de los programas a los cuales se le van a aplicar los protocolos de pruebas, se hizo de tal forma que se pueda verificar el funcionamiento de cada uno de los ejemplos propuestos en el kit de aprendizaje.

- Uso de LEDs: En este programa se declaran los puertos de los LED como salida y se encienden y apagan haciendo uso de los *delay*.
- Botones: Este programa consiste en un control de encendido y apagado de LEDs mediante una máquina de estados finita.
- LEDs con *Timers*: Se realiza el encendido y apagado de LEDs controlado por medio de *timers*.
- Semáforo: Se sigue una secuencia de encendido y apagado de LEDs controlado por *timers*.
- Envío de datos: Se realiza el envío de datos haciendo uso de la USART.
- Recepción de datos: Se reciben datos haciendo uso de la USART.
- Lectura ADC Potenciómetro: Se leen el potenciómetro, y de acuerdo con el valor medido se encienden los LEDs determinando una escala.
- Lectura ADC Temperatura: Se lee el valor de la temperatura y se encienden LEDs determinando una escala.
- Semáforo PWM: Se replica el programa del semáforo, pero haciendo uso del LED RGB.
- Tonalidades RGB: Por medio del potenciómetro y de los botones, se controla la tonalidad del LED RGB, el valor de cada uno de los colores se muestra por medio del puerto serie.
- Control Motores: Se ajusta la velocidad de los motores DC por medio del potenciómetro, con los botones se elige el motor al que se le desea controlar la velocidad. El valor se visualiza por medio del puerto serie.

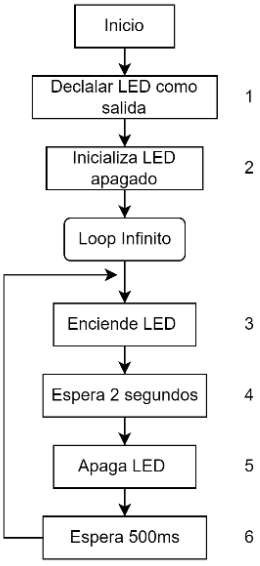
6. APLICACIÓN DEL PROTOCOLO

Una vez se conocen los programas a los cuales se les aplica el protocolo de pruebas, se inicia con el procedimiento:

1. Uso de LEDs:
    a. Diagrama de caja negra:



    b. Diagrama y grafo de flujo general:



c. Listado de Posibles Flujos:
Sólo hay un posible flujo: 1, 2, 3, 4, 5, 6

d. Valores de entrada y salida esperados:
No se tienen valores de entrada, las salidas esperadas son el encendido y apagado del LED.

e. Compilación del programa:
En el programa Atmel Studio:

```
Done building project "LedBlink.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

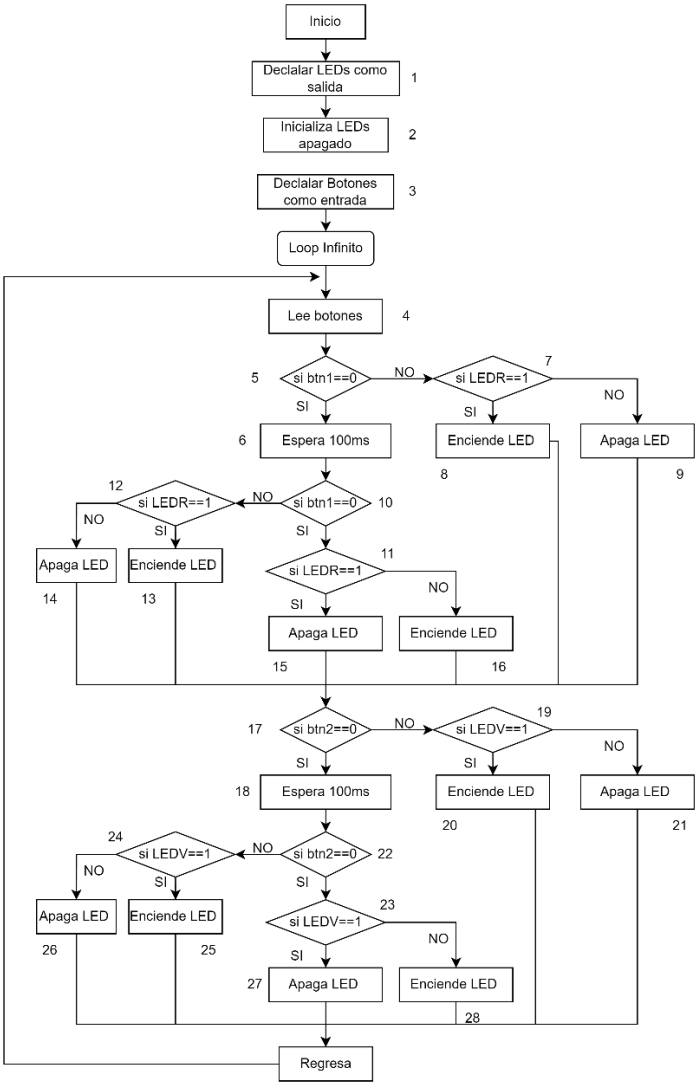
En el IDE de Arduino:

