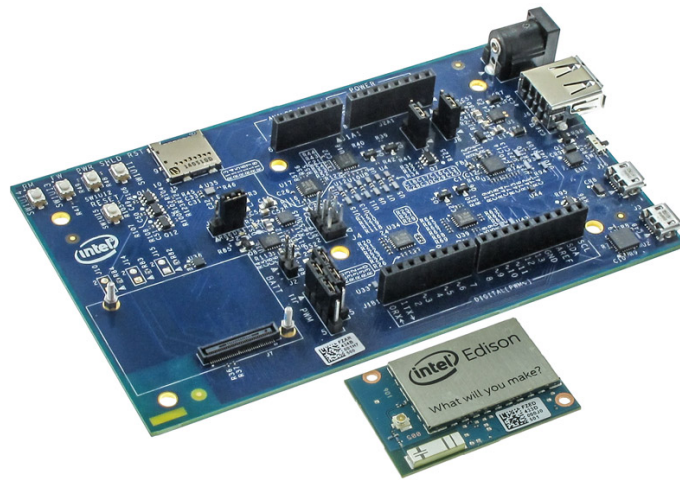


ALDO NÚÑEZ TOVAR

INICIANDO CON EDISON



FREE BOOKS

Índice general

| | | |
|-------|--|----|
| 1 | <i>Intel Edison</i> | 7 |
| 1.1 | <i>Arranque</i> | 8 |
| 1.2 | <i>Comunicación serie</i> | 8 |
| 1.3 | <i>Configuración de la conexión inalámbrica</i> | 9 |
| 1.3.1 | <i>Localización del IP</i> | 10 |
| 1.4 | <i>Configuración de repositorios e instalación de paquetes</i> | 10 |
| 2 | <i>Ambientes de desarrollo y programación</i> | 13 |
| 2.1 | <i>Arduino IDE</i> | 13 |
| 2.2 | <i>Intel XDK</i> | 14 |
| 2.3 | <i>Intel System Studio</i> | 14 |
| 2.4 | <i>Intel System Studio para microcontroladores</i> | 15 |
| 2.5 | <i>Desarrollo en la propia tarjeta Edison</i> | 15 |
| 3 | <i>Biblioteca MRAA y herramientas</i> | 17 |
| 3.1 | <i>Qué es la biblioteca MRAA</i> | 17 |
| 3.2 | <i>Herramientas</i> | 17 |
| 3.3 | <i>Editores</i> | 17 |
| 3.4 | <i>Compiladores e intérpretes</i> | 18 |
| 3.5 | <i>Cuenta de usuario para editar y compilar</i> | 18 |
| 4 | <i>Programación de los puertos de entrada y salida</i> | 19 |

4.0.1 *Salidas digitales* 19

4.0.2 *Entradas digitales* 27

4.0.3 *PWM* 29

4.0.4 *Entradas analógicas* 35

5 *Ejemplos* 39

5.1 *Servo* 39

Glosario 43

Siglas 45

Este tutorial va dirigido a todos aquellos desarrolladores que estén interesados en empezar a utilizar la tarjeta Edison de Intel como plataforma para la implementación de sus proyectos.

A pesar de que existen diferentes ambientes de desarrollo, tal como el IDE de Arduino, el enfoque escogido ha sido la programación directa en la tarjeta Edison. Esto nos impone un lector con conocimientos básicos de programación en cualquier lenguaje.

Los programas y ejemplos están escritos en los lenguajes C, C++, Python y JavaScript. Se ha creado un repositorio en el sitio github, a través del cual se puede acceder a este tutorial y a todo el código que se incluye en este documento. El enlace es el siguiente:

<https://github.com/lizard20/edison>.

Todo el desarrollo de este tutorial, incluso la elaboración de este documento, se ha realizado en el sistema operativo Linux.

Para cualquier comentario o sugerencia pueden dirigirse al autor a través de su correo electrónico:

Aldo Núñez Tovar
anunez20@gmail.com



Este documento se distribuye bajo una licencia Creative Common. Reconocimiento – NoComercial – SinObraDerivada (by-nc-nd): No se permite un uso comercial de la obra original ni la generación de obras derivadas.

1 Intel Edison

La tarjeta Edison fue presentada por Intel en el año 2014. Tiene un procesador Atom de 32 bits de doble núcleo y una frecuencia de 500 MHz. Además, tiene un procesador Quark a 100 MHz, que se encarga de las operaciones de entrada y salida. Tiene una memoria flash de 4GB y memoria RAM de 1GB. La memoria flash viene pre programada con una versión del sistema operativo Linux, llamado proyecto [Yocto](#).

Para la comunicación inalámbrica cuenta con WiFi y [Bluetooth](#). Además, tiene puertos para la comunicación serie del tipo: [I2C](#), [SPI](#) y [UART](#).

Tiene salidas y entradas digitales, salidas [PWM](#) y entradas para leer señales analógicas.

Características principales:

- Procesadores
 - Intel dual-core Atom a 500MHz
 - Coprocesador Quark a 100 MHz
- Memoria
 - 4GB de memoria flash
 - 1GB de memoria RAM
- Sistema operativo Yocto
- Conectividad
 - WiFi
 - BlueTooth 4.0
- Comunicación serie
 - [I2C](#)
 - [SPI](#)
 - [UART](#)
- GPIO's
 - E/S digitales
 - 6 entradas analógicas [ADC](#)
 - 4 salidas [PWM](#)

1.1 Arranque

La tarjeta incluye el módulo Edison y el Kit Edison para Arduino.

Opción 1:

En esta opción necesitamos 2 cables usb tipo micro usb.

1. Mueva el switch1 en la dirección del conector micro usb J16,
- Figura 1.1
2. Conecte su computadora a través del conector micro usb J16
3. Si la tarjeta Edison arrancó correctamente, debe encenderse el led DS1
4. Ahora, para establecer la comunicación serie conecte su computadora a la tarjeta Edison, a través del conector micro usb J3

Opción 2:

En esta opción necesitamos una fuente de poder de 7-15 volts de CD y 1500 mA, aunque una fuente de 5 volts también funciona, y un cable usb tipo micro usb.

1. Mueva el switch1 en la dirección del conector micro usb J16
2. Conecte la fuente de alimentación de 7.5 - 15 volts a J1
3. Si la tarjeta Edison arrancó correctamente, debe encenderse el led DS1
4. Ahora, para establecer la comunicación serie conecte su computadora a la tarjeta Edison, a través del conector micro usb J3



Figura 1.1: Tarjeta de desarrollo Intel Edison Arduino

1.2 Comunicación serie

1. En su computadora abra una consola y busque en el subdirectorio /dev el archivo ttyUSB0. Para esto, ejecute el siguiente comando:

```
$ ls -l /dev | grep ttyUSB0
```


2. Para conectarse a la tarjeta Edison, ejecute en su computadora, como súper usuario, el comando `screen`:

```
$ sudo screen /dev/ttyUSB0 115200
```

3. Una vez establecida la comunicación le pedirá el nombre de usuario. Introduzca `root`

```
edison login: root
```

4. Ahora debe aparecer en consola:

```
root@edison:~#
```

1.3 Configuración de la conexión inalámbrica

Una vez establecida la comunicación serial con la tarjeta Edison, vamos a configurar la conexión inalámbrica.

