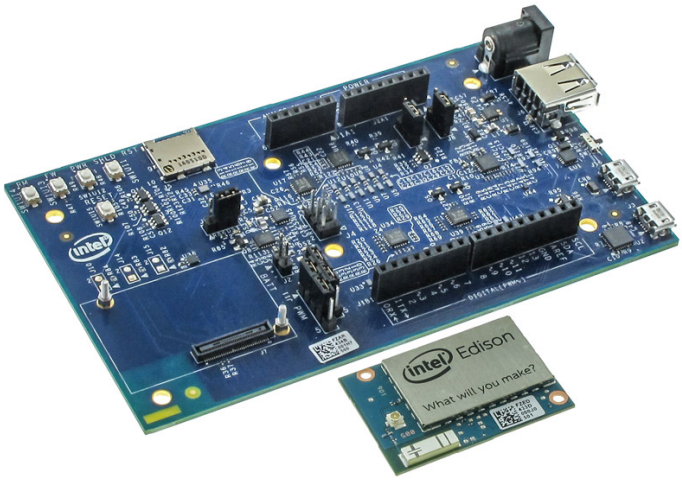
ALDO NÚÑEZ TOVAR

INICIANDO CON EDISON



FREE BOOKS

Índice general

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 lenguages 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`

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 opcíon 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:255.25
5.255.0
```

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

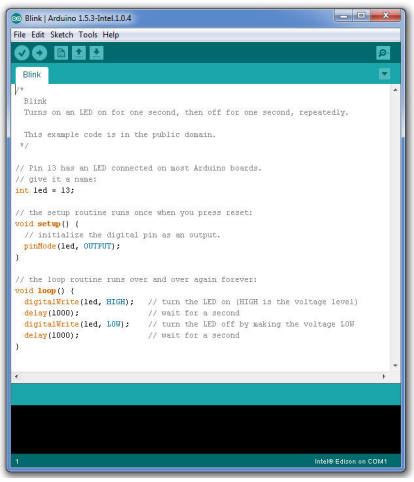


Figura 2.1: Arduino IDE

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

El lenguage 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`

2.2 *Intel XDK*

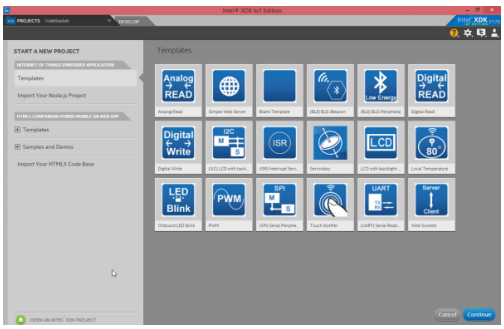


Figura 2.2: Intel XDK

Es un ambiente de desarrollo orientado a aplicaciones web y dispositivos móviles. Se programa básicamente en `JavaScript`. Contiene muchos ejemplos y plantillas (templates) para el desarrollo de aplicaciones.

Para instalar este ambiente de desarrollo consulte el siguiente enlace: `https://xdk.intel.com`

2.3 *Intel System Studio*

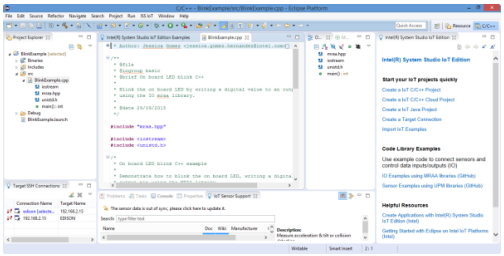


Figura 2.3: Intel System Studio

Está basado en Eclipse, se puede integrar con las bibliotecas `UPM` (biblioteca para el manejo de sensores y actuadores) y `MRAA` (biblioteca para el manejo de entradas y salidas). Se puede programar en `C/C++` y `Java`.

Puede consultar el siguiente enlace:
`https://software.intel.com/es-es/iot/tools-ide/ide/iss-iot-edition`

2.4 *Intel System Studio para microcontroladores*

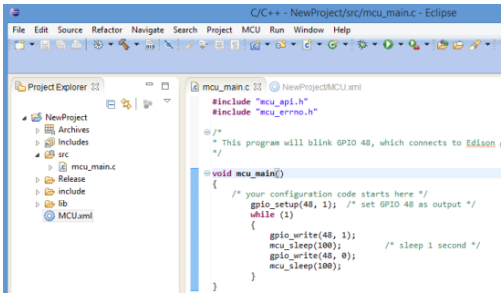


Figura 2.4: Intel System Studio para Microcontroladores

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*

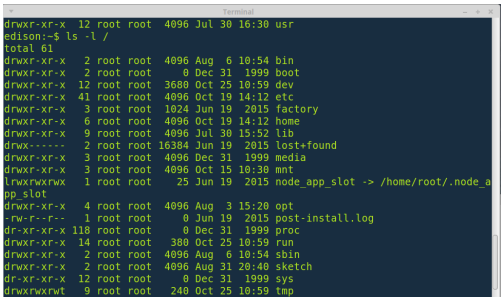


Figura 2.5: Consola

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 inalambrica 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 Uno[1]. 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

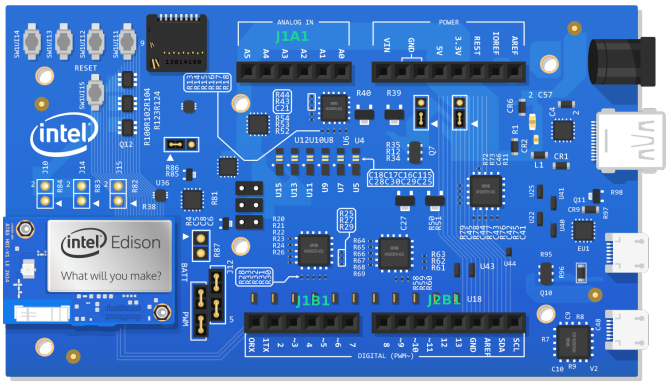


Figura 4.1: Headers de los puertos de entradas y salidas

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`

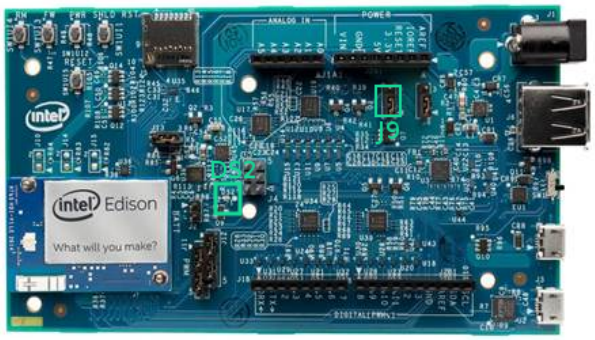
```
7  #include <mraa.h>
```

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:

```c
/*
** Program: blink.c
** Description: blink LED DS2
** Author: Aldo Nunez
*/

#include <mraa.h>

int
main ( void )
{
    /* declare led  variable as a gpio type */
    mraa_gpio_context led;

    /* initialize led with a pin number*/
    led = mraa_gpio_init ( 13 );

    /* set gpio direction to out */
    mraa_gpio_dir ( led, MRAA_GPIO_OUT );

    /* blink indefinitely until (Ctrl + c) */
    while ( 1 )
    {
        /* turn on led */
        mraa_gpio_write ( led, 1 );
        /* wait for 1 second */
        sleep ( 1 );
        /* turn off  led */
        mraa_gpio_write ( led, 0 );
        /* wait for 1 second */
        sleep ( 1 );
    }
    /* close the gpio context */
    mraa_gpio_close ( led );

    return ( MRAA_SUCCESS );
}
```

Compilar:

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

Ejecutar como súper usuario:

```
# ./blink
```

Hemos incluido en el repositorio de GitHub[2], versiones del programa blink, en C++, Python y JavaScript.

[2] https://github.com/lizard20/edison

Blink mejorado

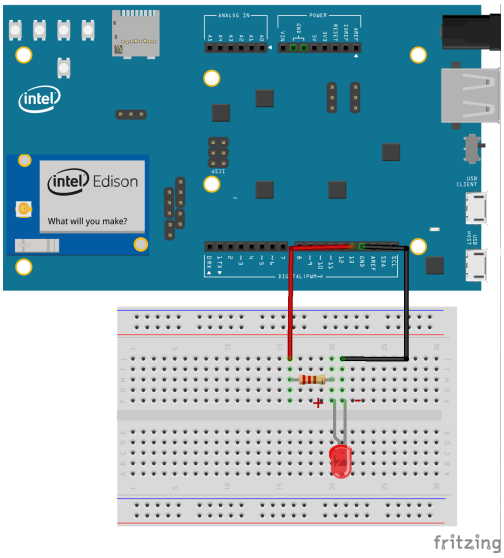


Figura 4.3: Led conectado al puerto 13

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 verificacion 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 caida 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:

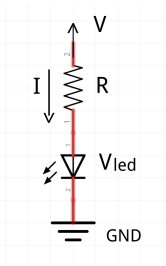


Figura 4.4: Circuito equivalente cuando la salida está en alto

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:

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

Escojemos el valor comercial más cercano, `220` Ω.

Componentes:

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

Versión en `C`:

```c
/*
** Program: blink_io.c
** Description: blink any LED connected to ports
                IO0 to IO13
** Author: Aldo Nunez
*/

#include <stdio.h>
#include <signal.h>
#include <mraa.h>

/* error messages */
const char* message1 = "Usage: %s <port> \n";
const char* message2 = "<port>: 0 | 1 | 2 | 3 | 4 | 5 |.....| 13 \n";
```

```
15  const char* message3 = "<port> must be between: 0..13 \n";
16
17  /* false = 0, true = 1 */
18  typedef enum { false, true } bool;
19
20  /* OFF = 0, ON = 1 */
21  enum STATE { OFF, ON };
22
23  /* number of digital outputs ports */
24  const unsigned int PORTS = 14;
25
26  /* 1e5 ~ 0.1 sec */
27  const useconds_t T = 1e5;
28
29  volatile sig_atomic_t  flag = 1;
30
31  /************  Functions prototypes ************/
32  /*
33  ** Name:        isValidArgument
34  ** Parameters:  1.- Integer,  number of arguments
35  **              2.- Initial address of  pointers array to char
36  ** Output:      Boolean
37                  true -  if it is an integer number and  it
38                          is between 0 and 13
39  **              false - if it is not an integer number between
40  **              0 - 13
41  ** Description: Check if the inputs are valid arguments
42  */
43  bool isValidArgument ( int, char* [] );
44
45  /*
46  ** Name:        isNumber
47  ** Parameters:  string input
48  ** Output:      Boolean
49                  true - if the input is an integer number.
50  **              false - if the input is not an integer number
51  ** Description: Check if the input is an integer number
52  */
53  bool isNumber ( char* );
54
55  /*
56  ** Name: manage_signal
57  ** Input: Integer
58  ** Output: None
59  ** Description: Catch the signal interrupt,
60  **              'Ctrl + c' signal  generated
61  **              by the user  and  modify
62  **              flag variable
63  */
64  void manage_signal ( int );
65  /*******  End of functions prototypes *********/
66
67  int
68  main ( int argc, char* argv [] )
69  {
70      /* Check if the argumentis is valid*/
71      if ( !isValidArgument ( argc, argv ) )
72      {
73          exit ( 1 );
74      }
75
```

```
76      int port =  atoi ( argv [ 1 ] );
77
78      /* create access to gpio  pin */
79      mraa_gpio_context led;
80
81      /* Initialize pin LED_PIN for led */
82      led = mraa_gpio_init ( port );
83      if ( led == NULL )
84      {
85          fprintf (stderr, "The port %d that you requested is not valid!\n
                ", port );
86          exit ( 1 );
87      }
88
89      /* set gpio drection to  out */
90      mraa_result_t response = mraa_gpio_dir ( led, MRAA_GPIO_OUT );
91      if ( response != MRAA_SUCCESS )
92      {
93          mraa_result_print ( response );
94      }
95
96      printf ( "MRAA Version: %s\n", mraa_get_version () );
97      printf ( "Platform: %s\n", mraa_get_platform_name () );
98      printf ( "blinking port %d...\n", port );
99      printf ( "To finish press: Ctrl + c \n" );
100
101     signal ( SIGINT, manage_signal );
102
103     /* blink indefinitely until (Ctrl + c) */
104     while ( flag )
105     {
106         response = mraa_gpio_write ( led, ON );
107         if ( response != MRAA_SUCCESS )
108         {
109             mraa_result_print ( response );
110         }
111
112         usleep ( T );
113
114         response = mraa_gpio_write ( led, OFF );
115         if ( response != MRAA_SUCCESS )
116         {
117             mraa_result_print ( response );
118         }
119
120         usleep ( T );
121     }
122
123     mraa_gpio_close ( led );
124     return 0;
125 }
126 /* end of main */
127
128 bool
129 isValidArgument ( int argc, char* argv [] )
130 {
131     /* Check the number of arguments*/
132     if ( argc < 2 )
133     {
134         fprintf ( stderr, message1, argv [ 0 ] );
135         fprintf ( stderr, message2 );
```

