DOCUMENTO NÚMERO 3: ANEXOS

ÍNDICE DE ANEXOS

ANEXO 1 – CONFIGURACIÓN DEL MICROCONTROLADOR

CAPÍTULO 1 - CREACIÓN DEL PROYECTO

1.0 Programa informático

El archivo para la programación se realizará mediante el programa informático Arduino Software (IDE) (versión 1.6.8). Este programa pertenece a Arduino.

Este programa informático está disponible tanto para Windows, Linux como Mac OS. En nuestro caso lo ejecutaremos desde Linux.

1.1 Creación del archivo del proyecto

Se crea un proyecto mediante la opción: Archivo; Nuevo

Automáticamente se nos abre una ventana con el entorno grafico preparado para escribir código. Incluso nos genera los encabezados del “setup”, el cual se ejecuta solo una vez al encender el dispositivo, y el “loop”, este se ejecutará cíclicamente.

De esta forma se crea el archivo del proyecto. Esto se hará de igual forma tanto para programar al dispositivo emisor como para la baliza receptora.

CAPÍTULO 2 - CONEXIONES

2.0 Introducción

Se definirán las conexiones de ambos dispositivos y el uso de ellas en los siguientes apartados.

2.1 Emisor

La única conexión existente en este dispositivo es un USB mini a través del cual se programará el arduino nano. Como uso más frecuente del mismo conector, este se dispondrá para la carga de la batería recargable una vez el dispositivo se encuentre apagado.

2.2 Receptor

Este dispositivo dispone de un puerto USB tipo B para poder tanto alimentar el Arduino UNO como para programarlo.

Adicionalmente, si el dispositivo compone al Shield Ethernet, tendrá también un puerto un puerto Ethernet desde el que tomará acceso a internet.

ANEXO 2 – CÓDIGO DEL MICROCONTROLADOR

CAPÍTULO 1 – CÓDIGO EN LENGUAJE ARDUINO

Para la comunicación entre el emisor y el receptor se ha implantado un sistema de comunicación por el cual se transmitirán 2 bytes. El primero tendrá una doble funcionalidad ya que lo dos últimos bits se emplearán como bit de paridad y bit identificativo de alerta, mientras que el resto serán parte del número identificativo del paciente. El último se empleará para transmitir la segunda parte del número identificativo del paciente.

En el caso en el que lleven acelerómetros el propio microcontrolador determinará con los valores que le facilite el acelerómetro si el paciente ha sufrido una caída o no.

1.0 Código Emisor

En este apartado se definirás múltiples códigos para el emisor en función si este dispone de un acelerómetro analógico, uno digital, o ninguno.

1.0.1 Código para emisor sin acelerómetro

Este código ha sido probado y verificado su correcto funcionamiento.

```
#include <VirtualWire.h>

unsigned char id[2] = {0x1B, 0x8A};
unsigned long tm = millis(); // timeout counter

void setup()
{
   Serial.begin(9600);
   Serial.println("Setup...");
   vw_setup(2000);
}

void loop()
{
   if ((unsigned long) abs((millis() - tm)) > 10000)
```

```
   {
      //send data every 10 seconds
      send();
      tm = (unsigned long) millis(); // restart timeout counter
   }
}

void send ()
{
  vw_send((uint8_t *)id, 2); //Envia el mensaje
  vw_wait_tx(); //Espera hasta que se haya acabado de transmitir todo

  Serial.print(id[0], HEX);
  Serial.println(id[1], HEX);
}
```

