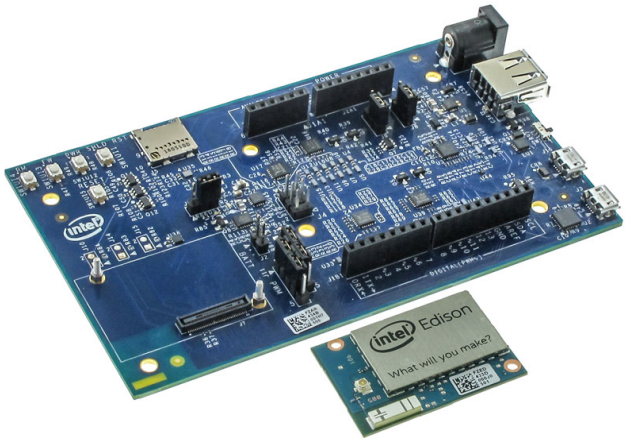
Edición en español



Empezando con

Edison

Free Books

Aldo Núñez Tovar

Índice general

1. Intel Edison **5**
- 1.1. Arranque . . . 6
- 1.2. Comunicación serie . . . 6
- 1.3. Configuración de la conexión inalámbrica . . . 7
- 1.4. Configuración de repositorios e instalación de paquetes . . . 8

2. Ambientes de desarrollo y programación **11**
- 2.1. Arduino IDE . . . 11
- 2.2. Intel XDK . . . 12
- 2.3. Intel System Studio . . . 12
- 2.4. Intel System Studio para microcontroladores . . . 13
- 2.5. Desarrollo en Linux en la propia tarjeta Edison . . . 13

3. Programación **15**
- 3.1. Biblioteca MRAA . . . 15
- 3.2. Compiladores e intérpretes . . . 15
- 3.3. Editores . . . 16
- 3.4. Programación de los GPIO's . . . 16
- 3.5. Salidas digitales . . . 16
- 3.6. Entradas digitales . . . 20
- 3.7. PWM . . . 22
- 3.8. Entradas analógicas . . . 26

4. Ejemplos **31**
- 4.1. Servo . . . 31

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.

El autor se ha enfocado en un lector que tenga conocimientos básicos de programación. A pesar de que existen distintos ambientes de desarrollo, como el IDE de Arduino, el enfoque escogido ha sido la programación directa en la tarjeta Edison.

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`.

Para cualquier comentario o sugerencia pueden dirigirse al autor a través del siguiente correo electrónico: `anunez20@gmail.com`

Capítulo 1

Intel Edison

La tarjeta Edison fue presentada por Intel en el año 2014. Cuenta con un procesador dual Atom a 500 MHz. Además de un procesador Quark a 100 MHz, que se encarga de las operaciones de entrada y salida. Tiene una memoria flash de 4GB, 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: Inter Integrated Circuits (I2Cs), Serial Peripheral Interface buss (SPIs) y UART.

Tiene salidas y entradas digitales, salidas PWM y entradas para leer señales analógicas.

Características principales:

1. Intel dual-core Atom a 500MHz
2. Coprocesador Quark a 100 MHz
3. 4GB de memoria flash
4. 1GB de memoria RAM
5. Sistema operativo Yocto
6. WiFi
7. BlueTooth 4.0
8. I2C
9. SPI
10. UART
11. 20 salidas digitales, 4 salidas PWM
12. 6 entradas analógicas

1.1. Arranque

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

Opción 1:

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

Opción 2:

1. Mueva el `switch1` en la dirección del conector micro usb `J16`
2. Conecte una fuente de alimentación de 7.5 - 12 Volts a `J1`. Aunque una fuente de 5 Volts también funciona.
3. Ahora, para establecer la comunicación serie conecte su computadora a la tarjeta Edison, a través del conector micro usb `J3`
4. Si la tarjeta Edison arrancó correctamente, debe encenderse el led `DS1`

Figura 1.1: Tarjeta de desarrollo Intel Edison Arduino

1.2. Comunicación serie

1. En su computadora abra una consola, busque en el subdirectorio `/dev` el archivo `ttyUSB0`. Puede ejecutar 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`.
El IP asignado puede ser diferente en cada caso. Además, cada vez que arranque Edison el valor del IP también puede cambiar.