1. Ejecute:

```
root@edison:~# configure_edison --wifi
```

A continuación, aparecerá un conjunto de opciones y las redes inalámbricas disponibles. Seleccione la red inalámbrica correspondiente e introduzca la contraseña.

2. Si todo se realizó correctamente, antes de establecer comunicación a través de la red inalámbrica, debemos conocer el IP asignado a la tarjeta Edison. Para esto, ejecute el comando `ifconfig`:

```
root@edison:~# ifconfig wlan0 | grep "inet_addr:"
```

este comando debe imprimir en consola algo parecido al siguiente resultado:

```
inet addr:192.168.1.66 Bcast:192.168.1.255 Mask:255.255.255.0
```

El IP asignado es: 192.168.1.66.

La asignación del IP, por defecto, es dinámica. Es decir, que cada vez que arranque la tarjeta Edison el valor del IP cambiará.

3. Ahora ya podemos conectarnos a través de la red inalámbrica. Para esto vamos a ejecutar el comando `ssh`. En su computadora abra una consola y ejecute:

```
$ ssh root@192.168.1.66
```

en consola se desplegará:

```
root@edison:~#
```

Ahora ya establecimos la conexión inalámbrica.

1.3.1 Localización del IP

Para localizar el IP asignado a la tarjeta Edison puede instalar en su computadora el program Avahi daemon. En Debian:

```
$ sudo apt-get install avahi-daemon
```

ejecute en su computadora

```
$ avahi-browse -a --resolve
```

Otra alternativa para localizar el IP asignado a la tarjeta Edison es la herramienta: "Angry IP Scanner". Ésta se puede bajar del sitio: <http://angryip.org/>

1.4 Configuración de repositorios e instalación de paquetes

El sistema operativo instalado en Edison está basado en el proyecto Yocto. La versión instalada es Poky 3.10.17.

Una vez establecida la conexión inalámbrica ya podemos instalar paquetes. Pero antes, debemos configurar los repositorios donde residen estos paquetes.

1. Verificamos si el archivo `base-feeds.conf` existe y desplegamos la información que contiene.

```
root@edison:~# cat /etc/opkg/base-feeds.conf
```

si se despliega la siguiente información:

```
src/gz all http://repo.opkg.net/edison/repo/all
src/gz edison http://repo.opkg.net/edison/repo/edison
src/gz core2-32http://repo.opkg.net/edison/repo/core2-32
```

entonces pasamos al punto 3.

2. Si el archivo no existe, entonces tenemos que crearlo y añadir las direcciones de los repositorios.

Si el sistema tiene instalado algún editor de texto, ya sea nano o vim, hacemos uso de cualquiera de estos para crear el archivo: `/etc/opkg/base-feeds.conf`, y añadimos la información de los repositorios mostrada en el punto 1.

Si no disponemos de un editor de texto, a través del comando `echo` podemos crear el archivo de repositorios `base-feeds.conf`. Para esto ejecute los siguientes comandos:

```
root@edison:~# echo "src/gz_all_http://repo.opkg.net/edison/repo/all" >> /etc/opkg/base-feeds.conf
```

```
root@edison:~# echo "src/gz_edison_http://repo.opkg.net/edison/repo/edison" >> /etc/opkg/base-feeds.conf
```

```
root@edison:~# echo "src/gz_core2-32_http://repo.opkg.net/edison/repo/core2-32" >> /etc/opkg/base-feeds.conf
```

Si todo se realizó de forma correcta verifique que el archivo exista y tenga la información de los repositorios.

3. El comando **opkg** se utiliza para la administración de paquetes. Y, cada vez que se modifique la lista de repositorios ésta debe actualizarse para que tenga efecto.

```
root@edison:~# opkg update
```

Si el comando anterior manda errores, hay que revisar los archivos que residen en el subdirectorio `/etc/opkg/`. Pues, a veces, existen repositorios duplicados.

Revise la información de los archivos de este subdirectorio y verifique.

En caso de que existan repositorios duplicados elimine o comente esa línea. Para comentar una línea anteponga el símbolo `#` al inicio de la línea en cuestión.

Si el sistema no marca errores, ya está disponible para la instalación de paquetes.

Para instalar un paquete, ejecute:

```
root@edison:~# opkg install <paquete>
```

Para remover un paquete, ejecute:

```
root@edison:~# opkg remove <paquete>
```

Para conocer las opciones del comando **opkg**, ejecute:

```
root@edison:~# opkg help
```


2 Ambientes de desarrollo y programación

Existen distintas opciones para programar la tarjeta Edison. Vamos a mencionar las que existen actualmente.

2.1 *Arduino IDE*



Figura 2.1: Arduino IDE

Está basado en la popular plataforma Arduino. Para programar se debe instalar el ambiente de desarrollo de Arduino -Arduino IDE-.

El language de programación se llama processing, un dialecto de C/C++. Fácil de usar, aunque muy básico. Los programas o sketches, se ejecutan sobre un emulador del microcontrolador de Arduino. No utiliza todos los recursos de la plataforma Edison. No es recomendable para un desarrollo demandante.

Para mayor información acerca de la instalación del IDE, consulte el siguiente enlace:

<https://software.intel.com/en-us/get-started-arduino-install>

2.4 Intel System Studio para microcontroladores



Figura 2.4: Intel System Studio para Microcontroladores

Está basado en Eclipse y hasta el momento es la única herramienta disponible para interactuar con el microcontrolador Quark. Se puede programar en C/C++.

Para la instalación e información consulte el siguiente enlace:

<https://software.intel.com/es-es/intel-system-studio-microcontrollers>

2.5 Desarrollo en la propia tarjeta Edison



Figura 2.5: Consola

Esta es la opción más versátil y utilizada por los desarrolladores, no necesita instalar ningún programa o ambiente de desarrollo en su computadora. Simplemente requiere conectarse a través de la conexión inalámbrica o del puerto usb.

Se necesitan conocimientos básicos del sistema operativo Linux. Se puede programar en C/C++, Python, Java y JavaScript.

Puede utilizar las bibliotecas MRAA y UPM, para interactuar con los puertos y sensores. Estas bibliotecas existen también para otras plataformas, de tal manera que el código generado puede ser portable.

3 Biblioteca MRAA y herramientas

3.1 Qué es la biblioteca MRAA

La biblioteca MRAA, <http://iotdk.intel.com/docs/master/mraa/>, es un desarrollo de código abierto implementado en C/C++ con enlaces a Python, Javascript y Java, y sirve como interfaz para el manejo de E/S en Edison y otras plataformas.

Esta biblioteca no lo ata a ningún hardware específico lo cual nos permite crear código portable.

La biblioteca MRAA viene instalada en el sistema Yocto.

Puede consultar los ejemplos para el uso de esta biblioteca, en el siguiente subdirectorio de su tarjeta Edison: `/usr/share/mraa/examples/`

3.2 Herramientas

Antes de empezar a programar haciendo uso de la biblioteca MRAA, debemos tener acceso a un editor de texto, a los compiladores y a los intérpretes.

Antes de instalar cualquier paquete debemos tener conexión con la tarjeta Edison desde nuestra computadora, ya sea a través del puerto usb o de la conexión inalámbrica. De lo contrario, vea las secciones 1.2 y 1.3.