1.0.2 Código para emisor con acelerómetro analógico

Este código es un modelo a seguir el cual solo lee los datos obtenidos por el acelerómetro analógico.

```
#include <VirtualWire.h>
#include <math.h>

//Constantes
#define DELAY 1000
#define RAD 57.2958

//Variables globales
int xVal = 0;  //eje x del acelerometro
int yVal = 0;  //eje y del acelerometro
int zVal = 0;  //eje z del acelerometro

double angleYZ = 0;
double angleXZ = 0;

bool caida = false;

unsigned char id[2] = {0x13, 0x45};
unsigned char alarma[2] = {0x93, 0x45};
```

```
void setup()
{
    //Acelerometro
  analogReference(EXTERNAL);
  //Serial.begin(9600);

  //Iniciamos el Serial y la comunicacion por radio
  Serial.begin(9600);
  Serial.write("Sistema encendido\n");
  vw_setup(2000);
}

void loop()
{
  calculaCaida();
  send(id);
  delay(DELAY);
}

void calculaCaida(){
  //Acelerometro
  xVal = analogRead(0);
  Serial.print("Valor x sin mapear: ");
  Serial.print(xVal);
  xVal = map(xVal, 0, 1023, -500, 500);
  Serial.print(" Valor x mapeado: ");
  Serial.print(xVal);
  Serial.println();

  yVal = analogRead(1);
  Serial.print("Valor y sin mapear: ");
  Serial.print(yVal);
  yVal = map(yVal, 0, 1023, -500, 500);
  Serial.print(" Valor y mapeado: ");
  Serial.print(yVal);
  Serial.println();

  zVal = analogRead(2);
  Serial.print("Valor z sin mapear: ");
  Serial.print(zVal);
  zVal = map(zVal, 0, 1023, -500, 500);
  Serial.print(" Valor z mapeado: ");
  Serial.print(zVal);
  Serial.println();
```

```
  angleYZ = atan((double)yVal / (double)zVal);
  angleYZ = angleYZ*(RAD);

  angleXZ = atan((double)xVal / (double)zVal);
  angleXZ = angleXZ*(RAD);

  Serial.print("Angulo YZ: ");
  Serial.print(angleYZ);
  Serial.print(" Angulo XZ: ");
  Serial.print(angleXZ);
  Serial.println();
  Serial.println();

  if (caida){
    send(alarma);
  }
}

//Funcion para enviar el mensaje
void send (unsigned char * message)
{
  vw_send((uint8_t *)message, 2); //Envia el mensaje
  vw_wait_tx(); //Espera hasta que se haya acabado de transmitir todo

  Serial.println(id); //Muestra el mensaje por Serial
}
```

