Laboratori 2:

Processament de senyal amb ADC

**Informe de la pràctica**

*Sistemes Encastats i Ubics (SEU)*

Màster en Enginyeria Informàtica - FIB UPC

**Autores:**

Cristina Migó Lluis

Àlex Ollé Parcerisas

**Professor:** Manel Frigola Bourlon

*21 d’octubre del 2024*

# Introducció

L’objectiu d’aquesta pràctica consisteix a realitzar un codi que permeti monitorar a través d’un pulsòmetre la freqüència cardíaca d’una persona per tal de familiaritzar-nos amb les entrades analògiques i les tècniques de processament de senyal.

Per a fer-ho, usarem un pulsòmetre que retorna valors entre 0 i 1 per una de les entrades analògiques que té el nostre microxip LPC1768 i processar el senyal per tal de reconèixer el senyal del batec del cor i mesurar-ne la freqüència cardíaca.

Un dels problemes amb els quals ens hem trobat, ha estat en el fet que hi havia molt soroll en el senyal, la qual cosa feia que fos complicat obtenir el pols correcte. No sempre podíem donar els valors com a senyals del batec i, per tant, caldria usar diverses tècniques que ens permetessin netejar els valors obtinguts i finalment detectar de manera més o menys precisa el pols.

# Obtenció del Threshold

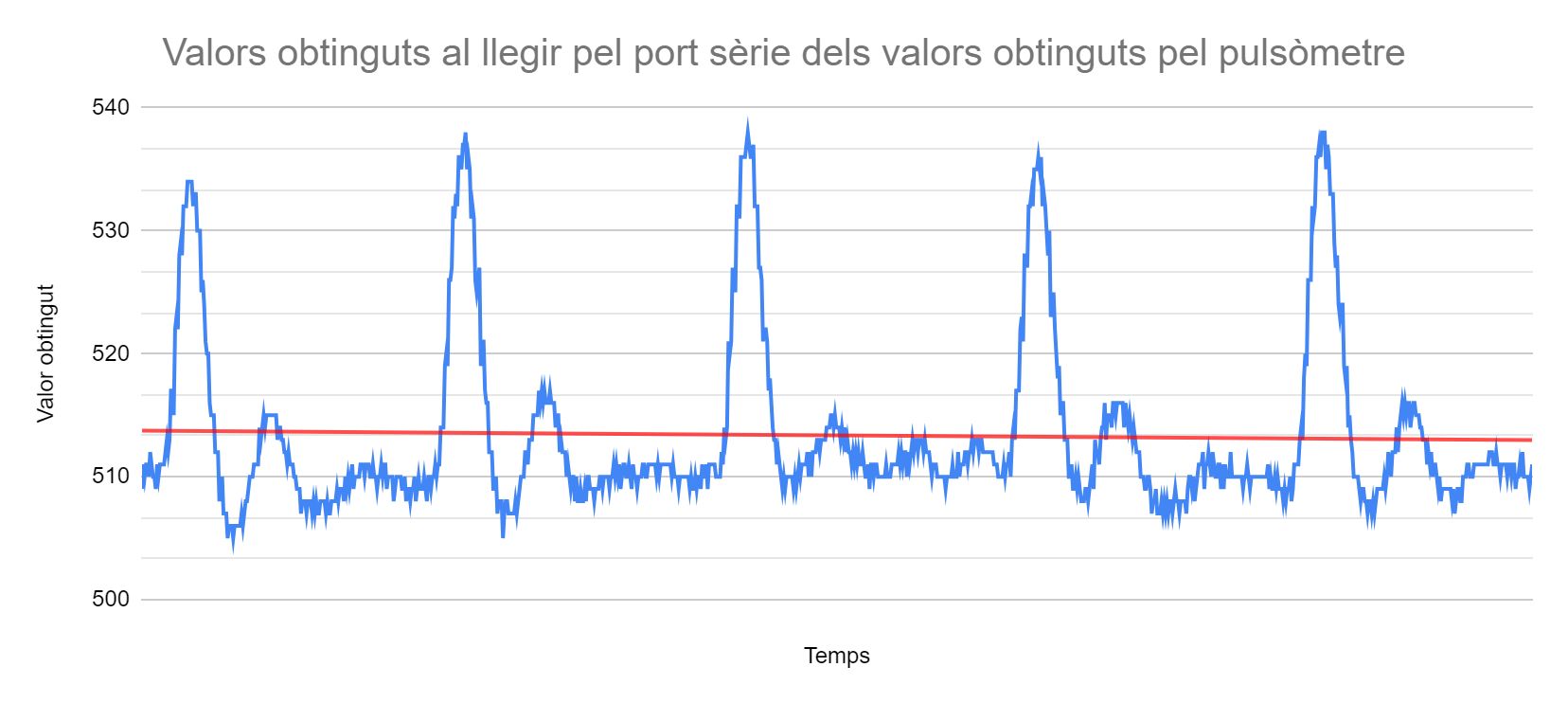
El primer que vam intentar, va ser calcular de manera dinàmica el llindar, fent que es guardessin els màxims i mínims i es busqués un llindar, de manera en què no calia donar un valor fix, sinó que es podia autocalcular. El problema que vam tenir va ser que amb el soroll produït, era complicat tenir un valor de threshold estable i que ens donés sempre un valor correcte, per la qual cosa vam desestimar la idea i vam decidir obtenir un valor fix.

