

Embedded system – Proof Of Concept Report

Pulse Sensor

A4 NE2 Mokhtari – Moukembé – Petovello – Picard

Sommaire

Description du projet.....	3
Détail technique des composants	3
Détail du branchement électronique	4
Détail du code.....	5
Résultat et conclusion	10

Description du projet

Dans le cadre du projet de fin de semestre d'Embedded System, le Proof Of Concept, nous avons dû réaliser un système embarqué comptant au minimum 4 interactions. En ce qui nous concerne, nous avons opté pour une station médicale portable, car une partie du groupe travaillait (pour son PI²) sur la sécurité des personnes âgées et la surveillance de leur état de santé. Par ailleurs, ce même groupe possédait déjà un capteur cardiaque, il nous a donc parut évident de concevoir cette station médicale portable.

Détail technique des composants



Une Arduino Yun



Une Raspberry Pi

Pour le projet, nous avons choisi différents composants pour des raisons bien précises. Tout d'abord une Arduino Yun pour commander le capteur et une Led car nous maîtrisons bien cette partie (une Arduino Yun plutôt que Uno pour des raisons pratiques puisque nous travaillions déjà dessus durant les TP précédents). Les informations de la Arduino sont ensuite envoyées à une Raspberry Pi 3 par une communication I2C afin de traiter les données dans le but de les utiliser pour un site internet.

Pour afficher les données en temps réel, nous utilisons un écran LCD connecté en I2C avec la Raspberry, la Led (qui brille en fonction du battement cardiaque) et un site internet (en serveur local) qui affiche, sur une courbe, le suivi des battements cardiaques captés par le pulse sensor.



Le Pulse sensor



Une Led (rouge, bleu, ...)

Nous avons dans un premier temps pensé à utiliser des panneaux Led 8x8 pour un côté plus « esthétique » du visuel d'un battement de cœur, mais les panneaux utilisés étant défectueux, nous avons dû nous rabattre sur une Led tout simple.