```
El Sketch usa 1656 bytes (0%) del espacio de almacenamiento de programa. El máximo es 253952 bytes.
Las variables Globales usan 9 bytes (0%) de la memoria dinámica, dejando 8183 bytes para las variables locales. El máximo es 8192 bytes.
```

2. Botones:

a. Diagrama de caja negra:

b. Diagrama y grafo de flujo general:



c. Listado de Posibles Flujos:
Flujo1: 1, 2, 3, 4, 5, 6, 10, 11, 15, 17, 18, 22, 23, 27
Flujo2: 1, 2, 3, 4, 5, 6, 10, 11, 16, 17, 18, 22, 23, 28
Flujo3: 1, 2, 3, 4, 5, 6, 10, 12, 13, 17, 18, 22, 24, 25
Flujo4: 1, 2, 3, 4, 5, 6, 10, 12, 14, 17, 18, 22, 24, 26
Flujo5: 1, 2, 3, 4, 5, 7, 8, 17, 19, 20
Flujo6: 1, 2, 3, 4, 5, 7, 9, 17, 19, 21

d. Valores de entrada y salida esperados:
Flujo1: Boton1=0, Boton1=0, Apaga LED Rojo, Boton2=0, Boton2=0, Apaga LED Verde.
Flujo2: Boton1=0, Boton1=0, Enciende LED Rojo, Boton2=0, Boton2=0, Enciende LED Verde.
Flujo3: Boton1=0, Boton1=1, Enciende LED Rojo, Boton2=0, Boton2=1, Enciende LED Verde.
Flujo4: Boton1=0, Boton1=1, Apaga LED Rojo, Boton2=0, Boton2=1, Apaga LED Verde.
Flujo5: Boton1=1, Enciende LED Rojo, Boton2=1, Enciende LED Verde.
Flujo6: Boton1=1, Apaga LED Rojo, Boton2=1, Apaga LED Verde.

e. Compilación del programa:
En el programa Atmel Studio:

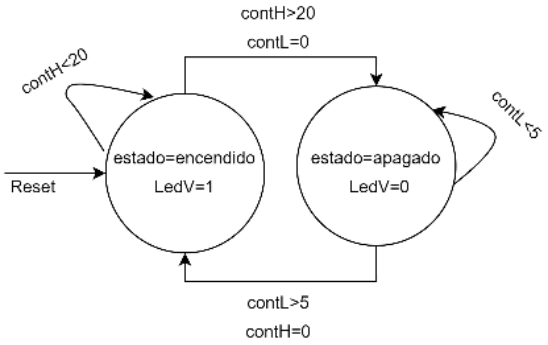
```
Done building target "Build" in project "Botones.cproj".
Done building project "Botones.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

En el IDE de Arduino:

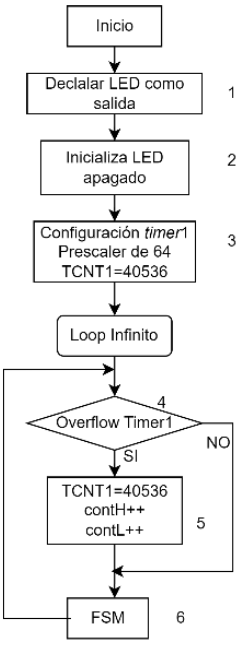
```
El Sketch usa 1656 bytes (0%) del espacio de almacenamiento de programa. El máximo es 253952 bytes.
Las variables Globales usan 9 bytes (0%) de la memoria dinámica, dejando 8183 bytes para las variables locales. El máximo es 8192 bytes.
```

3. LEDs con timers:
   a. Diagrama de caja negra:

   b. Máquina de estados finita:



d. Diagrama y grafo de flujo general:



e. Listado de Posibles Flujos:
Flujo1: 1, 2, 3, 4, 5, 6
Flujo2: 1, 2, 3, 4, 6

f. Valores de entrada y salida esperados:
Flujo1: Incrementa las variables de conteo.
Flujo1 – FSM estado encendido: Si cumple contH=20, Apaga LED.
Flujo1 – FSM estado apagado: Si cumple contL=5, Enciende LED.
Flujo2: Variables de conteo permanecen igual.
Flujo2 – FSM estado encendido: Enciende LED.
Flujo2 – FSM estado apagado: Apaga LED.

g. Compilación del programa:

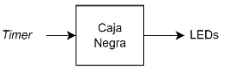
```
Done building target "Build" in project "LedBlink.cproj".
Done building project "LedBlink.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

En el IDE de Arduino:

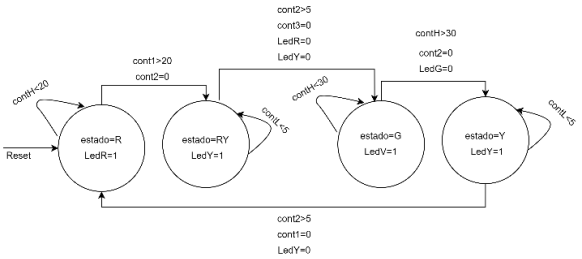
```
El Sketch usa 1656 bytes (0%) del espacio de almacenamiento de programa. El máximo es 253952 bytes.
Las variables Globales usan 9 bytes (0%) de la memoria dinámica, dejando 8183 bytes para las variables locales. El máximo es 8192 bytes.
```

4. Semáforo:

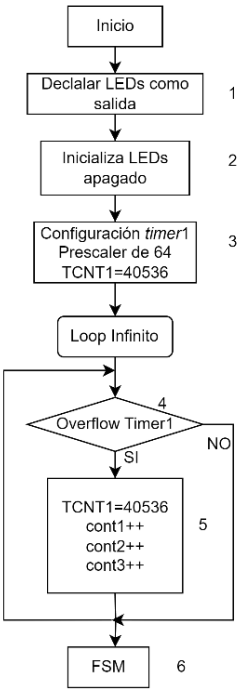
a. Diagrama de caja negra:



b. Máquina de estados finita:



c. Diagrama y grafo de flujo general:



d. Listado de Posibles Flujos:
Flujo1: 1, 2, 3, 4, 5, 6
Flujo2: 1, 2, 3, 4, 6

e. Valores de entrada y salida esperados:
Flujo1: Incrementa las variables de conteo.
Flujo1 – FSM estado R: Si cumple cont1=20, Enciende LED Amarillo.
Flujo1 – FSM estado RY: Si cumple cont2=5, Enciende LED Verde, Apaga LEDs Rojo y Amarillo.
Flujo1 – FSM estado G: Si cumple cont3=30, Enciende LED Amarillo, Apaga LED Verde
Flujo1 – FSM estado Y: Si cumple cont2=5, Enciende LED Rojo, Apaga LED Amarillo.
Flujo2: Variables de conteo permanecen igual.
Flujo2 – FSM estado R: Enciende LED Rojo.
Flujo2 – FSM estado RY: Enciende LED Amarillo.
Flujo2 – FSM estado G: Enciende LED Verde.
Flujo2 – FSM estado Y: Enciende LED Amarillo.

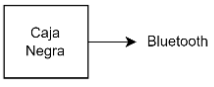
f. Compilación del programa:

