Paralelo: 1 Grupo: 2

Materia: Aplicaciones Móviles y Servicios Telemáticos

Profesora: Msig. Adriana Collaguazo Jaramillo

Fecha de inicio: 14 de julio del 2019

Fecha de finalización: 20 de agosto del 2019

**INTEGRANTES**

* Luis Macas
* Viviana Mero
* Allan Orellana
* Richard Ruales

MANUAL TÉCNICO

Registro y notificación de pulsos de equipo médico en ambulancias

# Tabla de contenido

[Tabla de contenido 2](#_Toc17327784)

[MANUAL TÉCNICO 3](#_Toc17327785)

[1. Diagramas UML 3](#_Toc17327786)

[1.1. Diagrama de Comunicación del Sistema 3](#_Toc17327787)

[1.2. Diagrama de Actividades 3](#_Toc17327788)

[1.2.1. Historia de Usuario 1 3](#_Toc17327789)

[1.2.2. Historia de Usuario 2 3](#_Toc17327790)

[1.2.3. Historia de Usuario 3 4](#_Toc17327791)

[1.2.4. Historia de Usuario 4 4](#_Toc17327792)

[1.3. Diagrama Casos de Uso 4](#_Toc17327793)

[1.4. Diagrama de Clases 4](#_Toc17327794)

[1.5. Diagrama de Despliegue 5](#_Toc17327795)

[1.6. Diagrama Entidad - Relación 6](#_Toc17327796)

[1.7. Diagrama de Circuito 6](#_Toc17327797)

[2. Programación del microcontrolador PIC 6](#_Toc17327798)

[2.1 Requerimientos de hardware 6](#_Toc17327799)

[2.2Requerimientos de software 6](#_Toc17327800)

[2.3 Configuración de registros en MPLAB X 7](#_Toc17327801)

[2.3.1 Crear un proyecto en MPLAB X 7](#_Toc17327802)

[2.3.2 Configurar Timer0 13](#_Toc17327803)

[2.3.3 Configuración del módulo EXT\_INT 14](#_Toc17327804)

[2.3.4 Configuración del módulo EUSART 15](#_Toc17327805)

[2.4 Cargar el código al PIC 21](#_Toc17327806)

[2.5 Programación del Thinxtra 21](#_Toc17327807)

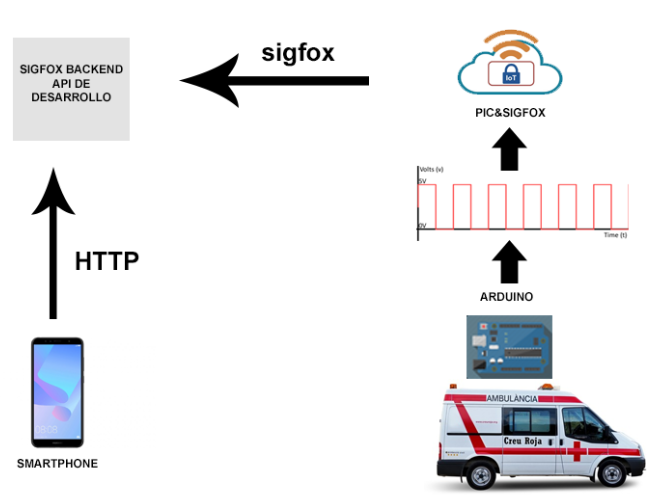
[3. Creación de callback 26](#_Toc17327808)

[4. Recursos de Software 29](#_Toc17327809)

# MANUAL TÉCNICO

## Diagramas UML

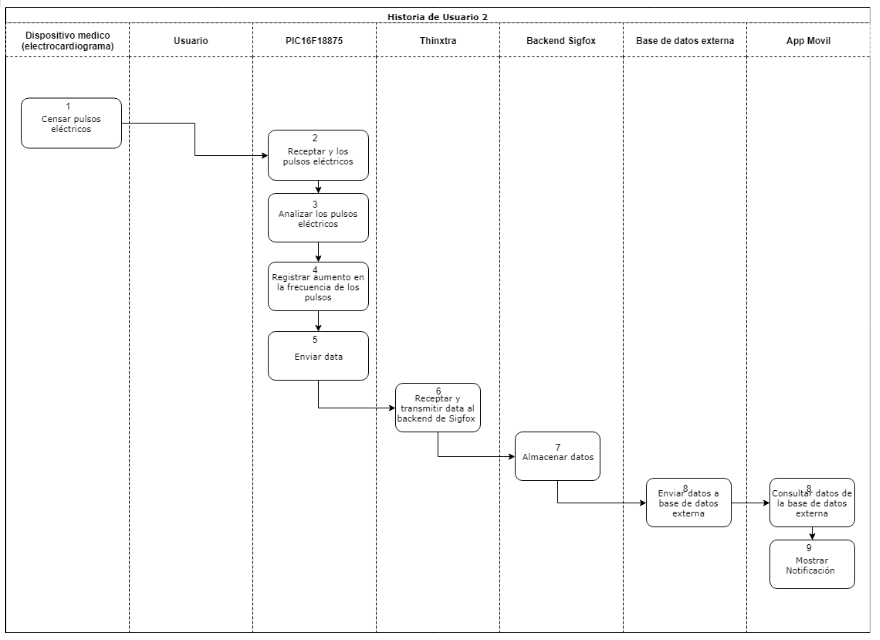
### Diagrama de Comunicación del Sistema



### Diagrama de Actividades

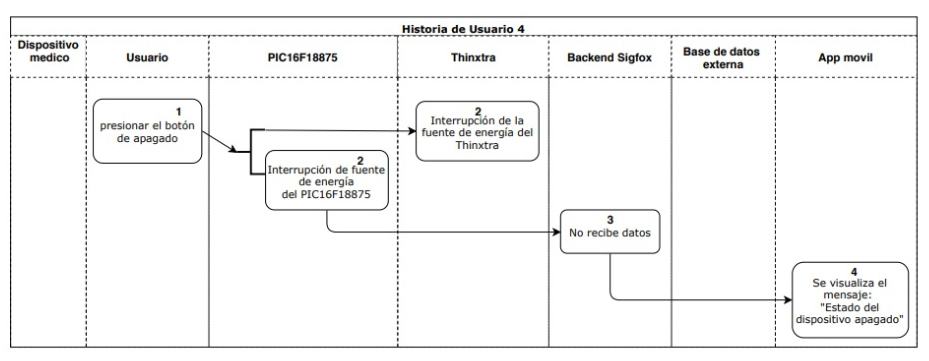
#### Historia de Usuario 1

#### Historia de Usuario 2

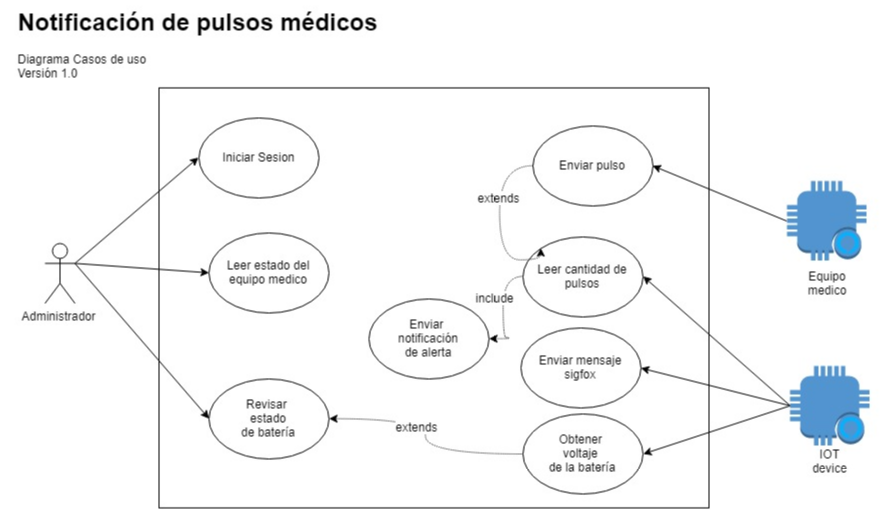


#### Historia de Usuario 3

#### Historia de Usuario 4

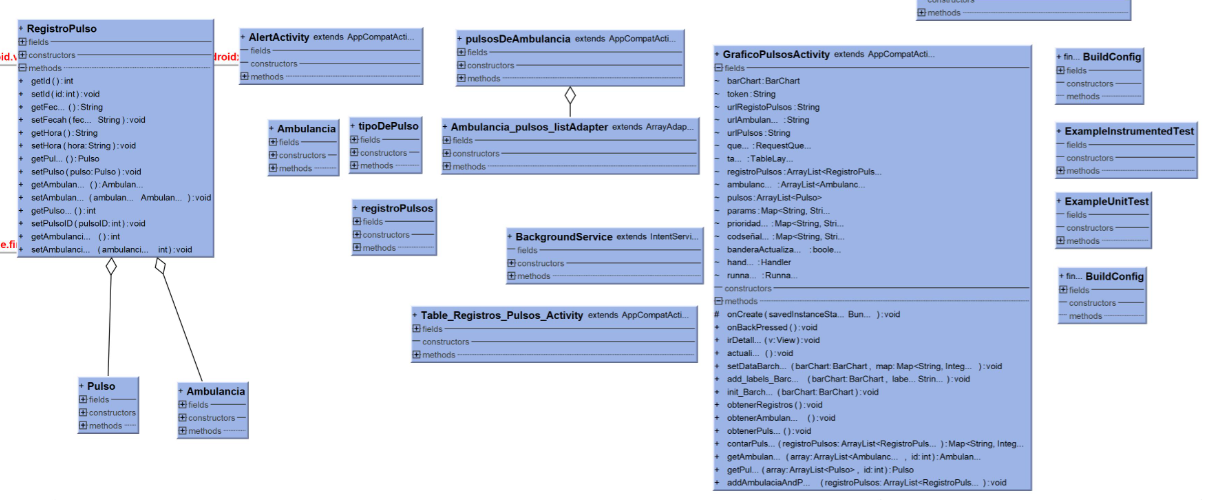


### Diagrama Casos de Uso

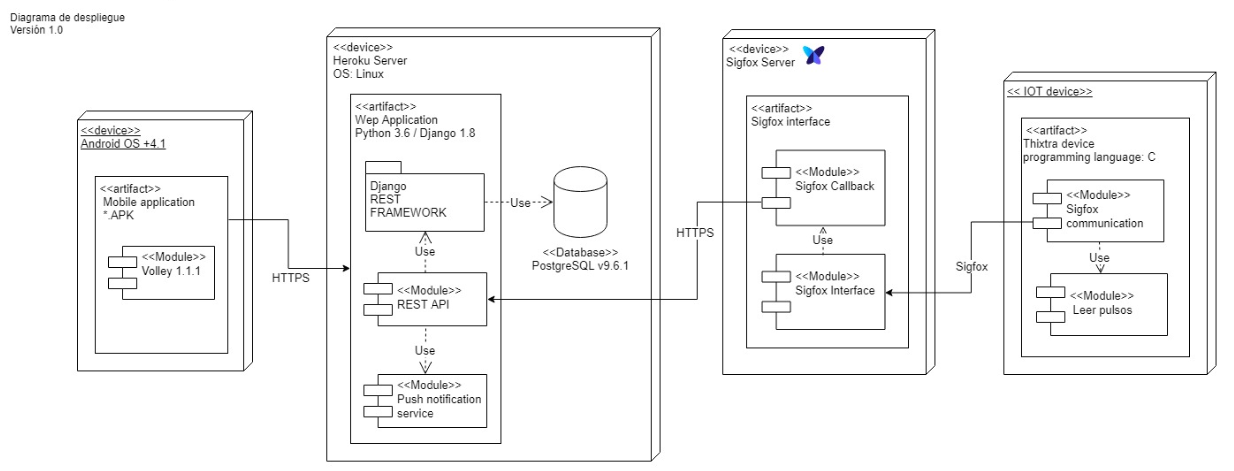


### Diagrama de Clases

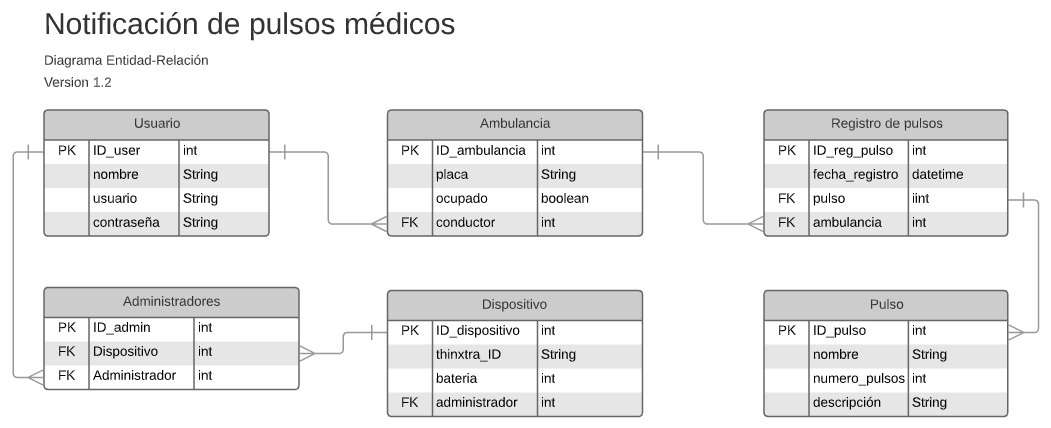




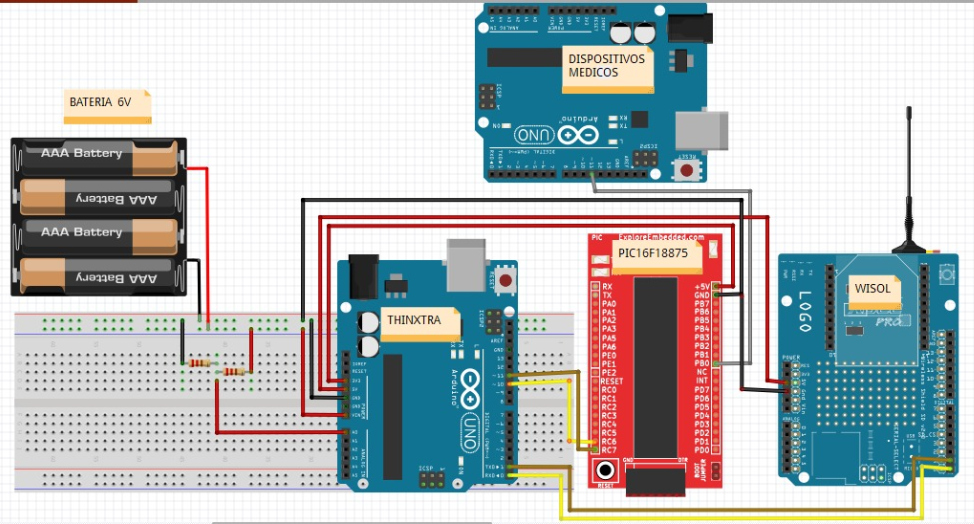
### Diagrama de Despliegue



### Diagrama Entidad - Relación



### Diagrama de Circuito



## Programación del microcontrolador PIC

## Requerimientos de hardware

PIC16F18875

Board Curiosity HPC

Cable USB

Jumper macho-macho

Jumper macho-hembra

Arduino Uno

Thinxtra sigfox Dev kit pro

6 pilas de 1.5 voltios

## 2.2Requerimientos de software

MPLAB X IDE v5.10

MPLAB Code Configurator (MCC) plug-in

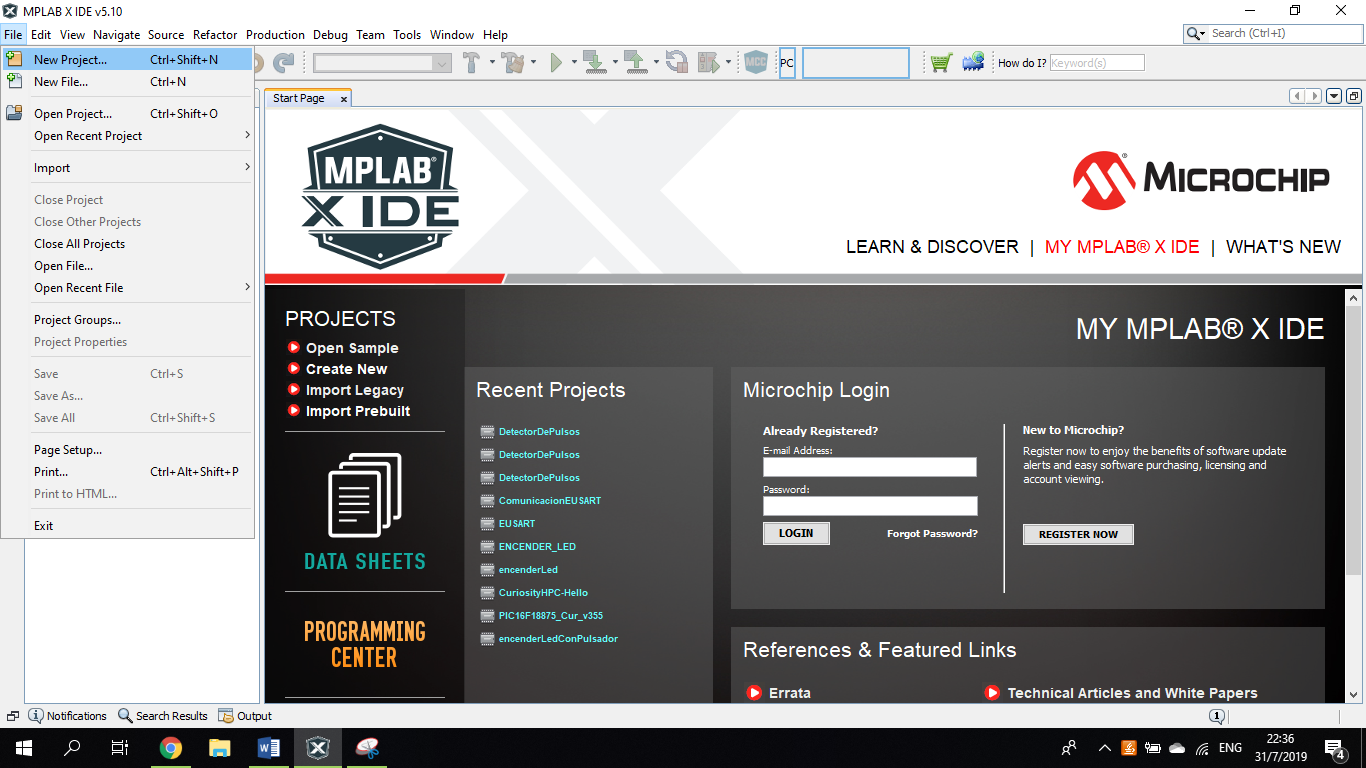
Arduino IDE

Github(enlace de proyecto):