```
136             return false;
137         }
138
139         /* Check if the argument is an integer number */
140         if ( !isNumber ( argv [ 1 ] ) )
141         {
142             fprintf ( stderr, message1, argv [ 0 ] );
143             fprintf ( stderr, message2 );
144             return false;
145         }
146
147         int port =  atoi ( argv [ 1 ] );
148         /* Check if the number is between: 0 - 13 */
149         if ( ( port < 0 ) || ( port  > PORTS - 1 ) )
150         {
151             fprintf ( stderr, message3 );
152             fprintf ( stderr, message2 );
153             return false;
154         }
155         return true;
156     }
157
158     bool
159     isNumber ( char* str )
160     {
161         while ( *str != 0 )
162         {
163             if ( !isdigit ( *str ) )
164             {
165                 return false;
166             }
167             str++;
168         }
169         return true;
170     }
171
172     void
173     manage_signal ( int sig )
174     {
175         if ( sig == SIGINT )
176         {
177             flag = 0;
178         }
179         printf ("\nprogram is closing ... \n" );
180         return;
181     }
```

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*

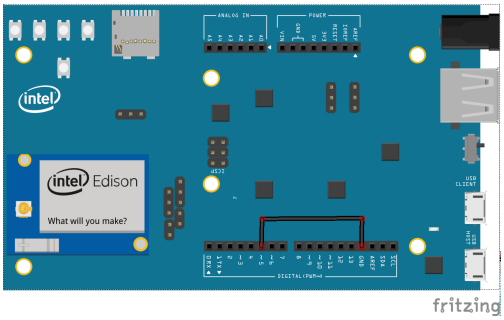


Figura 4.5: Entradas

En esto 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`:

```c
/*
** Program: digital_input.c
** Description: Read a digital input and display its value
** Author: Aldo Nunez
*/

#include <stdio.h>
#include <mraa.h>

const int PORT = 5;

int
main ( void )
{
    /* declare variable as a gpio type */
    mraa_gpio_context port_input;

    /* initialize variable with a port number*/
    port_input = mraa_gpio_init ( PORT );

    /* set gpio direction to input */
    mraa_gpio_dir ( port_input, MRAA_GPIO_IN );

    /* read port value */
    int value = mraa_gpio_read ( port_input );

    /* print port number and its value */
    printf ( "MRAA Version: %s \n", mraa_get_version () );
    printf ( "Platform: %s \n", mraa_get_platform_name () );
    printf ( "Input port %d: %d\n", PORT, value );

    /* close port */
    mraa_gpio_close ( port_input );

    return ( MRAA_SUCCESS );
}
```

Compilar:

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

Ejecutar como súper usuario:

```
# ./digital_input
```

Versión en Python:

```
'''
Program: digital_input.py
Description: Read a digital input and display its value
Author: Aldo Nunez
'''

import mraa as m

# main
def main ():
    PORT = 5

    # initialize variable with a port number
    port_input = m.Gpio ( PORT )

    # set gpio direction to input
    port_input.dir ( m.DIR_IN )

    # read port value
    value  = port_input.read ()

    print ( "MRAA Version: %s" %  m.getVersion () )
    print ( "Platform: %s" % m.getPlatformName () )
    print ( "Input port %d: %d" % (  PORT , value ) )

if __name__ == "__main__":
    main ()
```

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*

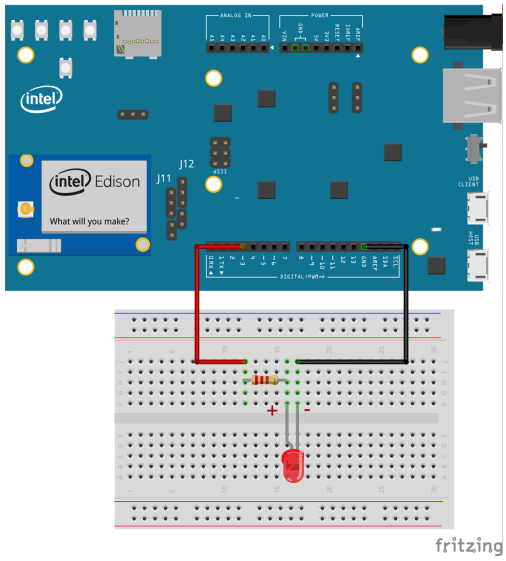


Figura 4.6: Salidas PWM

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.

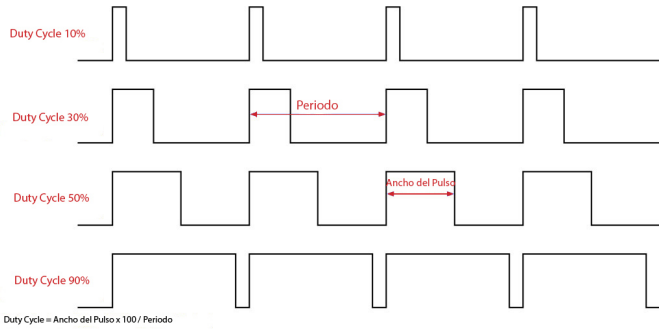


Figura 4.7: Ciclo útil

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++

