Xavier Del Campo Romero

Analitzador de xarxes elèctriques amb Arduino

TREBALL DE FÍ DE GRAU

Dirigit Jose Luis Ramírez Falo

Grau en Enginyeria Elèctrica

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2015

Índex

1 Introducció

1.1 Descripció del TFG

El Treball Fi de Grau consisteix en el disseny d'un analitzador de xarxes elèctriques realitzat utilitzant la plataforma Arduino, desenvolupament de hardware i software obert. També s'ha construït un prototip i s'han realitzat alguns assaigs d'exemple.

1.2 Què és Arduino?

Arduino és una plataforma de desenvolupament creada per l'equip d'Arduino (amb base a Itàlia) i la comunitat dedicada a la creació de hardware i software de fàcil ús i intuïtiu, enfocat per a tot tipus de públic: des de dissenyadors, artistes i aficionats a l'electrònica fins a empreses.

Es distribueixen una àmplia gamma de productes, essent l'Arduino UNO el buc insígnia. Consta d'una PCB amb un microcontrolador ATMega 328P com a cor del dispositiu. Amb dues files de connectors connectades al xip a través de les pistes, la UNO disposa de pins d'entrada i sortida digitals i analògics que permeten interactuar amb hardware i dispositius externs, com motors, LED, pantalles... La majoria de plaques Arduino s'alimenten de dues maneres: a 9 V a través d'un adaptador de tensió o mitjançant connexió USB a un ordinador. Aquest últim mode permet també interactuar amb el PC (pujada de codi, comunicació per serial, etc.).

La programació en Arduino està dissenyada per ser el més senzilla possible. Des de la web oficial, es proporciona un IDE (*Integrated Development Environment*, en anglès), un editor de text avançat que incorpora les llibreries oficials. El llenguatge de programació utilitzat és C++, un llenguatge orientat a objectes molt conegut i estès en el mon de la informàtica.

Les funcionalitats d'Arduino poden estendre's amb l'ús de shields, plaques de mida estàndard que s'apilen a la placa principal. Existeixen multitud de productes diferents: adaptadors d'Ethernet, plaques protoboard, pantalles, mòduls Bluetooth, etcètera.

La naturalesa oberta i transparent d'Arduino ha permès una evolució molt rellevant en els últims anys. Multitud d'empreses i aficionats han mostrat interès per la plataforma, ja sigui creant nous shields, llibreries o fins i tot projectes comercials basats en plaques Arduino.

1.3 Què és un analitzador de xarxes?

En l'enginyeria elèctrica, els analitzadors de xarxes són dispositius utilitzats per multitud d'aplicacions. D'una manera resumida, es tracta d'aparells que permeten fer mesures de tensió i corrent d'un sistema elèctric, com faria un tester convencional, amb moltes més funcionalitats. Algunes d'elles són la capacitat d'emmagatzemar les dades per a una posterior interpretació, mostrar informació sobre potències real, reactiva i aparent, factor de potència, distorsió harmònica total de l'ona, etcètera.

Figura 1. Vista frontal de dos analitzadors de xarxes: un de la casa Fluke (esquerra) i un de HT Instruments (dreta).

Aquestes dades esdevenen de gran utilitat per conèixer en profunditat el funcionament d'una xarxa elèctrica. Per exemple, un analitzador de xarxes elèctriques es pot utilitzar per determinar la qualitat de subministrament i consum de l'energia elèctrica que arriba a un determinat abonat.

Els analitzadors de xarxes elèctriques són aparells de precisió homologats, destinats a empreses i enginyers elèctrics. Són comercialitzats a un preu molt elevat, superant molts d'ells el miler d'euros. La necessitat de cercar una solució amb prestacions més bàsiques però un cost més assequible esdevé la raó d'aquest TFG.

2 Antecedents

Actualment, existeixen tota una sèrie d'analitzadors de xarxes comercials de diversos fabricants: Fluke, HT Instruments, PCE Instruments, etc. Aquests dispositius compten amb una sèrie de característiques diferents en funció de la gamma, però la majoria d'ells destaquen per poder registrar dades en un sistema d'emmagatzematge concret, ser de gran versatilitat i amb una gran quantitat de paràmetres ajustables. Tanmateix, són instruments de gran complexitat i precisió que són comercialitzats a preus molt elevats, ja que estan destinats normalment a grans consumidors d'energia elèctrica o a empreses vinculades al subministrament d'enginyeria elèctrica (certificacions d'eficiència energètica, anàlisi del funcionament de la maquinària, mesura de la qualitat del subministrament, etcètera).

En aquest treball es comprova la viabilitat d'implementar un analitzador de xarxes elèctriques mitjançant una placa Arduino. Si bé és cert que existeixen una sèrie de limitacions, principalment donades pel hardware, que no permetran assolir el nivell qualitatiu ni les innumerables opcions dels equips comercials, el cost del prototip d'analitzador de xarxes construït se situa entorn dels 50,00 € (150,00 € amb pinces incloses), un preu molt inferior a qualsevol model comercial.

3 Hardware

Per al nostre TFG, es va proposar la següent arquitectura de hardware:

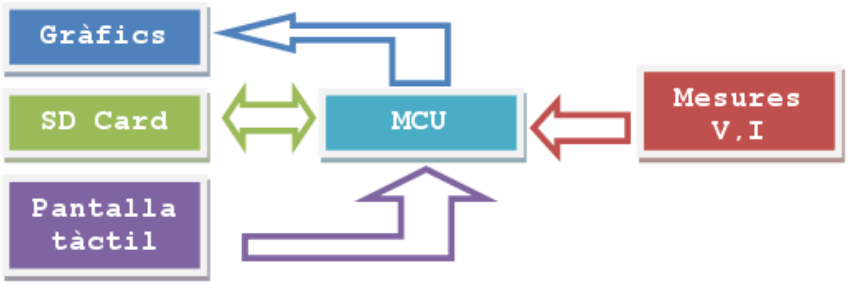


Figura 2. Diagrama de principi de funcionament del dispositiu.

El MCU (sigles de *Micro-Controller Unit*) o microcontrolador contindrà dins de la seva memòria el programa i rebrà la informació dels perifèrics d'entrada (lectura de la targeta SD, pantalla tàctil i mesures de corrent i tensió). Aquest flux de dades serà interpretat pel microcontrolador i executarà una sèrie d'instruccions que permetran fer actuar els perifèrics de sortida (escriptura a la targeta SD i gràfics per pantalla). Així doncs, l'Arduino esdevé el cor del sistema.

Les funcionalitats d'Arduino es poden estendre mitjançant mòduls anomenats *shields*. Les mesures de tensió i corrent poden realitzar-se gràcies a un shield anomenat EmonTX Shield, mentre que la pantalla tàctil i la targeta SD formen part d'un altre fabricat per Itead Studio. A continuació es descriuen les característiques d'ambdós dispositius i del microcontrolador Arduino utilitzat.

3.1 EmonTX Shield

Desenvolupat per la comunitat OpenEnergyMonitor, aquest shield inclou quatre connectors jack de 3,5 mm i una entrada de 5,5 x 2,1 mm. En els connectors jack s'inseriran les pinces amperimètriques per a les lectures de corrent, i al connector de tipus barril es connectarà un adaptador de 9 V AC per a les lectures de tensió. L'EmonTX inclou un conjunt de circuits senzills per tal d'adaptar els paràmetres d'entrada en tensions amb valors entre 0 i 5 V, adaptades per tal de ser llegides pel convertidor analògic/digital de l'Arduino (d'ara en endavant, ADC).