Per tal d’obtenir un valor mitjà que ens permetés detectar quan teníem un senyal ascendent i, per tant, un batec del cor. Per a fer-ho, ens vam fer un programa molt senzill el qual simplement llegia del port analògic el valor obtingut i l’escalava a valors de zero a 1000 per tal que fos més precisa la sortida (al no permetre valors decimals). Un cop vam registrar una quantitat significativa de valors llegint cada 50 ms (més de 1000), vam posar-los en un Excel i en vam calcular els valors, obtenint com a resultat els següents valors:

| **Tipus** | **Valor** |
| --- | --- |
| Mínim | 505 |
| Màxim | 538 |
| Valor mitjà | 513 |

Com podem veure a la taula, tenim que el valor mínim és de 505, és a dir, per sobre de 0,5 i que el valor màxim estava a 32 valors de diferència, això feia que no tinguéssim gaires valors entre l'un i l’altre i, per tant, no fos tan notable la diferència.

Amb les dades vam poder crear un gràfic, el qual també ens va ajudar a entendre i verificar que els valors que estàvem llegint eren correctes i anàvem per bon camí.



En el gràfic podem veure clarament els batecs del cor amb el batec principal i el secundari, on el primer es pot apreciar molt millor que el segon. En vermell també hem volgut marcar el valor mitjà obtingut, on és el 513 mencionat anteriorment. Així i tot, podem veure com el batec secundari també hi arriba, a més que alguns sorolls també s’hi acosten molt. És per aquest motiu, que vam decidir obtenir un valor on només els pics poguessin arribar i on el senyal fos tan neta com sigui possible.

És per això que vam decidir agafar el valor de 520 com a llindar a superar i treballar en la neteja del soroll per tal d’evitar obtenir falsos positius.

# Explicació del codi

En aquest apartat es dividirà el codi en petits fragments i s’explicaran per parts per tal de facilitar-ne la comprensió.

### 3.1. Declaració de variables globals

En aquest primer fragment de codi, podem veure les variables globals declarades en el programa per tal de mantenir la consistència. És important destacar el pulseMinThreshold, el qual conté el valor del threshold que es tindrà en compte per a veure si estem en un flanc de pujada, el pulseReadings que és un array per poder calcular el valor mitjà de batecs per tal d’evitar valors erronis o fora de rang en escriure-ho per pantalla. Finalment, tenim els valors de batec mínim i màxim per tal d’identificar valors fora de rang i no tenir-los en compte.

| #define WAITING\_TIME 50  AnalogIn heart\_sensor(p20);  // Paràmetres de configuració float pulseMinThreshold = 520; // Umbral de detecció flanc ascendent const int minPulseInterval = 300; // Mínim temps entre pulsos (calculat fent 1/200BPM = 300ms)  const int windowSize = 20; // Tamany del array de mostres float pulseReadings[windowSize] = {0}; // Array de mostres inicialitzat amb zeros int readingIndex = 0; // Index per saber quina mostra sobrescriure  float currentBPM = 0; // Valor actual  float minBPM = 30; // Valor mínim float maxBPM = 220; // Valor màxim de BPM int lastPrintTime = 0;  Timer pulseTimer; Timer printTimer; |
| --- |