```
// Name: pwm.cpp
// Description: Generates a PWM output. The user enters
//              the port name  and the percentage of duty cycle
// Author: Aldo Nunez

#include <iostream>
#include <string>
#include <mraa.hpp>

using namespace mraa;
using namespace std;

const int gpio [] = { 3, 5, 6, 9 };
const string pwm [] = { "P0", "P1", "P2", "P3" };
const int T ( 20000 );      // T - period in usec

/************  Functions prototypes ************/
/*
** Name:        isValidArgument
** Parameters:  1.- Integer,  number of arguments
**              2.- Initial address of  pointers array to char
** Output:      Boolean
                true -  if the first parameter is a valid port:
                        P0, P1, P2, P3
                        if the second parameter is a valid
                        Duty Cycle percentage: 0% - 100%
**              false - if it is not a valid Port.
**                      if it is not a valid Duty Cycle
** Description: Check if the inputs are valid arguments
*/
bool isValidArgument ( int , char* [] );

// Name:        isValidPort
// Parameters:  String
// Output:      Integer:
//                  port index number
//                  -1, if it is not a valid port.
// Description: Check if the input is a valid port.
```

```
//              If it is a valid port, return port index number
//              If it is not a valid port, return -1
int isValidPort ( string );

// Name:        isValidDC
// Parameters:  String
// Output:      Boolean, true - if it is a valid input.
//              false - if it is not a valid input
// Description: Check if the input is a number or a
//              decimal point.
bool isValidDC ( string );

//  Name:       turnOff
// Parameters:  int
// Output:      void
// Description: Turn off the output
void turnOff ( int );

int
main ( int argc, char* argv [] )
{
    // Check if the input arguments are valid
    if ( !isValidArgument ( argc, argv ))
    {
        return 1;
    }

    int port_index ( isValidPort ( argv [ 1 ] ) );
    float dutyCycle ( atof ( argv [ 2 ] ) );        // percentage value
    int pwm_port ( gpio [ port_index ] );
    Pwm* pwm = new Pwm ( pwm_port );

    if ( pwm == NULL )
    {
        return ERROR_UNSPECIFIED;
    }

    pwm -> enable ( true );
    pwm -> period_us ( T );     // f ~ 1/T
    cout << dutyCycle << endl;
    pwm -> write ( dutyCycle / 100.0 );

    cout << "MRAA Version: " << mraa_get_version () << endl;
    cout << "Platform: " << mraa_get_platform_name () << endl;
    cout << "PWM: " << port_index << endl;
    cout << "Port Number: " << pwm_port << endl;
    cout << "Period: " << T * 1.0e-6 << " sec" << endl;
    cout << "Frequency: " << 1.0e6 / T << " Hz" << endl;
    cout << "Percentage of PWM: " << 100 *  pwm -> read () << endl;
    cout << "Press \"Enter\" to finish." << endl;

    cin.ignore ();
    delete ( pwm );
    turnOff ( pwm_port );

    return ( MRAA_SUCCESS );
}
// end of main

bool
isValidArgument ( int argc, char* argv [] )
```

```
{
    if ( argc < 3 )
    {
        cerr << "Usage: " << argv [ 0 ] << " <port>" << " <duty cycle>"
             << endl;
        cerr << "<port>: " << pwm [ 0 ] << " | " << pwm [ 1 ] << " | "
             <<  pwm [ 2 ] <<  " | " <<  pwm [ 3 ] << endl;
        cerr << "<duty cycle>: 0.0 - 100.0" << endl;

        return false;
    }

    // check if it is a valid  port
    int port_index;
    if ( ( port_index = isValidPort ( argv [ 1 ] ) ) == -1 )
    {
        cerr <<  "Invalid port...." << endl;
        cerr << "<port> must be: " << pwm [ 0 ] << " | " << pwm [ 1 ] <<
              " | " <<  pwm [ 2 ] <<  " | " <<  pwm [ 3 ] << endl;

        return false;
    }

    // check if it is a valid duty cycle
    if ( !isValidDC ( argv [ 2 ] ) )
    {
        cerr << "Invalid argument...." << "\n";
        cerr << "<duty cycle> must be a number between: 0.0 - 100.0" <<
             endl;

        return false;
    }

    // convert the input argument to floating point
    // and check if it is in the valid interval
    float dutyCycle ( atof ( argv [ 2 ] ) );        // percentage value
    if ( ( dutyCycle > 100.0 ) || ( dutyCycle  < 0.0 ) )
    {
        cerr << "<duty cycle> must be between: 0.0 - 100.0" << endl;

        return false;
    }
    return true;
}

bool
isValidDC ( string s )
{
    int i ( 0 );

    while ( s [ i ] != 0 )
    {
        if ( !isdigit ( s [ i ] )  &&  ( s [ i ] != '.' ) )
        {
            return false;
        }
        i++;
    }
    return true;
}

```