1.0.3 Código para emisor con acelerómetro digital

Este código está finalizado pendiente de pruebas, debido a la necesidad de un convertidor de tensión de 5V a 3V.

```
#include <Wire.h>
#include <WiredDevice.h>
#include <RegisterBasedWiredDevice.h>
#include <Accelerometer.h>
#include <AccelerometerMMA8451.h>
#include <VirtualWire.h>
#include <math.h>
```

```
AccelerometerMMA8451 acc(0); // SA0 of MMA8451 is LOW so Address is 0x1C
volatile bool ready = false;
unsigned char id[2] = {0x13, 0x45};
unsigned char alarma[2] = {0x93, 0x45};

unsigned long tm = millis(); // timeout counter

void isr()
{
   ready = true;
}

void setup()
{
   Serial.begin(9600);
   Serial.print("Setup...");

   // Step 1: Put the device into Standby Mode: Register 0x2A CTRL_REG1
   acc.standby();

   // Step 2: Set Configuration Register for Motion Detection by setting the
   // "OR" condition OAE = 1, enabling X, Y, and the latch
   acc.configureRegisterBits(AccelerometerMMA8451::FF_MT_CFG,
AccelerometerMMA8451::FF_MT_CFG_OAE, 0x00);

   // Event flag enable on X, Y and Z event.
   acc.configureRegisterBits(AccelerometerMMA8451::FF_MT_CFG,
AccelerometerMMA8451::FF_MT_CFG_ZEFE, 0x20);
   acc.configureRegisterBits(AccelerometerMMA8451::FF_MT_CFG,
AccelerometerMMA8451::FF_MT_CFG_YEFE, 0x10);
   acc.configureRegisterBits(AccelerometerMMA8451::FF_MT_CFG,
AccelerometerMMA8451::FF_MT_CFG_XEFE, 0x08);

   // Step 3: Threshold Setting Value for the Motion detection of > 3g
   // Note: The step count is 0.063g/count
   // 3g/0.063g = 47.6;
   // Round up to 48
   acc.configureRegisterBits(AccelerometerMMA8451::FF_MT_THS,
AccelerometerMMA8451::FF_MT_THS_THS, 0x10);

   // Step 4: Set the debounce counter to eliminate false readings for 100 Hz
   // sample rate with a requirement of 100 ms timer.
   // Note: 100 ms/10 ms (steps) = 10 counts
   acc.writeRegister(AccelerometerMMA8451::FF_MT_COUNT, 0x0a);
```

```
    // Configure the INT pins for Open Drain
    acc.setPushPullOpenDrain(AccelerometerMMA8451::PUSH_PULL);

    // Step 5: Enable Motion/Freefall Interrupt Function in the System
    // disable all interrupt sources we only want Freefall or Motion interrupt
    acc.disableInterrupt(AccelerometerMMA8451::INT_ALL);
    acc.enableInterrupt(AccelerometerMMA8451::INT_FF_MT, 1);
    //acc.enableInterrupt(AccelerometerMMA8451::INT_DRDY, 1);

    // Step 7: Put the device in Active Mode
    acc.activate();

    // display attached device id
    unsigned char b;
    b = acc.readRegister(AccelerometerMMA8451::WHO_AM_I);
    switch (b) {
       case(0x2A):
          Serial.println("MMA8452 present");
          break;
       case(0x1A):
          Serial.println("MMA8451 present");
          break;
       default:
          Serial.print("unknown device present 0x");
          Serial.println(b, HEX);
    }

    // Step 8: Write a Service Routine to Service the Interrupt
    attachInterrupt(0, isr, FALLING);
    vw_setup(2000);
    Serial.println("done.");
}

void loop()
{
    AccelerometerMMA8451::INT_SOURCEbits source;

    // 10 seconds since last interrupt is something wrong? read most of the
MMA8451's status regs
    if ((unsigned long) abs((millis() - tm)) > 10000)
    {
      source.value                                                    =
acc.readRegister(AccelerometerMMA8451::INT_SOURCE);
       //send id
```

```
        send(id);
        tm = (unsigned long) millis(); // restart timeout counter
    }

    if (ready)
    {
        // Serial.println("ready");
        ready = false;

        source.value                                                    =
acc.readRegister(AccelerometerMMA8451::INT_SOURCE);

        Serial.print("source: ");
        Serial.print(source.value, HEX);
        Serial.print(" ");

        // Set up Case statement here to service all of the possible interrupts
        // Freefall or Motion Detection Interrupt (bit 2 of source)
        if (source.SRC_FF_MT)
        {
            //Read the FF_MT State from the Status Register,
            //clear the interrupt, by reading the FF_MT_SRC Register
            AccelerometerMMA8451::FF_MT_SRCbits ff_mt_src;
            ff_mt_src.value                                             =
acc.readRegister(AccelerometerMMA8451::FF_MT_SRC);

            Serial.print("FreeFall Motion Interrupt: ");
            Serial.println(ff_mt_src.value, BIN);

            //send alarm
            for(int i = 0; i < 3; i++)
            {
              send(alarma);
              delay(100);
            }
        }
    }
}

void send (unsigned char * message)
{
  vw_send((uint8_t *)message, 2); //Envia el mensaje
  vw_wait_tx(); //Espera hasta que se haya acabado de transmitir todo

  Serial.println(message); //Muestra el mensaje por Serial
```

```
}
```

1.2 Código Receptor

En este apartado se definirán dos códigos para la baliza receptora, uno para que transmita al servidor vía WiFi y otro para que transmita vía conexión Ethernet.

1.2.1 Código receptor con Ethernet

Este código ha sido probado y verificado su correcto funcionamiento.

```
#include <SPI.h>         // needed for Arduino versions later than 0018
#include <Ethernet.h>
#include <EthernetUdp.h> // UDP library from: bjoern@cs.stanford.edu 12/30/2008
#include <VirtualWire.h>

unsigned char id[2];
// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] =
{
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
IPAddress ip(192, 168, 1, 177);
IPAddress dst(192, 168, 1, 253);

unsigned int localPort = 8888;      // local port to listen on
unsigned int destinationPort = 9876;// destination port to send packets

// An EthernetUDP instance to let us send and receive packets over UDP
EthernetUDP Udp;

byte message[VW_MAX_MESSAGE_LEN];
byte messageLength = VW_MAX_MESSAGE_LEN;

void setup()
{
  // start serial com
```

```
  Serial.begin(9600);
  // start the Ethernet and UDP:
  Ethernet.begin(mac, ip);
  Udp.begin(localPort);
  // start the RF:
  vw_set_rx_pin(8);
  vw_setup(2000);
  vw_rx_start();
  Serial.println("Setup done");
}

void loop()
{
  if (vw_get_message(message, &messageLength))
  {
    id[0]=message[0];id[1]=message[1];
    if (Udp.beginPacket(dst, destinationPort)){
      Udp.write(id[0]);Udp.write(id[1]);
      Serial.print(id[0], HEX);Serial.print(id[1], HEX);
      Serial.println();
      if (Udp.endPacket())
      {
        Serial.println("Packet sent");
      }
      else
      {
        Serial.println("Error sending");
      }
    }
  }
}
```