### 3.2. Funció passa baixos

Aquesta funció permet aplicar un filtre passa baixos per tal de netejar el soroll que es pugui produir en llegir de l’analògic i fer servir valors que segueixin la tendència. El que fa aquest filtre és que cada valor només compti una petita part (en el nostre cas alpha serà 0.2), fent que si la tendència és ascendent, aquest valor vagi formant-se a partir de diverses lectures i, per tant, si hi ha un valor de soroll, no farà un canvi radical, sinó que simplement farà un petit canvi. Amb això, hem evitat gran part del soroll produït i aconseguir uns valors més lineals.

| float lowPassFilter(float currentValue, float previousValue, float alpha) {  return alpha \* currentValue + (1.0 - alpha) \* previousValue; } |
| --- |

### 3.3. Funció d’impressió per pantalla

Aquesta funció s’encarrega d’imprimir les pulsacions per minut cada segon a partir de calcular la mitjana entre els últims valors llegits, podent aconseguir un valor que no oscil·li de manera molt exagerada, sinó que vagi modificant-se gradualment. Com que a la versió del programari mbed OS 6 el SerialPC ja no és necessari per escriure per pantalla, podem usar el *printf* normal per tal d’obtenir els valors necessaris. Per a calcular la mitjana entre els últims 20 valors de BPM obtinguts, el que fem és sumar tots els valors diferents de zero (per evitar acumular els zeros amb el que l’iniciem) i fer-ne la divisió.

| void printBPM() {  int currentTime = duration\_cast<milliseconds>(printTimer.elapsed\_time()).count();  if (currentTime - lastPrintTime > 1000) { // Imprimim el valor mitjà cada segon  int bpm = 0;  int counter = 0;  for(int num : pulseReadings) {  if (num > 0) {  bpm += num;  counter++;  }  }  if(counter > 0) {  printf("Pulsaciones: %d \n", bpm / counter);  lastPrintTime = currentTime;  }   } } |
| --- |

### 3.4. Lectura del port analògic

En aquest fragment, on podem veure el main de l’aplicació, inicialitzem els dos timers (cosa que ens permetrà saber el temps que ha passat des de l’anterior pols i el que ens permet imprimir cada segon), així com inicialitzarem algunes variables locals que ens permetran validar els valors obtinguts i mantenir-ne la coherència.

Llavors tenim un bucle infinit el que cada 50 ms, llegeix del sensor, converteix el valor en una escala major fent-lo múltiple de 1024 bits i mirem si el senyal està entre el mínim i el màxim que hem cregut raonables després de l’estudi del threshold donant-los un petit marge de tolerància per evitar retallar en excés. Finalment, apliquem el filtre passa baixos explicats anteriorment per normalitzar el valor obtingut i ens guardem el valor com a lastFilteredValue. Amb això, hem aconseguit un valor prou estable per a començar a treballar per veure si és candidat a marcar una pulsació.

Per tal de comprovar si cal imprimir el valor de BPM, cridem la funció a cada iteració per tal d’evitar desajustos.

| int main() {  pulseTimer.start();  printTimer.start();  int lastPulseTime = 0;  float lastValidBPM = 0;  float lastFilteredValue = pulseMinThreshold;   while (1) {  float sensorValue = heart\_sensor.read() \* 1023;  if(sensorValue >= 500.f && sensorValue <= 550.f) {  float filteredValue = lowPassFilter(sensorValue, lastFilteredValue, 0.2);  lastFilteredValue = filteredValue;  *...* // Analitzarem aquest codi posteriorment  }  printBPM();  wait\_us(50);  } } |
| --- |

### 3.5. Obtenció de les pulsacions

Aquest fragment de codi s’encarrega de validar si realment el valor obtingut és candidat d’indicar que hi ha hagut un nou pols i, per tant, de poder calcular el valor.