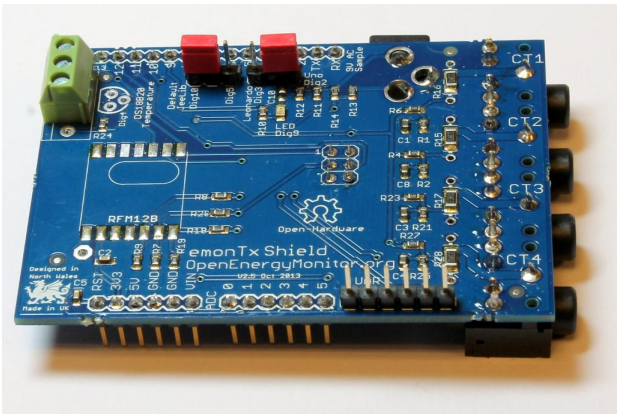


Figura 3. Vista superior de l'EmonTX Shield.

De forma opcional, pot instal·lar-se una antena de tipus RFM12B per a la transmissió de dades a un servidor. Aquesta funcionalitat no ha estat d'interès per al nostre TFG, ja que volíem un sistema autónom.

3.2 Itead Studio 2.8" Screen Shield

El conjunt inclou un altre shield, fabricat per Itead Studio, que consta d'una pantalla tàctil resistiva LCD de 2,8 polzades amb una ranura per a targeta SD. Té un selector de tensió lògica 5/3,3 V, afegint així doncs compatibilitat amb les noves plaques Arduino Due (basades en arquitectura ARM i operant a 3,3 V en comptes dels 5 V usats en els microcontroladors AVR). El shield és plenament compatible amb les llibreries UTFT, UTouch i SDFatLib.

Figura 4. Vistes frontal i posterior de la pantalla LCD amb ranura per a SD usada.

3.3 Arduino UNO a Mega 2560

En un primer moment, la placa de desenvolupament utilitzada fou l'Arduino UNO R3, la qual està formada per una MCU ATMega 328p funcionant a 16 MHz, 32 kB de memòria flash i 4 kB de SRAM. L'Arduino UNO compta amb 13 entrades digitals i 6 entrades analògiques. Els pins estan distribuïts per la placa de tal forma que es poden comprar uns mòduls anomenades shields, amb una grandària estandarditzada, que s'acoblen damunt de la placa i permeten afegir noves funcionalitats.

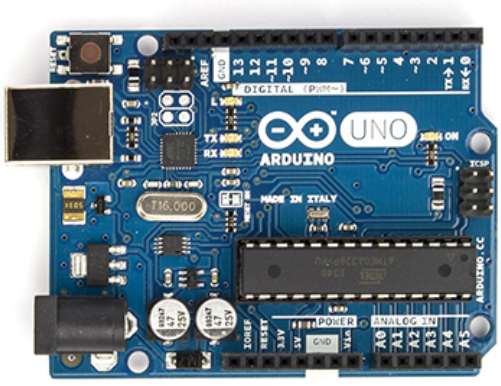


Figura 5. Vista superior de l'Arduino UNO R3.

L'Arduino UNO ràpidament va ser descartada com a opció per a realitzar el TFG. Les raons principals van ser la falta de pins quan es connectaven els dos shields i la falta de memòria flash. A la Fig. 6 es mostra l'esquema de l'EmonTX. Observem que els jacks estèreo utilitzen les entrades analògiques 0 a 5, de manera que ja s'estan utilitzant totes les entrades analògiques disponibles en l'Arduino UNO. Les entrades digitals, si observéssim l'esquema als annexes, són utilitzades per a l'antena RFM12B, el sensor d'humitat i el LED integrat, elements que no seran utilitzats en el nostre projecte; així que, resumidament, l'EmonTX Shield només utilitza (a més de les entrades de tensió i reset, que són comunes a tots els shields) les entrades ADC 0 a 5.

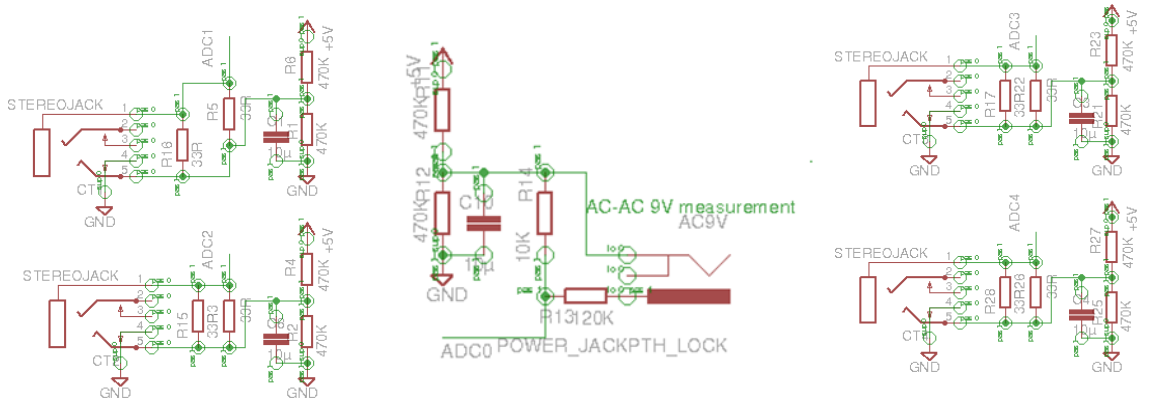


Figura 6. Detall dels esquemàtics de l'Emon TX Shield.

El problema més greu es va trobar en la utilització de la pantalla LCD. Si n'observéssim l'esquema, aquest shield fa ús de pràcticament tots els pins anàlogics i digitals de l'Arduino UNO. Això, per tant, ens impossibilita utilitzar al mateix temps l'EmonTX Shield, ja que necessitàvem les entrades analògiques ADC 0 a 5.

D'altra banda, 32 kB d'espai a la memòria flash ràpidament van esdevenir insuficients per a un treball d'aquesta grandària. Diverses llibreries de tercers, especialment UTFT i UTouch, tenen funcions que són molt pesades en termes memòria ocupada. De fet, únicament un petit exemple "Hello World!" que funcionés en la nostra pantalla LCD podria arribar a consumir fins a 22 dels 32 kB de la memòria flash disponible a l'Arduino UNO. Si hi haguéssim d'afegir el codi que permet la lectura dels paràmetres procedents de l'ADC, i el processament d'aquestes dades, la memòria s'exhauriria molt ràpidament.

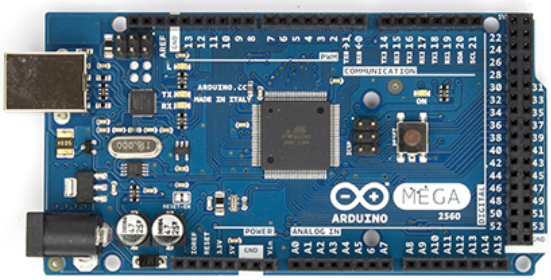


Figura 7. Vista superior de l'Arduino MEGA 2560.

Es va decidir canviar la placa de desenvolupament, buscant-ne una amb més entrades analògiques i major memòria flash. Alhora, havia de trobar-se dins de la mateixa arquitectura (AVR) que permetés la compatibilitat absoluta amb els dos shields ja adquirits. L'Arduino Mega 2560 es va convertir en la nova placa de desenvolupament. L'Arduino Mega 2560 compta amb 256 kB de memòria flash, MCU ATMega 2560 a 16 Mhz, 4 kB de SRAM, 54 entrades/sortides digitals i 16 entrades analògiques. Aquesta nova placa permetia no només fer treballar ambdós shields alhora, sinó que també afegia molt més espai a la memòria que es tradueix en més característiques per al nostre analitzador.