```
157 int
158 isValidPort ( string port )
159 {
160     for ( int i ( 0 ); i < sizeof ( pwm ) / sizeof ( pwm [ 0 ] ); i++ )
161     {
162         if ( port.compare ( pwm [ i ] ) == 0 )
163         {
164             return i;
165         }
166     }
167     return -1;
168 }
169
170 void
171 turnOff ( int port )
172 {
173     int OFF ( 0 );
174
175     mraa::Gpio p ( port );
176     mraa::Result response = p.dir ( mraa::DIR_OUT );
177     p.write ( OFF );
178
179     return;
180 }
```

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:

```
'''
 Name: pwm.py
 Description: Generate a PWM output. The user enters
              the port and the percentage of duty cycle
 Author: Aldo Nunez
'''

import mraa as m
import sys

PWM = { 'P0':3, 'P1':5, 'P2':6, 'P3':9 }
T = 20000    # Period: 20000 usec ~ 50 Hz

'''
 Name:          isValidPort
 Parameters:    string
 Output:         True - if it is a valid input.
        False - if it is not a valid input
 Description:   Check if the input is a valid number
'''
def isValidPort ( string ):
    if string in PWM:
        return PWM [ string ]
    return False


'''
 Name:          isValidDC
 Parameters:    string
 Output:        boolean, True - if it is a valid input.
        False - if it is not a valid input
 Description:   Check if the input is a valid number
'''
def isValidDC ( string ):
    try:
        float ( string )
        return True
    except:

        return False

'''
 Name:          turnOff
 Parameters:    number
 Output:        None
 Description:   Turn off the output
'''
def turnOff ( port ):
    OFF = 0;
    p = m.Gpio ( port )
    p.dir ( m.DIR_OUT )
    p.write ( OFF )

# main
def main ( argv ):
    if len ( sys.argv ) < 3:
        print ( "Usage: python " + sys.argv [ 0 ] +  " <port>" + " <
             duty_cycle>" )
```

```
58          print ( "<port>: P0 | P1 | P2 | P3" )
59          print ( "<duty_cycle>: 0.0 - 100.0" )
60          sys.exit ()
61
62  # check if it is a valid  port
63      p = isValidPort ( sys.argv [ 1 ] )
64      if p  == False:
65          print ( "Invalid port...." )
66          print ( "<port>, must be: P0 | P1 | P2 | P3" )
67          sys.exit ()
68
69  # check if it is a valid duty cycle
70      if isValidDC ( sys.argv [ 2 ] ) == False:
71          print ( "Invalid argument...." )
72          print ( "<duty cycle>, must be between: 0.0 - 100.0" )
73          sys.exit ()
74
75  # convert the input argument to floating point
76  # and check if it is in the valid interval
77      value = float ( sys.argv [ 2 ] );
78      if value < 0.0 or value > 100.0:
79          print ( "<duty cycle>, must be between: 0.0 - 100.0" )
80          sys.exit ()
81
82      PIN_PORT = p
83      out = m.Pwm ( PIN_PORT )
84      out.period_us ( T )
85      out.enable ( True )
86      out.write ( value / 100.0 )
87
88      print ( "MRAA Version: " + m.getVersion () )
89      print ( "Platform: " + m.getPlatformName () )
90      print ( "PWM: " + sys.argv [ 1 ] )
91      print ( "Port Number: " +  str ( PIN_PORT ) )
92      print ( "Period: " + str ( T * 1.0e-6 ) + " sec" )
93      print ( "Frequency: " + str ( 1.0e6 / T ) + " Hz" )
94      print ( "Percentage of PWM: "),
95      print ( "{0:.2f}".format ( round ( 100 *  out.read (), 2 ) ) )
96
97      c = raw_input ( "Press \"Enter\" to finish." )
98      turnOff ( PIN_PORT )
99
100 if __name__ == "__main__":
101     main ( sys.argv [ 1: ] )
```

Ejecutar como súper usuario:

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