1.2.2 Código receptor con WiFi

Este código está finalizado pendiente de ser probado.

```
#include <WiFiEsp.h>
#include <WiFiEspUdp.h>
#include <VirtualWire.h>

// Emulate Serial1 on pins 6/7 if not present
```

```
#ifndef HAVE_HWSERIAL1
#include "SoftwareSerial.h"
SoftwareSerial Serial1(6, 7); // RX, TX
#endif

char ssid[] = "XXXXXXXXX";            // your network SSID (name)
char pass[] = "XXXXXXXXX";        // your network password
int status = WL_IDLE_STATUS;     // the Wifi radio's status

unsigned int localPort = 8888;  // local port to listen on
unsigned int destinationPort = 9876;// destination port to send packets

char packetBuffer[255];          // buffer to hold incoming packet
char ReplyBuffer[] = "ACK";      // a string to send back

WiFiEspUDP Udp;
IPAddress dst(192, 168, 1, 253);

unsigned char id[2];
byte message[VW_MAX_MESSAGE_LEN];
byte messageLength = VW_MAX_MESSAGE_LEN;

void setup() {
  // initialize serial for debugging
  Serial.begin(115200);
  // initialize serial for ESP module
  Serial1.begin(9600);
  // initialize ESP module
  WiFi.init(&Serial1);

  // check for the presence of the shield:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // don't continue:
    while (true);
  }

  // attempt to connect to WiFi network
  while ( status != WL_CONNECTED) {
    Serial.print("Attempting to connect to WPA SSID: ");
    Serial.println(ssid);
    // Connect to WPA/WPA2 network
    status = WiFi.begin(ssid, pass);
  }
```

```
  Serial.println("Connected to wifi");
  printWifiStatus();

  vw_set_rx_pin(8);
  vw_setup(2000);
  vw_rx_start();
  Serial.println("Setup done");
}

void loop()
{
  if (vw_get_message(message, &messageLength))
  {
    id[0]=message[0];id[1]=message[1];
    if (Udp.beginPacket(dst, destinationPort)){
      Udp.write(id[0]);Udp.write(id[1]);
      Serial.print(id[0], HEX);Serial.print(id[1], HEX);
      Serial.println();
      if (Udp.endPacket())
      {
        Serial.println("Packet sent");
      }
      else
      {
        Serial.println("Error sending");
      }
    }
  }
}


void printWifiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
```

```
  Serial.println(" dBm");
}
```

ANEXO 3 – MANUAL DE USUARIO

CAPÍTULO 1 – INICIALIZACIÓN DEL SISTEMA

1.1 Inicialización del emisor

El sistema presenta una gran simplicidad para su uso. Este simplemente se sujeta al usuario que lo vaya a portar y se enciende. Cada dispositivo tendrá un ID determinado el cuál será característico del paciente, de esta forma envía información de donde se encuentra el paciente, incluso informa de si esta ha sufrido una caída o no.

1.2 Inicialización del receptor

En caso de que la baliza receptora tenga un puerto Ethernet requerirá que, aparte de ser conectada a la alimentación vía USB tipo B, también se disponga cerca de un Router Wifi al que conectarse.

Si la baliza receptora dispone de un Shield WiFi solo requerirá conectarse a la alimentación.

CAPÍTULO 2 – CARGA DEL SISTEMA EMISOR

El sistema tiene un método de carga que presenta un método de seguridad para que la parte que ejecuta su función habitual no se vea afectada mientras este carga. De esta forma, simplemente hay que apagar el dispositivo para poder conectar a la alimentación el micro USB del que dispone.