```
Done building target "Build" in project "Semaforo.cproj".
Done building project "Semaforo.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

En el IDE de Arduino:

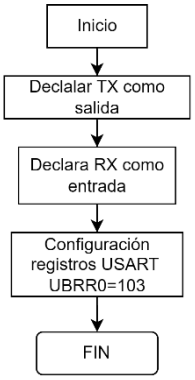
```
El Sketch usa 1656 bytes (0%) del espacio de almacenamiento de programa. El máximo es 253952 bytes.
Las variables Globales usan 9 bytes (0%) de la memoria dinámica, dejando 8183 bytes para las variables locales. El máximo es 8192 bytes.
```

5. Envío de datos:
   a. Diagrama de caja negra:

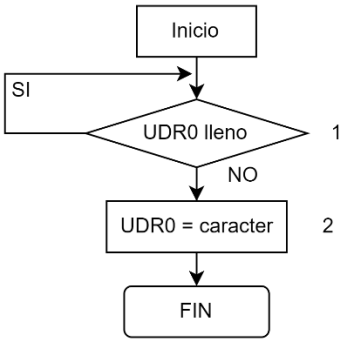


b. Diagrama y grafo de flujo general:

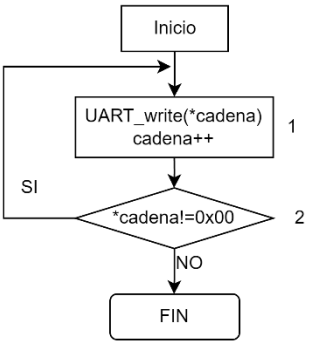
Función UART_init:

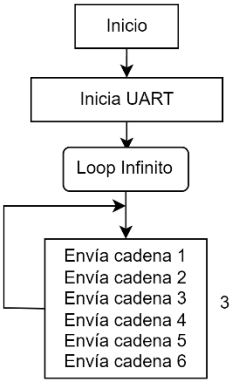


Función UART_write:



Función UART_write_txt:





c. Listado de Posibles Flujos:
Flujo general: 1, 2, 3
En el bloque 1, se ejecuta la función UART_init, después en el bloque 3, en enviar cadena se llama la función UART_write_txt, que llama a la función UART_write.

d. Valores de entrada y salida esperados:
Los valores de salida son los siguientes:
“Primero” , salto de línea, “Segundo”, salto de línea, “Hola Mundo”, salto de línea.

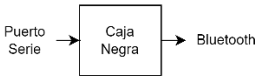
e. Compilación del programa:

```
Done building target "Build" in project "Envio.cproj".
Done building project "Envio.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

En el IDE de Arduino:

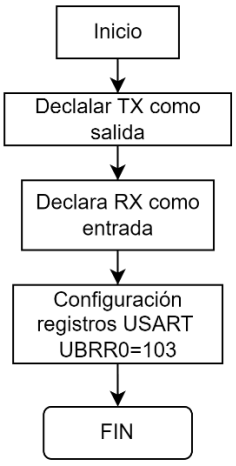
```
El Sketch usa 1656 bytes (0%) del espacio de almacenamiento de programa. El máximo es 253952 bytes.
Las variables Globales usan 9 bytes (0%) de la memoria dinámica, dejando 8183 bytes para las variables locales. El máximo es 8192 bytes.
```

6. Recepción de datos:
   a. Diagrama de caja negra:

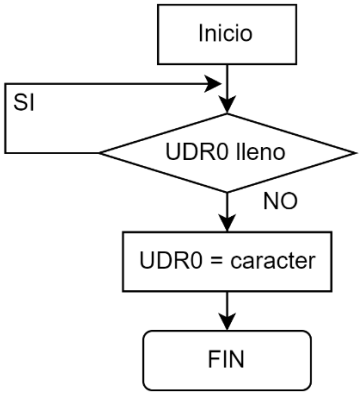


   b. Diagrama y grafo de flujo general:

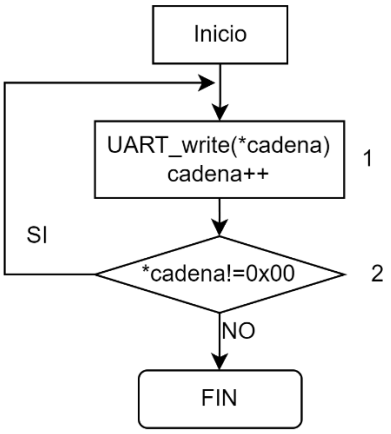
Función UART_init:



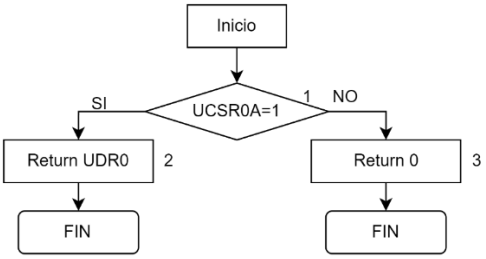
Función UART_write:



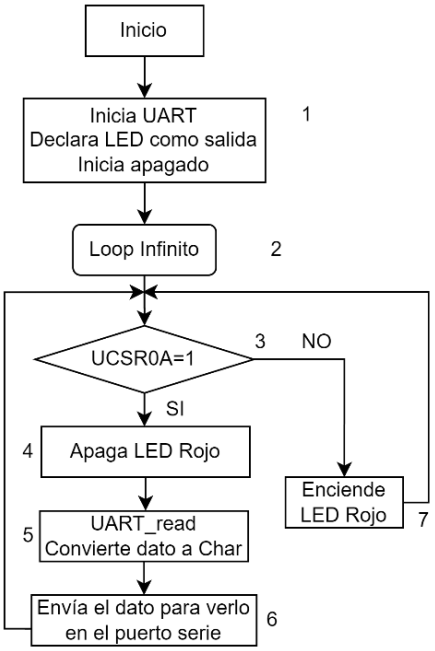
Función UART_write_txt:



Función UART_read:



Flujo general



c. Listado de Posibles Flujos:
Flujo 1: 1, 2, 3, 4, 5, 6

Flujo 2: 1, 2, 3, 7

d. Valores de entrada y salida esperados:
Flujo 1: Se inicia la UART, inicia el led rojo apagado, al escribir algo en el puerto serie, el dato es mostrado en la salida del puerto serie.
Flujo 2: Se inicia la UART, inicia el LED rojo apagado, mientras no se escriba nada, el LED rojo permanece encendido.

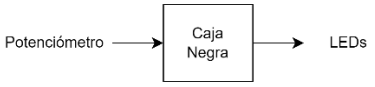
e. Compilación del programa:

```
Done building target "Build" in project "Recepcion.cproj".
Done building project "Recepcion.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

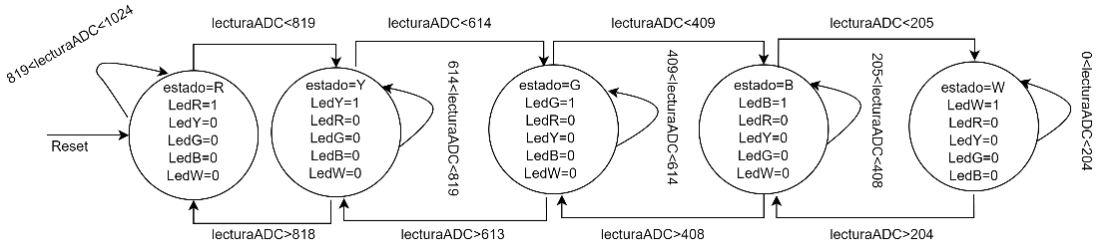
En el IDE de Arduino:

```
El Sketch usa 1656 bytes (0%) del espacio de almacenamiento de programa. El máximo es 253952 bytes.
Las variables Globales usan 9 bytes (0%) de la memoria dinámica, dejando 8183 bytes para las variables locales. El máximo es 8192 bytes.
```

7. Lectura Potenciómetro:
   a. Diagrama de caja negra:

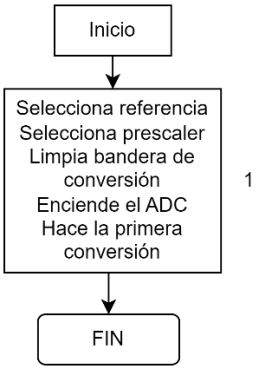


   b. Máquina de estados finita:

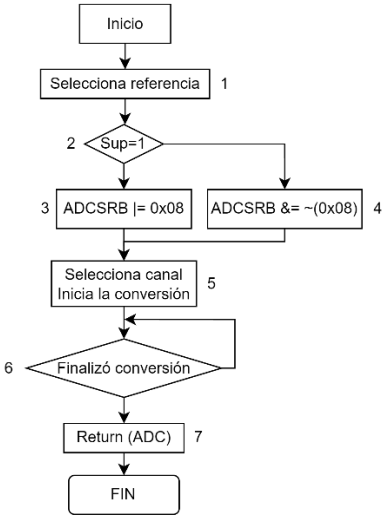


   c. Diagrama y grafo de flujo general:

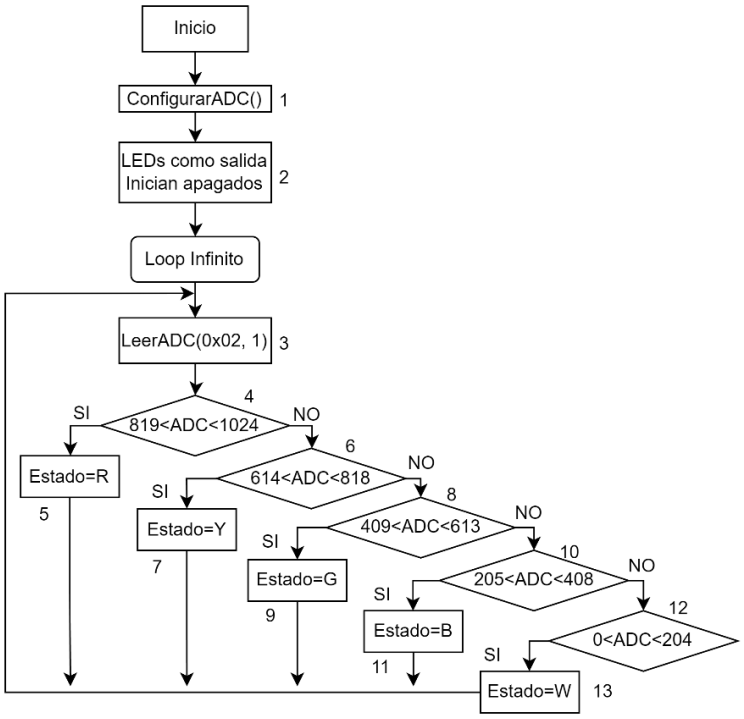
ConfigurarADC()