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
Si desea crear una nueva cuenta de usuario que utilice para editar y compilar sus programas, use el comando `addduser`:

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

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 core2-32 http://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 la forma correcta verifique que el archivo existe y tiene la información de los repositorios.

3. El comando opkg Open PaKage Managements (OPKGs) 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. Despliegue la información de los archivos de este subdirectorio y verifique.
   En caso de que existan repositorios duplicados elimine o comente esa línea (para comentar anteponga el símbolo # al inicio de la línea en cuestion).
   Si no marca errores el sistema 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
```

Capítulo 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

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.

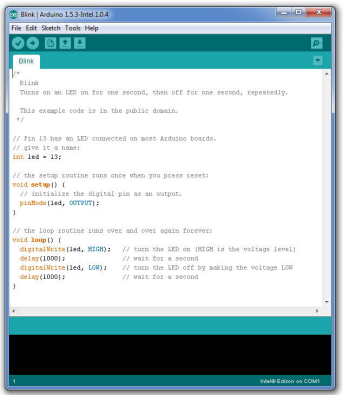


Figura 2.1: Arduino IDE

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

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`

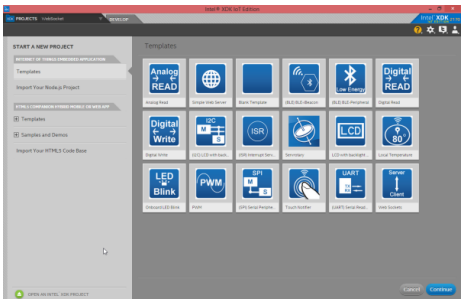


Figura 2.2: Intel XDK

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`

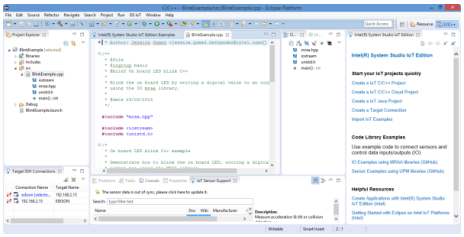


Figura 2.3: Intel System Studio

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:

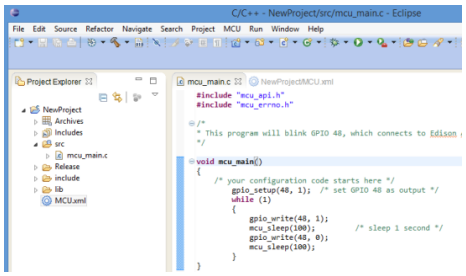


Figura 2.4: Intel System Studio para Microcontroladores

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

2.5. Desarrollo en Linux en la propia tarjeta Edison

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

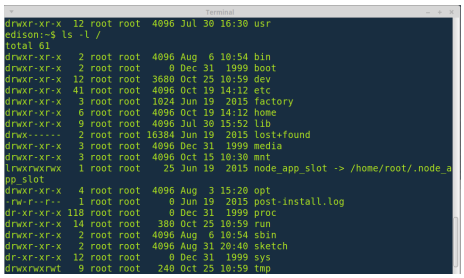


Figura 2.5: Consola

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.

Capítulo 3

Programación

La programación vamos a llevarla a cabo directamente en la tarjeta Edison. Para ello, nos conectamos a través del puerto `usb` o de la conexión inalámbrica, como ya se ha mencionado.

Para el manejo de los `GPIOs`, vamos a utilizar la biblioteca `libmraa`, los compiladores e intérpretes instalados y un editor de texto.

Es importante recalcar que para programar directamente en la tarjeta Edison, se requieren conocimientos básicos del sistema operativo Linux. Así como del proceso de compilación y uso de intérpretes.

3.1. Biblioteca MRAA

La biblioteca `libmraa`, `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 `libmraa` viene pre instalada en el sistema Yocto.

3.2. Compiladores e intérpretes

Los compiladores de `C` y `C++` vienen instalados. Estos compiladores son libres, del proyecto `GNU`. La versión instalada es la `4.9.1`

El intérprete de `Python` también viene instalado, la versión es `2.7.3`.

Para ejecutar los programas en JavaScript se usa Node.js, la versión instalada es la v.4.4.3

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

3.3. Editores

Antes de empezar a programar debemos instalar un editor de texto. Podemos escoger nano o vim

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

o

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

3.4. Programación de los GPIO's

3.5. Salidas digitales

Vamos a mostrar cómo se programan las salidad digitales utilizando la biblioteca libmraa. Se va a programar una de las salidas digitales de la figura 3.1. El programa producirá el parpadeo del led DS2 que está conectado, a su vez, a la terminal digital 13. En este ejemplo, hemos incluido versiones en C, C++, Python y JavaScript para mostrar la gama de opciones disponible.

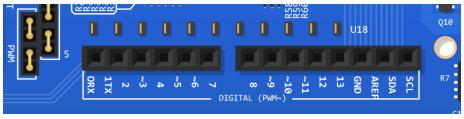


Figura 3.1: E/S digitales

Versión en C:

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

#include <mraa.h>

enum STATE { OFF, ON };      /* OFF = 0, ON = 1 */
const int LED_PIN = 13;      /* 13 ~ led DS2 */
```

```
11 const useconds_t T = 1e6;   /* 1e6 ~ 1 sec */
12
13 int
14 main ( void )
15 {
16     mraa_init ();                              /* initialize mraa */
17     mraa_gpio_context led;                     /* create access to gpio
           pin */
18     led = mraa_gpio_init ( LED_PIN );
19     mraa_gpio_dir ( led, MRAA_GPIO_OUT );   /* set gpio direction to
           out */
20
21     while ( 1 )                                /* blink indefinitely
          until (Ctrl + c) */
22     {
23         mraa_gpio_write ( led, ON );           /* turn on led */
24         usleep ( T );                          /* wait for 1 second */
25         mraa_gpio_write ( led, OFF );          /* turn off  led */
26         usleep ( T );                          /* wait for 1 second */
27     }
28
29     mraa_gpio_close ( led );                   /* close and exit */
30
31     return ( MRAA_SUCCESS );
32 }
```