3.3 Editores

Para editar nuestros programas vamos a instalar un editor de texto. Podemos escoger entre nano o vim

```
root@edison:~# opkg install nano
```

o

```
root@edison:~# opkg install vim
```

3.4 *Compiladores e intérpretes*

La tarjeta Edison tiene instalados los compiladores de C y C++. Estos compiladores son libres, del proyecto [GNU](#). La versión instalada es la 4.9.1

Para verificar si se ha instalado el compilador de C, ejecute:

```
edison:~$ gcc --version
```

debe desplegarse un mensaje como el siguiente:

```
gcc (GCC) 4.9.1
Copyright (C) 2014 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
```

Si no obtiene este mensaje o, algo similar, puede proceder a instalar el compilador.

Como súper usuario, ejecute:

```
root@edison:~# opkg install gcc
```

De la misma manera, para verificar que el compilador de C++ ha sido instalado, ejecute:

```
edison:~$ g++ --version
```

Para verificar si tiene instalado el intérprete de Python, ejecute:

```
edison:~$ python --version
```

Finalmente, para verificar si tiene instalado el intérprete de JavaScript, Node, ejecute:

```
edison:~$ node --version
```

También se puede programar en Java, aunque la máquina virtual de Java no viene instalada.

3.5 *Cuenta de usuario para editar y compilar*

Es una buena práctica crear una cuenta de usuario para editar y compilar programas.

Para crear una nueva cuenta de usuario use el comando `adduser`:

```
root@edison:~# adduser <nombre-de-usuario>
```

Finalmente, es importante mencionar que para programar directamente en la tarjeta Edison, se requieren conocimientos básicos del sistema operativo Linux, así como de sus principales comandos.

Así también, conocer los comandos para compilar programas y el uso de intérpretes.

4 Programación de los puertos de entrada y salida

Los puertos de E/S de la tarjeta Edison son compatibles con la tarjeta Arduino Uno¹. Esta tarjeta cuenta con 20 puertos de E/S para las conexiones externas, distribuidos entre los headers J1A1, J1B1 y J2B1.

Se dispone de seis entradas analógicas, dispuestas en el header 'ANALOG IN' (J1A1), Figura 4.1.

Catorce entradas y salidas digitales en los headers 'DIGITAL (PWM~)' (J1B1 y J2B1), Figura 4.1.

¹ Consultar: Intel Edison Kit for Arduino

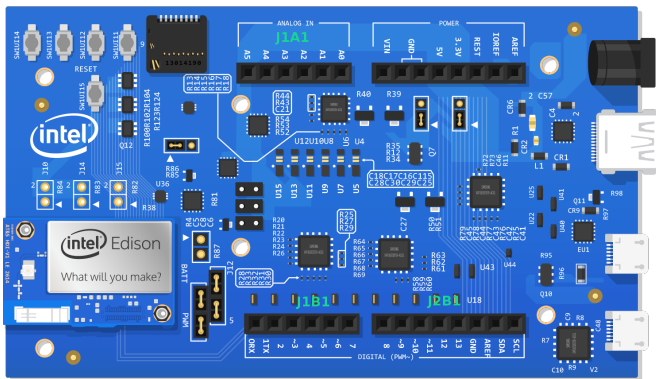


Figura 4.1: Headers de los puertos de entradas y salidas

4.0.1 Salidas digitales

Vamos a mostrar un programa sencillo que producirá el parpadeo de un led utilizando la biblioteca MRAA. Como salida usamos el puerto 13, que a su vez está conectada al led DS2, Figura 4.2.

En la línea 7, incluimos el encabezado de la biblioteca MRAA

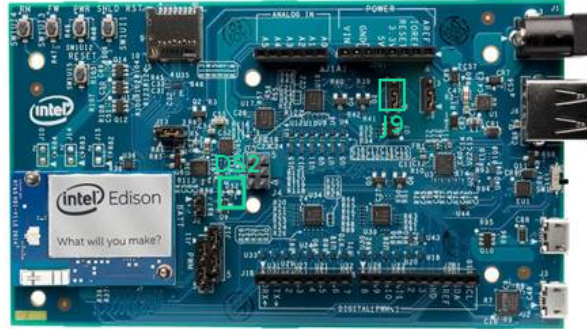
```
7 #include <mraa.h>
```

En la línea 13, se declara la variable led del tipo GPIO

```
13 mraa_gpio_context led;
```

En la línea 16, inicializamos la variable led con un número de puerto

Figura 4.2: Led DS2 y jumper J9



```
16 led = mraa_gpio_init ( 13 );
```

En la línea 19, establecemos la dirección del puerto, en este caso como salida

```
19 mraa_gpio_dir ( led, MRAA_GPIO_OUT );
```

En la línea 22, iniciamos un ciclo infinito while.

```
22 while ( 1 )
```

En las líneas 25 y 29, encendemos y apagamos el led, para ello escribimos un 1 o un 0 en el puerto correspondiente.

```
25 mraa_gpio_write ( led, 1 );
```

```
29 mraa_gpio_write ( led, 0 );
```

En las líneas 27 y 31 se incluyen pausas de 1 segundo.

```
27 sleep ( 1 );
```

Versión en C:

```

1  /*
2  ** Program: blink.c
3  ** Description: blink LED DS2
4  ** Author: Aldo Nunez
5  */
6
7  #include <mraa.h>
8
9  int
10 main ( void )
11 {
12     /* declare led variable as a gpio type */
13     mraa_gpio_context led;
14
15     /* initialize led with a pin number*/
16     led = mraa_gpio_init ( 13 );
17
18     /* set gpio direction to out */
19     mraa_gpio_dir ( led, MRAA_GPIO_OUT );
20
21     /* blink indefinitely until (Ctrl + c) */
22     while ( 1 )
23     {
24         /* turn on led */
25         mraa_gpio_write ( led, 1 );
26         /* wait for 1 second */
27         sleep ( 1 );
28         /* turn off led */
29         mraa_gpio_write ( led, 0 );
30         /* wait for 1 second */
31         sleep ( 1 );
32     }
33     /* close the gpio context */
34     mraa_gpio_close ( led );
35
36     return ( MRAA_SUCCESS );
37 }

```

Compilar:

```
$ gcc -o blink blink.c -Wall -lmraa
```

Ejecutar como súper usuario:

```
# ./blink
```

Hemos incluido en el repositorio de GitHub², versiones del programa blink, en C++, Python y JavaScript.

² <https://github.com/lizard20/edison>

Blink mejorado

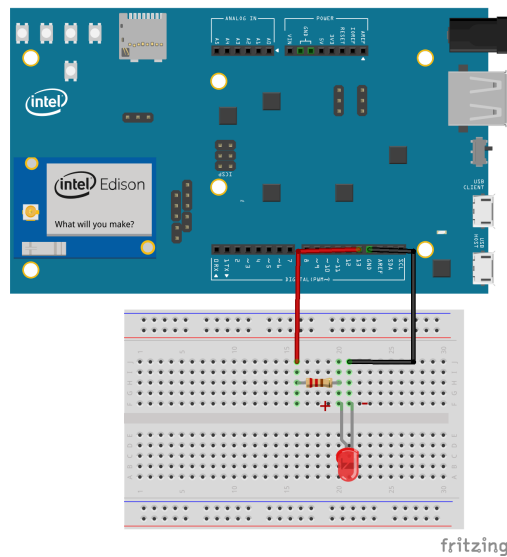


Figura 4.3: Led conectado al puerto 13

Vamos a dar un paso adelante y mejorar el programa blink. Las limitaciones que tiene el programa blink son:

- Cada vez que queremos escoger un puerto distinto, tenemos que compilar el programa nuevamente
 - Cuando decidimos terminar la ejecución de blink, el led puede terminar en cualquier estado: encendido o apagado
 - En el programa no se realiza ninguna verificación del valor retornado por las llamadas a funciones de la biblioteca MRAA
 - No finaliza el programa, a través de Ctrl + c, de forma controlada
- En el siguiente programa hemos resuelto estas limitaciones.