LeerADC



Flujo General



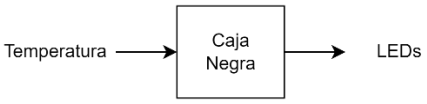
d. Listado de Posibles Flujos:
   Flujo 1: 1, 2, 3, 4, 5
   Flujo 2: 1, 2, 3, 4, 6, 7
   Flujo 3: 1, 2, 3, 4, 6, 8, 9
   Flujo 4: 1, 2, 3, 4, 6, 8, 10, 11
   Flujo 5: 1, 2, 3, 4, 6, 8, 10, 12, 13
e. Valores de entrada y salida esperados:
   Flujo1: El valor leído por el potenciómetro está entre los valores de 1024 y 819, se enciende el LED Rojo .
   Flujo2: El valor leído por el potenciómetro está entre los valores de 818 y 614, se enciende el LED Amarillo.
   Flujo3: El valor leído por el potenciómetro está entre los valores de 613 y 409, se enciende el LED Verde.
   Flujo4: El valor leído por el potenciómetro está entre los valores de 408 y 205, se enciende el LED Azul.
   Flujo5: El valor leído por el potenciómetro está entre los valores de 204 y 0, se enciende el LED Blanco.
f. Compilación del programa:

```
Done building target "Build" in project "Potenciometro.cproj".
Done building project "Potenciometro.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

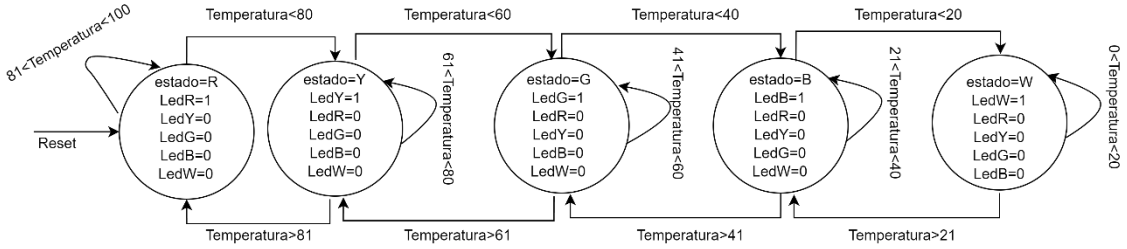
En el IDE de Arduino:

```
El Sketch usa 1656 bytes (0%) del espacio de almacenamiento de programa. El máximo es 253952 bytes.
Las variables Globales usan 9 bytes (0%) de la memoria dinámica, dejando 8183 bytes para las variables locales. El máximo es 8192 bytes.
```

8. Lectura Temperatura:
   a. Diagrama de caja negra:

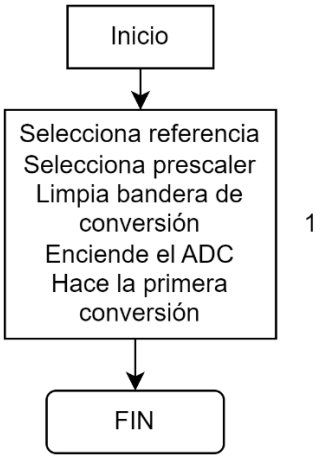


   b. Máquina de estados finita:

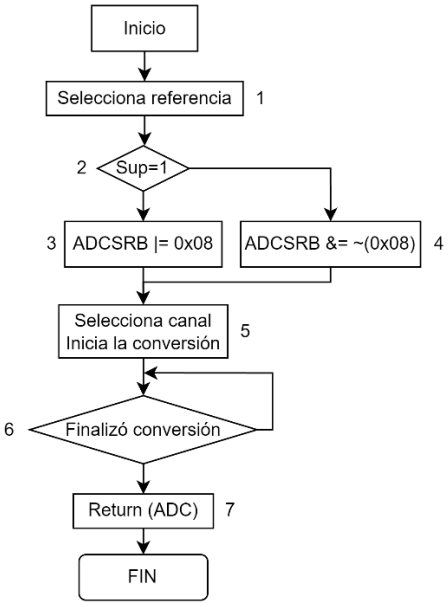


   c. Diagrama y grafo de flujo general:

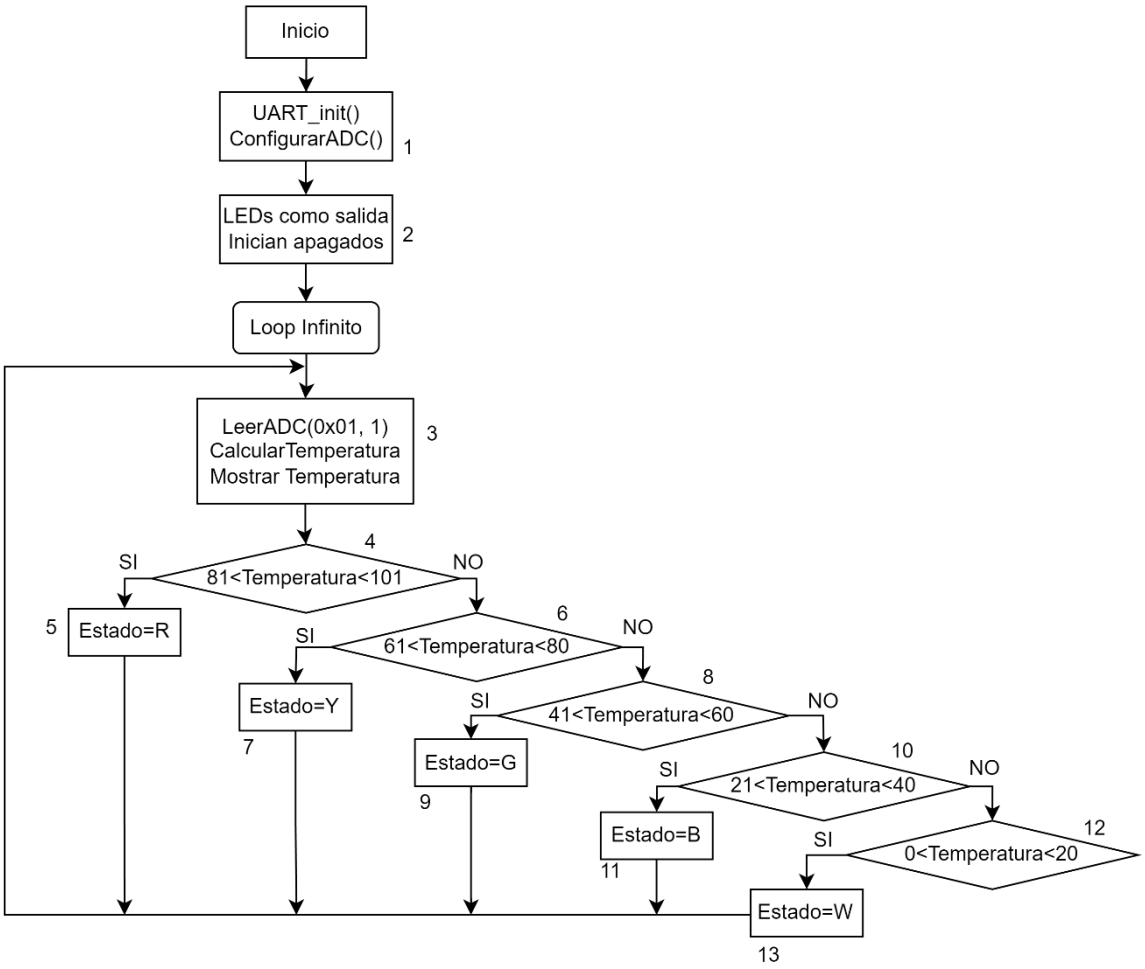
ConfigurarADC:



LeerADC:



Flujo General



d. Listado de Posibles Flujos:
   Flujo 1: 1, 2, 3, 4, 5
   Flujo 2: 1, 2, 3, 4, 6, 7
   Flujo 3: 1, 2, 3, 4, 6, 8, 9
   Flujo 4: 1, 2, 3, 4, 6, 8, 10, 11
   Flujo 5: 1, 2, 3, 4, 6, 8, 10, 12, 13

e. Valores de entrada y salida esperados:
   Flujo1: La Temperatura está entre los valores de 100 y 81, se enciende el LED Rojo.
   Flujo2: La Temperatura está entre los valores de 80 y 61, se enciende el LED Amarillo.
   Flujo3: La Temperatura está entre los valores de 60 y 41, se enciende el LED Verde.
   Flujo4: La Temperatura está entre los valores de 40 y 21, se enciende el LED Azul.
   Flujo5: La Temperatura está entre los valores de 20 y 0, se enciende el LED Blanco.

f. Compilación del programa:

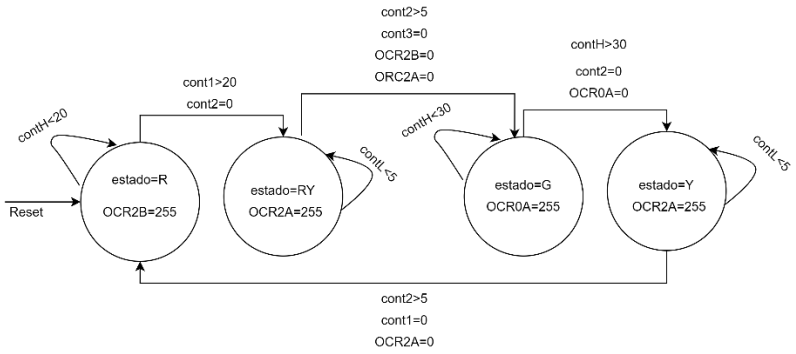
```
Done building project "Temperatura.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