4 Software

4.1 Característiques bàsiques

El nostre analitzador de xarxes permet a l'usuari obtenir dades sobre tensió a través d'un adaptador 230 – 9 V i sobre corrent a través d'unes pinces amperimètriques de 100A:50mA, tot mitjançant una interfície gràfica molt intuïtiva. Les dades poden visualitzar-se de dues formes: a través de la pantalla que incorpora el dispositiu o des d'un dispositiu extern que pugui llegir les dades emmagatzemades a la targeta SD. A continuació tenim una llista de les funcionalitats bàsiques de l'aparell:

- Lectura en temps real del valor eficaç de tensió monofàsica a la xarxa.
- Lectura en temps real dels valors eficaços de corrent, amb capacitat fins a 4 pinces amperimètriques (sistema trifàsic + neutre).
- Possibilitat d'emmagatzemar les dades en una targeta SD en arxius de versió .TXT per ser fàcilment manipulats amb un full de càlcul com Microsoft Excel o OpenOffice Calc.
- Possibilitat d'escollir el temps de mostreig a l'hora d'emmagatzemar les dades en una targeta SD:
    - Valors instantanis
    - Valors eficaços: aproximadament 10 períodes d'ona
    - Mitjana aritmètica dels valors eficaços acumulats durant un cert període de temps completament ajustable (per exemple, 10 minuts).
- Oscil·loscopi bàsic amb capacitat de mostrar els senyals de tensió i corrent per pantalla amb resolució i temps de mostreig ajustables.
- Interfície gràfica molt intuïtiva amb menú de configuració i calibratge.

4.2 Criteri de disseny

En el següents apartats s'explicarà l'evolució del software des d'una perspectiva temporal. Es començarà per una explicació sobre el criteri de decisió utilitzat per escollir les eines de desenvolupament i la forma en què s'han estructurat els directoris del programa. Més endavant, es detallarà el funcionament intern de les parts més importants del programa i s'esmentaran algunes preses de decisions dutes a terme al llarg del TFG.

4.3 De l'Arduino IDE a Makefiles

L'Arduino IDE és un entorn integrat de desenvolupament proporcionat per l'equip Arduino. És un programa multi-plataforma, amb versions per a Windows, Linux i Mac OS X. Les seves funcions principals són les de proporcionar un editor de text avançat per a la creació de codi i una interfície per poder pujar aquest codi a la placa a través de la connexió USB entre ordinador i l'Arduino en si. Es tracta d'un software dedicat a l'usuari novell sense coneixements en programació de dispositius embedded, o amb poques nocions de programació general, pel que la seva interfície ha estat dissenyada de manera que sigui molt senzilla d'utilitzar.

L'Arduino IDE està basat en Java, AVR-GCC, Processing i altre software de codi obert. Té compatibilitat amb totes les plaques oficials d'Arduino distribuïdes actualment.

Es tracta sens dubte de la millor manera de començar en el món d'Arduino i de la programació de dispositius embedded, ja que afegeix una sèrie de capes d'abstracció que permeten realitzar tasques amb gran facilitat.

Per exemple, es pot verificar si el codi s'ha escrit correctament clicant el botó "Verifica" i pujar aquest treball al dispositiu únicament clicant el botó "Puja". Si el codi s'ha escrit correctament, no cal preocupar-se de res, ja que l'IDE s'encarrega de cridar el programari necessari per compilar-lo i pujar-lo. D'altra banda, l'Arduino IDE permet l'addició de llibreries (usades per poder fer anar els shields) de forma senzilla, les quals també són compilades juntament amb el codi escrit per l'usuari, i la lectura d'exemples i exercicis. Destaquem també el monitor serial, que permetrà mostrar els missatges impresos per pantalla procedents de la placa i a través del port USB (gràcies al xip convertidor Serial – USB integrat dins de la placa Arduino). Finalment, tenim manuals de referència i primers passos i enllaços a la web de la comunitat Arduino.

Tot això constitueix un sistema fàcil d'aprendre i manipular. Si l'usuari compta amb un mínim de nocions sobre programació en C o C++, en qüestió de minuts podrà tindre un petit codi funcional dins de la memòria del xip que realitzarà una sèrie de tasques, com per exemple encendre i apagar el LED integrat dins de la placa.

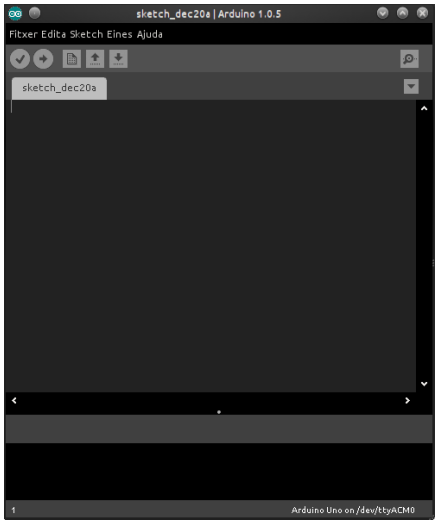


Figura 8. Pantalla de l'Arduino IDE amb un arxiu en blanc.

Malauradament, tanta simplicitat passa factura per a l'usuari més avançat. Una de les problemàtiques més importants relacionades amb l'ús de l'IDE és l'absència total d'advertències (o "warnings" en anglès) llançades pel compiladors. Malgrat que això per a l'usuari novell pot ser d'ajuda, ja que els errors produïts per compiladors són usualment críptics i difícils d'entendre a primera vista, pot ser quelcom molt perjudicial. Al cap i a la fi, s'estaran ignorant errors potencials en el codi, complicant severament la tasca de depurar el codi i trobar bugs. Això es tracta, exclusivament, d'una decisió presa per l'equip d'Arduino amb l'únic objectiu de no "espantar" els usuaris novells, ja que de vegades les advertències poden no afectar la funcionalitat del codi.

D'altra banda, l'Arduino IDE compta amb una limitació relacionada amb la declaració dels tipus de dades. Si creem un arxiu de capçalera (popularment anomenats "header files"), el compilador ens retornarà un error si intentem fer ús de tipus de dades com byte o uint8_t. Malgrat que això pot no ser de rellevància, pot limitar la compatibilitat amb algunes llibreries, com per exemple LibFixMath, la qual fa ús de uint32_t per poder declarar la seva variable de coma fixa, fix16_t. En diversos fòrums d'Internet s'ha proposat solucionar aquest problema de disseny, però encara l'equip de desenvolupament de l'Arduino IDE no s'hi ha pronunciat.

Altres raons que em van obligar a deixar d'utilitzar aquest entorn de desenvolupament van ser l'elevat ús de RAM, la lentitud de càrrega i execució (tenint amb compte que aquest Projecte s'ha desenvolupat íntegrament en un netbook amb processador Intel Atom de 1,66 GHz i 1 GB de RAM), la incomoditat d'enllaçar la combinació de tecles Shift+Esborrar a Supr i les poques possibilitats de configuració. Em vaig veure obligat a migrar de l'Arduino IDE completament. Així doncs, vaig optar per utilitzar un altre editor de text avançat anomenat Geany per a l'escriptura del codi.