El programa puede controlar cualquier led que esté conectado a uno de los 14 puertos de los headers J1B1 y J2B2, Figura 4.1, sin necesidad de compilar el programa nuevamente.

Hemos incluido un procedimiento para la terminación controlada del programa, de tal manera que cuando se decide finalizar éste, el led siempre terminará en el estado de apagado.

Para empezar, vamos a conectar un led al puerto 13 a través de una resistencia para limitar la corriente.

Conexiones eléctricas:

Esta placa de expansión tiene disponibles en sus salidas voltajes de 3.3 volts y 5 volts. Para escoger uno de estos voltajes simplemente mueva el jumper J9, Figura 4.2.

Las corrientes máximas de salida son: si usamos 5 volts la máxima corriente será de 32 mA. Y, si seleccionamos 3.3 volts, la máxima corriente será de 24 mA.

Un led comercial puede soportar una corriente máxima aproxima-

da de 20 mA, vamos a proponer 15 mA. El led, cuando está polarizado de forma directa, tiene una caída de voltaje alrededor de 1.6 volts.

Cuando la salida se pone en alto y enciende el led, tenemos el circuito equivalente de la Figura 4.4. Este circuito nos sirve para calcular la resistencia necesaria que limite la corriente, y así, proteger el puerto de salida y el led.

Circuito de salida:

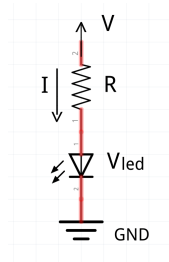


Figura 4.4: Circuito equivalente cuando la salida está en alto

Aplicamos la LVK (Ley de voltajes de Kirchhoff):

$$V = V_{led} + I \times R$$

despejamos R:

$$R = \frac{V - V_{led}}{I}$$

Sustituimos y obtenemos el valor de la resistencia:

$$R = \frac{5V - 1,6V}{15mA} = 226\Omega$$

Escojemos el valor comercial más cercano, 220 Ω .

Componentes:

- 1 led
- 1 resistencia de 220 Ω
- Alambres
- Placa de prueba (breadboard)

Versión en C:

```

1  /*
2  ** Program: blink_io.c
3  ** Description: blink any LED connected to ports
4  **              I00 to I013
5  ** Author: Aldo Nunez
6  */
7
8  #include <stdio.h>
9  #include <signal.h>
10 #include <mraa.h>
11
12 /* false = 0, true = 1 */
13 typedef enum { false, true } bool;
14

```

```

15  /* OFF = 0, ON = 1 */
16  enum STATE { OFF, ON };
17
18  /* number of digital outputs ports */
19  const unsigned int PORTS = 14;
20
21  /* 1e6 ~ 1 sec */
22  const useconds_t T = 1e6;
23
24  volatile sig_atomic_t flag = 1;
25
26  /*
27  ** Name:      isValid
28  ** Parameters: string input
29  ** Output:     Boolean
30  **             true - if the input is an integer number.
31  **             false - if the input is not an integer number
32  ** Description: Check if the input is an integer number
33  */
34  bool isValid ( char* );
35
36  /*
37  ** Name: manageSignal
38  ** Input: Integer
39  ** Output: None
40  ** Description: Catch the signal interrupt,
41  **             'Ctrl + c' signal generated
42  **             by the user and modify
43  **             flag variable
44  */
45  void manageSignal ( int );
46
47  int
48  main ( int argc, char* argv [] )
49  {
50      if ( argc < 2 )
51      {
52          fprintf ( stderr, "Usage: %s <port> \n", argv [ 0 ] );
53          fprintf ( stderr, "<port>: 0 | 1 | 2 | 3 | 4 | 5 | ..... | 13 \n"
54                  );
55          return 1;
56      }
57      /* Check if the argument is a number */
58      if ( !isValid ( argv [ 1 ] ) )
59      {
60          fprintf ( stderr, "<port> must be an integer number between: 0 -
61                  13 \n" );
62          return 1;
63      }
64
65      int port = atoi ( argv [ 1 ] );
66
67      /* Check if the number is between: 0 - 13 */
68      if ( ( port < 0 ) || ( port > PORTS - 1 ) )
69      {
70          fprintf ( stderr, "<port> must be an integer number between: 0 -
71                  13 \n" );
72          return 1;

```



```

73     }
74
75     /* create access to gpio pin */
76     mraa_gpio_context led;
77
78     /* Initialize pin LED_PIN for led */
79     led = mraa_gpio_init ( port );
80     if ( led == NULL )
81     {
82         fprintf (stderr, "The port %d that you requested is not valid!",
83                 port );
84         return 1;
85     }
86
87     /* set gpio direction to out */
88     mraa_result_t response = mraa_gpio_dir ( led, MRAA_GPIO_OUT );
89     if ( response != MRAA_SUCCESS )
90     {
91         mraa_result_print ( response );
92     }
93
94     printf ( "MRAA Version: %s\n", mraa_get_version () );
95     printf ( "Platform: %s\n", mraa_get_platform_name () );
96     printf ( "blinking port %d...\n", port );
97     printf ( "To finish press: Ctrl + c \n" );
98
99     signal ( SIGINT, manageSignal );
100
101     /* blink indefinitely until (Ctrl + c) */
102     while ( flag )
103     {
104         response = mraa_gpio_write ( led, ON );
105         if ( response != MRAA_SUCCESS )
106         {
107             mraa_result_print ( response );
108         }
109
110         usleep ( T );
111
112         response = mraa_gpio_write ( led, OFF );
113         if ( response != MRAA_SUCCESS )
114         {
115             mraa_result_print ( response );
116         }
117
118         usleep ( T );
119     }
120
121     mraa_gpio_close ( led );
122
123     return 0;
124 }
125
126 bool
127 isValid ( char* str )
128 {
129     while ( *str != 0 )
130     {
131         if ( !isdigit ( *str ) )
132         {
133             return false;

```

```

133     }
134     str++;
135 }
136 return true;
137 }
138
139 void
140 manageSignal ( int sig )
141 {
142     if ( sig == SIGINT )
143     {
144         flag = 0;
145     }
146     printf ( "\nprogram is closing ... \n" );
147
148     return;
149 }

```

Compilar:

```
$ gcc -o blink_io blink_io.c -Wall -lmraa
```

Ejecutar como súper usuario:

```
# ./blink_io <port>
```

```
<port> 0 | 1 | 2 | 4 |...| 13
```

4.0.2 Entradas digitales

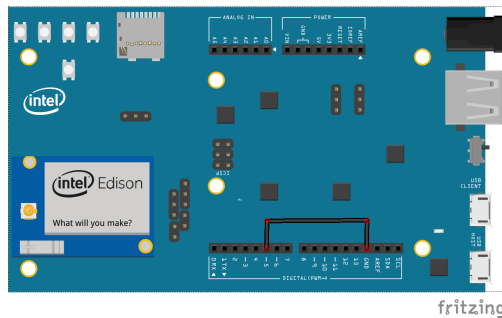


Figura 4.5: Entradas

En este ejemplo, se ha escogido el puerto 5. Por defecto, las entradas digitales están en alto. Para leer un valor 0, conectamos el puerto 5 a tierra (GND).