En el IDE de Arduino:

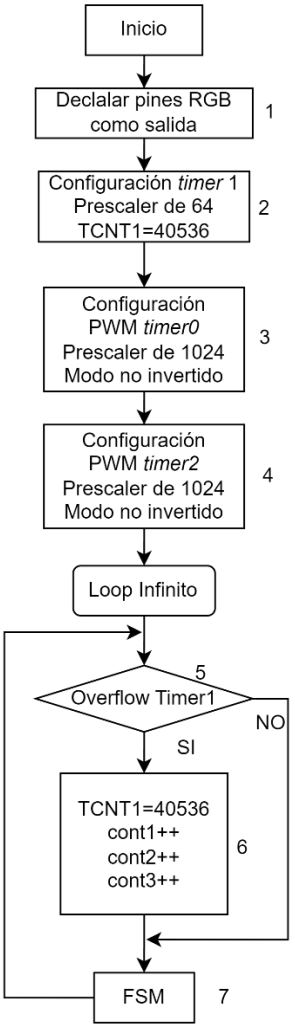
```
El Sketch usa 1656 bytes (0%) del espacio de almacenamiento de programa. El máximo es 253952 bytes.
Las variables Globales usan 9 bytes (0%) de la memoria dinámica, dejando 8183 bytes para las variables locales. El máximo es 8192 bytes.
```

9. Semáforo PWM:
    a. Diagrama de caja negra:

    b. Máquina de estados finita:



    c. Diagrama y grafo de flujo general:



d. Listado de Posibles Flujos:
Flujo1: 1, 2, 3, 4, 5, 6, 7
Flujo2: 1, 2, 3, 4, 5, 7

e. Valores de entrada y salida esperados:
Flujo1: Incrementa las variables de conteo.
Flujo1 – FSM estado R: Si cumple cont1=20, Activa PWM Azul a 255.
Flujo1 – FSM estado RY: Si cumple cont2=5, Activa PWM Verde a 255, Activa PWM Rojo a 0, y PWM Azul a 0.
Flujo1 – FSM estado G: Si cumple cont3=30, Activa PWM Azul a 255, Activa PWM Verde a 0.
Flujo1 – FSM estado Y: Si cumple cont2=5, Activa PWM Rojo a 255, Activa PWM Azul a 0.
Flujo2: Variables de conteo permanecen igual.
Flujo2 – FSM estado R: Activa PWM Rojo a 255.
Flujo2 – FSM estado RY: Activa PWM Rojo a 255 y PWM Azul a 255.
Flujo2 – FSM estado G: Activa PWM Verde a 255.
Flujo2 – FSM estado Y: Activa PWM Azul a 255.