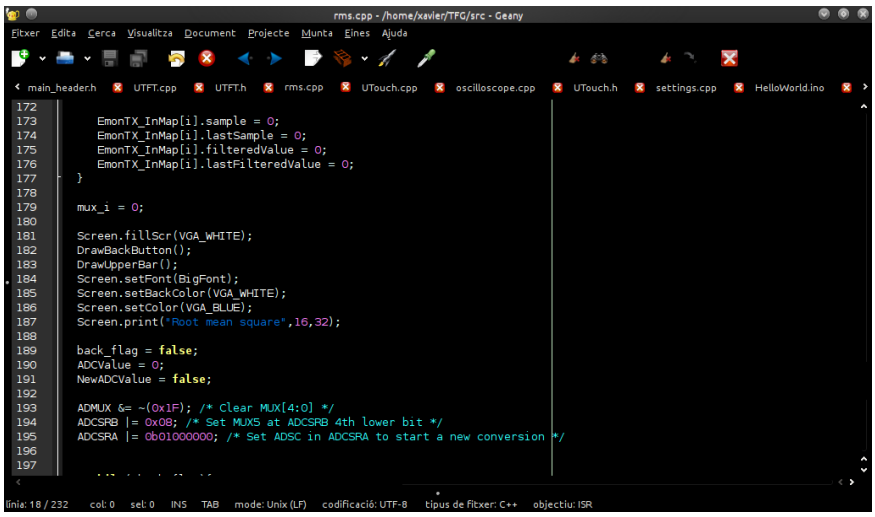


Figura 9. Pantalla del Geany IDE mostrant part d'un codi.

No obstant, malgrat que el Geany pot ésser configurat internament per compilar el codi, executar-lo i salvar aquesta configuració en un arxiu amb format propi, vaig limitar el seu ús únicament a editor de text. En combinació al Geany, així doncs, pot fer-se ús de qualsevol emulador de terminal (en el meu cas, Konsole per a Kubuntu).

D'altra banda, per a la compilació del codi i la pujada a la placa, s'hauran de definir una sèrie de Makefiles (arxius que automatitzen una sèrie d'ordres) que permetin compilar el codi i convertir-lo en un arxiu de tipus hexadecimal que finalment es pujarà al xip.

4.4 Estructura del codi font del Projecte

El codi font del Projecte s'ha estructurat de la forma típica en què ho fan els programes escrits per a sistemes de tipus UNIX, com Linux o Mac OS X. En primer lloc, tindrem els següents directoris:

- **$ROOT/include:** s'inclouen tots els arxius de capçalera (arxius en format .h o .hpp) utilitzats en el Projecte, així com també els procedents de les llibreries.
- **$ROOT/lib:** s'hi inclouen totes les llibreries estàtiques (arxius en format .a) de tercers, com les llibreries d'Arduino, LibFixMath o UTF.
- **$ROOT/src:** s'hi inclou el codi font tant del propi Projecte en si com de les llibreries de tercers. Cada llibreria ha estat separada de la resta en diferents subdirectoris per evitar conflictes. Dins de cada subdirectori (així com també al propi directori) hi ha un arxiu Makefile que construeix les diferents parts que composen el Projecte.

4.5 Llibreries i programari de tercers utilitzat

Aquest Projecte hagués estat impossible (o almenys hagués portat molt més temps) de no ser per la col·laboració desinteressada de voluntaris que contribueixen en la creació de llibreries que permeten estendre les funcionalitats d'Arduino i els seus shields. En el nostre cas, l'ús de llibreries ens ha facilitat molt tasques com interactuar amb la pantalla tàctil o la targeta SD.

4.5.1 LibArduino

A més de proporcionar un hardware base i un entorn de desenvolupament, l'equip d'Arduino es va prendre la molèstia de crear un set de llibreries que permeten interactuar fàcilment amb el dispositiu. A més de proporcionar un conjunt de funcions i classes com Serial, analogRead(), digitalWrite(), pinMode()... les llibreries d'Arduino estableixen un conjunt important de definicions i macros utilitzats tant per l'usuari com per altres llibreries compatibles.

Les llibreries d'Arduino constitueixen la base de qualsevol sketch (nom que rep el codi creat utilitzant l'Arduino IDE), ja que defineixen la seva típica estructura:

```
void setup(){
  //Enter code
}
void loop(){
  //Enter code
}
```

Codi 1. Esquema bàsic d'un programa d'Arduino.

Que en realitat és una abstracció del següent codi:

```
int main(void)
{
  init();

#if defined(USBCON)
  USBDevice.attach();
#endif

  setup();

  for (;;) {
        loop();
        if (serialEventRun) serialEventRun();
  }

  return 0;
}
```

Codi 2. Esquema bàsic d'un programa d'Arduino sense abstraccions.

Les llibreries d'Arduino estan empaquetades en l'arxiu libarduino.a dins de la carpeta $ROOT/lib, i el codi font s'ha guardat a $ROOT/src/libarduino.

4.5.2 UTFT i UTouch

Aquestes llibreries, creades per Henning Karlsen, són força conegudes en el món d'Arduino perquè permeten interactuar amb una gran gamma de pantalles LCD de diferents fabricants. Essencialment, UTFT proporciona un conjunt de funcions i definicions per dibuixar gràfics simples com píxels, línies o rectangles, així com també arxius bitmap. D'altra banda, UTouch és la llibreria encarregada d'interactuar amb la pantalla tàctil. Igual que UTFT, la seva utilització pot estendre's a multitud de dispositius diferents i el seu ús és molt senzill.

Per poder utilitzar UTFT, s'ha de crear un objecte de la classe UTFT i declarar els pins de l'Arduino Mega 2560 on anirà connectada la pantalla. Del mateix mode, ha de

declarar-se una instància de la classe UTouch i definir els pins de la pantalla utilitzats per al control tàctil. El manual del nostre model de pantalla especifica que:

Arduino PIN	Description
D0	DB8
D1	DB9
D2	DB10
D3	DB11
D4	DB12
D5	DB13
D6	DB14
D7	DB15
D8	Touch_Dout
D9	Touch_IRQ
D10	SD_CS
D11	SD_MOSI
D12	SD_MISO
D13	SD_SCK
A0	Touch_Din

Taula 1. Taula de distribució de pins de la pantalla LCD.

No obstant, cal tindre amb compte que els noms dels pins estan referits a l'Arduino UNO. Per al nostre TFG, que fa servir l'Arduino Mega 2560:

```
//UTFT(byte model, int RS, int WR, int CS, int RST, int SER=0);
UTFT Screen(ITDB28,59,58,57,56);
//UTouch(byte tclk, byte tcs, byte tdin, byte dout, byte irq);
UTouch Touch(A1,A3,A0,8,9);
```

Codi 3. Instruccions d'inicialització de la LCD i la pantalla tàctil.

La llibreria UTFT ha estat lleugerament modificada per fer la funció UTFT::print() compatible amb la classe __FlashStringHelper. Aquesta classe permet que els strings d'un programa no s'emmagatzemin a la SRAM, que és d'espai molt limitat, sinó a la memòria flash. Es tracta d'una manera òptima d'estalviar espai a la SRAM, ja que recordem que només té 4 kB.

```
void UTFT::print(const __FlashStringHelper * ifsh, int x, int y, intdeg){
  intstl = 0, i;
  constchar PROGMEM *p= (constchar PROGMEM *)ifsh;
  unsignedchar c;

  do {
          c= pgm_read_byte(p+stl);
          if(c)
          stl++;
  } while(c != 0);
  if (orient==PORTRAIT)
  {
```

```
if (x==RIGHT)
        x=(disp_x_size+1)-(stl*cfont.x_size);
if (x==CENTER)
        x=((disp_x_size+1)-(stl*cfont.x_size))/2;
}
else
{
if (x==RIGHT)
        x=(disp_y_size+1)-(stl*cfont.x_size);
if (x==CENTER)
        x=((disp_y_size+1)-(stl*cfont.x_size))/2;
}
for (i=0; i<stl; i++)
        if (deg==0)
                printChar(pgm_read_byte(p++), x + (i*(cfont.x_size)), y);
        else
                rotateChar(pgm_read_byte(p++), x, y, i, deg); }
```

Codi 4. Implementació alternativa de UTFT::print per fer-la compatible amb la classe __FlashStringHelper.