Versión en C:

```

1  /*
2  ** Program: digital_input.c
3  ** Description: Read a digital input and display its value
4  ** Author: Aldo Nunez
5  */
6
7  #include <stdio.h>
8  #include <mraa.h>
9
10 const int PORT = 5;
11
12 int
13 main ( void )
14 {
15     /* declare variable as a gpio type */
16     mraa_gpio_context port_input;
17
18     /* initialize variable with a port number*/
19     port_input = mraa_gpio_init ( PORT );
20
21     /* set gpio direction to input */
22     mraa_gpio_dir ( port_input, MRAA_GPIO_IN );
23
24     /* read port value */
25     int value = mraa_gpio_read ( port_input );
26
27     /* print port number and its value */
28     printf ( "Input port %d: %d\n", PORT, value );
29
30     /* close port */
31     mraa_gpio_close ( port_input );
32
33     return ( MRAA_SUCCESS );
34 }

```

Compilar:

```
$ g++ -o digital_input digital_input.cpp -Wall -lmraa
```

Ejecutar como súper usuario:

```
# ./digital_input
```

Versión en Python:

```

1  '''
2  Program: digital_input.py
3  Description: Read a digital input and display its value
4  Author: Aldo Nunez
5  '''
6
7  import mraa as m
8
9  PORT = 5
10
11 # initialize variable with a port number
12 port_input = m.Gpio ( PORT )
13
14 # set gpio direction to input
15 port_input.dir ( m.DIR_IN )
16
17 # read port value
18 value = port_input.read ()
19
20 print ( "Input port %d: %d" % ( PORT , value ) )

```

Ejecutar como súper usuario:

```
# python digital_input
```

En el repositorio de [GitHub](#) hemos incluido las versiones en C++ y JavaScript de este programa.

Como en el caso del programa blink, hemos mejorado el programa digital_input, y lo hemos incluido en el repositorio de GitHub. El programa se llama digital_input.io.

4.0.3 PWM

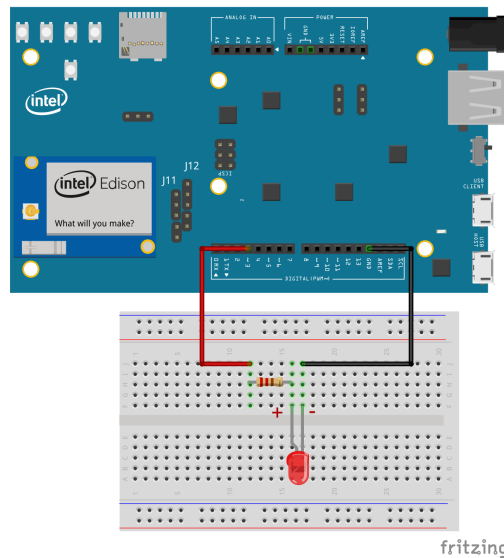


Figura 4.6: Salidas PWM

Edison solo tiene 4 salidas PWM (Pulse Width Modulation). Por defecto, los puertos 3, 5, 6, 9 están configuradas como salidas PWM. También se pueden usar los puertos 10 y 11, como salidas PWM, moviendo los jumpers J11 y J12, Figura 4.6.

Qué es el ciclo útil (duty cycle)? Es el porcentaje de tiempo en la que una señal se mantiene en alto sobre un período de tiempo. En la Figura 4.7 se observan diferentes valores de ciclo útil.

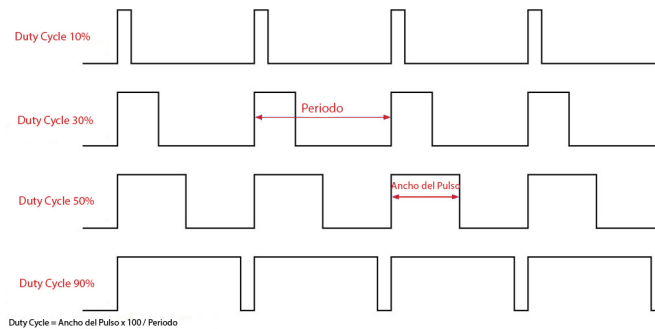


Figura 4.7: Ciclo útil

Para calcular el porcentaje de ciclo útil podemos usar la siguiente ecuación:

$$D = 100 \times \frac{T_h}{P}$$

T_h - tiempo de la señal en alto

P - período de la señal

Si no dispone de un osciloscopio para observar el resultado puede usar el circuito de la Figura 4.6. La intensidad de la luz emitida por el led debe variar conforme varía el porcentaje del ciclo útil.

Componentes:

- 1 led
- 1 resistencia de 220 Ω
- Alambres
- Placa de prueba (breadboard)

Código:

El siguiente programa genera una salida PWM en el puerto especificado en la entrada. El usuario debe introducir el puerto y luego el porcentaje de ciclo útil (duty cycle) deseado; valores entre 0.0 y 100.0.

Es importante comentar que se ha añadido la función

```
void turnOff ( int );
```

con el fin de que cuando el programa finalice, el estado de la salida PWM sea de apagado.

Versión en C++

```

1  // Name: pwm.cpp
2  // Description: Generates a PWM output. The user enters
3  //               the port name and the percentage of duty cycle
4  // Author: Aldo Nunez
5
6  #include <iostream>
7  #include <string>
8  #include <mraa.hpp>
9
10 using namespace mraa;
11 using namespace std;
12
13
14 const int gpio [] = { 3, 5, 6, 9 };
15 const string pwm [] = { "P0", "P1", "P2", "P3" };
16 const int T ( 20000 );      // T - period in usec
17
18 // Name:      isValidPort
19 // Parameters: String
20 // Output:     Integer:
21 //               port index number
22 //               -1, if it is not a valid port.
23 // Description: Check if the input is a valid port.
24 //               If it is a valid port, return port index number
25 //               If it is not a valid port, return -1
26 int isValidPort ( string );
27
28 // Name:      isValidDC
29 // Parameters: String
30 // Output:     Boolean, true - if it is a valid input.
31 //               false - if it is not a valid input
32 // Description: Check if the input is a number or a
33 //               decimal point.
34 bool isValidDC ( string );
35
36 // Name:      turnOff
37 // Parameters: int
38 // Output:     void