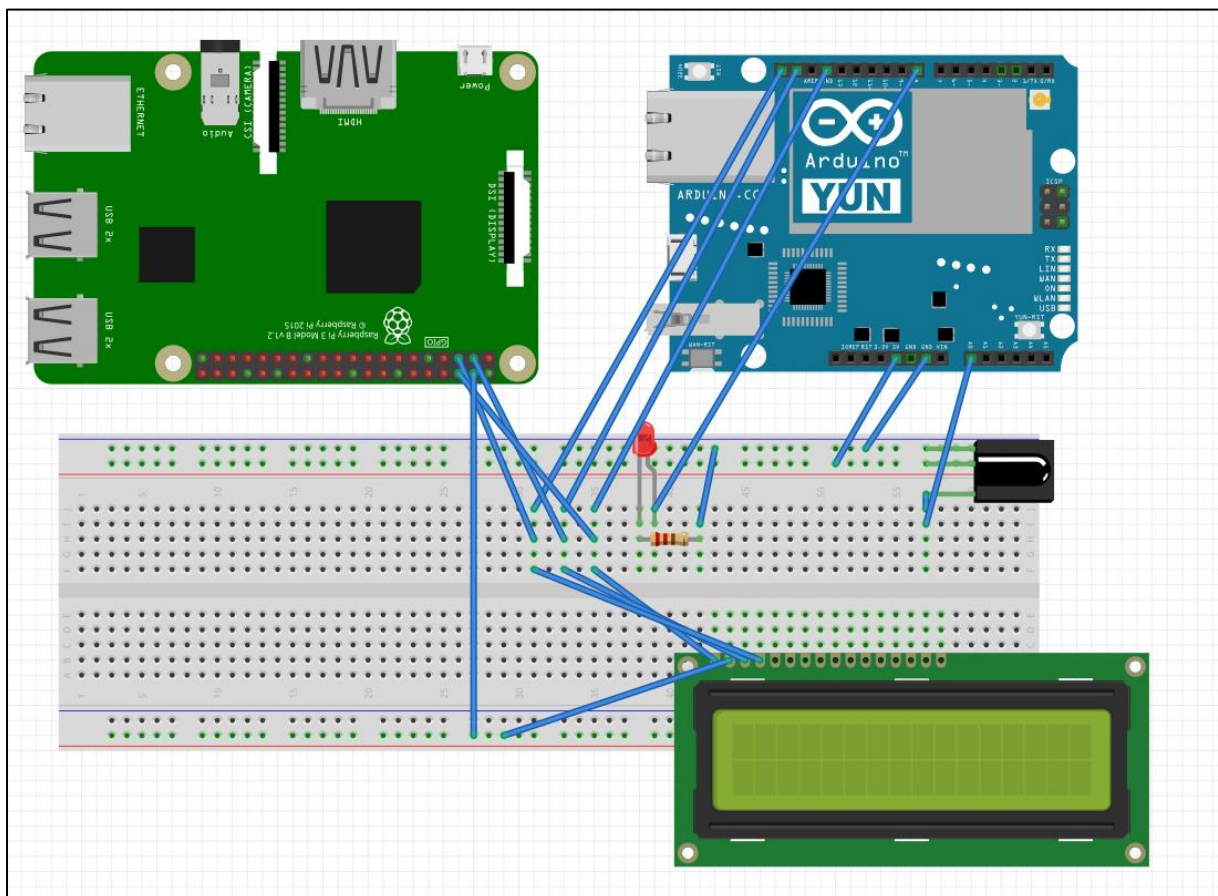


Un écran LCD



Une résistance 220 Ohm

Détail du branchement électronique



Ci-dessus, nous pouvons observer le schéma électrique et le lien entre chaque composant. Ainsi nous pouvons voir que la carte Arduino utilise ses ports SDA et SCL ainsi qu'un GND pour faire une communication I2C avec la Raspberry Pi sur les mêmes ports (il en est de même pour l'écran LCD, auquel on a juste rajouté une alimentation de 5V de la Raspberry Pi. On utilise aussi sur la Arduino la pin A0 et la pin 8 respectivement pour gérer le clignotement de la Led et le retour des valeurs analogiques du pulse sensor.

NB : le schéma électrique a été fait sur Fritzing, mais comme le pulse sensor n'était pas disponible dans la bibliothèque de composants, nous avons utilisé un capteur à 3 branches quelconque (pour modéliser l'alimentation, le GND et le port analogique).

Détail du code

Dans cette partie nous allons détailler les différents codes mis en place pour pouvoir exécuter les actions suivantes :

- Faire clignoter la Led en fonction de la pulsation cardiaque
- Afficher le nombre de battement par minute sur un écran LCD
- Faire communiquer la Arduino et la Raspberry
- Récupérer les données du capteur cardiaque
- Créer une page web depuis la Raspberry Pi

```
pi@192.168.137.15:22 - Bitvise xterm - pi@raspberrypi ~
GNU nano 2.7.4

import smbus
import time
import datetime
import shutil

# Define some device parameters
I2C_ADDR = 0x27 # I2C device address
LCD_WIDTH = 16 # Maximum characters per line

# Define some device constants
LCD_CHR = 1 # Mode - Sending data
LCD_CMD = 0 # Mode - Sending command

LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line
LCD_LINE_3 = 0x94 # LCD RAM address for the 3rd line
LCD_LINE_4 = 0xD4 # LCD RAM address for the 4th line

LCD_BACKLIGHT = 0x08 # On
#LCD_BACKLIGHT = 0x00 # Off

ENABLE = 0b00000100 # Enable bit

# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005

I2C_ARDUINO = 0x01
BPM = 0

#Open I2C interface
#bus = smbus.SMBus(0) # Rev 1 Pi uses 0
bus = smbus.SMBus(1) # Rev 2 Pi uses 1

def lcd_init():
    # Initialise display
    lcd_byte(0x33,LCD_CMD) # 110011 Initialise
    lcd_byte(0x32,LCD_CMD) # 110010 Initialise
    lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
    lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
    lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
    lcd_byte(0x01,LCD_CMD) # 000001 Clear display
    time.sleep(E_DELAY)

def lcd_byte(bits, mode):
    # Send byte to data pins
    # bits = the data
    # mode = 1 for data
    #       0 for command

    bits_high = mode | (bits & 0xF0) | LCD_BACKLIGHT
    bits_low = mode | ((bits<<4) & 0xF0) | LCD_BACKLIGHT

    # High bits
    bus.write_byte(I2C_ADDR, bits_high)
    lcd_toggle_enable(bits_high)

    # Low bits
    bus.write_byte(I2C_ADDR, bits_low)
    lcd_toggle_enable(bits_low)

def lcd_toggle_enable(bits):
    # Toggle enable
    time.sleep(E_DELAY)
    bus.write_byte(I2C_ADDR, (bits | ENABLE))
    time.sleep(E_PULSE)
    bus.write_byte(I2C_ADDR,(bits & ~ENABLE))
    time.sleep(E_DELAY)

def lcd_string(message,line):
    # Send string to display

    message = message.ljust(LCD_WIDTH, " ")

pi@192.168.137.15:22 - Bitvise xterm - pi@raspberrypi ~
GNU nano 2.7.4

E_DELAY = 0.0005

I2C_ARDUINO = 0x01
BPM = 0

#Open I2C interface
#bus = smbus.SMBus(0) # Rev 1 Pi uses 0
bus = smbus.SMBus(1) # Rev 2 Pi uses 1

def lcd_init():
    # Initialise display
    lcd_byte(0x33,LCD_CMD) # 110011 Initialise
    lcd_byte(0x32,LCD_CMD) # 110010 Initialise
    lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
    lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
    lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
    lcd_byte(0x01,LCD_CMD) # 000001 Clear display
    time.sleep(E_DELAY)

def lcd_byte(bits, mode):
    # Send byte to data pins
    # bits = the data
    # mode = 1 for data
    #       0 for command

    bits_high = mode | (bits & 0xF0) | LCD_BACKLIGHT
    bits_low = mode | ((bits<<4) & 0xF0) | LCD_BACKLIGHT

    # High bits
    bus.write_byte(I2C_ADDR, bits_high)
    lcd_toggle_enable(bits_high)

    # Low bits
    bus.write_byte(I2C_ADDR, bits_low)
    lcd_toggle_enable(bits_low)

def lcd_toggle_enable(bits):
    # Toggle enable
    time.sleep(E_DELAY)
    bus.write_byte(I2C_ADDR, (bits | ENABLE))
    time.sleep(E_PULSE)
    bus.write_byte(I2C_ADDR,(bits & ~ENABLE))
    time.sleep(E_DELAY)

def lcd_string(message,line):
    # Send string to display

    message = message.ljust(LCD_WIDTH, " ")
```