```
<port>: P0 | P1 | P2 | P3
<number>: 0.0 - 100.0
```

4.0.4 *Entradas analógicas*

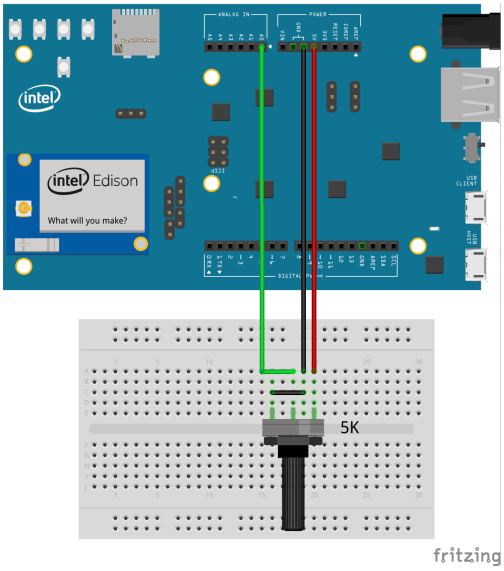


Figura 4.8: Entradas analógicas

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

```
69      while ( 1 )
70      {
71          intValue = aInput -> read ();           // reading integer value
72          floatValue = aInput -> readFloat ();    // reading float value
73          cout << "ADC " << port <<  " read integer: " << intValue << "\n"
                ;
74          cout << "ADC " << port <<  " read float:  " <<  setprecision ( 5
                ) << floatValue << "\n";
75          sleep ( 1 );
76      }
77
78      delete ( aInput );
79
80      return ( MRAA_SUCCESS );
81  }
82  //end of main
83
84  bool
85  isValid ( char* str )
86  {
87      while ( *str != 0 )
88      {
89          if ( !isdigit ( *str ) )
90          {
91              return false;
92          }
93          str++;
94      }
95      return true;
96  }
```

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

```
'''
 Program: analog_input.py
 Description: Read analog input and displays its value
 Author: Aldo Nunez
'''

import time
import sys
import mraa as m

# ADC resolution
NBITS = 12

# Number of analog ports
N = 6

PORTS = list ( range ( N ) )

def main ( argv ):
    try:
        port =  int ( sys.argv [ 1 ] )
    except:
        print ( "Usage: python " + sys.argv [ 0 ] + " <port>" )
        print ( "<port>: 0 | 1 | 2 | 3 | 4 | 5 " )
        sys.exit ()

    #Check if the number is between: 0 - 5
    if not ( port in PORTS ):
        print ( "<port>: 0 | 1 | 2 | 3 | 4 | 5 " )
        sys.exit ()

    analogIn = m.Aio ( port )
    analogIn.setBit ( NBITS )

    print ( "MRAA Version: " +  m.getVersion () )
    print ( "Platform: " + m.getPlatformName () )
    print ( "Analog port: " + str ( port ) )

    while True:
        intValue =  analogIn.read ()
        floatValue =  analogIn.readFloat ()
        print ( "ADC " + sys.argv [ 1 ] + " read integer: " + str (
             intValue )  )
        print ( "ADC " + sys.argv [ 1 ] + " read float: "  + "%.5f" %
             round (  floatValue, 5 ) )
        time.sleep ( 1 )

if __name__ == "__main__":
    main ( sys.argv [ 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.

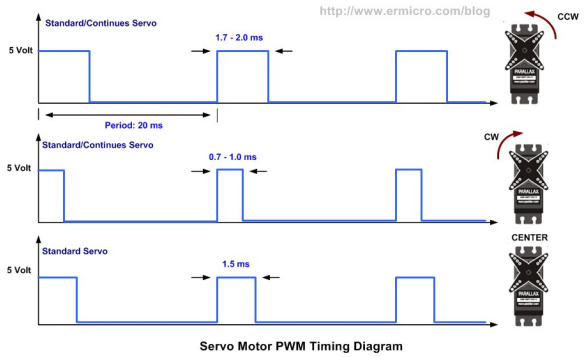


Figura 5.1: Diagrama de tiempo

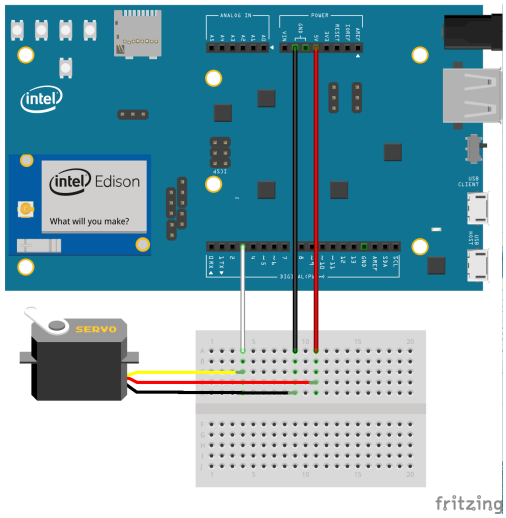


Figura 5.2: Diagrama de conexiones

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`).