```

```

39 // Description: Turn off the output
40 void turnOff ( int );
41
42 int
43 main ( int argc, char* argv [] )
44 {
45     if ( argc < 3 )
46     {
47         cout << "Usage: " << argv [ 0 ] << "<port>" << " <duty cycle>"
48             << endl;
49         cout << "<port> must be: " << pwm [ 0 ] << " | " << pwm [ 1 ] <<
50             " | " << pwm [ 2 ] << " | " << pwm [ 3 ] << endl;
51         cout << "<duty cycle> 0.0 - 100.0" << endl;
52         return 1;
53     }
54
55     // check if it is a valid port
56     int port_index;
57     if ( ( port_index = isValidPort ( argv [ 1 ] ) ) == -1 )
58     {
59         cout << "Invalid port..." << endl;
60         cout << "<port> must be: " << pwm [ 0 ] << " | " << pwm [ 1 ] <<
61             " | " << pwm [ 2 ] << " | " << pwm [ 3 ] << endl;
62         return 1;
63     }
64
65     // check if it is a valid duty cycle
66     if ( !isValidDC ( argv [ 2 ] ) )
67     {
68         cout << "Invalid argument..." << "\n";
69         cout << "<duty cycle> must be a number between: 0.0 - 100.0" <<
70             endl;
71         return 1;
72     }
73
74     // convert the input argument to floating point
75     // and check if it is in the valid interval
76     float dutyCycle ( atof ( argv [ 2 ] ) ); // percentage value
77     if ( ( dutyCycle > 100.0 ) || ( dutyCycle < 0.0 ) )
78     {
79         cout << "<duty cycle> must be between: 0.0 - 100.0" << endl;
80         return 1;
81     }
82
83     int pwm_port ( gpio [ port_index ] );
84     Pwm* pwm = new Pwm ( pwm_port );
85     if ( pwm == NULL )
86     {
87         return ERROR_UNSPECIFIED;
88     }
89
90     pwm -> enable ( true );
91     pwm -> period_us ( T ); // f ~ 1/T
92     pwm -> write ( dutyCycle / 100.0 );
93
94     cout << "MRAA Version: " << mraa_get_version () << endl;
95     cout << "Platform: " << mraa_get_platform_name () << endl;
96     cout << "Port Number: " << pwm_port << endl;
97     cout << "Period: " << T * 1.0e-6 << " sec" << endl;
98     cout << "Frequency: " << 1.0e6 / T << " Hz" << endl;
99     cout << "Percentage of PWM: " << 100 * pwm -> read () << endl;

```

```

96     cout << "Press \"Enter\" to finish." << endl;
97
98     cin.ignore ();
99     delete ( pwm );
100    turnOff ( pwm_port );
101
102    return ( MRAA_SUCCESS );
103 }
104
105 bool
106 isValidDC ( string s )
107 {
108     int i ( 0 );
109
110     while ( s [ i ] != 0 )
111     {
112         if ( !isdigit ( s [ i ] ) && ( s [ i ] != '.' ) )
113         {
114             return false;
115         }
116         i++;
117     }
118     return true;
119 }
120
121 int
122 isValidPort ( string port )
123 {
124     for ( int i ( 0 ); i < sizeof ( pwm ) / sizeof ( pwm [ 0 ] ); i++ )
125     {
126         if ( port.compare ( pwm [ i ] ) == 0 )
127         {
128             return i;
129         }
130     }
131     return -1;
132 }
133
134 void
135 turnOff ( int port )
136 {
137     int OFF ( 0 );
138
139     mraa::Gpio p ( port );
140     mraa::Result response = p.dir ( mraa::DIR_OUT );
141     p.write ( OFF );
142     return;
143 }

```

Compilar:

```
$ g++ -o pwm pwm.cpp -Wall -lmraa
```

Ejecutar como súper usuario:

```
#./pwm <port> <number>
```

```
<port>: P0 | P1 | P2 | P3
```

```
<number>: 0.0 - 100.0
```

Versión en Python:

```

1  '''
2  Name: pwm.py
3  Description: Generate a PWM output. The user enters
4               the percentage of duty cycle
5  Author: Aldo Nunez
6  '''
7
8  import mraa as m
9  import sys
10
11  PWM = { 'P0':3, 'P1':5, 'P2':6, 'P3':9 }
12  T = 20000    # Period: 20000 usec ~ 50 Hz
13
14  '''
15  Name:          isValidPort
16  Parameters:    string
17  Output:        True - if it is a valid input.
18                False - if it is not a valid input
19  Description:   Check if the input is a valid number
20  '''
21  def isValidPort ( string ):
22      if string in PWM:
23          return PWM [ string ]
24      return False
25
26
27  '''
28  Name:          isValidDC
29  Parameters:    string
30  Output:        boolean, True - if it is a valid input.
31                False - if it is not a valid input
32  Description:   Check if the input is a valid number
33  '''
34  def isValidDC ( string ):
35      try:
36          float ( string )
37          return True
38      except:
39
40          return False
41
42  '''
43  Name:          turnOff
44  Parameters:    number
45  Output:        None
46  Description:   Turn off the output
47  '''
48  def turnOff ( port ):
49      OFF = 0;
50      p = m.Gpio ( port )
51      p.dir ( m.DIR_OUT )
52      p.write ( OFF )
53
54  if len ( sys.argv ) < 3:
55      print ( "Usage: python " + sys.argv [ 0 ] + " <port> " + " <number>"
56              )
57      print ( "<port>: P0 | P1 | P2 | P3" )
58      print ( "<number>: 0.0 - 100.0" )

```

```

58     sys.exit ()
59
60 # check if it is a valid port
61 p = isValidPort ( sys.argv [ 1 ] )
62 if p == False:
63     print ( "Invalid port..." )
64     print ( "Argument must be: P0 | P1 | P2 | P3" )
65     sys.exit ()
66
67 # check if it is a valid duty cycle
68 if isValidDC ( sys.argv [ 2 ] ) == False:
69     print ( "Invalid argument..." )
70     print ( "Argument must be a number between: 0.0 - 100.0" )
71     sys.exit ()
72
73 # convert the input argument to floating point
74 # and check if it is in the valid interval
75 value = float ( sys.argv [ 2 ] );
76 if value < 0.0 or value > 100.0:
77     print ( "<number> must be between: 0.0 - 100.0" )
78     sys.exit ()
79
80 PIN_PORT = p
81 out = m.Pwm ( PIN_PORT )
82 out.period_us ( T )
83 out.enable ( True )
84 out.write ( value / 100.0 )
85
86 print ( "MRAA Version: " + m.getVersion () )
87 print ( "Platform: " + m.getPlatformName () )
88 print ( "Port Number: " + str ( PIN_PORT ) )
89 print ( "Period: " + str ( T * 1.0e-6 ) + " sec" )
90 print ( "Frequency: " + str ( 1.0e6 / T ) + " Hz" )
91 print ( "Percentage of PWM: ",
92       "{0:.2f}".format ( round ( 100 * out.read (), 2 ) ) )
93
94 c = raw_input ( "Press \"Enter\" to finish." )
95 turnOff ( PIN_PORT )

```

Ejecutar como súper usuario:

```
# python pwm.py <port> <number>
```

```
<port>: P0 | P1 | P2 | P3
```

```
<number>: 0.0 - 100.0
```

4.0.4 Entradas analógicas

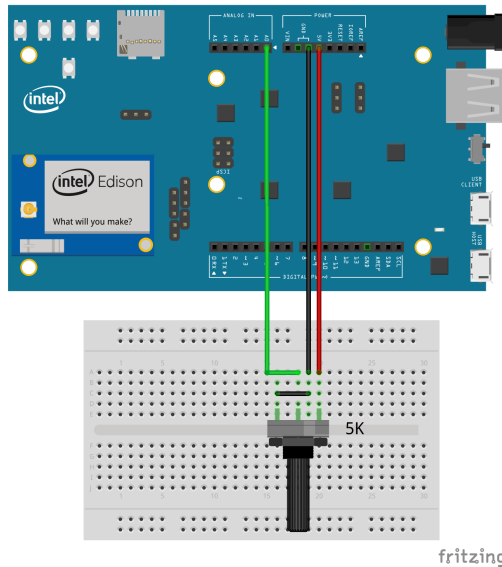


Figura 4.8: Entradas analógicas

Edison tiene 6 entradas analógicas (A0 .. A5). Usa un convertidor análogo digital (ADC) de 12 bits, el cual puede configurarse también como un convertidor de 10 bits. El rango dinámico es de 0 - 5 volts

Si lo configuramos como un ADC de 12 bits, el rango de valores enteros es el siguiente: $0 \leq n \leq 2^{12} - 1$

Si leemos estos valores en punto flotante (float), el rango es el siguiente: $0.0 \leq n \leq 1.0$

En el circuito de la Figura 4.8 el voltaje de entrada lo generamos a través de un potenciómetro de 5 K Ω .