D'aquesta manera, si volem que un string s'emmagatzemi a la memòria flash, només haurem d'escriure:

```
Screen.print(F("Helloworld!"),x,y);
```

Codi 5. Exemple d'impressió per pantalla d'un missatge "Hello World" amb la rutina UTFT::print().

El macro F( ) permet transformar, de forma molt senzilla, un string en un objecte de la classe __FlashStringHelper, amb la finalitat que aquesta cadena de caràcters sigui emmagatzemada a la memòria flash.

Les llibreries UTFT i UTouch estan empaquetades, respectivament, en els arxius libutft.a i libutouch.a dins de la carpeta $ROOT/lib, i el codi font s'ha guardat a $ROOT/src/libutft i $ROOT/src/libutouch.

4.5.3 SDFatLib

SDFatLib és una llibreria per a llegir i manipular informació amb una targeta SD que estiguin formatades en FAT16/32. Utilitza el conveni 8.3 per als noms dels arxius, que consisteix en un límit de 8 caràcters per al nom de l'arxiu i un límit de 3 caràcters per al nom de l'extensió (per exemple, ARXIUONE.TXT).

Per poder connectar la targeta SD a una Arduino, s'ha de connectar als pins de comunicació. En l'Arduino Mega 2560:

- MISO: a pin D50.
- SCK: a pin D52.
- SS (Select Slave): a pin D53.
- MOSI: a pin D51.

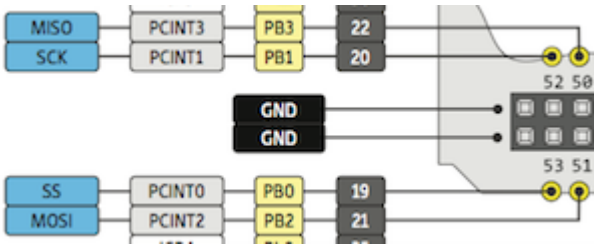


Figura 10. Detall dels pins que realitzen la comunicació per SPI.

Per poder inicialitzar la SD, en el codi del programa ha d'escriure's la següent línia:

```
sd.begin(53,SPI_FULL_SPEED);
```

Codi 6. Instrucció d'inicialització de la targeta SD.

On 53 fa referència al pin SS que hem esmentat anteriorment. El macro SPI_FULL_SPEED permet que la velocitat de transferència de dades sigui màxima, depenent de la qualitat de targeta SD instal·lada.

La manipulació d'arxius amb SdFatLib és molt semblant a l'estàndard de stdio: es declara una variable de tipus SdFile i s'obre l'arxiu amb la següent instrucció:

```
bool open(constchar* path, uint8_t oflag = O_READ);
```

Codi 7. Instrucció d'obertura de l'arxiu.

Aquesta funció retorna false en cas que hi hagi un error en la lectura de l'arxiu i true si s'ha aconseguit llegir correctament. La variable path és el directori en què es troba l'arxiu. D'altra banda, oflag especifica els flags que indicaran quines operacions es faran amb el fitxer; per defecte s'ajusta a O_READ / només lectura. Els flags més importants són:

```
// use the gnu style oflag in open()
/** open() oflag for reading */
uint8_t const O_READ = 0X01;
/** open() oflag for write */
uint8_t const O_WRITE = 0X02;
/** open() oflag - same as O_WRITE */
uint8_t const O_WRONLY = O_WRITE;
/** open() oflag for reading and writing */
uint8_t const O_RDWR = (O_READ | O_WRITE);
/** truncate thefile to zero length */
uint8_t const O_TRUNC = 0X10;
/** set the initial position at the end of the file */
uint8_t const O_AT_END = 0X20;
/** create the file if nonexistent */
uint8_t const O_CREAT = 0X40;
/** synchronous writes - call sync() after each write */
uint8_t const O_SYNC = 0X08;
```

Codi 8. Flags disponibles amb SDFatLib per a la manipulació d'arxius.

Per manipular informació, s'han de cridar les rutines read() i write(), que realitzen les funcions de lectura i escriptura, respectivament. En el cas d'enviar o rebre informació diferent de tipus uint8_t o char (variables de 8 bits), s'ha d'especificar quin tipus de variable s'està manipulant. Totes dues funcions retornen el nombre de bytes que s'han aconseguit llegir o escriure correctament.

```
int write(const void* buf, size_t nbyte);
int read(void* buf, size_t nbyte);
```

Codi 9. Instruccions d'escriptura i lectura a l'arxiu.

No obstant, fins ara la data de modificació no es veurà reflectida en les propietats de l'arxiu. Per aconseguir-ho, cal seguir el següent procediment:

1. Crear una funció amb la següent forma:

```
void dateTime(uint16_t* date, uint16_t* time) {
    uint16_t year;
   uint8_t month, day, hour, minute, second;
    // User gets date and time from GPS or real-time clock here        // return date using FAT_DATE macro to format fields
     *date = FAT_DATE(year, month, day);
      // return time using FAT_TIME macro to format fields
      *time = FAT_TIME(hour, minute, second);
    }
```

Codi 10. Codi que implementa la data i hora d'última modificació d'un arxiu.

En el nostre cas, les variables de data i hora són de tipus global i controlades per una funció externa anomenada Clock(), així que no és necessari declarar-les com a variables locals.

2. Obrir un arxiu per a escriptura amb els flags O_WR i O_SYNC.

3. Cridar la rutina dateTimeCallback() passant com a paràmetre un punter a la rutina que hem creat al pas 1. Per exemple:

```
file.dateTimeCallback(dateTime);
```

Codi 11. Instrucció necessària per cridar la rutina del pas anterior.

Finalment, per tancar l'arxiu s'ha d'utilitzar la següent funció:

```
bool close();
```

Codi 12. Instrucció de tancament de l'arxiu obert.

La llibreria SdFatLib està empaquetada en l'arxiu libsdfat.adins de la carpeta $ROOT/lib, i el codi font s'ha guardat a $ROOT/src/sdfatlib.

4.5.4 LibFixMath

Als éssers humans ens resulta molt còmode treballar amb nombres decimals. Quan programem, molts llenguatges de programació (com per exemple C, C++ o Java) tenen suport per a variables de coma flotant per tal d'operar amb nombres decimals de forma senzilla.

No obstant, molts microcontroladors no disposen d'unitat de càlcul de coma flotant (en anglès, Floating-point Unit o FPU). Això vol dir que els càlculs que involucrin nombres decimals hauran de ser processats per una llibreria externa, el qual significa que els temps de càlcul poden ser força lents.

Una manera alternativa de fer càlculs amb nombres decimals sense penalitzacions de temps és fer ús de nombre de coma fixa. "Coma fixa" (fixed-point en anglès) significa que la coma que separa la part entera de la decimal es troba sempre al mateix lloc. En altres paraules, es dediquen M bytes a la part entera i F bytes a la decimal. La notació estàndard que s'utilitza per denotar la distribució dels bytes és **Qm.f**.

Per poder convertir un nombre decimal en un nombre de coma fixa, es multiplica el nombre decimal per 2^f i ens quedem únicament amb la part entera d'aquest nombre. Es pot observar ràpidament que aquest truncament provocarà una pèrdua de precisió en els resultats que estarà directament relacionada amb el nombre de bytes que aniran destinats a la part decimal. També és obligatori que, en cas d'operacions amb altres nombres, siguin també de coma fixa i amb igual M i F.

Per implementar un suport bàsic de nombres de coma fixa en C/C++, existeix la llibreria LibFixMath. Crea el tipus de variable fix16_t, una variable de 32 bits de Q16.16, i un set de funcions i constants adaptades per ser utilitzades amb aquest tipus de dada.