f. Compilación del programa:

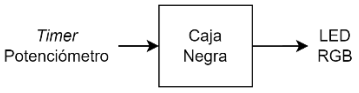
```
Done building target "Build" in project "Semaforo PWM.cproj".
Done building project "Semaforo PWM.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

En el IDE de Arduino:

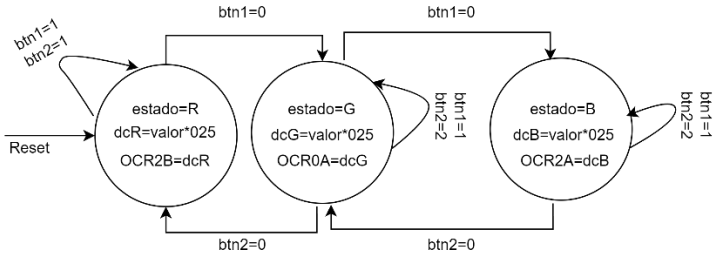
```
El Sketch usa 1656 bytes (0%) del espacio de almacenamiento de programa. El máximo es 253952 bytes.
Las variables Globales usan 9 bytes (0%) de la memoria dinámica, dejando 8183 bytes para las variables locales. El máximo es 8192 bytes.
```

10. Control Tonalidades:

a. Diagrama de caja negra:

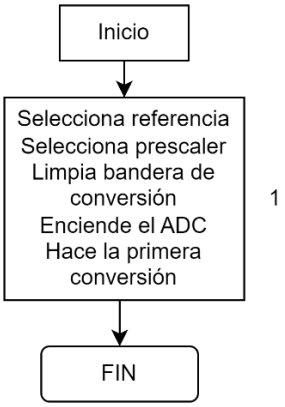


b. Máquina de estados finita:

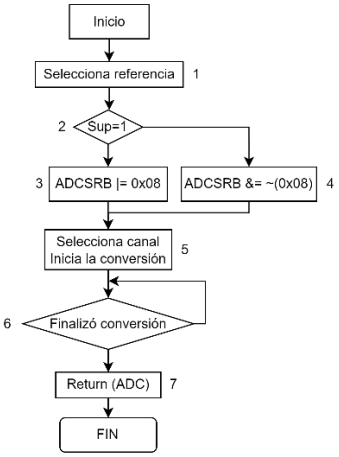


c. Diagrama y grafo de flujo general:

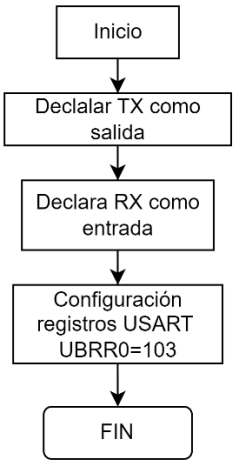
Configurar ADC:



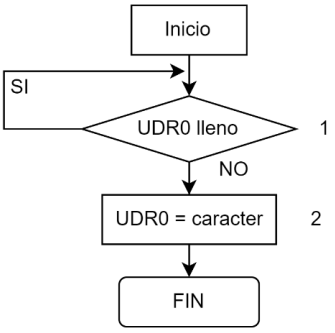
Leer ADC:



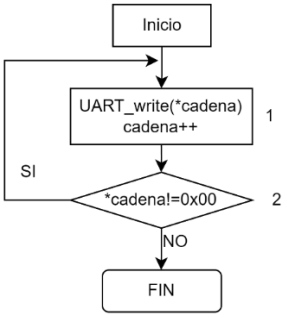
UART_init:



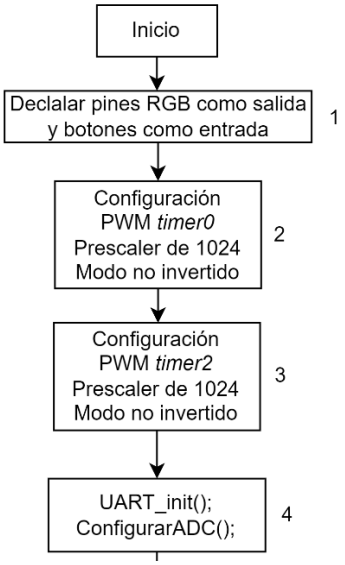
UART_write:

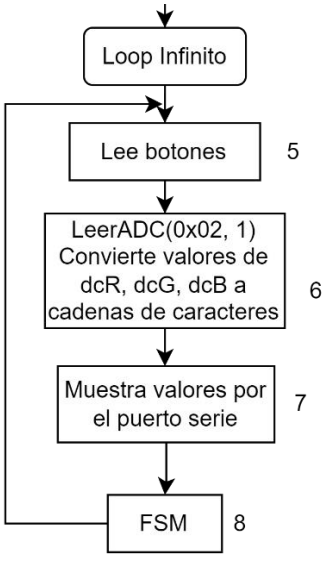


UART_write_txt:



Flujo General:





d. Listado de Posibles Flujos:
   Flujo1: 1, 2, 3, 4, 5, 6, 7, 8
e. Valores de entrada y salida esperados:
   Flujo1: Varía la FSM.
   Flujo1 – FSM estado R: Si no presiona botones, Se varía la tonalidad de Rojo.

Flujo1 – FSM estado G: Si no presiona botones, Se varía la tonalidad de Verde.
Flujo1 – FSM estado B: Si no presiona botones, Se varía la tonalidad de Azul.

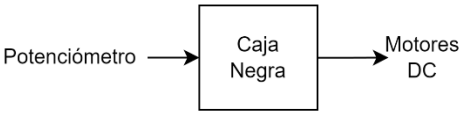
f. Compilación del programa:

```
Done building target "Build" in project "Tonalidades.cproj".
Done building project "Tonalidades.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