Componentes:

- 1 Potenciómetro de 5 K Ω
- Alambres
- Placa de prueba (breadboard)

Código

El siguiente programa lee la entrada analógica de cualquiera de las 6 entradas analógicas disponibles. El usuario debe introducir el puerto analógico deseado como argumento.

Versión en C++

```
1 // Program: analog_input.cpp
2 // Description: Read analog input and display its value
3 // Author: Aldo Nunez
4
5 #include <iostream>
6 #include <iomanip>
7 #include <mraa.hpp>
```

```

8
9 using namespace mraa;
10 using std::cout;
11
12 // Name:      isValid
13 // Parameters: pointer to string
14 // Output:     Boolean, true - if it is a valid input.
15 //            false - if it is not a valid input
16 // Description: Check if the input is an integer number
17 bool isValid ( char* );
18
19 enum ANALOG_IN { A0, A1, A2, A3, A4, A5 }; // analog ports: A0 - A5
20 const int NBITS ( 12 ); // number of bits
21
22 int
23 main ( int argc, char* argv[] )
24 {
25     // Check if the number of arguments is 2
26     if ( argc < 2 )
27     {
28         cout << "Usage: " << argv [ 0 ] << " <port>" << std::endl;
29         cout << "<port>: 0 | 1 | 2 | 3 | 4 | 5" << std::endl;
30         return 1;
31     }
32
33     // Check if the argument is a number
34     if ( !isValid ( argv [ 1 ] ) )
35     {
36         cout << "<port> must be an integer number: 0 | 1 | 2 | 3 | 4 | 5"
37             << "\n";
38         return 1;
39     }
40
41     // Check if the number is between: 0 - 5
42     int port ( atoi ( argv [ 1 ] ) );
43
44     if ( ( port < A0 ) || ( port > A5 ) )
45     {
46         cout << "<port> must be an integer number between: 0 - 5" << std
47             << endl;
48         return 1;
49     }
50
51     uint16_t intValue ( 0 ); // variable to read integer value
52     float floatValue ( 0.0 ); // variable to read float value
53
54     Aio* aInput = new Aio ( port );
55
56     if ( aInput == NULL )
57     {
58         return MRAA_ERROR_UNSPECIFIED;
59     }
60
61     aInput -> setBit ( NBITS ); // configure # of bits of ADC
62
63     cout << "MRAA Version: " << mraa_get_version () << "\n";
64     cout << "Platform: " << mraa_get_platform_name () << "\n";
65     while ( 1 )
66     {
67         intValue = aInput -> read (); // reading integer value

```

```

67     floatValue = aInput -> readFloat ();    // reading float value
68     cout << "ADC " << port << " read integer: " << intValue << "\n"
        ;
69     cout << "ADC " << port << " read float: " << std::
        setprecision ( 5 ) << floatValue << "\n";
70     sleep ( 1 );
71 }
72
73 delete ( aInput );
74
75 return ( MRAA_SUCCESS );
76 }
77
78
79 bool
80 isValid ( char* str )
81 {
82     while ( *str != 0 )
83     {
84         if ( !isdigit ( *str ) )
85         {
86             return false;
87         }
88         str++;
89     }
90     return true;
91 }

```

Compilar:

```
$ g++ -o analog_input analog_input.cpp -Wall -lmraa
```

Ejecutar como súper usuario:

```
# ./analog_input <number>
```

```
<number>: 0 | 1 | 2 | 3 | 4 | 5
```

Versión en Python

```

1  '''
2  Program: analog_input.py
3  Description: Read analog input and displays its value
4  Author: Aldo Nunez
5  '''
6
7  import time
8  import sys
9  import mraa as m
10
11  NBITS = 12
12
13  try:
14      port = int ( sys.argv [ 1 ] )
15  except:
16      print ( "Usage: python " + sys.argv [ 0 ] + " <port>" )
17      print ( "<port>: 0 | 1 | 2 | 3 | 4 | 5 " )
18      sys.exit ()
19
20  #Check if the number is between: 0 - 5
21  if ( port > 5 ) or ( port < 0 ):
22      print ( "<port> must be an integer number between: 0 - 5" );
23      sys.exit ()
24
25  analogIn = m.Aio ( port )
26  analogIn.setBit ( NBITS )
27
28  print ( "MRAA Version: " + m.getVersion () )
29  print ( "Platform: " + m.getPlatformName () )
30  print ( "Analog port: " + str ( port ) )
31
32  while True:
33      intValue = analogIn.read ()
34      floatValue = analogIn.readFloat ()
35      print ( "ADC " + sys.argv [ 1 ] + " read integer: " + str ( intValue
36      ) )
37      print ( "ADC " + sys.argv [ 1 ] + " read float: " + "%.5f" % round
38      ( floatValue, 5 ) )
39      time.sleep ( 1 )

```

Ejecutar como súper usuario:

```
# python analog_input.py <number>
```

```
<number>: 0 | 1 | 2 | 3 | 4 | 5
```

5 Ejemplos

5.1 Servo

Las salidas PWM se usan, entre otras aplicaciones, para controlar servo motores.

El sentido de giro de las flechas de estos servo motores se controlan variando el ciclo útil (duty cycle) de la señal PWM, Figura 5.1.

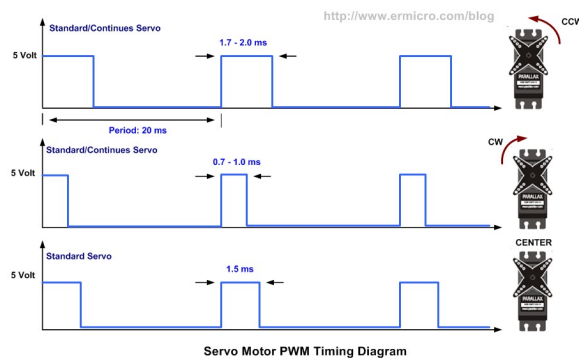


Figura 5.1: Diagrama de tiempo

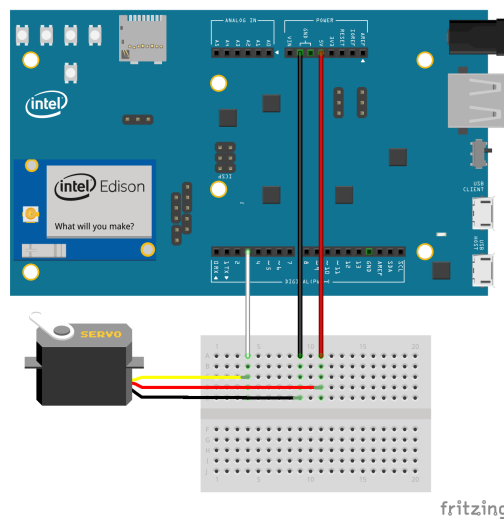


Figura 5.2: Diagrama de conexiones

En el diagrama de tiempo de la Figura 5.1 se observa que para

girar la flecha en sentido horario se requiere un pulso entre 0.7 ms a 1.0 ms. Para girar en sentido anti horario, un pulso entre 1.7 ms a 2 ms. La duración de los pulsos pueden cambiar dependiendo del fabricante. Esta prueba se hizo con servos de la marca Futaba y Vigor. Para mayor información consulte el manual del fabricante.