## 2.3 Configuración de registros en MPLAB X

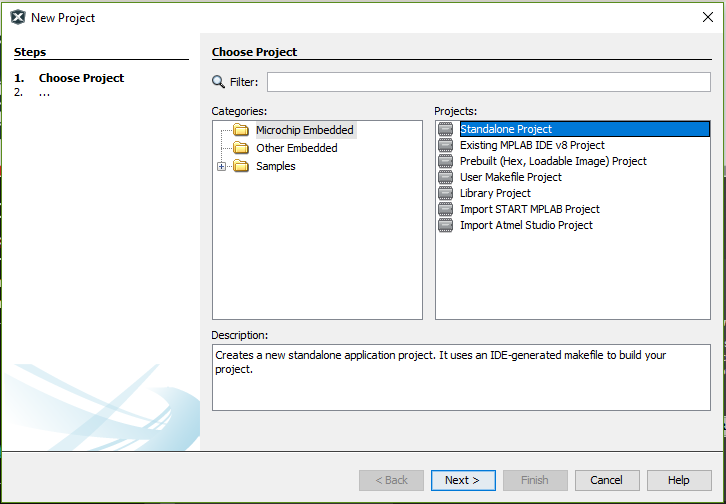
## 2.3.1 Crear un proyecto en MPLAB X

Ejecute el programa MPLAP X y proceda a seleccionar la opción **File ---> New Project**



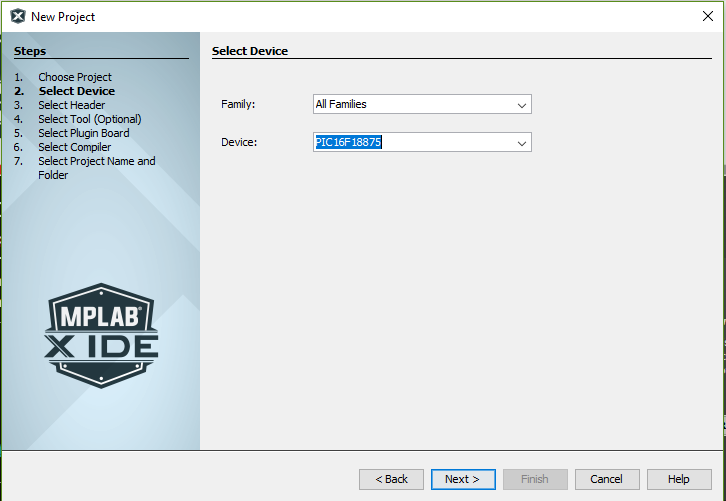
**Figura 1. Creación de un nuevo proyecto**

Se mostrará la ventana para seleccionar el tipo de proyecto. Se escogerá la opción **Standalone Project**, tal como se visualiza en la Figura 2. Luego haga clic en **Next.**



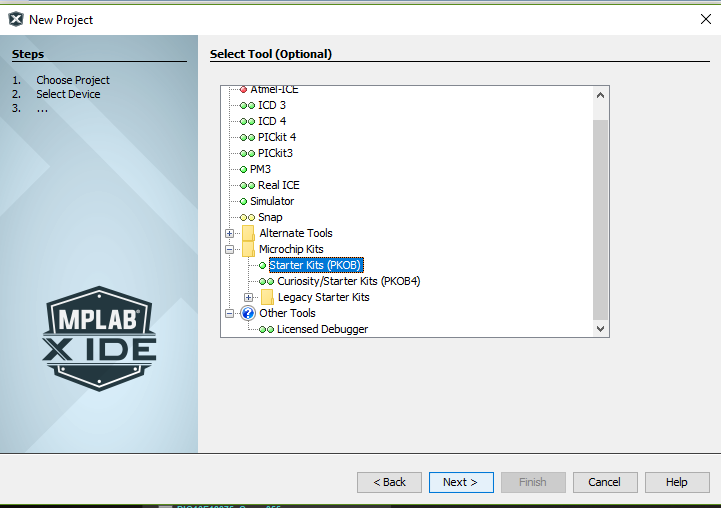
**Figura 2. Tipo de proyecto**

En la siguiente ventana **Select Device** proceda a escoger **All family** y el tipo de microcontrolador que se va a usar **PIC16f18875.** Tal como se muestra en la Figura 3. Luego haga clic en **Next**.



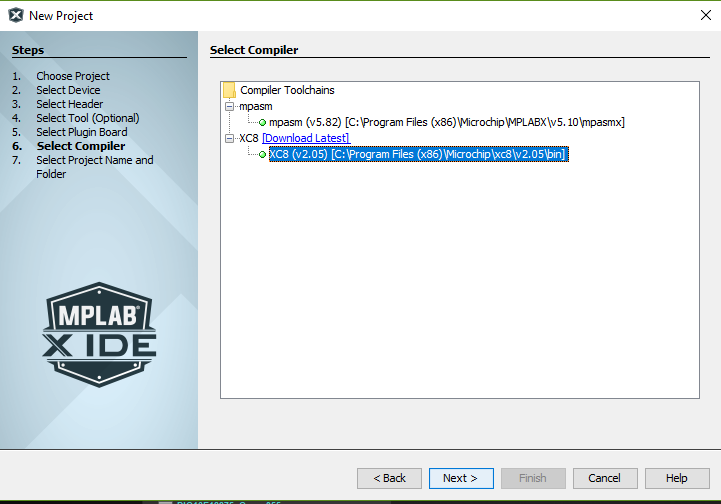
**Figura 3. Familia y tipo de microcontrolador**

Proceda a seleccionar **Starter kits (PKOB)** la cual es la herramienta que permitirá cargar el código al PIC por medio de la Board Curiosity. Tal como se muestra en la Figura 4. Luego haga clic en **Next**.



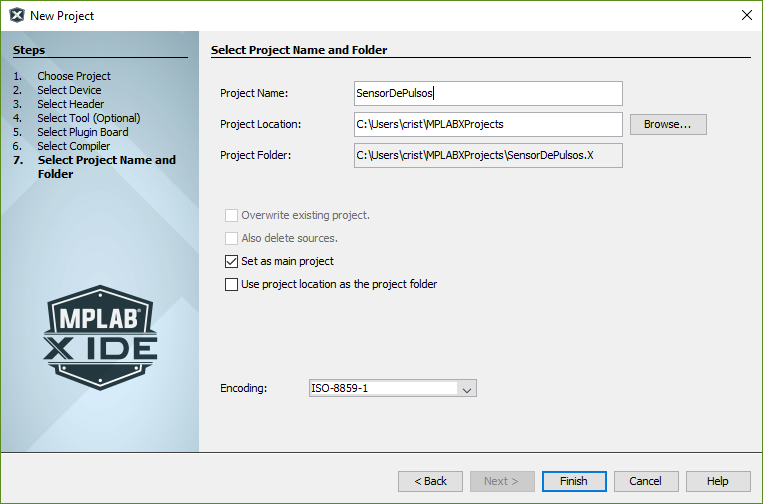
**Figura 4. Herramienta para cargar el código al PIC**

En la siguiente ventana **Select Compiler** proceda a escoger el compilador **XC8.** Tal como se muestra en la Figura 5. Luego haga clic en **Next**.



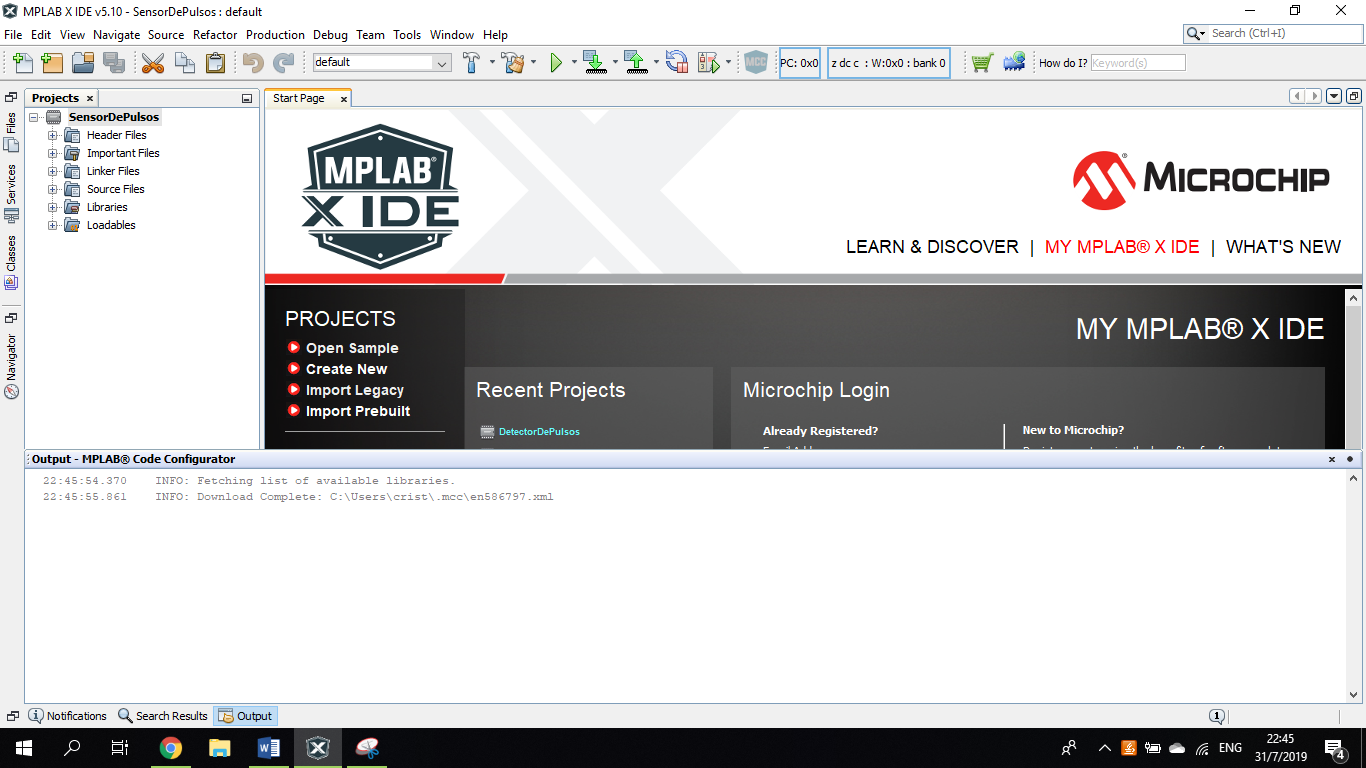
**Figura 5. Selección del compilador**

En la primera línea se coloca el nombre del proyecto y en la siguiente línea escoja la ruta donde va a crear su proyecto, en este caso, se dejará la ruta por defecto que asigna MPLAB X tal como se visualiza en la Figura 6. Para finalizar haga clic en Finish.



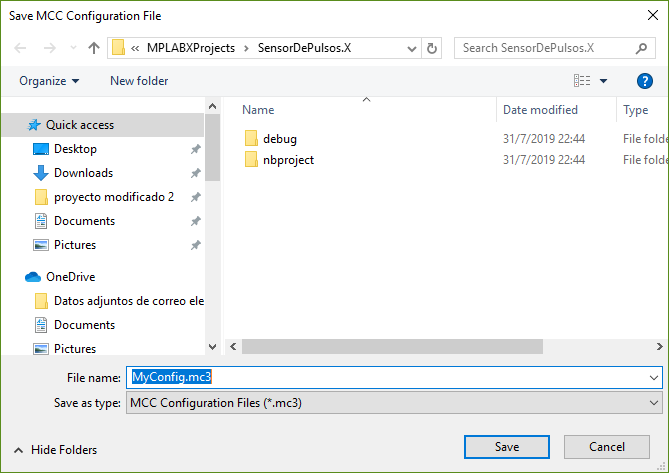
**Figura 6. Nombre y ruta del proyecto**

Una vez creado el proyecto en la parte izquierda de la ventana **Projects** observará todos los directorios que fueron creados. Haga clic el icono MCC (MPAB Code Configurator), esta herramienta permitirá la selección de modulos para la configuración de registros del PIC. Tal como se observa en la Figura 7.



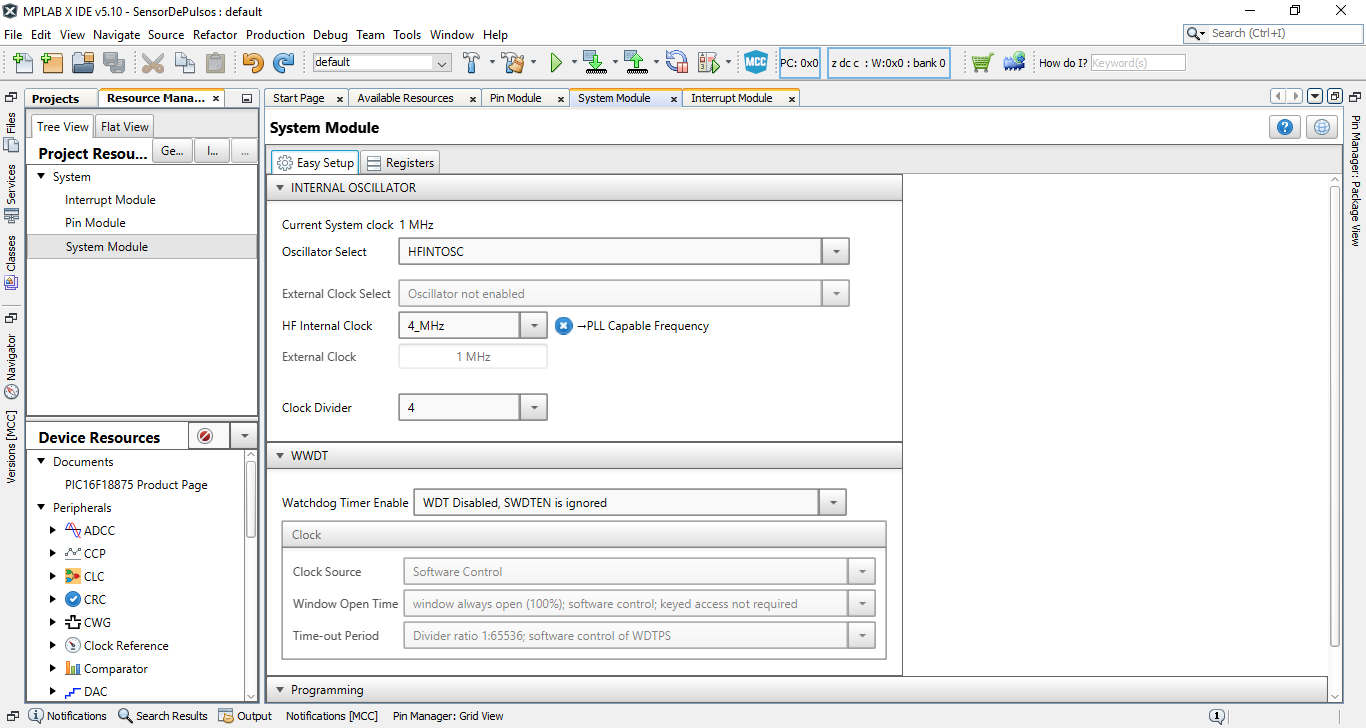
**Figura 7. Abriendo herramienta MCC**

Se abrirá una ventana que mostrara la carpeta donde se guardará, proceda a asignar un nombre al archivo .mc3 o mantenga el predeterminado, como en este caso, haga clic en **save**. Tal como se visualiza en la Figura 8.