Ci-dessus, nous pouvons observer la première partie du code en python qui nous permet de gérer à la fois l’affichage sur l’écran LCD, la communication entre la Raspberry Pi et la Arduino et la gestion des données récupérées.

Ainsi nous avons, dans un premier temps, le code qui implante la communication I2C entre la Raspberry Pi et l’écran LCD (notamment avec la création d’une ligne de bus 0x27 pour l’écran LCD par exemple), et le paramétrage de l’écran pour réussir l’affichage plus tard. Ce code est avant tout un code préexistant, modifié par nos soins pour s’adapter à notre problématique.

```
pi@192.168.137.15:22 - Bitvise xterm - pi@raspberrypi ~
GNU nano 2.7.4

bits_low = mode | ((bits<<4) & 0xF0) | LCD_BACKLIGHT

# High bits
bus.write_byte(I2C_ADDR, bits_high)
lcd_toggle_enable(bits_high)

# Low bits
bus.write_byte(I2C_ADDR, bits_low)
lcd_toggle_enable(bits_low)

def lcd_toggle_enable(bits):
    # Toggle enable
    time.sleep(E_DELAY)
    bus.write_byte(I2C_ADDR, (bits | ENABLE))
    time.sleep(E_PULSE)
    bus.write_byte(I2C_ADDR, (bits & ~ENABLE))
    time.sleep(E_DELAY)

def lcd_string(message,line):
    # Send string to display

    message = message.ljust(LCD_WIDTH," ")

    lcd_byte(line, LCD_CMD)

    for i in range(LCD_WIDTH):
        lcd_byte(ord(message[i]),LCD_CHR)

def call_i2c_arduino():
    bus.write_byte(I2C_ARDUINO, 1)
    # Pause de 1 seconde pour laisser le temps au traitement de se faire
    time.sleep(1)
    reponse = bus.read_i2c_block_data(I2C_ARDUINO, 0, 1)
    reponsestr = []
    for i in range(len(reponse)):
        BPM = reponse[i]
    return BPM

def main():
    # Main program block

    # Initialise display
    lcd_init()
    #fichier = open("/var/www/html/graph/data.csv", "a")
    fichier = open("data.csv", "a")
    fichier.write("date" + "," + "BPM" + ",")
    fichier.close()

    while True:

        BPM = call_i2c_arduino()

        # Send some test
        lcd_string("RPISpy" + "<",LCD_LINE_1)
        lcd_string("<3 BPM "+str(BPM),LCD_LINE_2)
        fichier = open("/var/www/html/graph/data.csv", "a")
        fichier.write(str(datetime.datetime.now()) + "," + str(BPM) + ",\n")
        fichier.close()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
    finally:
        lcd_byte(0x01, LCD_CMD)
```

Cette deuxième partie du code concerne la communication entre les 2 cartes et le stockage des données récupérées. Nous avons par exemple la fonction `call_i2c_arduino()` qui a pour but de commander à l’Arduino l’envoi de ses données vers la Raspberry Pi (via le bus 0x01). Une fois la connexion établit et la récupération des données effectuées, nous avons décidé de créer dans le `main()` un fichier .csv dans lequel nous voulions stocker toutes les données. Le fichier contient alors les valeurs récupérer par la fonction `call_i2c_arduino()` ainsi que l’heure et la date correspondante à cette donnée (nous avons plus tard pourquoi).