La llibreria LibFixMath està empaquetada en l'arxiu `libfixmath.a`, dins de la carpeta `$ROOT/lib`, i el codi font s'ha guardat a `$ROOT/src/fixmath`.

4.5.5 AVR-GCC

GCC són les sigles de GNU C Compiler. Tal i com diu el seu nom, es tracta d'un compilador per al llenguatge C (malgrat que actualment té compatibilitat per a molts altres llenguatges), i és l'eina per antonomàsia per construir executables en sistemes operatius de tipus UNIX utilitzat conjuntament amb binutils (software que inclou diferents eines de manipulació d'executables). GCC no només permet compilar codi per al sistema operatiu que l'executa, sinó que pot configurar-se per tal que creï codi que sigui entès per altres arquitectures, com AVR, ARM, MIPS, etcètera.

Per a la família de dispositius AVR (i, per tant, els xips ATMega que composen les plaques Arduino), existeix la variant avr-gcc. Es distribueix dins del paquet avr-libc, que inclou, a més del susdit compilador, la versió per a AVR de binutils, del depurador gdb, de les llibreries estàndard del llenguatge C (stdio, malloc, string...), un programari per interactuar amb el programador (avrdude) i un simulador (simulavr).

4.5.5.1 Configuració del compilador

La configuració del compilador es duu a terme dins del Makefile situat a la carpeta arrel de src/. S'hi realitza la crida a avr-gcc amb els següents paràmetres:

```
build: main.cpp buttons.cpp rms.cpp settings.cpp oscilloscope.cpp credits.cpp
  $(CC) $^  -o $(PROJECT).elf $(INCLUDE) $(CFLAGS) $(LIBS)
```

Codi 13. Fragment del Makefile amb la llista d'arxius a compilar a dalt i la instrucció per a dur a terme la compilació a sota.

Explicarem a continuació el significat d'aquestes línies:

- build: és el nom de l'objectiu de la instrucció. Al costat es presenta la llista d'objectes;en el nostre cas, els arxius de codi en C++ a compilar.
- $(CC) és el nom assignat a avr-gcc dins del Makefile.
- -o $(PROJECT).elf és la manera de configurar el codi compilat de tal manera que se n'extregui un arxiu de format ELF (Executable and Linkable Format) anomenat analiser.elf segons la configuració del paràmetre PROJECT del Makefile.
- $(INCLUDE) esmenta el directori on es troben els arxius de capçalera.
- $(CFLAGS) inclou altres paràmetres d'interès per a la compilació.
- $(LIBS) esmenta les llibreries utilitzades pel projecte.

Aplicant-hi els paràmetres que hi ha dins del Makefile, l'anterior es resumeix en la següent instrucció:

```
avr-gcc main.cpp buttons.cpp rms.cpp settings.cpp oscilloscope.cpp credits.cpp  -o analiser.elf -I ../include/ -I ../include/sdfat -I../include/arduino -mmcu=atmega2560 -DF_CPU=16000000UL -DARDUINO=150 -Wall -Os --function-sections -fdata-sections -Wno-write-strings -Wl,-u,vfprintf -lprintf_flt -L ../lib -lfixmath -lutouch -lutft -lsdfat -larduino -lm
```

Codi 14. Resultat del codi anterior un cop aplicats tots els paràmetres.

Això permet compilar el codi font en un arxiu anomenat analiser.elf. Es tracta d'un tipus d'arxiu del qual podem extreure l'executable que anirà a la nostra placa. Una altra utilitat que té també és la de poder mesurar el pes que té el programa a la memòria. Si utilitzem la utilitat avr-size passant com a paràmetre l'arxiu .elf que obtenim de la compilació del programa, ens en retornarà el pes en bytes:

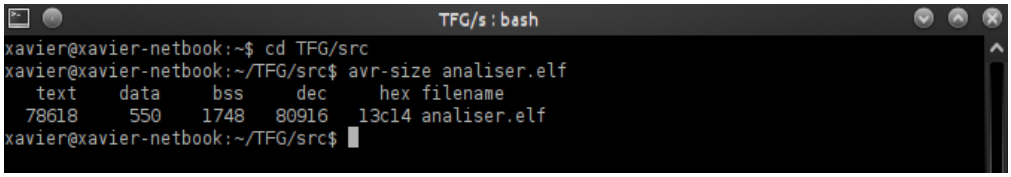


Figura 11. Pantalla de l'emulador de terminal amb el avr-size funcionant.

En aquest cas, el pes total en la memòria (sumant SRAM i flash) seria de 80.916 bytes. Tal i com es pot observar, aquest programa també desglossa el pes en bytes per a cada secció de la memòria. Les seccions que anirien a la memòria flash serien `text` i `data`, mentre que data i bss anirien a la SRAM.

4.6 Mesura de valors. De l'EmonLib a llibreries pròpies

La decisió més senzilla per obtenir els valors d'intensitat i tensió en l'Arduino UNO era fer ús de les llibreries proporcionades per l'equip d'OpenEnergyMonitor, anomenades EmonLib. Malgrat que la seva utilització és molt senzilla, la implementació no era del tot útil: a l'hora de llegir un valor procedent de l'ADC, el programa s'aturava fins que no finalitzés la conversió.

EmonLib fa ús de la rutina analogRead(), inclosa dins del set de funcions de les llibreries d'Arduino. Permet a l'usuari obtindre de forma puntual un valor de l'ADC, passant el pin a llegir com a paràmetre.

Aquest és el codi font de la funció analogRead(), l'utilitzada per EmonLib:

```
int analogRead(uint8_t pin)
{
	uint8_t low, high;

if (pin >= 14) pin -= 14; // allow for channel or pin numbers

  // set the analog reference (high two bits of ADMUX) and select the
	// channel (low 4 bits).  This also sets ADLAR (left-adjust result)
	// to 0 (thedefault).
ADMUX = (analog_reference<< 6) | (pin & 0x07);

	// start the conversion
sbi(ADCSRA, ADSC);

	// ADSC is cleared when the conversion finishes
while (bit_is_set(ADCSRA, ADSC));

// we have to read ADCL first; doing so locks both ADCL
	// and ADCH until ADCH is read.  Reading ADCL second would
	// cause the results of each conversion to be discarded,
	// as ADCL and ADCH would be locked when it completed.
low  = ADCL;
```

```
high = ADCH;

 // combine the two bytes
return (high<< 8) | low;
}
```

Codi 15. Implementació de la rutina analogRead().

El procediment és simple: s'ajusta ADMUX en funció al pin que l'usuari ha passat com a paràmetre, s'ajusta a 1 el bit ADSC i s'espera que finalitzi la conversió, moment en què el bit ADSC baixarà a zero de nou. Un cop finalitzada, s'emmagatzema el valor obtingut a partir dels registres ADCL i ADCH en les variables low i high. La combinació d'ambdues variables serà el valor a retornar a l'usuari.

La funció analogRead() és prou pràctica si volem obtenir valors digitals d'un sensor, potenciòmetre, font de tensió... de forma senzilla i puntual. L'inconvenient ve donat per aquesta línia:

```
      // ADSC is cleared when the conversion finishes
while(bit_is_set(ADCSRA, ADSC));
```

Codi 16. Detall de la rutina analogRead() mostrant les instruccions problemàtiques.

Aquesta instrucció paralitza el programa mentre el bit ADSC del registre ADSCRA es troba a 1. Aquest temps depèn de diversos factors com la configuració del prescaler, però en la configuració més típica (prescaler a 128, el que equival a f_{ADC} = 125 kHz) suposa un temps de 100 μs, aproximadament.