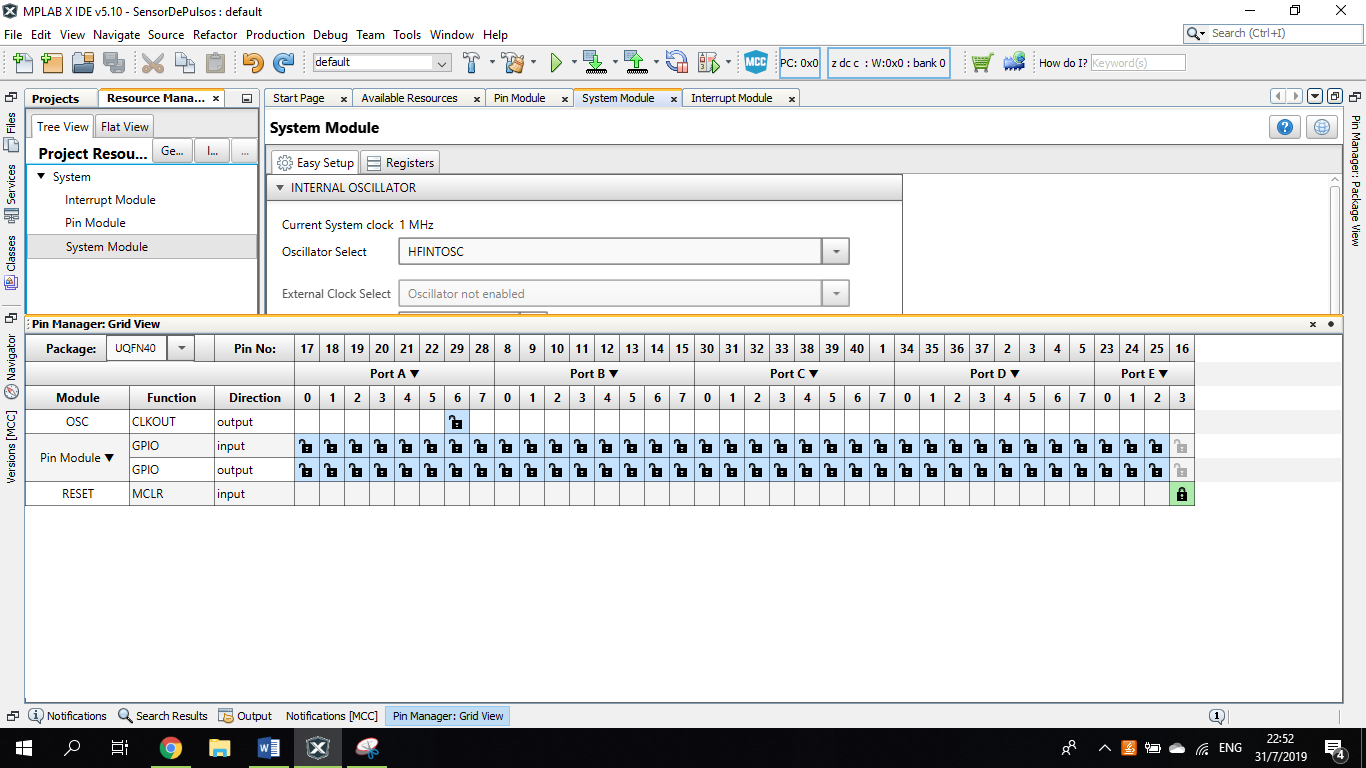
**Figura 8. Asignar nombre al archivo .mc3**

Una vez guardado el archivo en el paso anterior se mostrará la ventana de la herramienta MCC (MPAB Code Configurator), tal como se observa en la Figura 9.



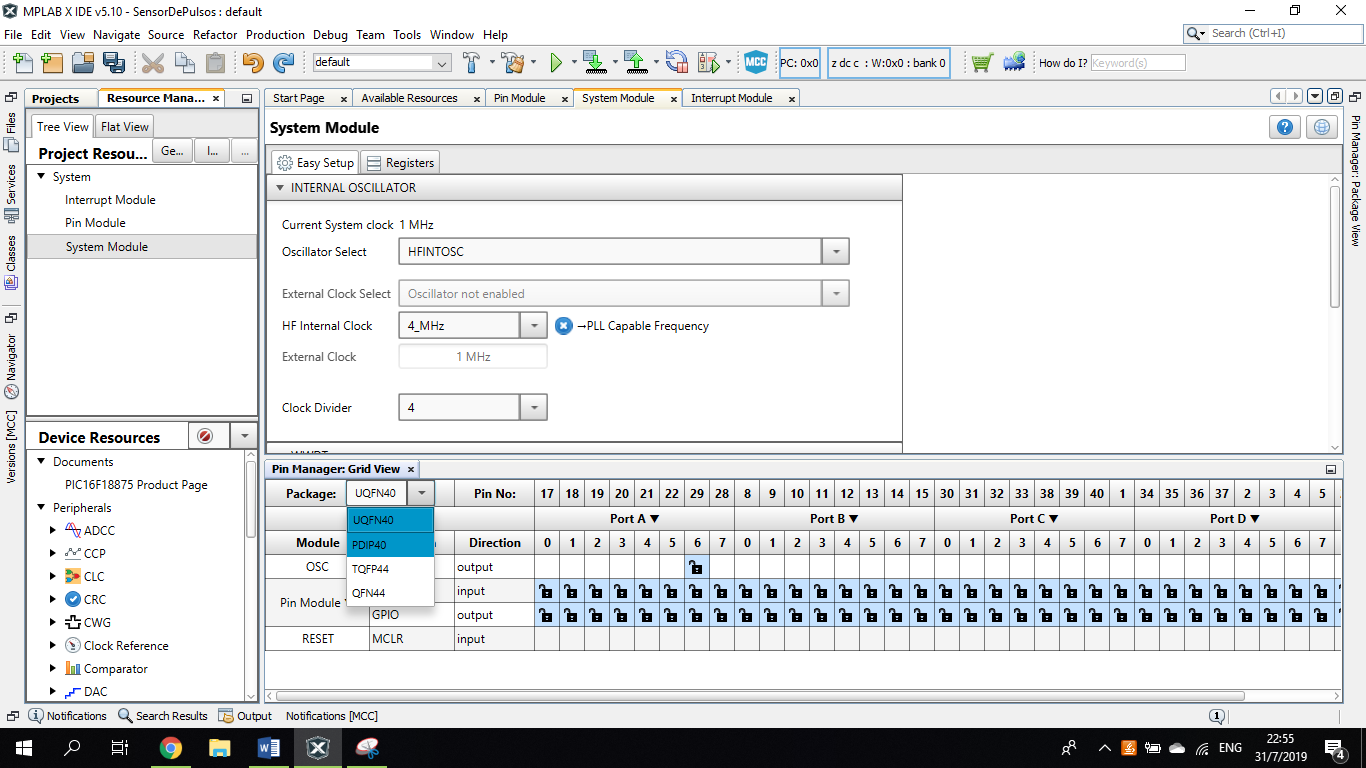
**Figura 9. Ventana MCC**

En la parte inferior de la ventana haga clic en **Pin Manage: Grid View** se desplegará un menú que muestra todos los modulos seleccionados por defecto.



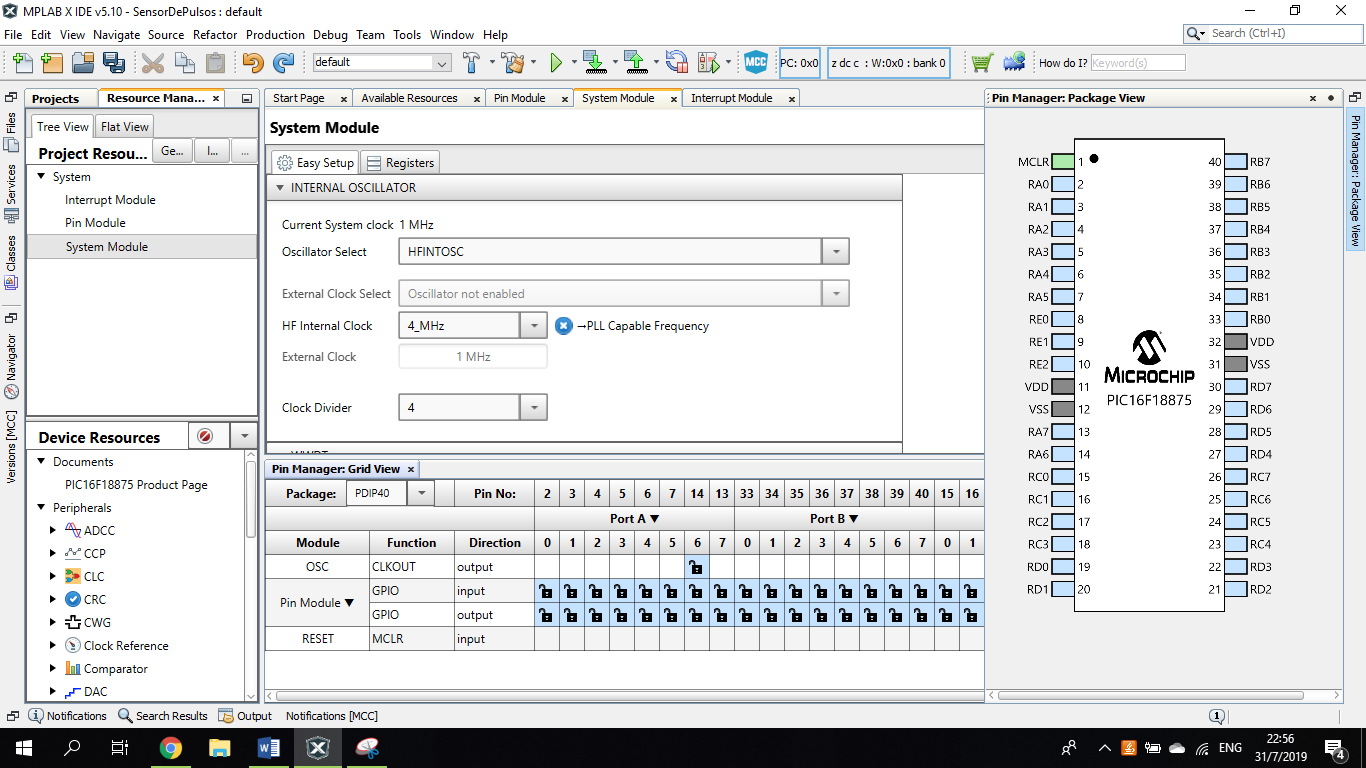
**Figura 10. Menú Pin Manager**

En **Pin Manage: Grid View** seleccione el **Package** PIDP40, tal como se observa en la Figura 11, correspondiente al empaquetado del microcontrolador PIC16F18875 que utilizaremos en este proyecto.



**Figura 11. Selección de Package PDIP40**

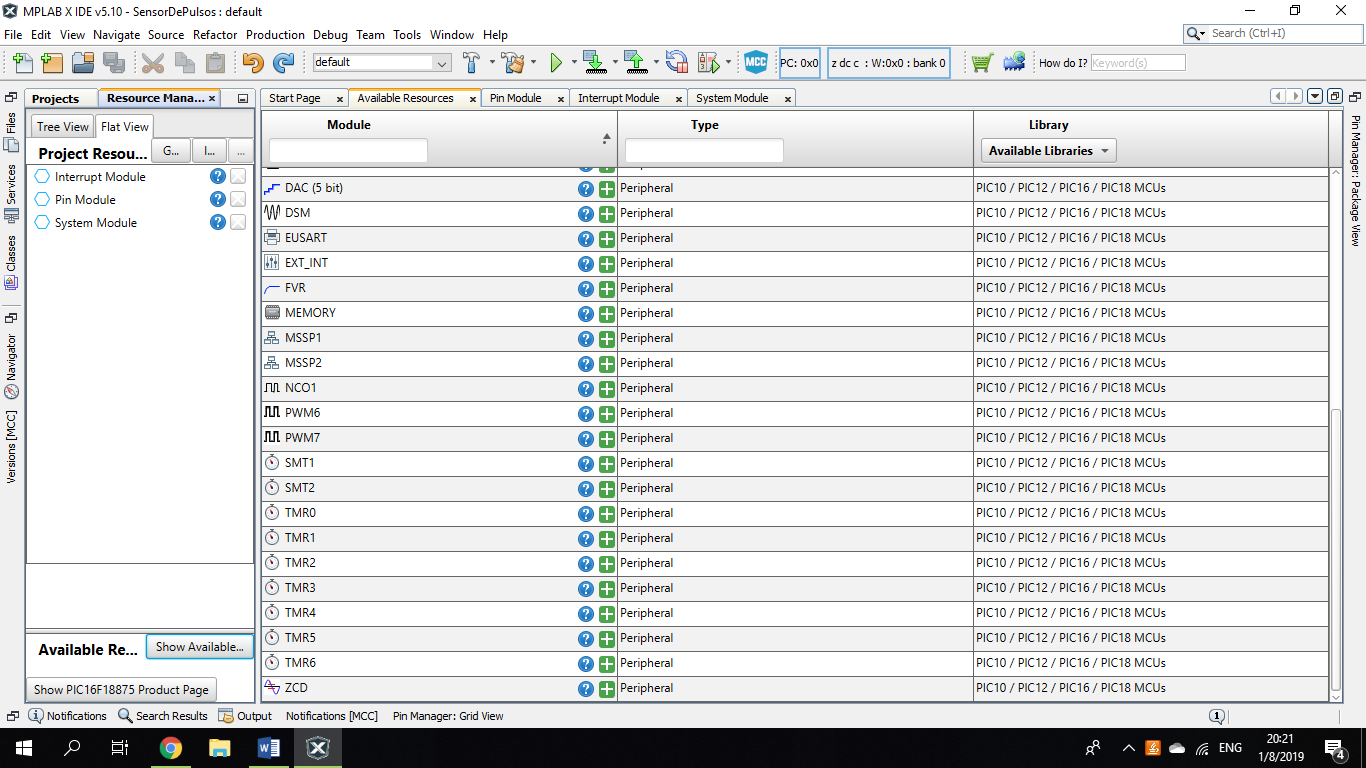
Proceda a dar clic en el menú **Pin Manager: Package View** para observar el grafico del PIC con sus respectivos pines, además proceda a configurar **Configurar System module**. Tal como se muestra en a Figura 12, además asegúrese Low-Voltage Programming este Enable.



**Figura 12. Configuración de System Module**

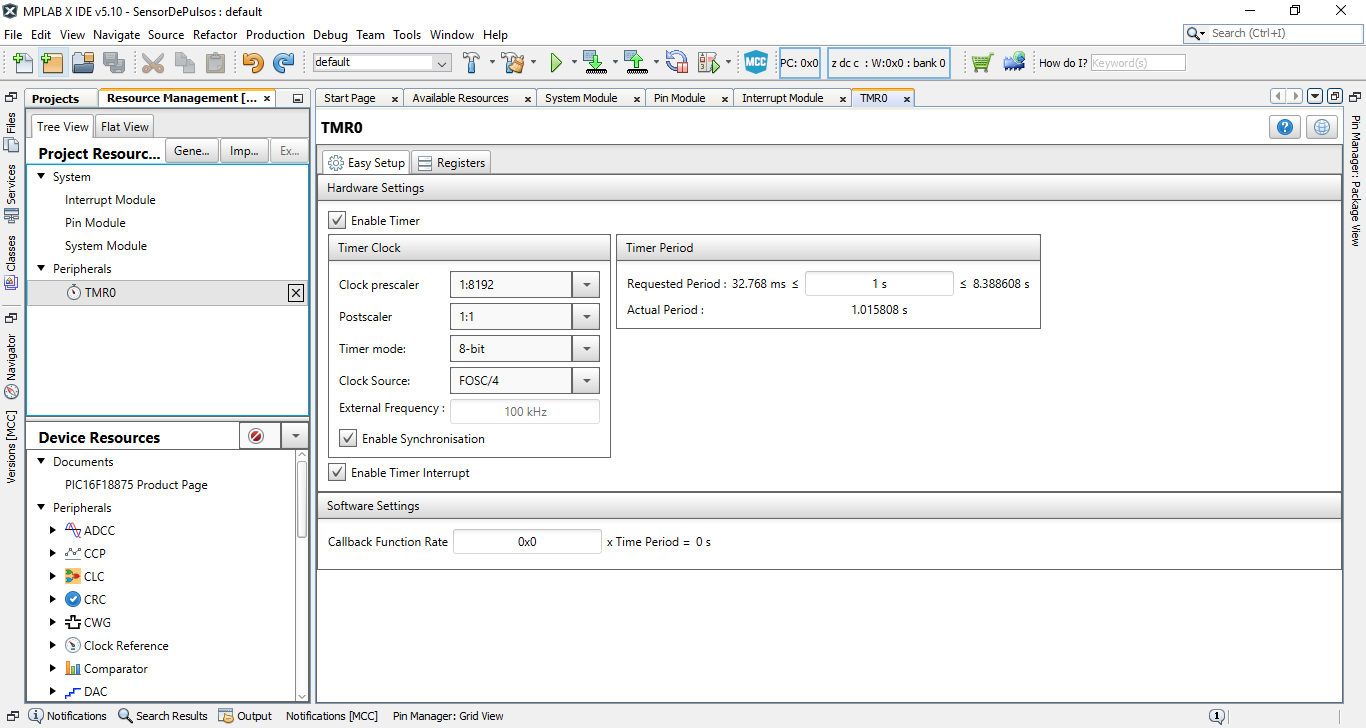
## 2.3.2 Configurar Timer0

Diríjase al menú **Resource Manager--->Flat View** y haga clic en **Show available Resources**, aparecerá una lista de modulos disponibles para agregar al proyecto, proceda a dar doble clic en TMR0(timer0). Tal como se muestra en a Figura 13.



**Figura 13. Agregando Timer0**

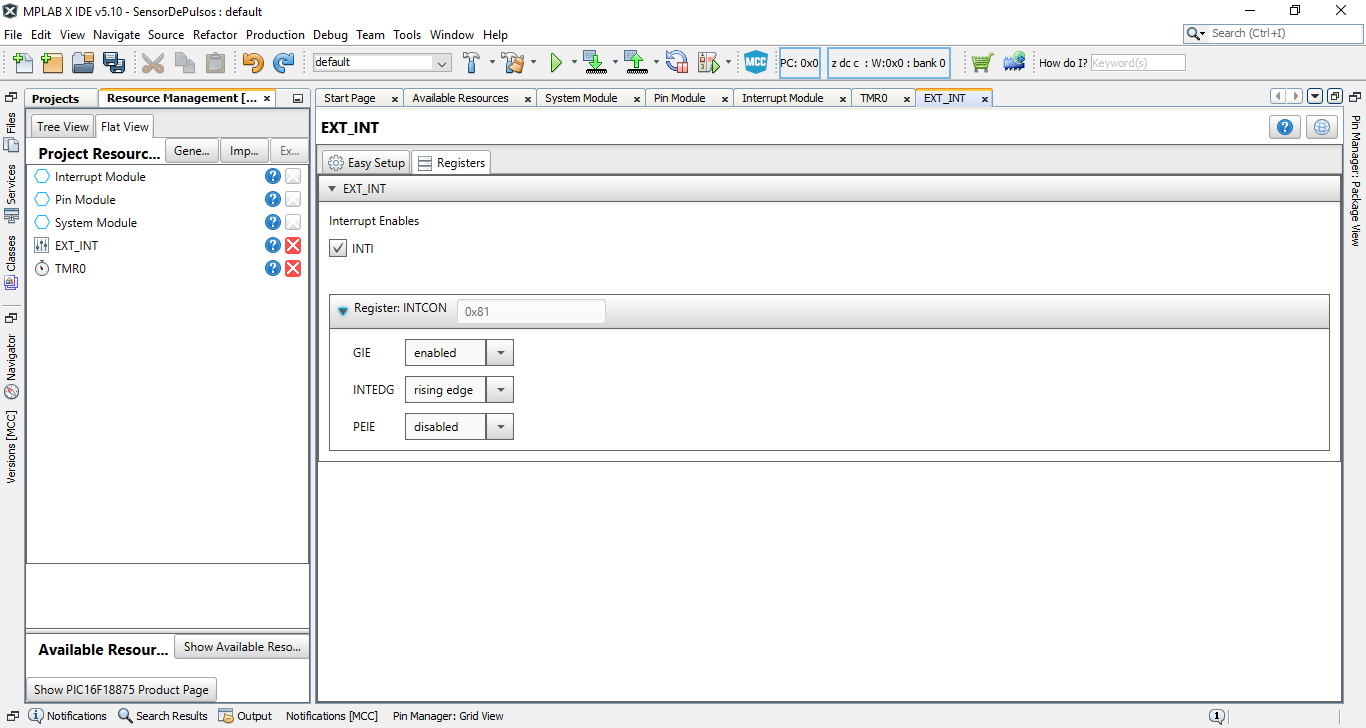
Proceda a realizar todas las configuraciones de los parámetros del timer0 tal como se observa en la Figura 14. Esto hará que cada segundo que pase se genere una interrupción.