Per a fer-ho, primer de tot comprovem si el valor filtrat està per sobre del threshold marcat i si és així, mirem si el temps que ha passat entre l’última lectura és suficient. Si és així, calculem els BPM i mirem que sigui un valor que tingui sentit comprovant que estigui entre el mínim i màxim marcat. En cas positiu, s’afegeix un nou registre a l’array de lectures perquè es tingui en compte a l’hora de mostrar-ho per pantalla.

| if(filteredValue > pulseMinThreshold) {  int currentTime = duration\_cast<milliseconds>(pulseTimer.elapsed\_time()).count();  if((currentTime - lastPulseTime) > minPulseInterval) {  float interval = (currentTime - lastPulseTime) / 1000.0;  float bpm = 60.0 / interval;  if(minBPM < bpm < maxBPM) {  pulseReadings[readingIndex % windowSize] = bpm;  readingIndex++;  }  lastPulseTime = currentTime;   }  } |
| --- |

# Resultats

Amb el codi anterior hem aconseguit obtenir uns resultats prou coherents, on no hi ha gaires variacions i que quan la persona força una pujada de pulsacions, aquestes augmenten en conseqüència. A continuació es poden veure una mostra dels valors obtinguts:

Pulsaciones: 67

Pulsaciones: 67

Pulsaciones: 67

Pulsaciones: 64

Pulsaciones: 64

Pulsaciones: 64

Pulsaciones: 64

Pulsaciones: 65

Pulsaciones: 66

Pulsaciones: 67

Pulsaciones: 68

Pulsaciones: 69

Pulsaciones: 70

Pulsaciones: 70

Pulsaciones: 69

Pulsaciones: 68

Pulsaciones: 68

Pulsaciones: 69

Pulsaciones: 72

Pulsaciones: 73

Pulsaciones: 74

Pulsaciones: 74

Pulsaciones: 74

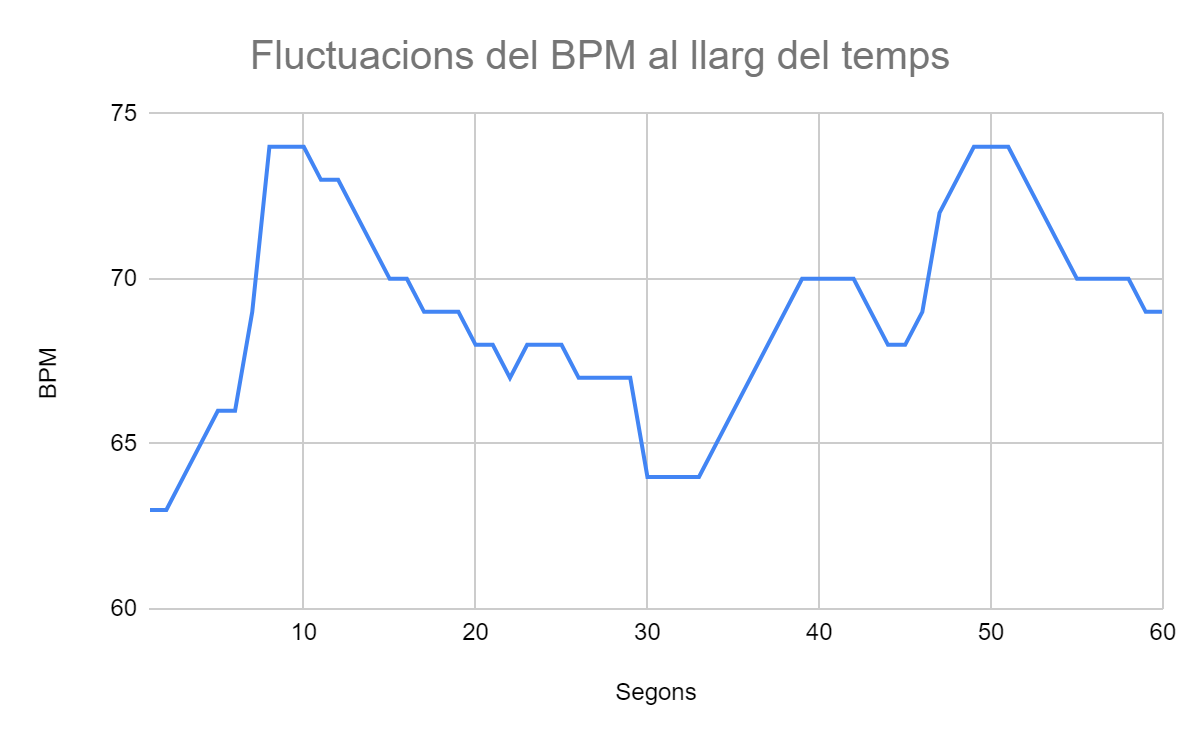
Pulsaciones: 73

Pulsaciones: 72

Pulsaciones: 71

Pulsaciones: 70

Si agafem els valors obtinguts i els posem en un full de càlcul, podem extreure un gràfic amb les fluctuacions al llarg del temps d’aquestes pulsacions.