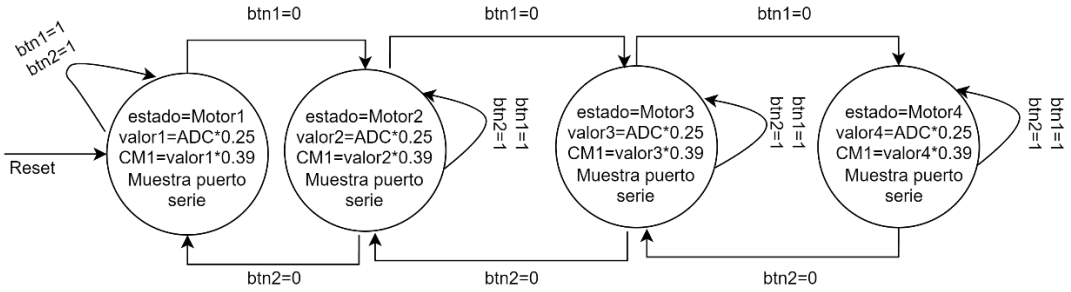
En el IDE de Arduino:

```
El Sketch usa 1656 bytes (0%) del espacio de almacenamiento de programa. El máximo es 253952 bytes.
Las variables Globales usan 9 bytes (0%) de la memoria dinámica, dejando 8183 bytes para las variables locales. El máximo es 8192 bytes.
```

11. Control Motores:
    a. Diagrama de caja negra:

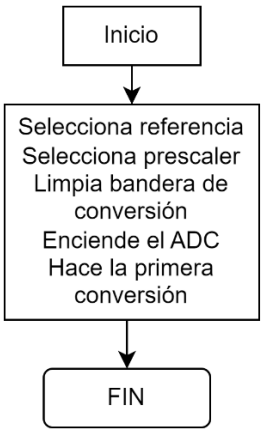


    b. Máquina de estados finita:

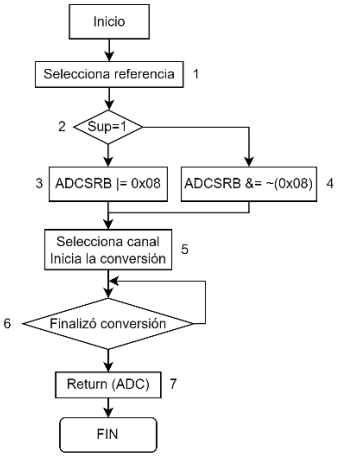


    c. Diagrama y grafo de flujo general:

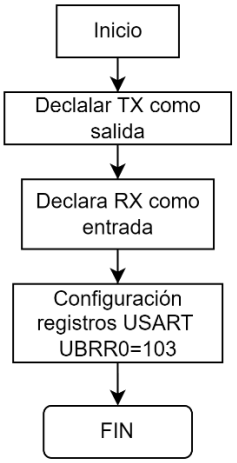
ConfigurarADC:



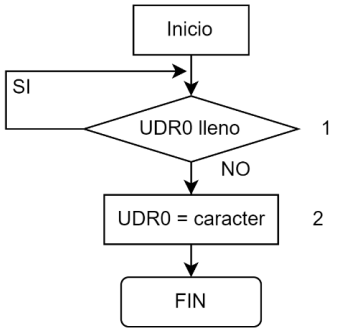
LeerAdDC:



UART_init:



UART_write:



UART_write_txt:

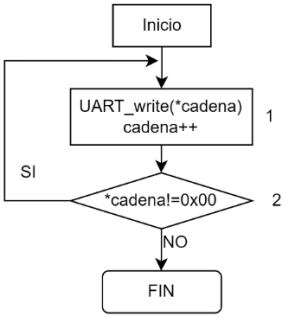
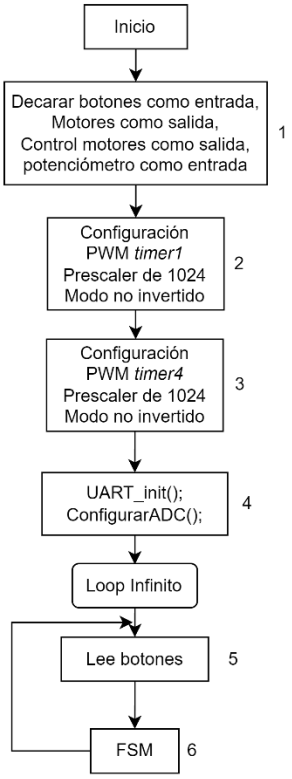


Diagrama de Flujo General



d. Listado de Posibles Flujos:
   Flujo único: 1, 2, 3, 4, 5, 6
e. Valores de entrada y salida esperados:
   Flujo1: Varía la FSM.
   Flujo1 – FSM estado Motor1: Si no presiona botones, Se varía la velocidad del motor1, y se muestra el valor por el puerto serie.
   Flujo1 – FSM estado Motor2: Si no presiona botones, Se varía la velocidad del motor2, y se muestra el valor por el puerto serie.
   Flujo1 – FSM estado Motor3: Si no presiona botones, Se varía la velocidad del motor3, y se muestra el valor por el puerto serie.
   Flujo1 – FSM estado Motor4: Si no presiona botones, Se varía la velocidad del motor4, y se muestra el valor por el puerto serie.

f. Compilación del programa:
En el IDE de Arduino:

```
El Sketch usa 1656 bytes (0%) del espacio de almacenamiento de programa. El máximo es 253952 bytes.
Las variables Globales usan 9 bytes (0%) de la memoria dinámica, dejando 8183 bytes para las variables locales. El máximo es 8192 bytes.
```

7. CONCLUSIONES

- Con la implementación del protocolo de pruebas de software de las guías de aprendizaje se pueden prever distintas fallas que se generan cuando se está codificando, ya que presenta una guía para entender mejor el código.
- Se cumplieron los objetivos propuestos a l inicio del documento.