**Figura 14. Configuración del timer0**

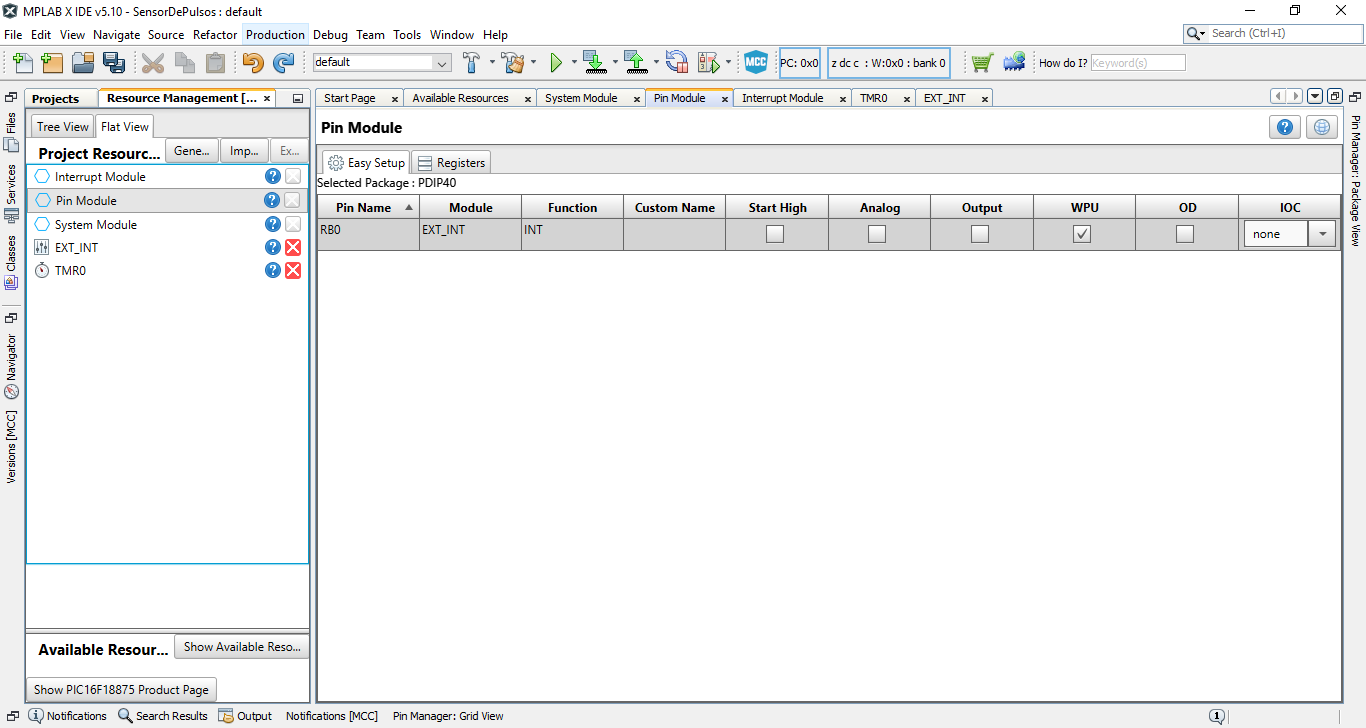
## 2.3.3 Configuración del módulo EXT\_INT

Diríjase al menú **Resource Manager--->Flat View** y haga clic en **Show available Resources**, proceda a dar doble clic en EXT\_INT (interrupciones externas) y proceda a configurarlo tal como se muestra en la Figura 15. Este módulo permite detectar los cambios que ocurran el pin RB0 del PIC.



**Figura 15. Configuración de interrupciones externas**

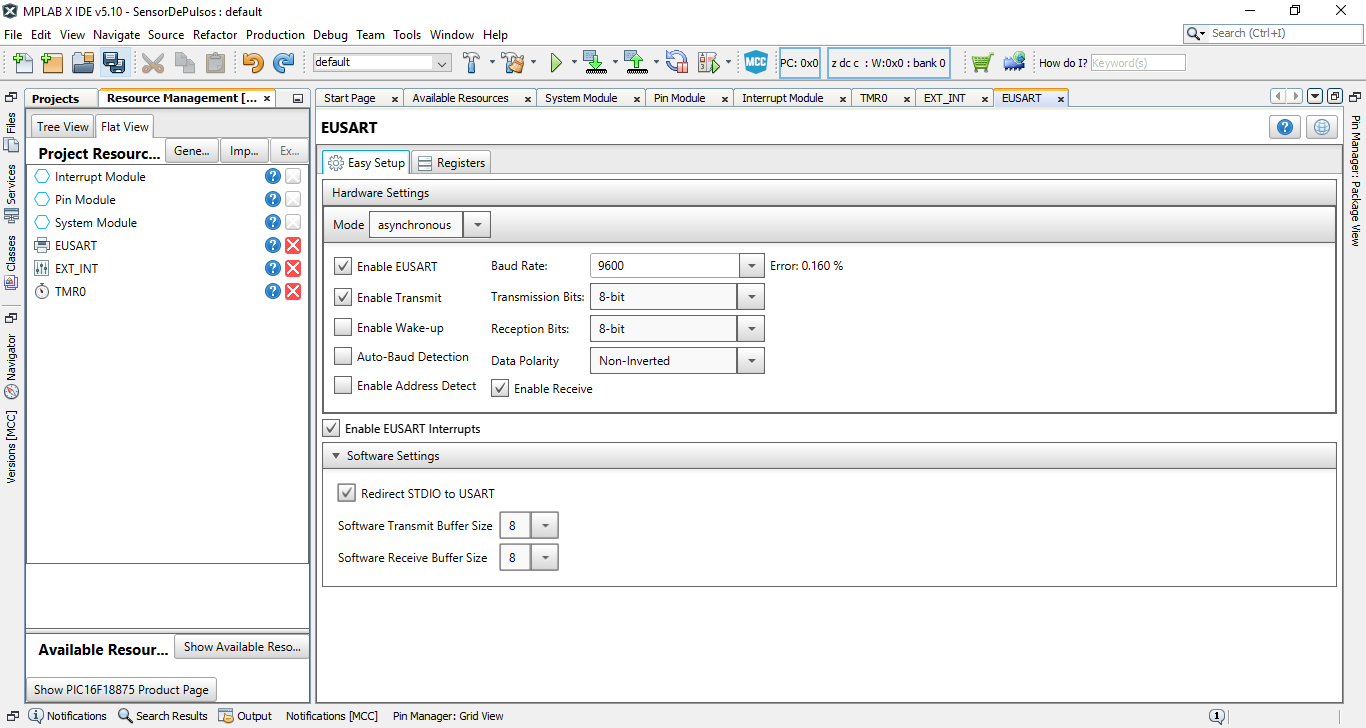
Diríjase al menú **Resource Manager--->Flat View** haga clic en **Pin Module** para ver el pin RB0 asignado por defecto, el cual se encargará de receptar algún cambio externo, además se debe habilitar WPDU.



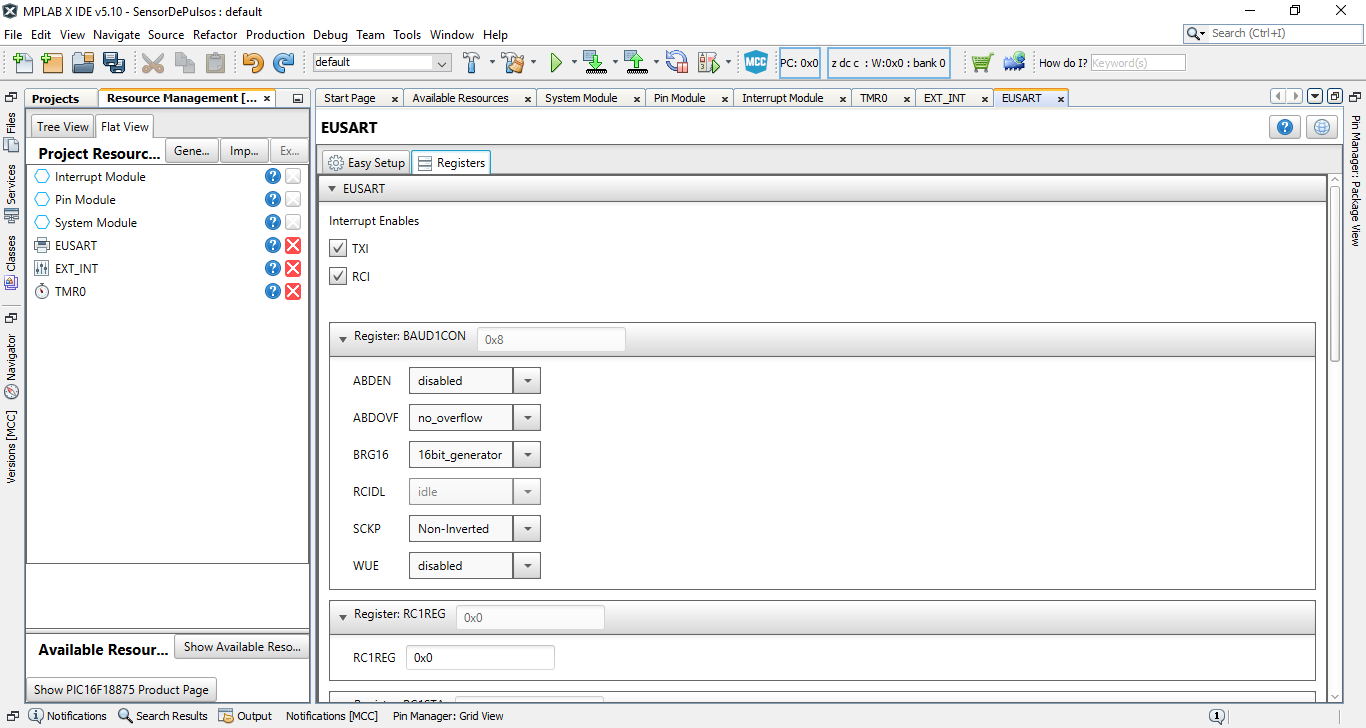
**Figura 16. Pines utilizados**

## 2.3.4 Configuración del módulo EUSART

Diríjase al menú **Resource Manager--->Flat View** y haga clic en **Show available Resources**, proceda a dar doble clic en EUSART y proceda a configurarlo tal como se muestra en la Figura 17.



**Figura 17. Agregando EUSART**

Diríjase al menú **Register** y asegúrese que se encuentren habilitados TXI y RXI. Tal como se observa en la Figura 18. TXI y RXI permiten el envió de datos desde el PIC16F18875 al Thinxtra por medio del protocolo serial UART.

**Figura 18. Habilitando TX y RX**

En la ventana **Pin Module** se puede observar cómo se establece predeterminadamente los pines RC6 y RC7 como TX y RX respectivamente.

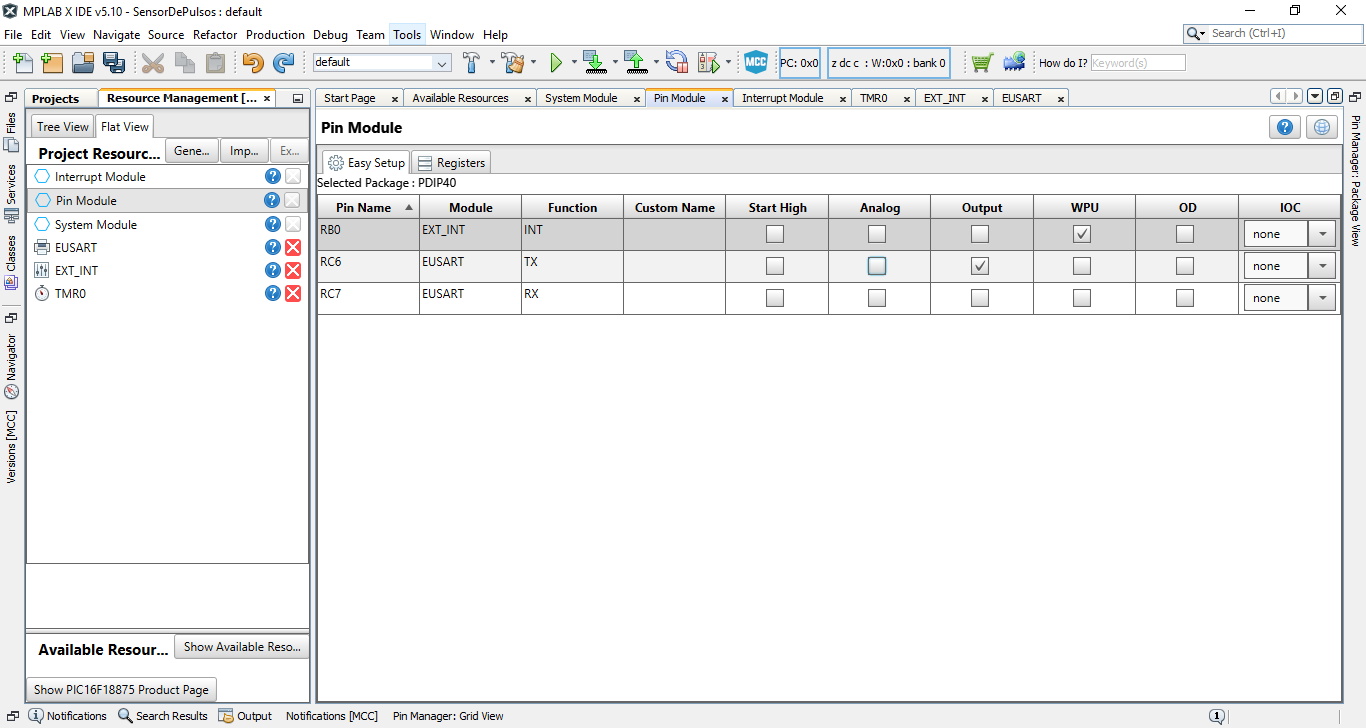
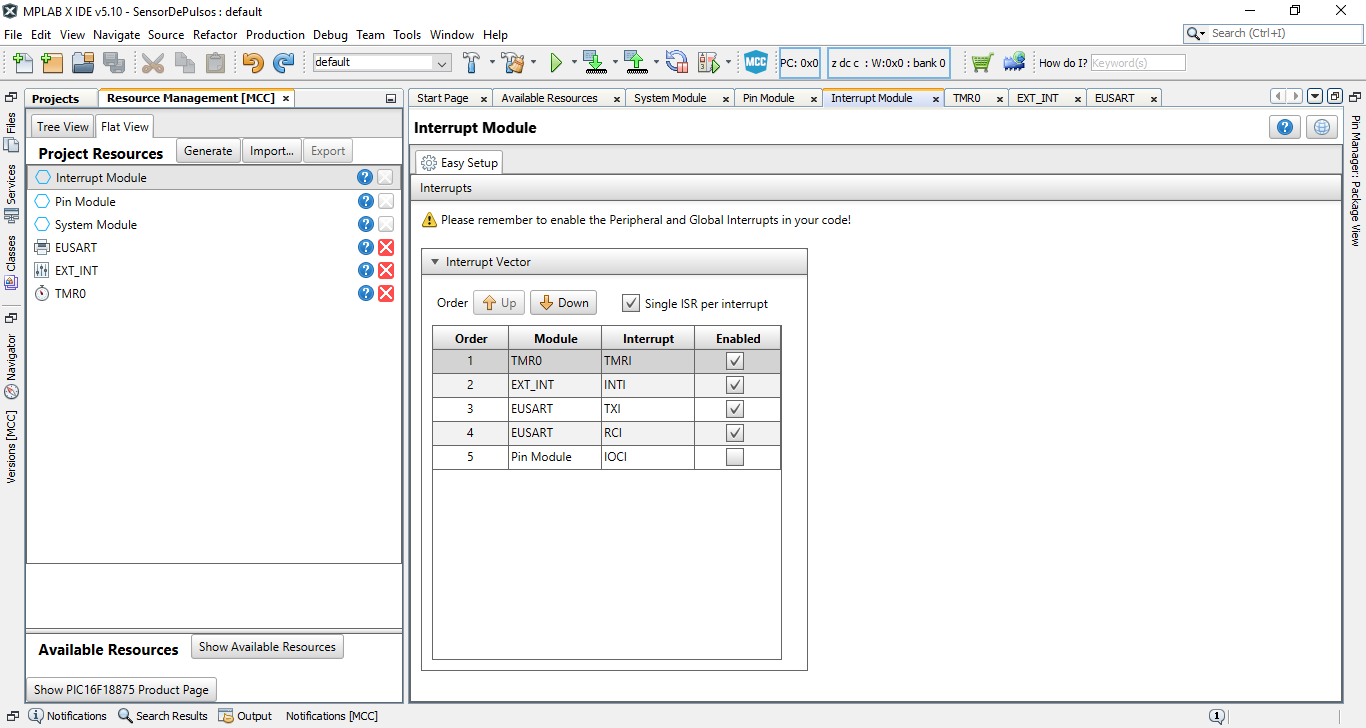


Figura 19. Pines utilizados como RX y TX

En la ventana **Interrupt Module** asegúrese que se encuentren marcados las opciones tal como se observa en la Figura 20. 