Componentes:

- `1` servo motor
- Alambres
- Placa de prueba (breadboard)

Código

El siguiente programa, genera la seña `PWM` adecuada para mover la flecha del motor en un sentido o, en el otro. El usuario ingresa la dirección de giro deseada como argumento. Ya sea `CW`, sentido de giro horario. O, `CCW`, sentido de giro anti horario.

Version en `C++`

```
// Name: servo.cpp
// Description: Generate a PWM signal to control a servo motor.
//              The user enters the turn direction: CW or CCW
// Author: Aldo Nunez

#include <iostream>
#include <string>
#include <mraa.hpp>

using namespace mraa;
using std::cout;

// Program: turnServo
// Parameters:  ptr - Pointer to Pwm
//              t   - Period in micro seconds
//              pw  - Pulse width
// Output:      None
// Description: Enable, set the period and send
//              the command to generate pwm signal
void turn_servo ( Pwm*, float, float );

enum PWM { PWM0 = 3, PWM1 = 5, PWM2 = 6, PWM3 = 9 } pwm_port; // PWM
     ports
const int T ( 20000 );              // T period in usec. 20000 ~ 20 ms;
      f ~ 50 hz
const float CLKWISE  ( 1.0e-3 );    // Clockwise, pulse width  1 ms
const float CCLKWISE ( 2.0e-3 );    // Counter Clockwise, pulse width 2
     ms

int
```

```
main ( int argc, char* argv [] )
{
    if ( argc < 2 )
    {
        cout << "<Usage>: " << argv [ 0 ] << " <DIR>" << "\n";
        cout << "DIR: <CW> or <CCW>" << "\n";

        return 1;
    }

    pwm_port = PWM0;
    Pwm* pwm =  new Pwm ( pwm_port );
    if ( pwm == NULL )
    {
        return ERROR_UNSPECIFIED;
    }

    std::string dir ( argv [ 1 ] );
    if ( !dir.compare ( "CW" ) )
    {
        turn_servo ( pwm, T, CLKWISE );
        dir = "Clockwise";
    }
    else if ( !dir.compare ( "CCW" ) )
    {
        turn_servo ( pwm, T, CCLKWISE );
        dir = "Counter clockwise";
    }
    else
    {
        delete ( pwm );
        cout << "Usage: ./servo <DIR>" << "\n";
        cout << "DIR: <CW> or <CCW>" << "\n";

        return 1;
    }

    cout << "MRAA Version: " << mraa_get_version () << "\n";
    cout << "Platform: " << mraa_get_platform_name () << "\n";
    cout << "PWM port: " << pwm_port << "\n";
    cout << dir << "\n";

    std::cin.ignore ();
    delete ( pwm );

    return ( MRAA_SUCCESS );
}

void
turn_servo ( Pwm* ptr, float t, float pw )
{
        ptr -> enable ( true );
        ptr -> period_us ( t );
        ptr -> pulsewidth ( pw );

        return;
}
```

Compilar:

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

Ejecutar como súper usuario:

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

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

Versión en Python:

```
'''
 Program: servo.py
 Description: Generates a PWM signal to control a servo motor
 Author: Aldo Nunez
'''
import sys
import mraa as m

'''
 Program: turnServo
 Parameters: p  - Reference to pwm
             t  - Period in micro seconds
             pw - Pulse width
 Output:     None
 Description: Enable, set the period and send
            the command to generate pwm signal
'''
def turn_servo ( p, t, d ):
    p.period_us ( t )
    p.enable ( True )
    p.pulsewidth ( d )
    return;

def main ( argv ):
    try:
        n =  sys.argv [ 1 ]
    except:
        print ( "<Usage>: " + sys.argv [ 0 ] + " <DIR>" )
        print ( "<DIR>: CW or CCW")
        sys.exit()

    PORT = 3
    T = 20000           # Period: 20 msec; f = 50 Hz
    CLKWISE = 1.0e-3
    CCLKWISE = 2.0e-3


    pwm = m.Pwm ( PORT )
    pwm.period_us ( T )
    turn_dir = sys.argv [ 1 ]

    if turn_dir == 'CW':
        turn_servo ( pwm, T, CLKWISE )
    elif turn_dir == 'CCW':
        turn_servo ( pwm, T, CCLKWISE )
    else:
        print ( "<Usage>: " + sys.argv [ 0 ] + " <DIR>" )
        print ( "<DIR>: CW or CCW")
        sys.exit ()

    print ( "MRAA Version: " + m.getVersion () )
    print ( "Platform: " + m.getPlatformName () )
```

```
53      print ( "Port Number: " +  str ( PORT) )
54      print ( "Clockwise" if turn_dir == "CW"  else "Counter clockwise" )
55      c = raw_input ( "" )
56
57  if __name__ == "__main__":
58      main ( sys.argv [ 1: ] )
```

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