Bien entendu ce fichier a été créé dans un répertoire spécifique « `/var/www/html/graph/data.csv` », répertoire qui nous servira pour la création de la page web.

FichierÉditionCroquisOutilsAide

Station_M_dicale

```
#define USE_ARDUINO_INTERRUPTS true
#include <PulseSensorPlayground.h>
#include <LiquidCrystal.h>
#include <Wire.h>

const int PulseWire = 0;
const int LED13 = 8;
int Threshold = 550;
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
PulseSensorPlayground pulseSensor;

void setup_PulseSensor()
{
    Serial.begin(9600);
    pulseSensor.analogInput(PulseWire);
    pulseSensor.blinkOnPulse(LED13);
    pulseSensor.setThreshold(Threshold);
    if (pulseSensor.begin())
    {
        Serial.println("We created a pulseSensor Object !");
    }
}

void setup_LCD()
```

Station_M_dicale | Arduino 1.8.9

FichierÉditionCroquisOutilsAide

Station_M_dicale

```
}

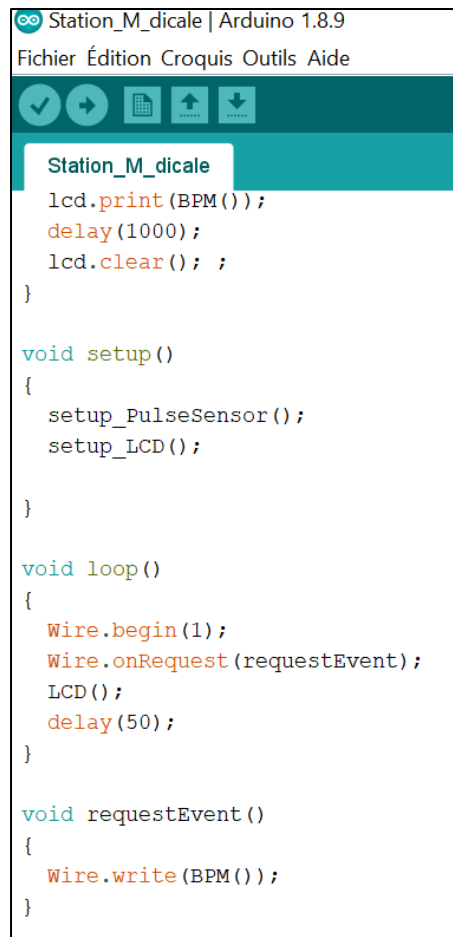
void setup_LCD()
{
    lcd.begin(16, 2);
}

int BPM()
{
    int myBPM = pulseSensor.getBeatsPerMinute();
    //delay(50);
    return myBPM;
}

void LCD()
{
    Serial.print("BPM:");
    Serial.print(BPM());
    Serial.print("\n");
    lcd.setCursor(0, 0);
    lcd.print("<3 BPM :");
    lcd.print(BPM());
    delay(1000);
    lcd.clear(); ;
}
```

Ce code Arduino nous permet de récupérer les données capter par le pulse sensor. A noter qu'une partie du code permettait d'afficher les battements par minute sur l'écran LCD directement depuis la carte Arduino, mais en raison de soucis technique (impossible d'afficher l'écran ET d'envoyer les données depuis l'Arduino vers la Raspberry Pi) nous avons choisi de privilégier la communication entre les 2 cartes. C'est d'ailleurs pour cette raison que la Raspberry Pi affiche les données en communication I2C sur l'écran LCD.