**Figura 20. Habilitación de interrupciones**

Finalmente guarde los cambios realizados y haga clic en **Generate**, lo cual nos generara un código en C que contiene todas las configuraciones de los registros de los modulos que hemos agregado para el correcto funcionamiento de este proyecto. Tal como se visualiza en la Figura 21.

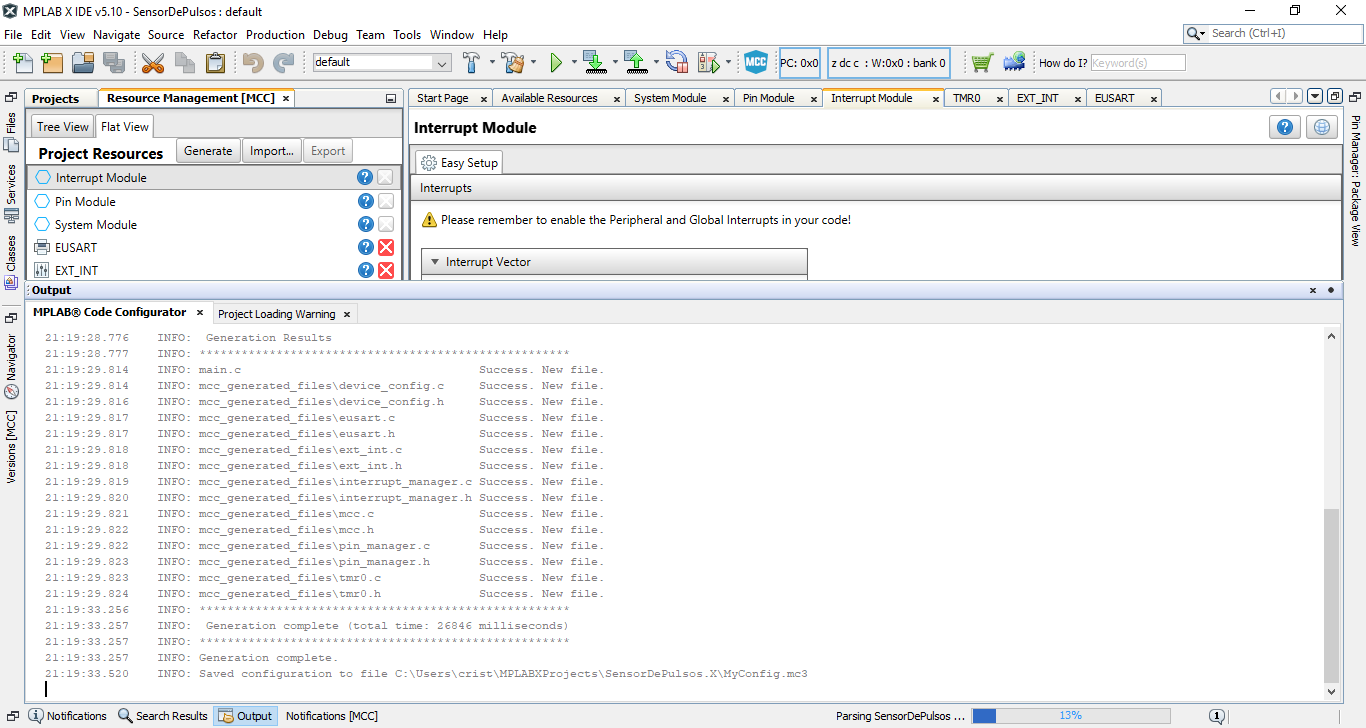


Figura 21. Generando código.

Cierre MPLAB Code Configurator (MCC) dando clic sobre el icono MCC, luego diríjase **Projects** donde observará todas las carpetas y archivos que fueron creados, abra el archivo **main.c** y proceda a descomentar **INTERRUPT\_GlobalInterruptEnable()** e **INTERRUPT\_PeripheralInterruptEnable()**, estas permiten escuchar las interrupciones globales y periféricas. Tal como se observa en la figura 22.

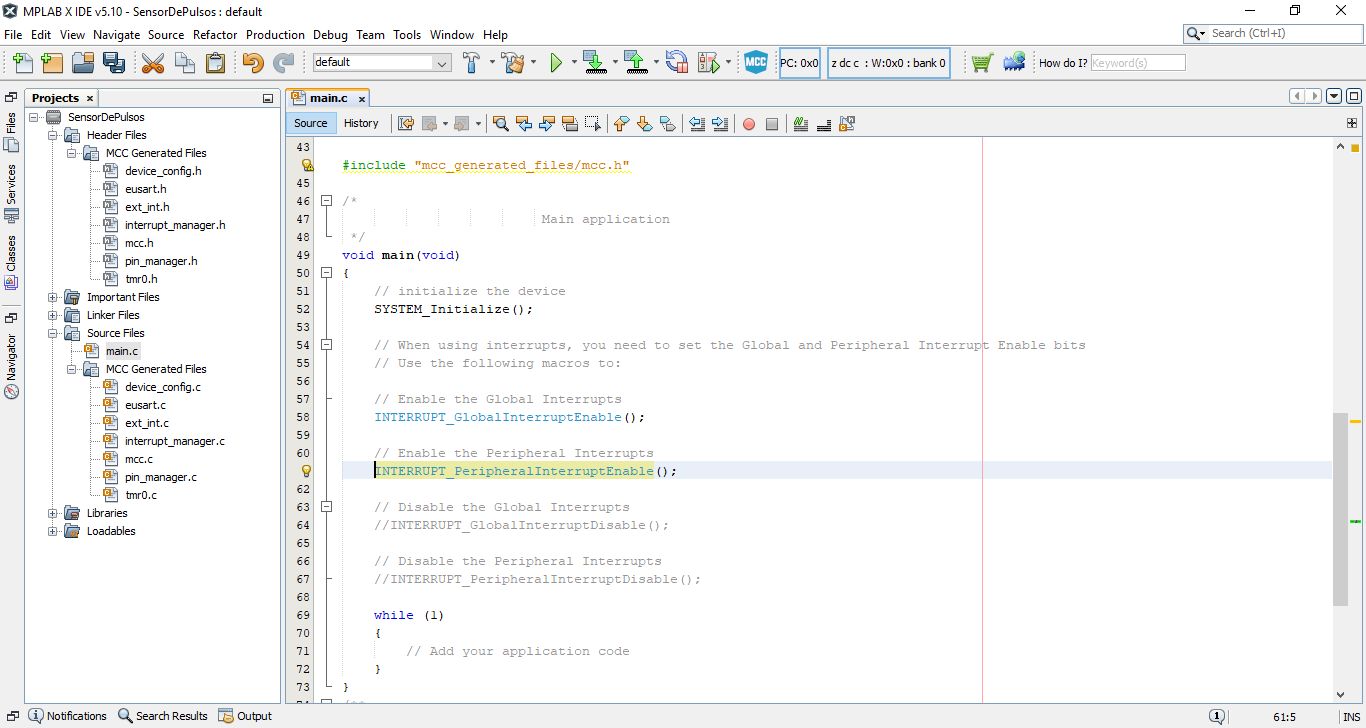


Figura 22. Main.c

Abra el archivo mcc.h y cree e inicialice la variable contDePulsos en con un valor de cero, esta se encargara de contar cada vez que se detecte un pulso en el pin RB0, luego Abra el archivo **ext\_int.c** y coloque dentro de **INT\_DefaultInterruptHandler(void)** contDePulsos++. Tal como se visualiza en la figura 23.

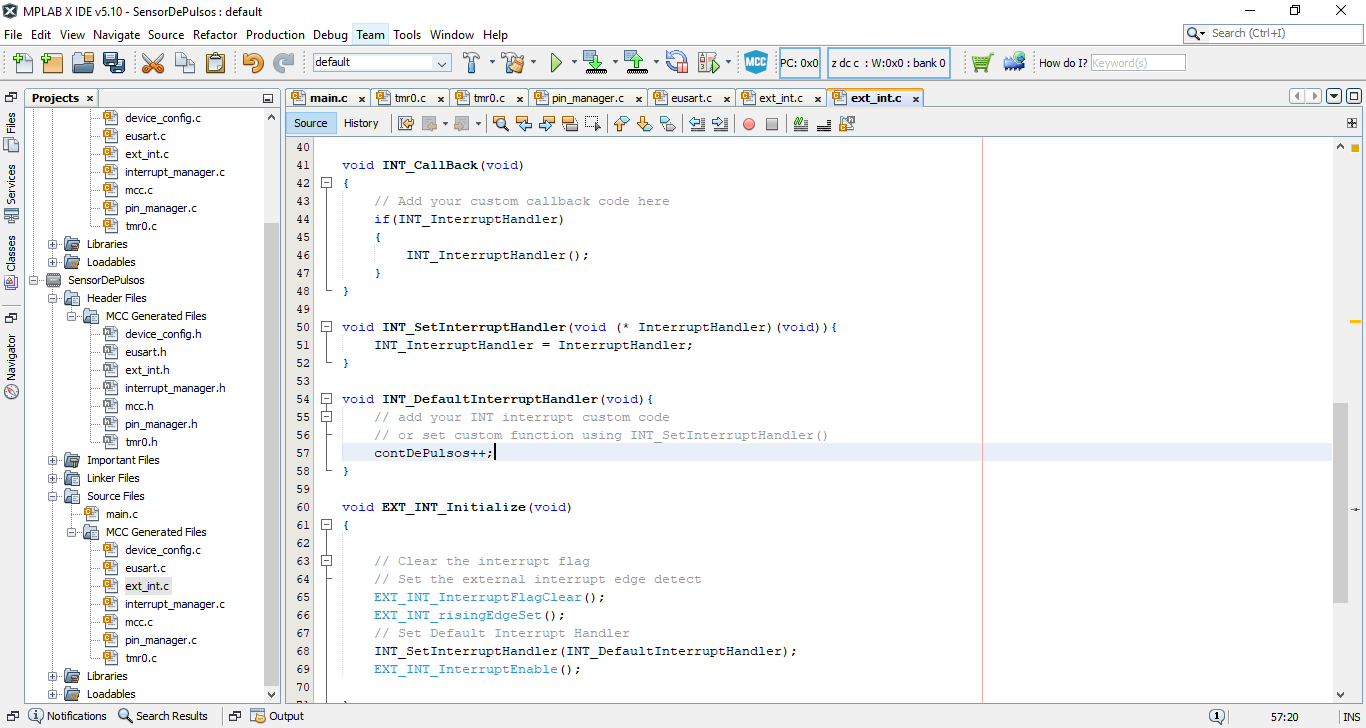


Figura 23. Creación de contador

Abra el archivo pin\_manager.c y agregue la función obtenerTipoPulso() la cual retorna un valor que representa el tipo de señal que el PIC está detectando dependiendo el número de pulsaciones que ha detectado el pin RB0 en un segundo.

/\*\*

\* Clasificara entre las senales disponibles

\*/

int obtenerTipoPulso(int \*contDePulsos){

int tipoDeSenal=-1;

if(\*contDePulsos == SIN\_SENAL){

tipoDeSenal = 0;

return tipoDeSenal;

}

else if(\*contDePulsos >= (PRESION\_ARTERIAL\_BAJA-1) && \*contDePulsos <= (PRESION\_ARTERIAL\_BAJA+1)){

tipoDeSenal = 1;

return tipoDeSenal;

}

else if(\*contDePulsos >= (PARO\_CARDIACO-1) && \*contDePulsos <= (PARO\_CARDIACO+1)){

tipoDeSenal = 2;

return tipoDeSenal;

}

else if(\*contDePulsos >= (ARRITMIA-1) && \*contDePulsos <= (ARRITMIA+1)){

tipoDeSenal = 3;

return tipoDeSenal;

}

else if(\*contDePulsos >= (HIPERPIREXIA-1) && \*contDePulsos <= (HIPERPIREXIA+1)){

tipoDeSenal = 4;

return tipoDeSenal;

}

else if(\*contDePulsos >= (PRESION\_ARTERIAL\_ALTA-2) && \*contDePulsos <= (PRESION\_ARTERIAL\_ALTA+2)){

tipoDeSenal = 5;

return tipoDeSenal;

}

else{

tipoDeSenal = 6;

return tipoDeSenal;

}

}

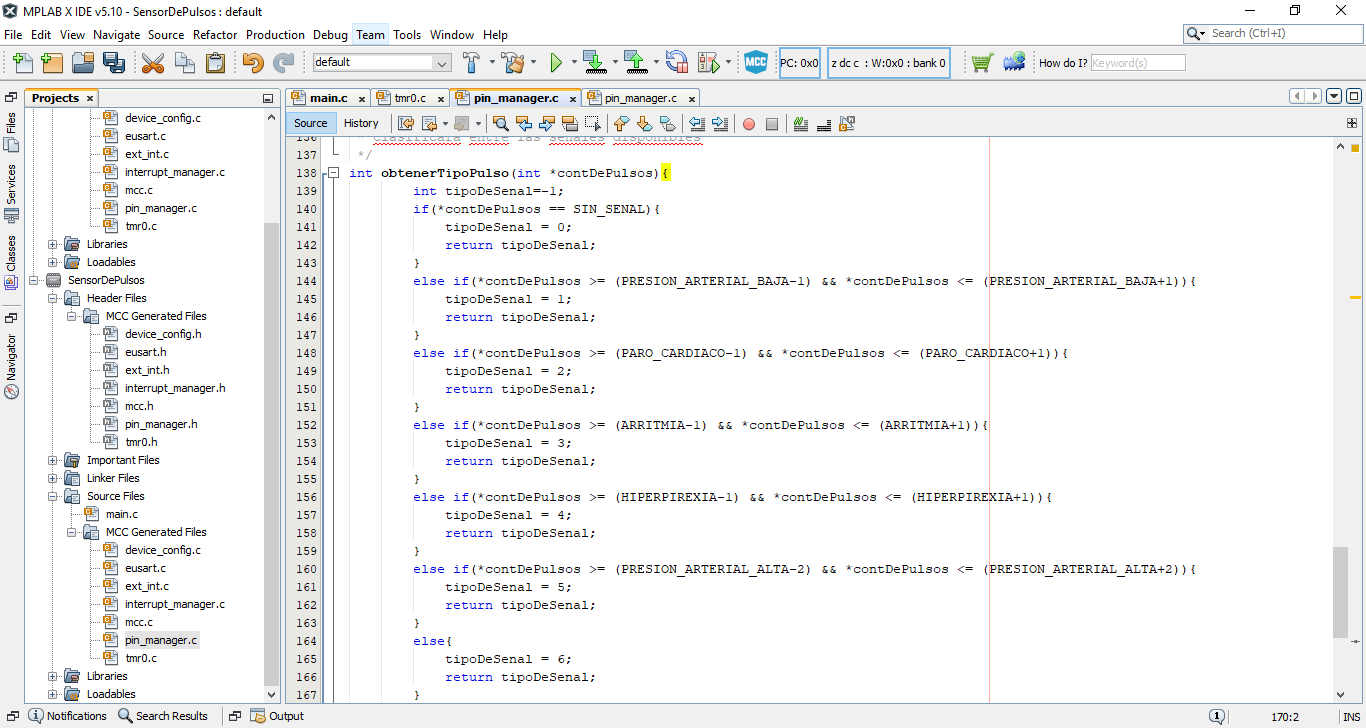


Figura 24. Agregando función ObtenerTipoPulso()

Abra el archivo tmr0.c y agregue el siguiente código dentro de la función void TMR0\_DefaultInterruptHandler(void):

void TMR0\_DefaultInterruptHandler(void){

// add your TMR0 interrupt custom code

// or set custom function using TMR0\_SetInterruptHandler()

int TipoPulso = obtenerTipoPulso(&contDePulsos);

if(TipoPulso != -1){

EUSART\_Write(TipoPulso); // send a byte to TX (from Rx)

contDePulsos = 0;

}

}

La función **obtenerTipoPulso**() se la llama cada segundo para conocer cuántos pulsos se ha contabilizado en 1 segundo y se envía por protocolo serial el **tipoDePulso** al Thinxtra.



Figura 25. Agregando código de envió de tipo de pulso

Guarde todos los cambios y proceda a compilar el proyecto dando clic en . Si todos los pasos anteriores se han hecho de forma correcta se mostrará el mensaje BUILD SUCCESSFUL. Tal como se observa en la figura 26.