D'altra banda, la implementació del càlcul de la RMS d'EmonLib és la següent:

```
double EnergyMonitor::calcIrms(int NUMBER_OF_SAMPLES)
{

  #ifdefined emonTxV3
  int SUPPLYVOLTAGE = 3300;
  #else
  int SUPPLYVOLTAGE = readVcc();
  #endif


  for (int n = 0; n < NUMBER_OF_SAMPLES; n++)
  {
lastSampleI = sampleI;
sampleI = analogRead(inPinI);
lastFilteredI = filteredI;
filteredI = 0.996*(lastFilteredI+sampleI-lastSampleI);

   // Root-mean-squaremethodcurrent
   // 1) squarecurrentvalues
sqI = filteredI * filteredI;
```

```
   // 2) sum
sumI += sqI;
 }

double I_RATIO = ICAL *((SUPPLYVOLTAGE/1000.0) / (ADC_COUNTS));
Irms = I_RATIO * sqrt(sumI / NUMBER_OF_SAMPLES);

 //Resetaccumulators
sumI = 0;
//--------------------------------------------------------------------------------

returnIrms;
}
```

Codi 17. Implementació de la rutina EnergyMonitor::calcIrms().

L'algorisme pot dividir-se en les següents parts:

- Definició de la tensió d'alimentació

Aquest primer pas té un motiu pràctic: no podem assegurar mai que tinguem el valor ideal de VCC a la nostra placa. És a dir, molts cops no tindrem 5 V exactes, sinó que poden fluctuar lleugerament. Això afectaria els resultats de les conversions, ja que és un dels factors de l'equació que converteix un valor de l'ADC a un valor real.

(1) $$Real_{value} = K_{cal} \cdot \frac{V_{CC}}{1023} \cdot ADC_{Value}:$$

Una manera de llegir indirectament la tensió d'alimentació és comparant el senyal de 1,1 V que incorpora el microcontrolador amb el valor obtingut d'una conversió amb l'ADC. En primer lloc, ajustem ADMUX de la següent manera:

```
// Read 1.1V reference against AVcc
ADMUX = _BV(REFS0) | _BV(MUX4) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
```

Codi 18. Instrucció per ajustar ADMUX a V_{BG}.

Table 26-4. Input Channel Selections (Continued)

MUX5:0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
011110	1.1V (V_{BG})	N/A		

Taula 2. Extracte del datasheet de l'ATmega 2560 descrivint els bits necessaris per ajustar ADMUX a V_{BG}.

És a dir, ajustem VCC com a tensió de referència (en el cas ideal de l'ATMega 2560 serien 5 V) i els bits MUX4 a MUX1 a 1. Això ajusta el pin d'entrada a VBG, que idealment es troba a 1,1 V.

Aleshores iniciem la conversió ajustant ADSC a 1. El resultat, guardat a ADCL i ADCH, s'emmagatzema a la variable result.

```
ADCSRA |= _BV(ADSC); // Convert
while (bit_is_set(ADCSRA,ADSC));
```

```
result = ADCL;
result |= ADCH<<8;
result = 1125300L / result; // Back-calculateAVcc in mV
```

Codi 19. Obtenció del valor real de V_{CC}.

I vet aquí la gràcia d'aquest mètode: en un cas ideal de 5000 mV, l'ADC retornaria un valor de 225. Donat que 5000*225 = 1125300, es fa una divisió per determinar el valor exacte de VCC. Per exemple, si l'ADC retornés un valor de 250, això significa que la senyal de 1,1 V realment és de 1,22 V; de manera que l'alimentació seria de 4501 mV.

Si l'expressió per calcular el valor eficaç d'un senyal de qualsevol tipus és:

$$\textbf{(2)} \qquad RMS = K_{cal} \cdot \frac{V_{CC}}{1023} \cdot \sqrt{\frac{\sum_{i=0}^{n} {ADC_i}^2}{n}}$$

Observem aleshores la importància de calcular VCC amb anterioritat.

No es tracta d'un inconvenient si el temps que això suposa no ens resulta d'importància. Però en un sistema en què fem actuar un conjunt d'elements alhora, una forma d'optimitzar els temps de processament i càlcul és fer treballar aquests elements en paral·lel tot el que sigui possible.

Una solució possible és aprofitar aquest temps que el microcontrolador perdria en un bucle. Per aconseguir-ho, hem de fer que l'ADC faci la conversió de forma paral·lela a l'execució del programa. L'ATMega 2560 permet això ajustant una sèrie de bits, de manera que les conversions es realitzaran de forma paral·lela a la resta del programa, i un cop la conversió hagi finalitzat el programa entrarà dins de la rutina de servei a la interrupció (en anglès, interrupt service routine o ISR).

Donat que el nostre prototip involucrava dos shields, un que prenia mesures de tensió/corrent i un altre que mostrava dades en temps real i les guardava en una targeta SD, era necessari poder processar les conversions analògiques-digitals amb altres tasques en paral·lel. No obstant, el codi original d'OpenEnergyMonitor i el seu principi de funcionament eren igualment molt útils, de manera que han estat aprofitats i adaptats a les nostres necessitats.

En el nostre cas, desitgem obtindre les mesures d'un canal de tensió (connectat a l'entrada ADC 8 de l'Arduino MEGA 2560) i quatre canals d'intensitat, connectats des de l'entrada ADC 9 fins a ADC12. Segons el datasheet de l'ATMega 2560, pot realitzar-se la selecció del canal manipulant el registre ADMUX a partir de la modificació dels bits MUX[4:0] i modificant el bit MUX5, que es troba dins del registre ADCSRB.

26.8.1 ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Taula 3. Detall del registre ADMUX segons el datasheet de l'ATmega 2560.

Explicació del registre:

- REFS[1:0]: bits de selecció de tensió de referència de l'ADC.

Table 26-3. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection[1]
0	0	AREF, Internal V_{REF} turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Internal 1.1V Voltage Reference with external capacitor at AREF pin
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Taula 4. Valors possibles de REFS[1:0] i les seves funcionalitats.

- ADLAR: determina la manera en què els bits s'ordenen als registres de resultat ADCL i ADCH.
- MUX[4:0]: bits de selecció de canal d'ADC. S'ha de tindre amb compte que també cal ajustar el bit MUX5, que es troba al registre ADCSRB.

25.2.1 ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	ACME	–	–	MUX5	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Taula 5. Detall del registre ADCSRB.

- MUX5: bit de major pes del grup MUX[5:0]. Necessari per ajustar el canal d'entrada de l'ADC.
- ACME: relacionat amb el comparador analògic. No és d'interès per a aquest Projecte.
- ADTS[2:0]: selecciona l'activador de la interrupció de l'ADC. Pot ajustar-se en mode freerunning (execució seguida) com en el nostre Projecte, o bé ajustar-lo en funció d'un temporitzador o del comparador analògic.

Table 26-6. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match A
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

Taula 6. Valors possibles dels bits ADTS[2:0] i les seves funcionalitats.

Donat que l'ATMega 2560 compta únicament amb un únic ADC, només podrem obtenir valors de altres pins canviant el valor dels bits de ADMUX. Això suposa una problemàtica perquè hi haurà un desfasament temporal entre les diferents mesures. Per tant, serà important tindre un seguiment del temps aproximat que triga el programa a manipular la informació.

A sota tenim descrits els bits que hem d'ajustar per llegir des dels pins que ens interessen:

100000	ADC8	N/A
100001	ADC9	
100010	ADC10	
100011	ADC11	
100100	ADC12	

Taula 7. Relació entre els bits MUX[5:0] i les entrades de l'ADC.

L'ATMega 2560 ofereix la possibilitat d'ajustar el guany de les mesures obtingudes, però no s'ha utilitzat en aquest Projecte. No obstant, en un futur podria ser interessant estudiar si aquesta opció podria incrementar la resolució de l'aparell en valors baixos de tensió o corrent.