Le reste du code ne comporte aucune difficulté particulière, nous avons initialisé une pin analogique (A0) pour la capteur cardiaque et la pin 8 pour le clignotement de la LED. Pour vérifier que tout fonction, quelques Serial.print() nous permette d'afficher les résultats sur la console de l'IDE Arduino.



```
Station_M_dicale | Arduino 1.8.9
Fichier Édition Croquis Outils Aide

Station_M_dicale
  lcd.print(BPM());
  delay(1000);
  lcd.clear();
}

void setup()
{
  setup_PulseSensor();
  setup_LCD();
}

void loop()
{
  Wire.begin(1);
  Wire.onRequest(requestEvent);
  LCD();
  delay(50);
}

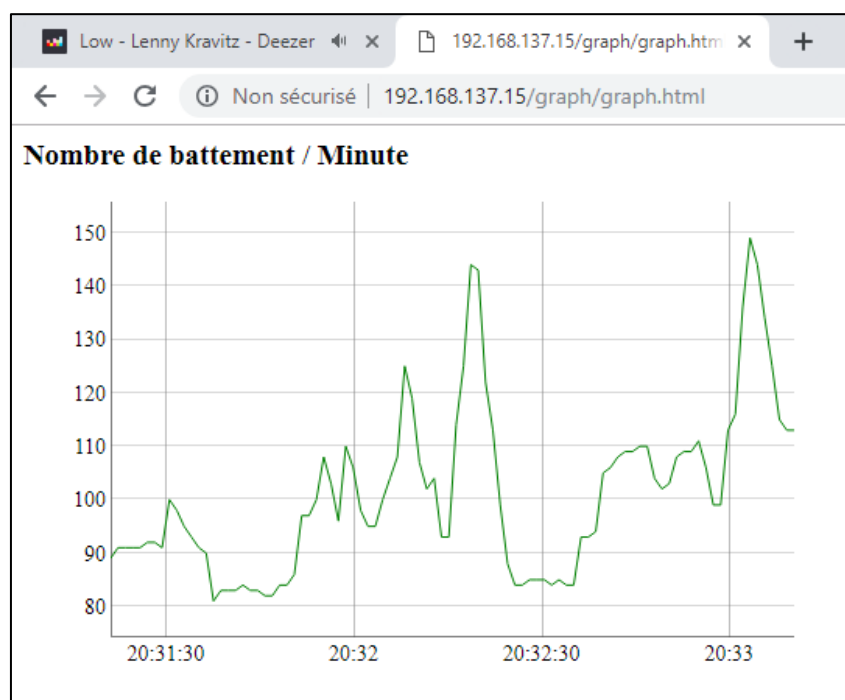
void requestEvent()
{
  Wire.write(BPM());
}
```

Cette dernière partie du code Arduino est là pour mettre en relation la carte Arduino avec la Raspberry Pi. Nous utilisons la fonction `Wire.begin()` pour définir l'adresse de communication de l'Arduino (0x01). Puis on définit un `Wire.onRequest(requestEvent)` l'ordre à exécuter tel que `requestEvent()` dit `Wire.write(BPM)`. Ainsi par ces deux fonctions nous avons récupéré puis permis la transmission de la donnée BPM (donnée qui détermine le nombre de battement par minute capté par le capteur à un instant T) de l'Arduino à la Raspberry Pi


```
pi@192.168.137.15:22 - Bitvise xterm - pi@raspberrypi: /var/www/html/graph
GNU nano 2.7.4

<html>
<head>
<script type="text/javascript"
  src="dygraph.js"></script>
<link rel="stylesheet" src="dygraph.css" />
</head>
<body>
<div>
<h3>Nombre de battement / Minute </h3>
</div>
<div id="graphdiv2"
  style="width:500px; height:300px;"></div>
<script type="text/javascript">
function reload()
{
    g2 = new Dygraph(document.getElementById("graphdiv2"),"data.csv",{ });
}
reload();
setInterval(function(){
    reload() // this will run after every 5 seconds
}, 1000);
</script>
</body>
</html>
```

Ce petit code html, dynamiquement produit en javascript, utilise le fichier .csv (produit par le code python) pour créer un graphique avec la valeur des battements par minute en ordonnée et le temps en abscisse. De plus, dans un souci de dynamisme visuel, nous relançons la création du graphique toutes les secondes. Ainsi, comme le .csv continue à se remplir au fur et mesure que le programme s'exécute, nous pouvons voir la courbe évoluer. On teste cette solution en local, c'est-à-dire qu'on utilise l'adresse IP de la Raspberry Pi sur un moteur de recherche quelconque pour lancer le fichier html (en écrivant par exemple : 192.168.137.42/graph/graph.html). Nous obtenons alors ceci.



Résultat et conclusion

Nous avons voulu, à travers ce projet, mettre en avant toutes les compétences acquises au cours de ce module. Ainsi nous pouvons retrouver du développement web, de la communication I2C entre cartes, de l'électronique ou encore de la programmation sur plusieurs langage (Arduino, python, javascript). Dans l'ensemble le projet a été un succès, nous avons abouti sur ce que nous envisagions, les rares changements de matériels (Led à la place de panneaux ou LCD en I2C avec la Raspberry à lieu de l'Arduino) sont des soucis mineurs que nous avons pu vite surmonter !