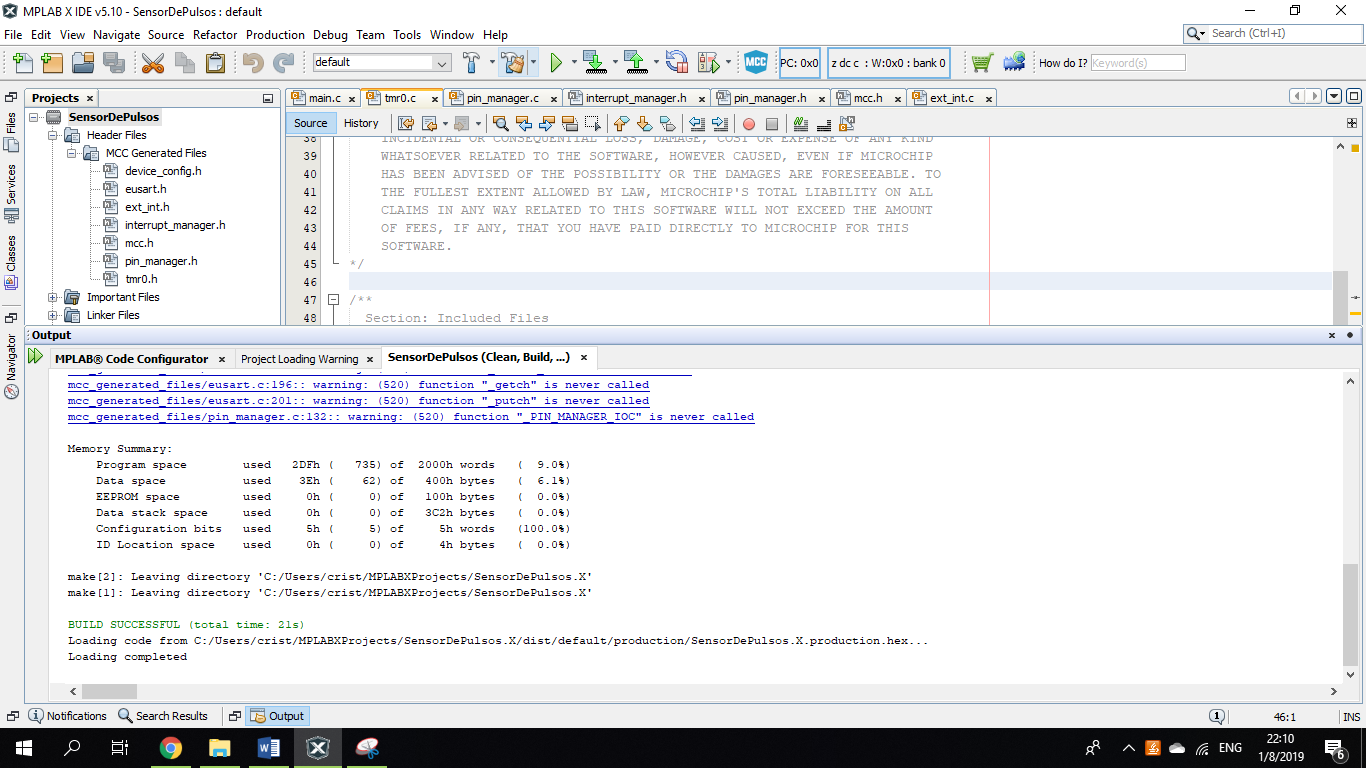


Figura 26. Compilación del proyecto

## 2.4 Cargar el código al PIC

Proceda a conectar el cable USB entre la computadora y la tarjeta Curiosity HPC, asegúrese que en la Curiosity HPC el jumper esté conectado en 3V3.

Imagen que contiene captura de pantalla

Descripción generada automáticamenteHaga clic en para cargar el código al PIC16F18875, si todo es correcto se observará un mensaje de programación exitosa tal como se observa en la Figura 27.Imagen que contiene captura de pantalla

Descripción generada automáticamente

Figura 27. Carga del código al PIC

## 2.5 Programación del Thinxtra

Abra el IDE de Arduino, cree un nuevo archivo con Archivo🡪nuevo y coloque el siguiente código:

// Include librairies

#include <SoftwareSerial.h>

#include <WISOL.h>

#include <Wire.h>

#include <math.h>

const int PRESION\_ARTERIAL\_ALTA = 5;

const int PRESION\_ARTERIAL\_BAJA = 1;

const int ARRITMIA = 3;

const int PARO\_CARDIACO = 2;

const int HIPERPIREXIA = 4;

const int SIN\_SENAL = 0;

float v1 = 4.98; // valor real de la alimentacion de Arduino, Vcc

float r1 = 200; //200ohm

float r2 = 1000; // 1K

int porcentaje=0;

int TipoDePulso;

const uint8\_t payloadSize = 4;

uint8\_t buf\_str[payloadSize];

SoftwareSerial PIC ( 10 , 11 ) ; // RX, TX

Isigfox \*Isigfox = new WISOL();

void setup(){

//inicia el puerto serial

Wire.begin();

Wire.setClock(100000);

// Init serial connection between Arduino and Modem

Serial.begin(9600);

PIC.begin(9600);//inicio de puerto serial entre PIC y thinxtra

Isigfox->initSigfox();

Isigfox->testComms();

}

void loop(){

float v = (analogRead(0) \* v1) / 1024.0;

float voltajeReal = v / (r2 / (r1 + r2));

porcentaje=(voltajeReal/6)\*100;

while (PIC.available()>0){

//leemos la opcion enviada desde el PIC

TipoDePulso=PIC.read();

Serial.println(TipoDePulso);

switch (TipoDePulso) {//selecciona senal

case SIN\_SENAL:

{

addSinSenal();

enviarDatos(buf\_str, payloadSize);

delay(20000);

break;

}

case 1:

{

addPresionArterialBaja();

enviarDatos(buf\_str, payloadSize);

delay(20000);

break;

}

case 2:

{

addParoCardiaco();

enviarDatos(buf\_str, payloadSize);

delay(20000);

break;

}

case 3:

{

addArritmia();

enviarDatos(buf\_str, payloadSize);

delay(20000);

break;

}

case 4:

{

addHiperpirexia();

enviarDatos(buf\_str, payloadSize);

delay(20000);

break;

}

case 5:

{

addPresionArterialAlta();

enviarDatos(buf\_str, payloadSize);

delay(20000);

break;

}

default:

{

addSenalDesconocida();

enviarDatos(buf\_str, payloadSize);

delay(20000);

break;

}

}

}

delay(1000);

}

// SendPayload Function => Send messages to the Sigfox Network

void enviarDatos(uint8\_t \*sendData, int len) {

recvMsg \*RecvMsg;

RecvMsg = (recvMsg \*)malloc(sizeof(recvMsg));

Isigfox->sendPayload(sendData, len, 0, RecvMsg);

for (int i = 0; i < RecvMsg->len; i++) {

Serial.print(RecvMsg->inData[i]);

}

Serial.println("");

free(RecvMsg);

}

//funciones que agregan a la trama la data de los pulsos respectivamente

void addSinSenal(void){

buf\_str[0] = 4;

buf\_str[1] = 555/256;

buf\_str[2] = 555%256;

buf\_str[3] = porcentaje;

}

void addPresionArterialBaja(void){

buf\_str[0] = 4;

buf\_str[1] = 550/256;

buf\_str[2] = 550%256;

buf\_str[3] = porcentaje;

}

void addPresionArterialAlta(void){

buf\_str[0] = 4;

buf\_str[1] = 554/256;

buf\_str[2] = 554%256;

buf\_str[3] = porcentaje;

}

void addParoCardiaco(void){

buf\_str[0] = 4;

buf\_str[1] = 551/256;

buf\_str[2] = 551%256;

buf\_str[3] = porcentaje;

}

void addArritmia(void){

buf\_str[0] = 4;

buf\_str[1] = 552/256;

buf\_str[2] = 552%256;

buf\_str[3] = porcentaje;

}

void addHiperpirexia(void){

buf\_str[0] = 4;

buf\_str[1] = 553/256;

buf\_str[2] = 553%256;

buf\_str[3] = porcentaje;

}

void addSenalDesconocida(void){

buf\_str[0] = 4;

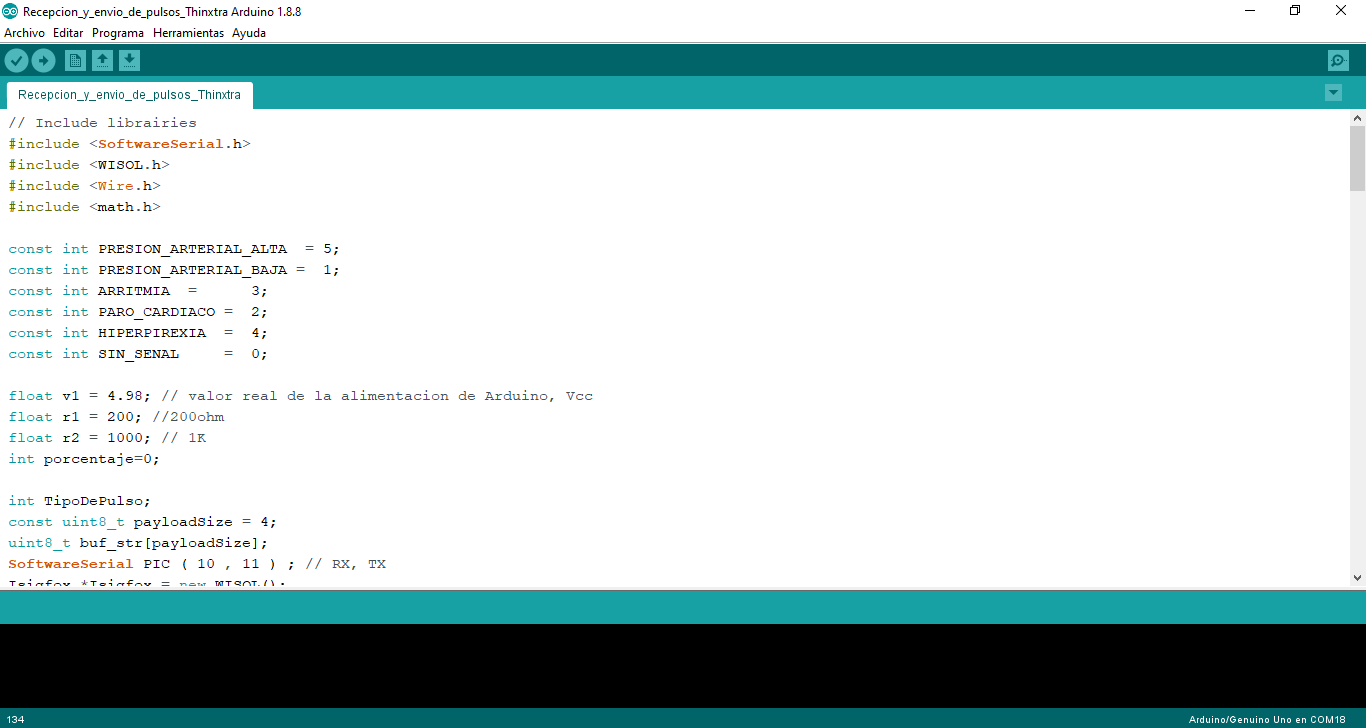
buf\_str[1] = 556/256;

buf\_str[2] = 556%256;

buf\_str[3] = porcentaje;

}

Guarde los cambios y asigne un nombre al archivo.



Conecte el cable USB entre la computadora y Arduino, luego en el IDE de Arduino seleccione el puerto al que se encuentra conectado con Herramientas puertoCOM17, este depende de su PC en este caso está conectado al COM17, tal como se visualiza en la figura 28.

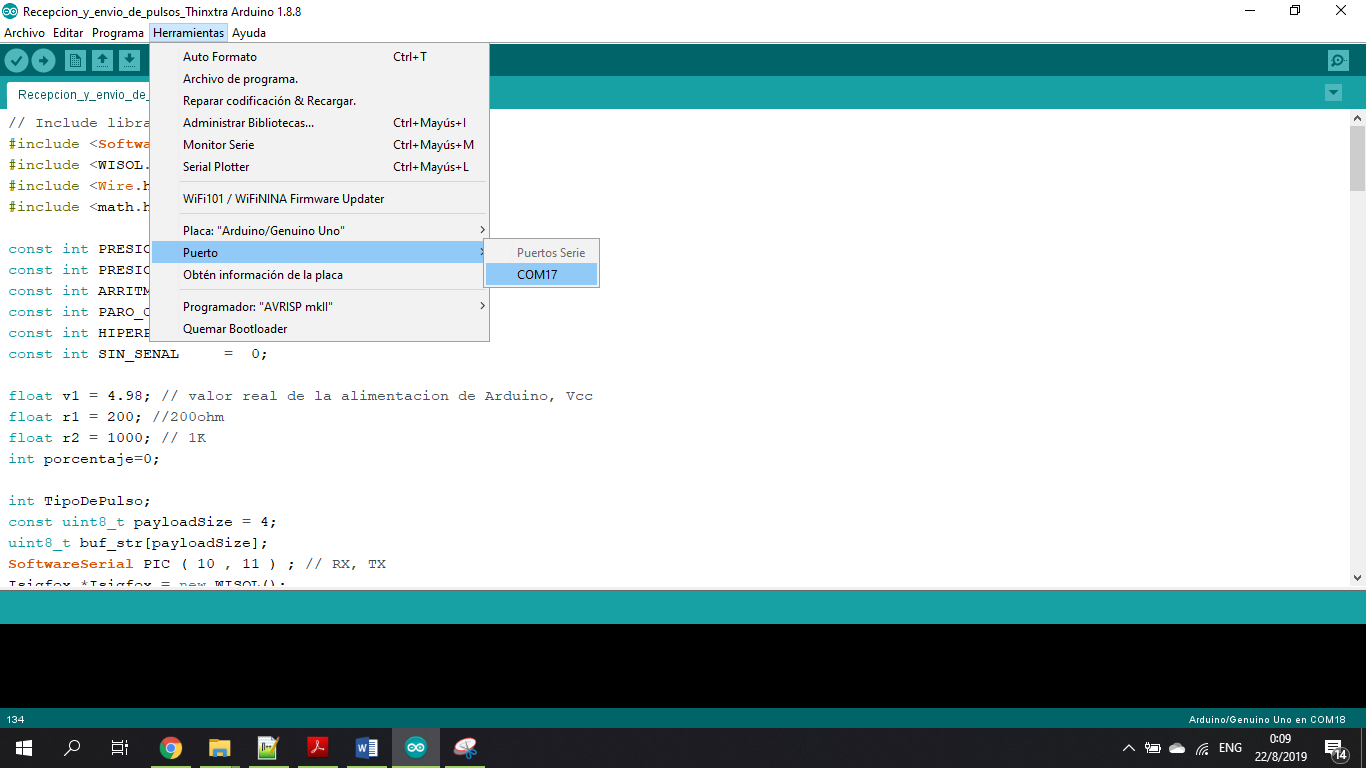


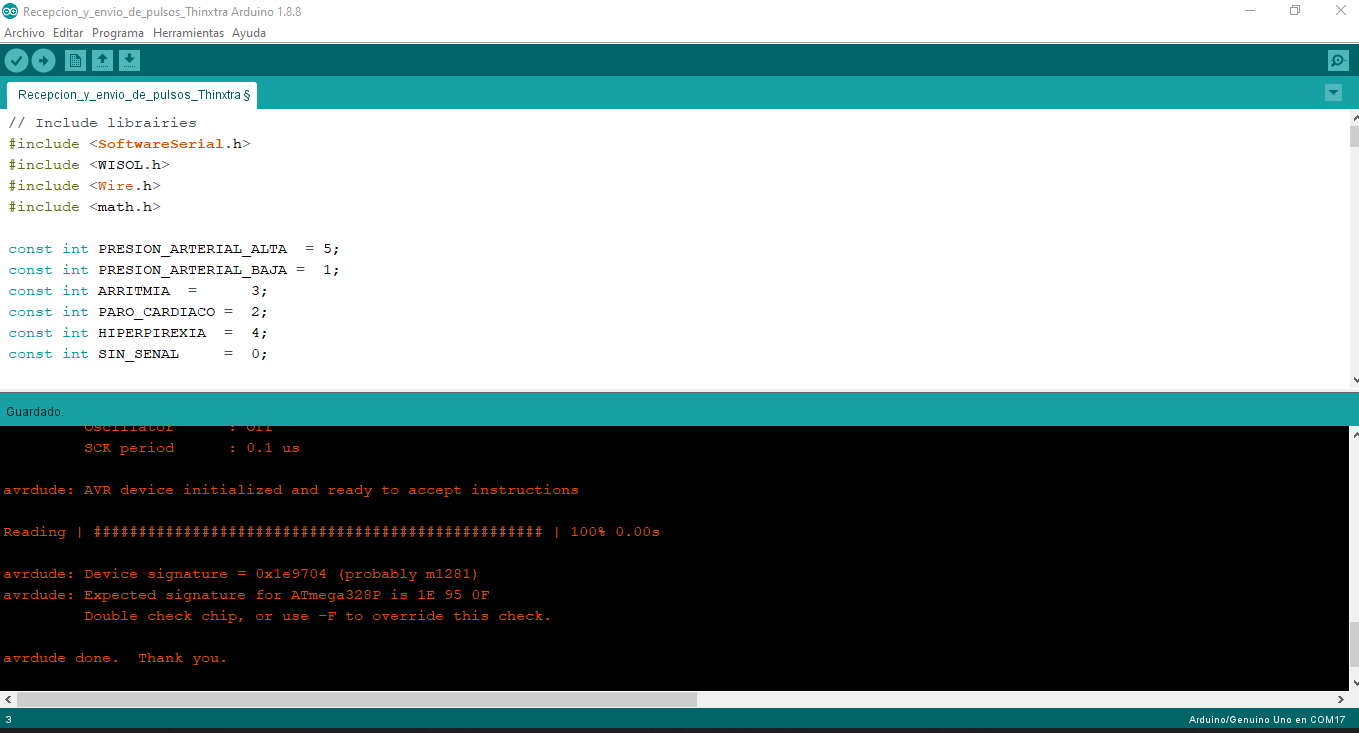
Figura 28. Seleccionando el puerto que está conectado Arduino

Proceda hacer clic en  para compilar el código, si se realizó con éxito la compilación se mostrará un mensaje, tal como se observa en la figura 29.