Compilar:

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

Ejecutar como súper usuario:

```
# ./blink
```

Versión en C++:

```
1  // Program: blink.cpp
2  // Description: blink LED
3  // Author: Aldo Nunez
4
5  #include <mraa.hpp>
6
7  enum STATE { OFF, ON };          // OFF = 0, ON = 1
8  const int LED ( 13 );            // 13 ~ led DS2
9  const useconds_t T ( 1e6 );      // 1e6 ~ 1 sec
10
11 int
```

```
12 main ( void )
13 {
14     // create and initialize gpio pin
15     mraa::Gpio led ( LED );
16     // set gpio direction to out
17     mraa::Result response = led.dir ( mraa::DIR_OUT );
18
19     while ( true )              // blink indefinitely until (Ctrl +
           c)
20     {
21         led.write ( ON );       // turn on led
22         usleep ( T );           // wait for 1 second
23         led.write ( OFF );      // turn off led
24         usleep ( T );           // wait for 1 second
25     }
26
27     return ( MRAA_SUCCESS );    // exit
28 }
```

Compilar:

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

Ejecutar como súper usuario:

```
# ./blink
```

Versión en Python;

```
1  '''
2   Program: blink.py
3   Description: blink LED
4   Author: Aldo Nunez
5  '''
6  import mraa as m
7  import time
8
9  # STATES
10 OFF = 0
11 ON = 1
12
13 # 13 ~ led DS2
14 LED = 13
15
16 # time sleep
17 T =  1.0     # 1.0 ~ 1 sec
18
```

```
led = m.Gpio ( LED )
led.dir ( m.DIR_OUT )

while True:
    led.write ( ON )
    time.sleep ( T )
    led.write ( OFF )
    time.sleep ( T )
```