Para alimentar y controlar este tipo de servo se dispone de tres conexiones: Vcc - voltaje de entrada (rojo), GND - tierra (negro o marrón), control (blanco o naranja). El voltaje de alimentación puede variar de 4.8 a 6 volts. Y las señal de control de 3 a 5 volts.

Las conexiones se muestran en la Figura 5.2.

Utilizamos una señal cuyo periodo es 20 ms (frecuencia 50 Hz).

Version en C++

```

1 // Name: pwm.cpp
2 // Description: Generate a PWM signal to control a servo motor.
3 //               The user enters the turn direction: CW or CCW
4 // Author: Aldo Nunez
5
6 #include <iostream>
7 #include <string>
8 #include <mraa.hpp>
9
10 using namespace mraa;
11 using std::cout;
12
13 // Program: turnServo
14 // Parameters: ptr - Pointer to Pwm
15 //             t   - Period in micro seconds
16 //             pw  - Pulse width
17 // Output:      None
18 // Description: Enable, set the period and send
19 //             the command to generate pwm signal
20 void turn_servo ( Pwm*, float, float );
21
22 enum PWM { PWM0 = 3, PWM1 = 5, PWM2 = 6, PWM3 = 9 } pwm_port; // PWM
                ports
23 const int T ( 20000 );           // T period in usec. 20000 ~ 20 ms;
                f ~ 50 hz
24 const float CLKWISE ( 1.0e-3 ); // Clockwise, pulse width 1 ms
25 const float CCLKWISE ( 2.0e-3 ); // Counter Clockwise, pulse width 2
                ms
26
27 int
28 main ( int argc, char* argv [] )
29 {
30     if ( argc < 2 )
31     {
32         cout << "<Usage>: " << argv [ 0 ] << " <DIR>" << "\n";
33         cout << "DIR: <CW> or <CCW>" << "\n";
34
35         return 1;
36     }
37
38     pwm_port = PWM0;
39     Pwm* pwm = new Pwm ( pwm_port );
40     if ( pwm == NULL )

```



```

41     {
42         return ERROR_UNSPECIFIED;
43     }
44
45     std::string dir ( argv [ 1 ] );
46     if ( !dir.compare ( "CW" ) )
47     {
48         turn_servo ( pwm, T, CLKWISE );
49         dir = "Clockwise";
50     }
51     else if ( !dir.compare ( "CCW" ) )
52     {
53         turn_servo ( pwm, T, CCLKWISE );
54         dir = "Counter clockwise";
55     }
56     else
57     {
58         delete ( pwm );
59         cout << "Usage: ./servo <DIR>" << "\n";
60         cout << "DIR: <CW> or <CCW>" << "\n";
61
62         return 1;
63     }
64
65     cout << "MRAA Version: " << mraa_get_version () << "\n";
66     cout << "Platform: " << mraa_get_platform_name () << "\n";
67     cout << "PWM port: " << pwm_port << "\n";
68     cout << dir << "\n";
69
70     std::cin.ignore ();
71     delete ( pwm );
72
73     return ( MRAA_SUCCESS );
74 }
75
76 void
77 turn_servo ( Pwm* ptr, float t, float pw )
78 {
79     ptr -> enable ( true );
80     ptr -> period_us ( t );
81     ptr -> pulsewidth ( pw );
82
83     return;
84 }

```

Compilar:

```
$ g++ -o servo servo.cpp -Wall -lmraa
```

Ejecutar como súper usuario:

```
# ./servo <DIR>
```

```
<DIR>: CW | CCW
```

Versión en Python:

```

1  '''
2  Program: servo.py
3  Description: Generates a PWM signal to control a servo

```

```

4   Author: Aldo Nunez
5   '''
6   import sys
7   import mraa as m
8
9   PORT = 3
10
11  T = 20000          # Period: 20 msec; f = 50 Hz
12  CLKWISE = 1.0e-3
13  CCLKWISE = 2.0e-3
14
15  '''
16  Program: turnServo
17  Parameters: p - Reference to pwm
18              t - Period in micro seconds
19              pw - Pulse width
20  Output:      None
21  Description: Enable, set the period and send
22              the command to generate pwm signal
23  '''
24  def turn_servo ( p, t, d ):
25      pwm.period_us ( t )
26      pwm.enable ( True )
27      pwm.pulsewidth ( d )
28      return;
29
30  try:
31      n = sys.argv [ 1 ]
32  except:
33      print ( "<Usage>: " + sys.argv [ 0 ] + " <DIR>" )
34      print ( "<DIR>: CW or CCW" )
35      sys.exit()
36
37  pwm = m.Pwm ( PORT )
38  pwm.period_us ( T )
39
40  if sys.argv [ 1 ] == 'CW':
41      turn_servo ( pwm, T, CLKWISE )
42  elif sys.argv [ 1 ] == 'CCW':
43      turn_servo ( pwm, T, CCLKWISE )
44  else:
45      print ( "<DIR>: CW or CCW" )
46      sys.exit ()
47
48  print ( "MRAA Version: " + m.getVersion () )
49  print ( "Platform: " + m.getPlatformName () )
50  print ( "Port Number: " + str ( PORT ) )
51
52  c = raw_input ( "Press \"Enter\" to finish." )

```

Ejecutar como súper usuario:

```
# python servo.py <DIR>
```

```
<DIR>: CW | CCW
```

Glosario

Bluetooth Bluetooth, es una tecnología inalámbrica para intercambio de datos a cortas distancias. [7](#)

GNU GNU's Not Unix, es un sistema operativo y una extensa colección de programas gratuitos. [18](#)

Yocto El Proyecto Yocto, es un proyecto de colaboración de código abierto que provee plantillas, herramientas y métodos para ayudar a crear sistemas personalizados basados en Linux dirigido a productos embebidos y con independencia de la arquitectura del hardware. [7](#)

Siglas

ADC Analog to Digital Converter. [7](#)

I2C Inter Integrated Circuit. [7](#)

opkg Open Package management. [11](#)

PWM Pulse Width Modulation. [7](#)

SPI Serial Peripheral Interface bus. [7](#)

UART Universal Asynchronous Receiver-Transmitter. [7](#)

Índice alfabético

ADC, 35
Angry IP Scanner, 10
Arduino, 13
avahi-daemon, 10

Biblioteca MRAA, 14, 15, 17
blink, 21
blink_io, 26
Bluetooth, 7

C, 18
C++, 18
ciclo útil, 39

Eclipse, 14, 15
Edison, 7
Entradas digitales, 27

I2C, 7
Intel, 7

Java, 18

nano, 17
Node, 18

opkg, 11

processing, 13
PWM, 29
Python, 18

Quark, 15

Salidas digitales, 19, 23
Servo, 39
 Futaba, 40
 Vigor, 40
SPI, 7

UART, 7
UPM, 14, 15

vim, 17

WiFi, 7

XDK, 14

Yocto, 10