Figura 29. Compilación del archivo

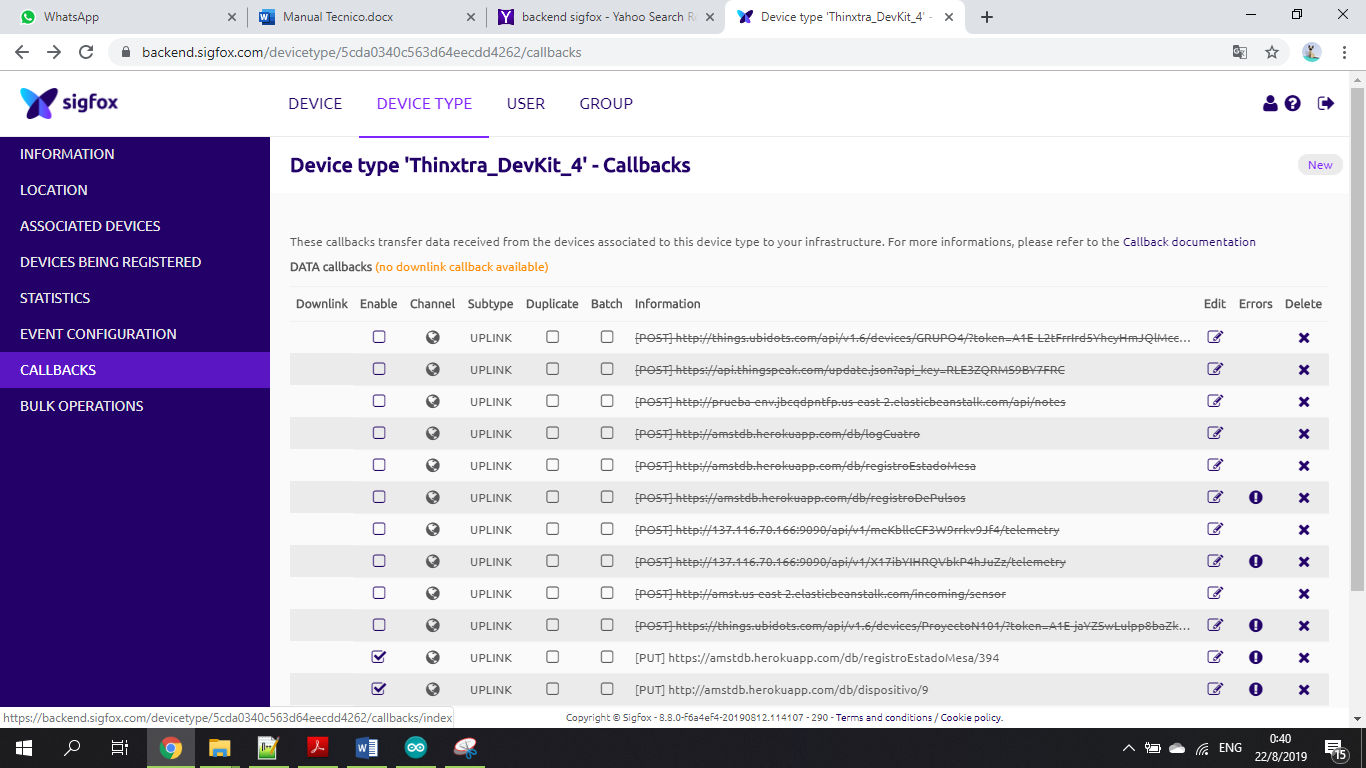
Finalmente proceda hacer clic en  para cargar el código al Arduino, asegúrese que el pin Tx y Rx estén desconectados.



# Creación de callback

Para poder almacenar la informacion en la base de datos de Heroku, se debe crear un callback que permita el envío de los datos que se encuentran almacenados en el backend de sigfox hacia la base de datos, esto se realiza debido a que la aplicación en Android mostrará información que se encuentre almacenada en la base de datos.

1.- Haga clic en **New🡪Custom callback**, tal como se muestra en la figura 30.



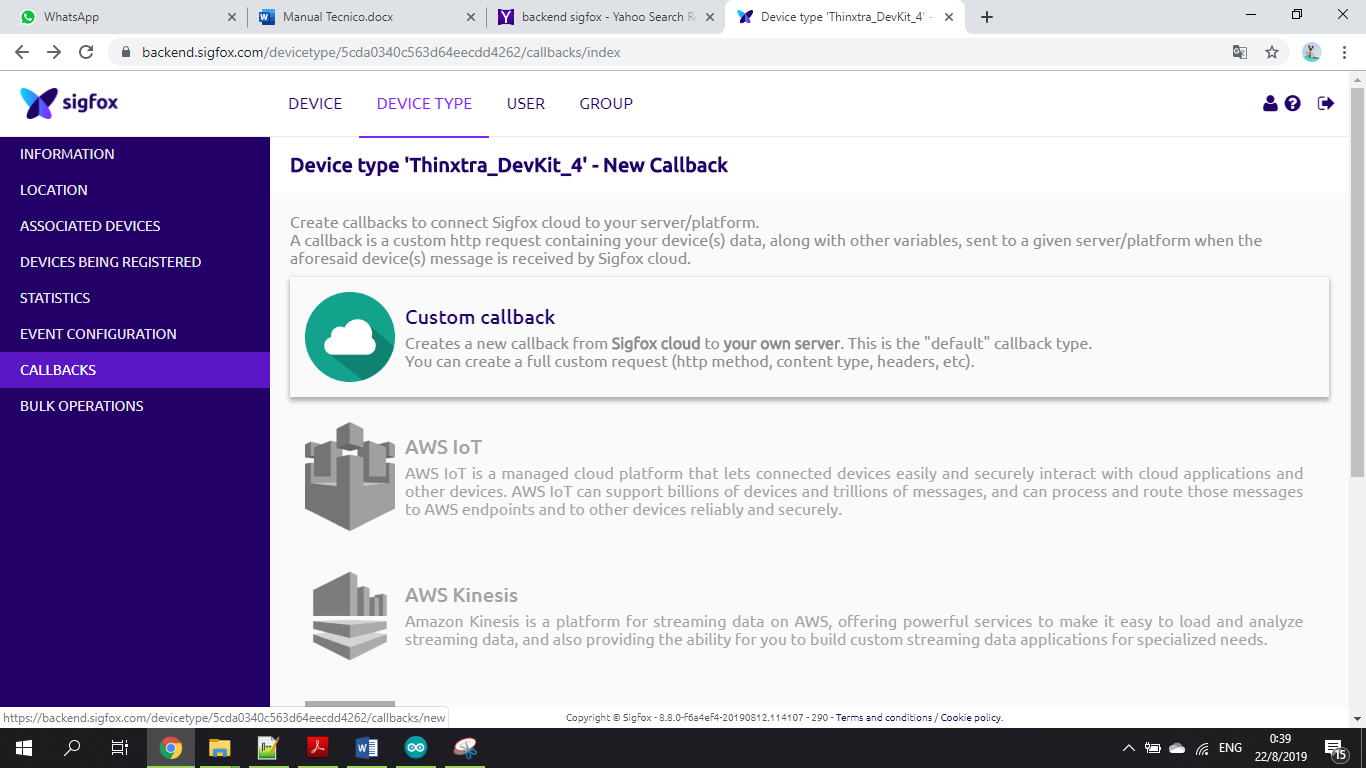


Figura 30. Creando un callback personalizado

2.- Proceda a realizar todas las configuraciones que se observan en la Figura 31.

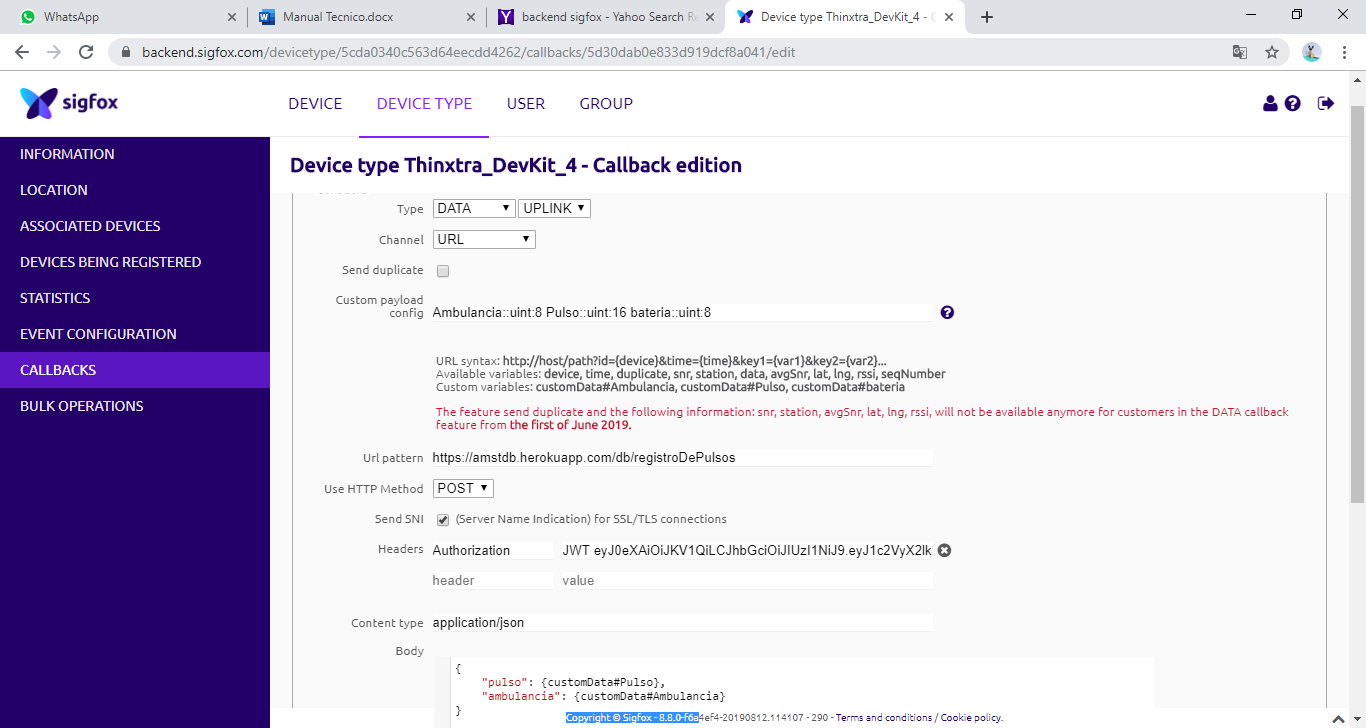


Figura 31. Configuración de callback

3.- Haga clic en **ok** y active el callback creado marcando la casilla del callback, tal como se muestra en la figura 32.

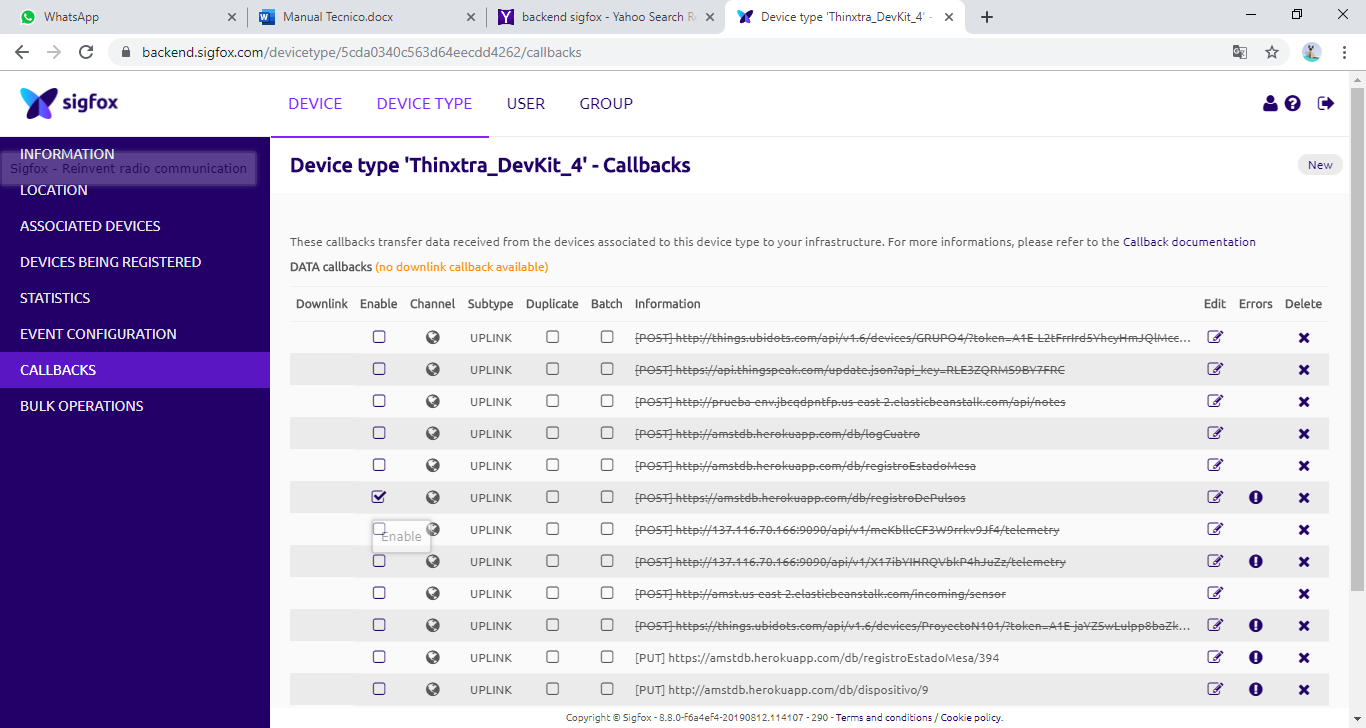


Figura 32. Activando callback de tipo de pulsos

Crear un callback que envié el estado de la batería del dispositivo de IoHealth.

1.- Realice el paso 1 del anterior callback

2.- Proceda a realizar todas las configuraciones que se observan en la Figura 33.

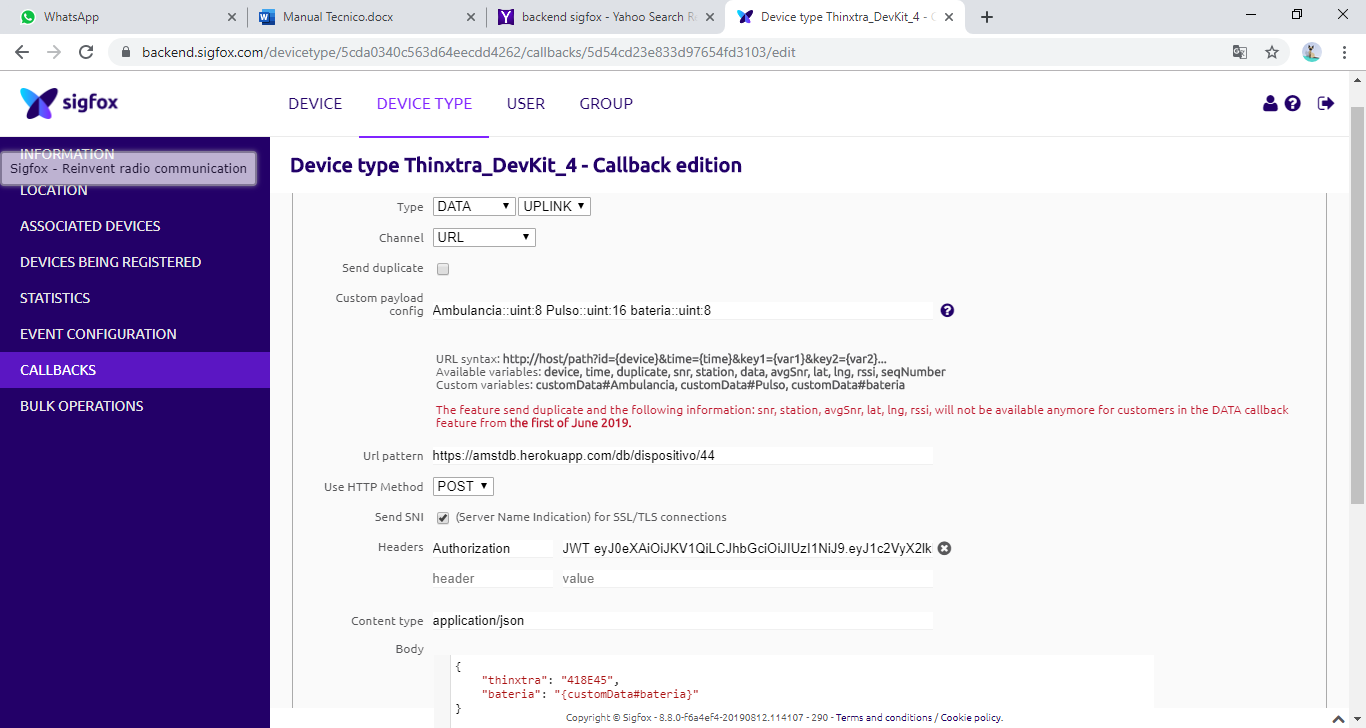


Figura 33. Configuración de callback estado de batería

3.- Haga clic en **ok** y active el callback creado marcando la casilla del callback, tal como se muestra en la figura 34.