Ejecutar como súper usuario:

```
# python blink.py
```

Versión en JavaScript:

```
// Program: blink.js
// Description: blink LED
// Author: Aldo Nunez


var m = require ( 'mraa' );      // module  mraa

const STATE  = { OFF: 0, ON: 1 };
const LED = 13;                  // 13 ~ led DS2
const T = 1000;                  // 1000 ~ 1 msec

var led = new m.Gpio ( LED );    // choose pin
led.dir ( m.DIR_OUT );           // set the gpio direction


// Name: sleep
// Input: The number of milliseconds
// Output:
// Description: Produce a delay of n milliseconds
function sleep ( n )
{
    var start = new Date().getTime();
    while ( ( new Date().getTime () - start ) < n )
    {}
}

while ( 1 )
{
    led.write ( STATE.ON );
    sleep ( T );
    led.write ( STATE.OFF );
```

```
32      sleep ( T );
33  }
```

Ejecutar como súper usuario:

```
# node blink.js
```

3.6. Entradas digitales

Vamos a leer el valor de una entrada digital. Seleccionamos uno de las 14 terminales de la figura 3.1. En esto ejemplo se ha escogido la terminal 5. Por defecto, las entradas digitales están en alto. Para leer un valor 0, conectamos la terminal 5 a tierra.

Versión en C++

```
1  // Program: digital_input.cpp
2  // Description: Read digital input and display its value
3  // Author: Aldo Nunez
4
5  #include <iostream>
6  #include <mraa.hpp>
7
8  using namespace mraa;
9  using std::cout;
10
11 const int PIN_INPUT ( 5 );
12
13 int
14 main ( void )
15 {
16     int iopin ( PIN_INPUT );
17     Gpio* gpio = new Gpio ( iopin );
18
19     if ( gpio == NULL )
20     {
21         return ERROR_UNSPECIFIED;
22     }
23
24     Result input = gpio -> dir ( DIR_IN );
25
26     if ( input != SUCCESS )
27     {
28         printError ( input );
29         return ERROR_UNSPECIFIED;
```

```
30      }
31
32      int value ( gpio -> read () );
33
34      cout << "MRAA Version: " << mraa_get_version () << "\n";
35      cout << "Platform: " << mraa_get_platform_name () << "\n";
36      cout << "input pin_" << PIN_INPUT << ": " << value  << "\n";
37
38      delete ( gpio );
39
40      return ( MRAA_SUCCESS );
41  }
```

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 digital input and display its value
4   Author: Aldo Nunez
5  '''
6
7  import mraa as m
8
9  PIN_INPUT = 5
10 pin_input = m.Gpio ( PIN_INPUT )
11 pin_input.dir ( m.DIR_IN )
12
13 value  = pin_input.read ()
14
15 print ( "MRAA Version: " +  m.getVersion () );
16 print ( "Platform: " +  m.getPlatformName () );
17 print ( "Input pin_" + str ( PIN_INPUT ) + ": " + str ( value ) )
```

Ejecutar como súper usuario:

```
# python digital_input
```

3.7. PWM

Edison tiene 4 salidas PWM (Pulse Width Modulation). Por defecto, están configuradas como salidas PWM las terminales 3, 5, 6 y 9. Aunque también se pueden usar las terminales 10 y 11, simplemente moviendo los jumpers J11 y J12.

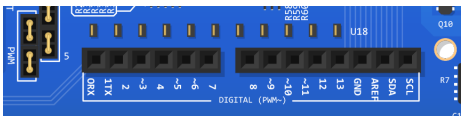


Figura 3.2: E/S digitales

Este programa genera una salida PWM. El usuario debe introducir el porcentaje de ciclo útil (ducty cycle) deseado.

Versión en C++

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

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

using namespace mraa;
using std::cout;


enum PWM { PWM0 = 3, PWM1 = 5, PWM2 = 6, PWM3 = 9 } pwm_port;
const int T ( 20000 );      // T ~ period in usec

// Name:         isValid
// Parameters:   pointer to 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 isValid ( char* );

int
main ( int argc, char* argv [] )
{
    if ( argc < 2 )
    {
        cout << "Usage: " << argv [ 0 ] << " <number>" << std::endl;
```

```
30          return 1;
31      }
32
33      if ( !isValid ( argv [ 1 ] ) )
34      {
35          cout << "Invalid argument...." << "\n";
36          cout << "Argument must be a number between: 0.0 - 100.0" <<
                "\n";
37          return 1;
38      }
39
40      float v ( atof ( argv [ 1 ] ) );        // percentage value
41
42      if ( ( v > 100.0 ) || ( v  < 0.0 ) )
43      {
44          cout << "<number> must be between: 0.0 and 100.0" << std::
                endl;
45          return 1;
46      }
47      pwm_port = PWM0;
48      Pwm* pwm = new Pwm ( pwm_port );
49      if ( pwm == NULL )
50      {
51          return ERROR_UNSPECIFIED;
52      }
53
54      pwm -> enable ( true );
55      pwm -> period_us ( T );     // f ~ 1/T
56      pwm -> write ( v / 100.0 );
57
58      cout << "MRAA Version: " << mraa_get_version () << "\n";
59      cout << "Platform: " << mraa_get_platform_name () << "\n";
60      cout << "Port Number: " << pwm_port << "\n";
61      cout << "Period: " << T * 1.0e-6 << " sec" << "\n";
62      cout << "Frequency: " << 1.0e6 / T << " Hz" << "\n";
63      cout << "Percentage of PWM: " << 100 *  pwm -> read ( ) <<  "%"
            << "\n";
64      cout << "Press \"Enter\" to finish." << "\n";
65
66      getchar ();
67      delete ( pwm );
68
69      return ( MRAA_SUCCESS );
70  }
71
72  bool
```

```
73  isValid ( char* str )
74  {
75      while ( *str != 0 )
76      {
77          if ( !isdigit ( *str )  &&  ( *str != '.' ) )
78          {
79              return false;
80          }
81          str++;
82      }
83      return true;
84  }
```

Compilar:

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

Ejecutar como súper usuario:

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

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 = { 'PWM0':3, 'PWM1':5, 'PWM2':6, 'PWM3':9 }
12  T = 20000     # Period: 20000 usec ~ 50 Hz
13
14  '''
15   Name:          isValid
16   Parameters:    string
17   Output:    boolean, 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 isValid ( string ):
22      try:
23          float (string)
24          return True
```