En el gràfic podem veure com en accelerar el ritme cardíac, aquest augmenta en conseqüència, així com baixa gradualment quan la persona para de relaxar-lo i torna a augmentar en tornar a forçar un augment de pulsacions.

Amb això, podem veure com realment la lectura de les pulsacions té sentit i fa una lectura aproximada de les pulsacions en cada segon de la persona.

# Conclusions

Creiem que hem estat capaços de superar el repte plantejat, podent obtenir el pols d’una persona al llarg del temps de manera més o menys precisa i on tingui un cert sentit.

En aquesta pràctica hem pogut aprendre com llegir d’una entrada analògica, així com filtrar-ne els valors per aconseguir el soroll que el detecto pugui tenir donat a factors externs i poder realitzar els càlculs amb aquests valors.

Per tant, podem donar la pràctica com a satisfactòria aconseguint un programa que resol el problema plantejat.

# Annex 1: Codi sencer

| #include "mbed.h" #include "Timer.h" #include <chrono> #include <cstdio>  using namespace std::chrono; #define WAITING\_TIME 50  AnalogIn heart\_sensor(p20);  // Paràmetres de configuració float pulseMinThreshold = 520; // Umbral de detecció flanc ascendent const int minPulseInterval = 300; // Mínim temps entre pulsos (calculat fent 1/200BPM = 300ms) const int windowSize = 20; // Tamany del array de mostres float pulseReadings[windowSize] = {0}; // Array de mostres inicialitzat amb zeros int readingIndex = 0; // Index per saber quina mostra sobrescriure  float minBPM = 30; // Valor mínim BPM float maxBPM = 220; // Valor màxim de BPM  int lastPrintTime = 0;  Timer pulseTimer; Timer printTimer;  float lowPassFilter(float currentValue, float previousValue, float alpha) {  return alpha \* currentValue + (1.0 - alpha) \* previousValue; }  void printBPM() {  int currentTime = duration\_cast<milliseconds>(printTimer.elapsed\_time()).count();  if (currentTime - lastPrintTime > 1000) { // Imprimim el valor mitjà cada segon  int bpm = 0;  int counter = 0;  for(int num : pulseReadings) {  if (num > 0) {  bpm += num;  counter++;  }  }  if(counter > 0) {  printf("Pulsaciones: %d \n", bpm / counter); // Calculem el valor mitjà  lastPrintTime = currentTime;  }   } }  int main() {  pulseTimer.start();  printTimer.start();  int lastPulseTime = 0;  float lastValidBPM = 0;  float lastFilteredValue = pulseMinThreshold;   while (1) {  float sensorValue = heart\_sensor.read() \* 1023; // Escalem el valor obtingut  if(sensorValue >= 500.f && sensorValue <= 550.f) { // Validem que el sensor tingui una lectura dins dels nostres marges  float filteredValue = lowPassFilter(sensorValue, lastFilteredValue, 0.2); // Apliquem filtre passa baixos  lastFilteredValue = filteredValue;   if(filteredValue > pulseMinThreshold) { // Comprovem si el valor està per sobre del threshold  int currentTime = duration\_cast<milliseconds>(pulseTimer.elapsed\_time()).count();  if((currentTime - lastPulseTime) > minPulseInterval) { // Comprovem que hagi passat el temps mínim entre polsos  float interval = (currentTime - lastPulseTime) / 1000.0;  float bpm = 60.0 / interval;  if(minBPM < bpm < maxBPM) { // Comprovem que el valor de BPM obtingut tingui sentit  pulseReadings[readingIndex % windowSize] = bpm; // Afegim el valor al array de lectures  readingIndex++;  }  lastPulseTime = currentTime;   }  }  }  printBPM();  wait\_us(50);  } } |
| --- |