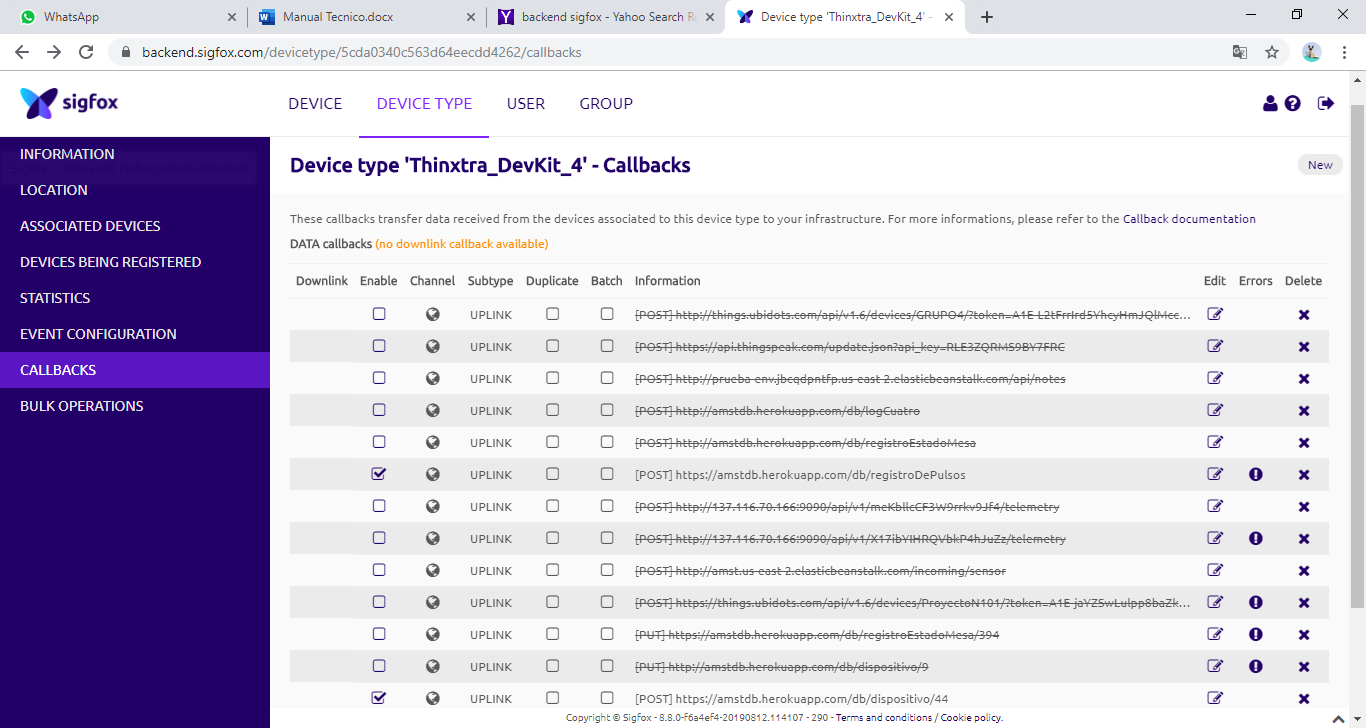


Figura 34. Activando callback de estado de batería

## Recursos de software

* 1. Version Android
  2. Dependendecias y Respositorios
     1. Dependencias utilizada

dependencies {  
 implementation fileTree(**dir**: **'libs'**, **include**: [**'\*.jar'**])  
 implementation **'androidx.appcompat:appcompat:1.0.2'**  
implementation **'androidx.constraintlayout:constraintlayout:1.1.3'**  
implementation **'com.android.volley:volley:1.1.1'**  
implementation **'com.github.PhilJay:MPAndroidChart:v3.1.0'**  
implementation **'com.google.firebase:firebase-messaging:17.3.4'**  
testImplementation **'junit:junit:4.12'**  
androidTestImplementation **'androidx.test:runner:1.2.0'**  
androidTestImplementation **'androidx.test.espresso:espresso-core:3.2.0'**  
implementation **"android.arch.persistence.room:runtime:1.0.0"**  
annotationProcessor **"android.arch.persistence.room:compiler:1.0.0"**  
implementation **'com.google.firebase:firebase-messaging:19.0.1'**  
**}**

* + 1. Repositorios añadidos

repositories {  
 maven { url **'https://jitpack.io'** }  
}

* 1. Permisos utilizados

Los permisos utilizados en la aplicación son unicamente para permitir las conexiones a Internet y verificar el estado de la red. Estos son añadidos en el archivo AndroidManifest.xml

<**uses-permission android:name="android.permission.INTERNET"** />  
<**uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE"** />  
<**uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"** />  
<**uses-permission android:name="android.permission.INTERNET"** />

* 1. Autentificacion

En la vista MainActivity se le perdirá al ususrio que ingrese usuario y contrañaseña para acceder a la aplicación, esto se realiza en la funcion **IniciarSesion(View view)**

Esta funcion es llamada por el boton **Iniciar Sesion.** Lo primero que realiza es leer los valores de los dos EditText que contienen los valores de usuario y contraseña y los guarda en sus respectivas variables. Despues utiliza la funcion errorConexion() para verificar si existe conexión a Internet, en caso de no existir se creara un mensaje mediante un Toast indicando el siguiente mensaje: “No hay conexión a Internet”, caso contrario llama a la funcion **inicirSesion(String usurio, String contraseña)**

**public void** IniciarSesion(View view){  
 **final** EditText dt1=(EditText) findViewById(R.id.***txtUser***);  
 **final** EditText dt2=(EditText) findViewById(R.id.***txtPasswd***);  
 String usuario=dt1.getText().toString();  
 String contrasena=dt2.getText().toString();  
  
 **if**(CheckInternet.*errorConexion*()){  
 Toast.*makeText*(**this**, **"No hay conexion a Internet"**, Toast.***LENGTH\_LONG***).show();  
 }  
 **else**{  
 iniciarSesion(usuario,contrasena);  
 }  
}

La Funncion **inicirSesion(String usurio, String contraseña)** t realiza una peticion http hacia el backend externo para verificar si las credenciales son correctas o no. Toma los parametros usuario y contraseña y los guarda en un Map para ser usados con cuerpo de la petición. La peticion se forma con el metodo POST y usa el cuerpo detallado anterior mente. Cuando el mensaje es éxito se ejecuata la funcion **onResponse(JsonObject response)** la cual se encargará de recibir el nuevo token de autentificacion enviado por el backend, almacenarlo en la base de datos e iniciar la actividad **MenuActivity.** En caso de no ser éxito la petición, se ejecuta el metodo **onErrorResponse(VolleyError error)** el cual mostrará un AlertDialog indicando que las credenciales estan incorrectas.

**private void** iniciarSesion(String usuario, String contrasena) {  
 Map<String, String> params=**new** HashMap<>();  
 params.put(**"username"**, usuario);  
 params.put(**"password"**, contrasena);  
  
 JSONObject parametros=**new** JSONObject(params);  
 String URL = **"https://amstdb.herokuapp.com/db/nuevo-jwt"**;  
  
 JsonObjectRequest request = **new** JsonObjectRequest(Request.Method.***POST***, URL, parametros, **new** Response.Listener<JSONObject>() {  
 @Override  
 **public void** onResponse(JSONObject response) {  
 System.***out***.println(response);  
 **try** {  
 **token** = response.getString(**"token"**);  
 Token tokenTosave = **new** Token(**token**);  
 **tokenDAO**.insert(tokenTosave);  
  
 Intent i = **new** Intent(getBaseContext(), MenuActivity.**class**);  
 i.putExtra(**"token"**, **token**);  
 startActivity(i);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 }  
 }, **new** Response.ErrorListener() {  
 @Override  
 **public void** onErrorResponse(VolleyError error) {  
  
 AlertDialog alertDialog = **new** AlertDialog.Builder(MainActivity.**this**).create();  
 alertDialog.setTitle(**"Alerta"**);  
 alertDialog.setMessage(**"Credenciales Incorrectas"**);  
 alertDialog.setButton(AlertDialog.***BUTTON\_NEUTRAL***, **"OK"**,  
 **new** DialogInterface.OnClickListener() {  
 **public void** onClick(DialogInterface dialog, **int** which) {  
 dialog.dismiss();  
 }  
 });  
 alertDialog.show();  
*;*  
}  
 });  
 **mQueue**.add(request);  
}

* 1. Peticiones a Tablas en la base de datos: Clase GetTablesHelper

Esta clase contiene ArrayLists que serviran para almacenar los registros de las tablas de interes, en este caso para las tablas Pulso, RegistroPulso, Ambulancia y Dispositivo; ademas de tener funciones que serviran para el post procesamiento de la informacion una vez ya pedida

* + 1. Peticiones al backend externo

Las funciones obtnerAmblancia (String url), obtenerPulso (String url), obtenerRegistros (String url) y obtenerBateria (String url) son funciones que tienen la misma estructura:

* Reciben como parametro la url a la cual haran la peticion GET
* En caso de una respuesta favorable por parte del backend externo, se ejecuta la funcion onResponse(JSONArray response) o onResponse(JSONObject response) en caso de recibir un JSONArray o un JSONObject respectivamente.
* En caso de no recibir una respuesta favorable se ejecuta la funcion onErrorResponse(VolleyError error)
* En la funcion getHeaders() se retorna la cabecera de la peticion, que estara almacenado en el mapa **params** tendrá la siguiente estructura:
  + {“**Authorization”: “JWT “+ token}**

Ejemplo:

**public void** obtenerAmbulancias(String url){  
 JsonArrayRequest request = **new** JsonArrayRequest(Request.Method.***GET***, url , **null**,  
 **new** Response.Listener<JSONArray>() {  
  
 @Override  
 **public void** onResponse(JSONArray response) {  
 **for** (**int** i = 0; i<response.length(); i++){  
 **try** {  
 JSONObject ambulanciaJ = response.getJSONObject(i);  
 **int** id = ambulanciaJ.getInt(**"id"**);  
 String placa = ambulanciaJ.getString(**"placa"**);  
 **boolean** ocuapdo = ambulanciaJ.getBoolean(**"ocupado"**);  
 **int** conductor = ambulanciaJ.getInt(**"conductor"**);  
 Ambulancia ambulancia = **new** Ambulancia(id,placa, ocuapdo,conductor);  
 **ambulancias**.add(ambulancia);  
 } **catch** (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }, **new** Response.ErrorListener() {  
 @Override  
 **public void** onErrorResponse(VolleyError error) {  
 System.***out***.println(**"Erooor respuesta"**);  
 System.***out***.println(error);  
 }  
 }) {  
 @Override  
 **public** Map<String, String> getHeaders() **throws** AuthFailureError {  
 **return params**;  
 }  
 };  
 **queue**.add(request);  
}

* 1. Diagrama de barras: GraficoPulsoActivity

En esta actividad se muestra un diagrama de barras que cuenta la cantidad de pulsos de cada tipo, para ello de la vista XML se obtiene una instancia de **BarChart.** Con ayuda de una instancia de GetTablesHelper, getTablesHelper, se obtienen los valores de las tablas RegistoPulso, Ambulancia, Pulso y todas las clases obtenidas son relacionadas a los RegistrosPulsos obtenidos.

**getTablesHelper**.getTables(**urlRegistoPulsos**, **urlPulsos**, **urlAmbulancia**);  
**getTablesHelper**.addAmbulaciaAndPulso();

Se utiliza un hilo Runnable para realizar de forma automatica la actualizacion de los resgitros, en donde se vuelve a ejecutar las peticiones a la base de datos. En concreto el hilo Runnable ejecuta la función **actualizar()**

La función actualizar() además de indicarle a la instancia de GetTablesHelper tambien usa sus datos para actualizar el diagrama de barras, esto se realiza con las funciones:

* ContarPulsos(List<RegistroPulso> registroPulso) que contará los pulsos de los diferente tipos y los almacenara en un mapa
* SetDataBarchart(BarChart, Map) que ingresara la informacion del mapa antes creado en el BarChart
* barChart.notifyDataSetChanged() y barChart.invalidate() que servirar para idicarle al Barchart que debe actualizarse

**public class** GraficoPulsosActivity **extends** AppCompatActivity {  
 */\*\**  
 *\* The Bar chart.*  
 *\*/*  
BarChart **barChart**;  
 */\*\**  
 *\* The Token.*  
 *\*/*  
String **token**;  
 */\*\**  
 *\* The Url registo pulsos.*  
 *\*/*  
String **urlRegistoPulsos** = **"https://amstdb.herokuapp.com/db/registroDePulsos"**;  
 */\*\**  
 *\* The Url ambulancia.*  
 *\*/*  
String **urlAmbulancia** = **"https://amstdb.herokuapp.com/db/ambulancia"**;  
 */\*\**  
 *\* The Url pulsos.*  
 *\*/*  
String **urlPulsos** = **"https://amstdb.herokuapp.com/db/pulsos"**;  
 */\*\**  
 *\* The Table.*  
 *\*/*  
TableLayout **table**;  
 */\*\**  
 *\* The Registro pulsos.*  
 *\*/*  
ArrayList<RegistroPulso> **registroPulsos**;  
 */\*\**  
 *\* The Ambulancias.*  
 *\*/*  
ArrayList<Ambulancia> **ambulancias**;  
 */\*\**  
 *\* The Pulsos.*  
 *\*/*  
ArrayList<Pulso> **pulsos**;  
 */\*\**  
 *\* The Get tables helper.*  
 *\*/*  
GetTablesHelper **getTablesHelper**;  
  
 */\*\**  
 *\* The Handler.*  
 *\*/*  
Handler **handler**;  
 */\*\**  
 *\* The Runnable.*  
 *\*/*  
Runnable **runnable**;  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_grafico\_pulsos***);  
  
 **token** = getIntent().getExtras().getString(**"token"**);  
  
 **getTablesHelper** = **new** GetTablesHelper(**token**, getApplicationContext());  
  
  
 **barChart** = (BarChart)findViewById(R.id.***barchart***);  
 **table** = (TableLayout) findViewById(R.id.***table1***);  
  
 **registroPulsos** = **new** ArrayList<>();  
 **ambulancias** = **new** ArrayList<>();  
 **pulsos** = **new** ArrayList<>();  
  
  
 init\_Barchart(**barChart**);  
  
 **handler** = **new** Handler();  
 **runnable** = **new** Runnable() {  
 @Override  
 **public void** run() {  
 **registroPulsos** = **new** ArrayList<>();  
 **ambulancias** = **new** ArrayList<>();  
 **pulsos** = **new** ArrayList<>();  
 actualizar();  
 **handler**.postDelayed(**this**, 10000);  
 }  
 };  
 **runnable**.run();  
  
 }  
  
 @Override  
 **public void** onBackPressed() {  
 **handler**.removeCallbacks(**runnable**);  
  
 **super**.onBackPressed();  
  
 }  
  
 */\*\**  
 *\* Ir detalles.*  
 *\**  
 *\** ***@param v*** *the v*  
 *\*/*  
**public void** irDetalles(View v){  
 **handler**.removeCallbacks(**runnable**);  
 Intent intent = **new** Intent(getApplicationContext(), Table\_Registros\_Pulsos\_Activity.**class**);  
 intent.putExtra(**"token"**, **token**);  
 startActivity(intent);  
 }  
  
 */\*\**  
 *\* Actualizar.*  
 *\*/*  
**public void** actualizar(){  
 System.***out***.println(**"Acccccccccc"**);  
 **getTablesHelper**.getTables(**urlRegistoPulsos**, **urlPulsos**, **urlAmbulancia**);  
  
  
  
 **new** Handler().postDelayed(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **getTablesHelper**.addAmbulaciaAndPulso();  
 **registroPulsos** = **getTablesHelper**.getRegistroPulsos();  
  
 System.***out***.println(**registroPulsos**);  
 Map<String, Integer> data = contarPulsos(**registroPulsos**);  
 setDataBarchart(**barChart**, data);  
 **barChart**.notifyDataSetChanged();  
 **barChart**.invalidate();  
 findViewById(R.id.***btnDetalles***).setVisibility(View.***VISIBLE***);  
  
 }  
 }, 3000);  
 }  
  
 */\*\**  
 *\* Set data barchart.*  
 *\**  
 *\** ***@param barChart*** *the bar chart*  
 *\** ***@param map*** *the map*  
 *\*/*  
**public void** setDataBarchart(BarChart barChart, Map<String, Integer> map){  
 String[] labels = **new** String[map.keySet().size()];  
 ArrayList<BarEntry> barEntries = **new** ArrayList<>();  
  
 **int** maxValue = 0;  
 Iterator <Map.Entry<String, Integer>> iterator = map.entrySet().iterator();  
 **int** counter = 0;  
 **while** (iterator.hasNext()){  
 Map.Entry<String, Integer> i = iterator.next();  
 String key = i.getKey();  
 Integer value = i.getValue();  
 barEntries.add(**new** BarEntry(counter, value));  
  
 labels[counter] = key;  
 counter ++;  
 **if** (value > maxValue)  
 maxValue = value;  
 }  
 BarDataSet barDataSet = **new** BarDataSet(barEntries, **"Pulsos"**);  
 barDataSet.setColors(ColorTemplate.***MATERIAL\_COLORS***);  
  
  
 BarData barData = **new** BarData(barDataSet);  
 *//barData.setBarWidth(0.9f);*  
  
barChart.setData(barData);  
 add\_labels\_Barchart(barChart, labels);  
  
 }  
  
 */\*\**  
 *\* Add labels barchart.*  
 *\**  
 *\** ***@param barChart*** *the bar chart*  
 *\** ***@param labels*** *the labels*  
 *\*/*  
**public void** add\_labels\_Barchart(BarChart barChart, String[] labels){  
 IndexAxisValueFormatter indexFormatter = **new** IndexAxisValueFormatter();  
 indexFormatter.setValues(labels);  
  
 XAxis xAxis = barChart.getXAxis();  
 xAxis.setValueFormatter(indexFormatter);  
 xAxis.setPosition(XAxis.XAxisPosition.***BOTTOM***);  
 xAxis.setGranularity(1);  
 xAxis.setCenterAxisLabels(**true**);  
 xAxis.setLabelRotationAngle(45);  
 xAxis.setAvoidFirstLastClipping(**true**);  
 xAxis.setGranularityEnabled(**true**);  
  
 xAxis.setXOffset(-20);  
  
 }  
  
 */\*\**  
 *\* Init barchart.*  
 *\**  
 *\** ***@param barChart*** *the bar chart*  
 *\*/*  
**public void** init\_Barchart(BarChart barChart){  
 barChart.setDrawBarShadow(**false**);  
 barChart.setDrawValueAboveBar(**false**);  
 *//barChart.setMaxVisibleValueCount(50);*  
barChart.setPinchZoom(**false**);  
 barChart.setDrawGridBackground(**true**);  
 barChart.getLegend().setEnabled(**false**);  
 barChart.getLegend().setHorizontalAlignment(Legend.LegendHorizontalAlignment.***CENTER***);  
  
 }  
  
  
 */\*\**  
 *\* Contar pulsos map.*  
 *\**  
 *\** ***@param registroPulsos*** *the registro pulsos*  
 *\** ***@return*** *the map*  
 *\*/*  
**public** Map<String, Integer> contarPulsos(ArrayList<RegistroPulso> registroPulsos){  
 Map<String, Integer> contador = **new** HashMap<>();  
 **for** (**int** i = 0; i<registroPulsos.size(); i++){  
 RegistroPulso registroPulso = registroPulsos.get(i);  
 Pulso pulso = registroPulso.getPulso();  
 String nombre = pulso.getNombre();  
 **try**{  
 Integer value = contador.get(nombre) + 1;  
 *//contador.remove(nombre);*  
contador.put(nombre,value);  
 }  
 **catch** (Exception e){  
 contador.put(nombre,1);  
 }  
 }  
 **return** contador;  
 }  
  
  
  
  
}