```
25      except:
26
27          return False
28
29  if len ( sys.argv ) < 2:
30      print ( "Usage: "+ sys.argv [ 0 ] + " <number>" )
31      sys.exit ()
32
33  if isValid ( sys.argv [ 1 ] ) == False:
34      print ( "Invalid argument...." )
35      print ( "Argument must be a number between: 0.0 - 100.0" )
36      sys.exit ()
37
38  value = float ( sys.argv [ 1 ] );
39  if value < 0.0 or value > 100.0:
40      print ( "<number> must be between: 0.0 - 100.0" )
41      sys.exit ()
42
43  PIN_PORT = PWM [ 'PWM0' ]
44  out = m.Pwm ( PIN_PORT )
45  out.period_us ( T )
46  out.enable ( True )
47  out.write ( value / 100.0 )
48
49  print ( "MRAA Version: " + m.getVersion () )
50  print ( "Platform: " + m.getPlatformName () )
51  print ( "Port Number: " +  str ( PIN_PORT ) )
52  print ( "Period: " + str ( T * 1.0e-6 ) + " sec" )
53  print ( "Frequency: " + str ( 1.0e6 / T ) + " Hz" )
54  print ( "Percentage of PWM: "),
55  print ( "{0:.2f}".format ( round ( 100 *  out.read (), 2 ) ) )
56
57  c = raw_input ( "Press \"Enter\" to finish." )
```

Ejecutar como súper usuario:

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

3.8. Entradas analógicas

Edison tiene 6 entradas analógicas (`A0 .. A5`). Usa un convertidor análogo digital (Analog to Digital Converters (ADCs)) de 12 bits, el cual puede configurarse también como un convertidor de 10 bits. El rango dinámico es de $0 - 5$ Volts

Figura 3.3: Entradas analógicas

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

Se pueden leer también los valores como tipo `float`, el rango de estos valores es: $0{,}0 \leq n \leq 1{,}0$

Este programa lee la entrada analógica de cualquiera de las 6 entradas disponibles. El usuario tiene que introducir como argumento el número del puerto analógico deseado.

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 <mraa.hpp>
7
8  using namespace mraa;
9  using std::cout;
10
11 // Name:         isValid
12 // Parameters:   pointer to string
13 // Output:       Boolean, true - if it is a valid input.
14 //               false - if it is not a valid input
15 // Description: Check if the input is an integer number
16 bool isValid ( char* );
17
18 enum ANALOG_IN { A0, A1, A2, A3, A4, A5 };  // analog ports: A0 - A5
19 const int NBITS  ( 12 );             // number of bits
20
21 int
22 main ( int argc, char* argv[] )
23 {
```

```
24      if ( argc < 2 )
25      {
26          cout << "Usage: " << argv [ 0 ] << " <port>" << std::endl;
27          cout << "<port>: 0..5" << std::endl;
28          return 1;
29      }
30
31      if ( !isValid ( argv [ 1 ] ) )
32      {
33          cout << "<port> must be between: 0..5" << "\n";
34          return 1;
35      }
36
37      float port ( atof ( argv [ 1 ] ) );
38
39      if ( ( port < A0 ) || ( port  > A5 ) )
40      {
41          cout << "<port> must be between: 0..5" << std::endl;
42          return 1;
43      }
44
45
46      uint16_t intValue ( 0 );        // variable  to read integer
            value
47      float floatValue ( 0.0 );       // variable to read float value
48
49      Aio* aInput = new Aio ( port );
50
51      if ( aInput == NULL )
52      {
53          return MRAA_ERROR_UNSPECIFIED;
54      }
55
56      aInput -> setBit ( NBITS );         // configure # of bits of
            ADC
57
58      cout << "MRAA Version: " << mraa_get_version () << "\n";
59      cout << "Platform: " << mraa_get_platform_name () << "\n";
60      while ( 1 )
61      {
62          intValue = aInput -> read ();           // reading integer
                value
63          floatValue = aInput -> readFloat ();    // reading float
                value
64          cout << "ADC " << port <<  " read integer: " << intValue <<
                "\n";
```

```
65          cout << "ADC " << port <<  " read float:  " <<  floatValue
                << "\n";
66          sleep ( 1 );
67      }
68
69      delete ( aInput );
70
71      return ( MRAA_SUCCESS );
72  }
73
74
75  bool
76  isValid ( char* str )
77  {
78      while ( *str != 0 )
79      {
80          if ( !isdigit ( *str ) )
81          {
82              return false;
83          }
84          str++;
85      }
86      return true;
87  }
```

Compilar:

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

Ejecutar como súper usuario:

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

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..5" )
18     sys.exit ()
19
20 if ( port > 5 ) or ( port < 0 ):
21     print ( "<Port>: 0..5" )
22     sys.exit ()
23
24 analogIn = m.Aio ( port )
25 analogIn.setBit ( NBITS )
26
27 print ( "MRAA Version: " +  m.getVersion () )
28 print ( "Platform: " + m.getPlatformName () )
29 print ( "Analog port: " + str ( port ) )
30
31 while True:
32     intValue =  analogIn.read ()
33     floatValue =  analogIn.readFloat ()
34         print ( "ADC " + sys.argv [ 1 ] + " read integer: " + str (
              intValue )  )
35         print ( "ADC " + sys.argv [ 1 ] + " read float: "  + str (
              floatValue ) )
36     time.sleep ( 1 )
```

Ejecutar como súper usuario:

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

Capítulo 4

Ejemplos

4.1. Servo

Las salidas PWM se usan, entre otras aplicaciones, para controlar los servo motores. El sentido de giro de las flechas de estos servo motores se controlan variando el ciclo útil (duty cicle) de la señal PWM. En el diagrama de tiempo de la figura 4.1 se observa que

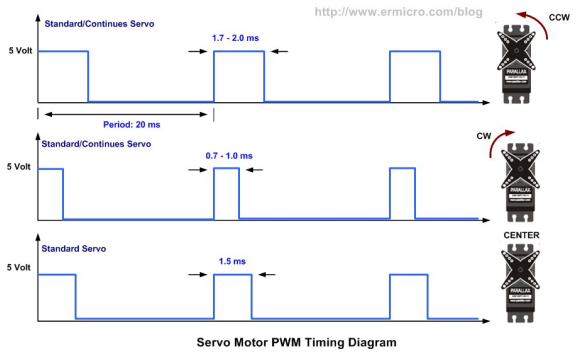


Figura 4.1: Diagrama de tiempo

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 de 3 a 5 volts.

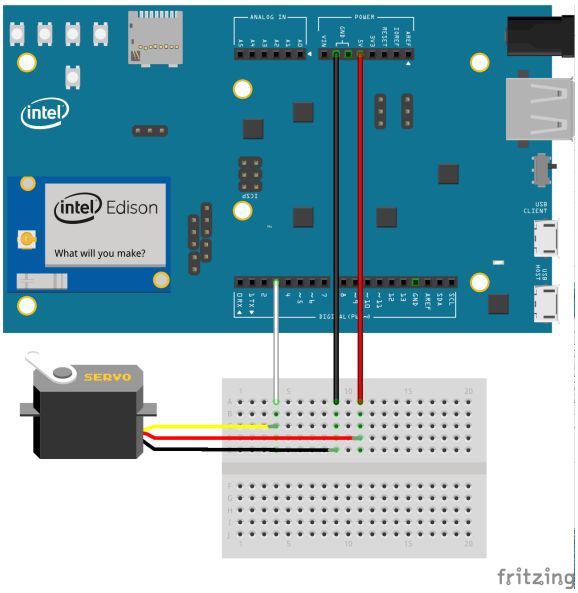


Figura 4.2: Diagrama de conexiones

Utilizamos una señal cuyo periodo es 20ms o frecuencia 50Hz

Version en C++

```
// Name: pwm.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 );
```

```
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      getchar ();
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 o 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
```

```
'''
 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 ):
    pwm.period_us ( t )
    pwm.enable ( True )
    pwm.pulsewidth ( d )
    return;

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

pwm = m.Pwm ( PORT )
pwm.period_us ( T )

if sys.argv [ 1 ] == 'CW':
    turn_servo ( pwm, T, CLKWISE )
elif sys.argv [ 1 ] == 'CCW':
    turn_servo ( pwm, T, CCLKWISE )
else:
    print ( "<DIR>: CW or CCW")
    sys.exit ()

print ( "MRAA Version: " + m.getVersion () )
print ( "Platform: " + m.getPlatformName () )
print ( "Port Number: " +  str ( PORT) )

c = raw_input ( "Press \"Enter\" to finish." )
```

Ejecutar como súper usuario:

```
# python servo.py <DIR>
<DIR>: CW o CCW
```

Glosario

ADC Analog to Digital Converter. 26

I2C Inter Integrated Circuit. 5

IDE Integrated Development Environment. 1, 11

OPKG Open PaKage Management. 9

SPI Serial Peripheral Interface bus. 5