El control d'inici de les conversions pot determinar-se de dues maneres: manualment o de forma automàtica, ja sigui en free runing o en funció d'un temporitzador. Un control manual de les conversions pot tenir sentit en el cas que vulguem realitzar-les donat un succés determinat o un temps mesurat amb un temporitzador.

Per iniciar la conversió A/D de forma manual, ha d'activar-se a 1 el bit ADSC, el qual tornarà a 0 automàticament un cop hagi finalitzat aquesta. A l'extracte de codi anterior s'observa com es realitza susdita operació:

```
// start the conversion
sbi(ADCSRA, ADSC);
 // ADSC is cleared when the conversion finishes
while (bit_is_set(ADCSRA, ADSC));
```

Codi 20. Instruccions que inicien una conversió A/D.

Tanmateix, donat que el nostre aparell realitzarà constantment mesures a la major velocitat possible, ajustar l'ADC al mode freerunning sembla la millor opció.

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Taula 8. Detall del registre ADCSRA segons el datasheet de l'ATmega 2560.

Explicació dels bits del registre ADCSRA:

- ADEN: habilita el convertidor analògic-digital. És activat automàticament en cridar la rutina init( ) de la llibreria Arduino, la qual ha d'ésser inclosa a setup( ). La rutina init( ) també inicialitza els timers i la comunicació sèrie (no necessària en aquest Projecte).
- ADSC: inicia una nova conversió.
- ADATE: habilita l'activació automàtica de les conversions. La font d'activació haurà d'ajustar-se amb els bits ADTS[2:0] del registre ADCSRB.
- ADIF: aquest bit puja quan una conversió s'ha finalitzat.
- ADIE: habilita la rutina de servei a la interrupció ISR(ADC_vect).
- ADPS[2:0]: ajusta la freqüència de rellotge de l'ADC per mitjà d'un factor divisor. A la taula de sota podem veure els valors possibles del prescaler.

Table 26-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Taula 9. Relació entre els bits ADPS[2:0] i el factor divisor del prescaler.

El factor divisor permet ajustar la freqüència de l'ADC agafant la del microcontrolador com a referència. La precisió de l'ADC està directament relacionada amb aquest factor, de manera que quant més gran és (o menor sigui la freqüència de rellotge de l'ADC), major serà la seva precisió. Inversament, a mesura que s'incrementa aquesta freqüència (és a dir, que es redueix el divisor), menor és la precisió.

- Per obtenir una millor precisió, cal una freqüència entre 50 kHz i 200 kHz.
- **La conversió requereix de 13 cicles de rellotge, amb la freqüència ajustada amb ADPS.**

El càlcul de la freqüència de rellotge de l'ADC es realitza de la següent manera:

$$f_{ADC} = \frac{f_{CPU}}{D} \quad \textbf{(3)}$$

On f_{CPU} és la freqüència de rellotge de l'Arduino Mega 2560 (16 Mhz) i D és el factor divisor. Per exemple:

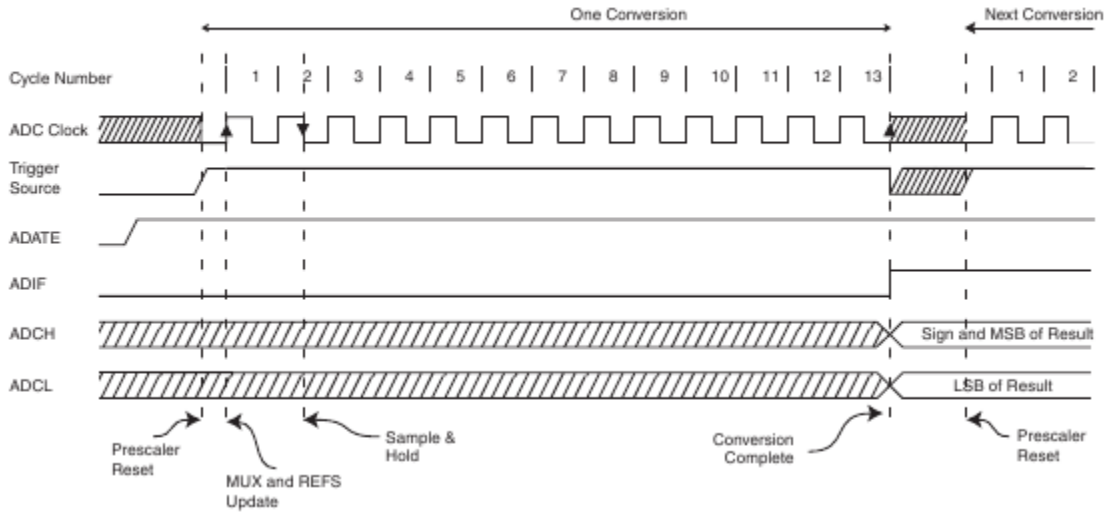
$$f_{ADC} = \frac{16\ MHz}{128} = 125kHz \quad \textbf{(4)}$$

Aquí hi ha la taula amb els valors de D possibles:

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

A sota podem veure el diagrama temporal del funcionament de l'ADC funcionant en mode free running.

Figure 26-6. ADC Timing Diagram, Auto Triggered Conversion



Quan el bit ADATE (ADC Auto Trigger Enable) es posa a 1, la font d'activació també es posa a 1,suposant que anteriorment hem ajustat ADTS[2:0] a freerunning mode. Aleshores, comença una conversió que dura 12 cicles de rellotge a freqüència f_{ADC}. Quan transcorre aquest temps, el resultat de la conversió s'emmagatzema als registres ADCH i ADCL i s'inicia automàticament una nova conversió.

Les pinces amperimètriques es basen en un transformador de corrent, el qual té una relació de transformació predeterminada tal que :

$$\textbf{(5)} \qquad N_{CT} = \frac{I_I}{I_{II}}$$

No importa si es tracten de valors RMS (els que s'especifiquen a la pròpia pinça) o valors de pic, ja que la constant sqrt(2) igualment desapareix.

En el nostre cas, les pinces YDHC SCT013 tenen una relació de 100 A : 50 mA, el qual vol dir que:

$$\textbf{(6)} \qquad N_{CT} = \frac{100\ A}{0{,}050\ A} = 2000$$

Aquesta intensitat, per tal que pugui ser llegida per un convertidor analògic-digital, haurà de ser convertida a una tensió. OpenEnergyMonitor implementa aleshores una resistència anomenada *burden* resistor (que a la placa s'identifiquen com a R16, R15, R17 i R28). Com que l'ADC pot mesurar valors compresos entre 0 V i Vcc (5 V, alimentació d'Arduino), els quals corresponen als valors 0 i 1024, respectivament, hem d'establir un valor mig de tensió per a quan el corrent sigui nul, de manera que:

- $0\ A \implies 512 (\frac{5}{2} = 2{,}5\ V)$0 A -> 512 (5/2 = 2,5 V)
- $100 \cdot \sqrt{2}\ A \implies 1023\ (5\ V)$
- $-100 \cdot \sqrt{2}\ A \implies 0\ (0\ V)$

Per crear aquest desplaçament (DC Offset en anglès), s'implementa un divisor de tensió amb dues resistències de valor idèntic.

D'aquesta manera, per a 2,5 V entre els terminals del *burden resistor* i una intensitat màxima donada per les característiques de la pinça, obtindrem que:

$$\textbf{(7)} \qquad R_b = \frac{{}^{V_{cc}}/_{2}}{I_{II,màx}} = \frac{2{,}5\ V}{50\sqrt{2}\ mA} = 35{,}4\Omega$$

Malauradament, el valor comercial més proper és 33 Ω, de manera que si el corrent és fix, la caiguda de tensió serà menor, assolint 2,333 V.

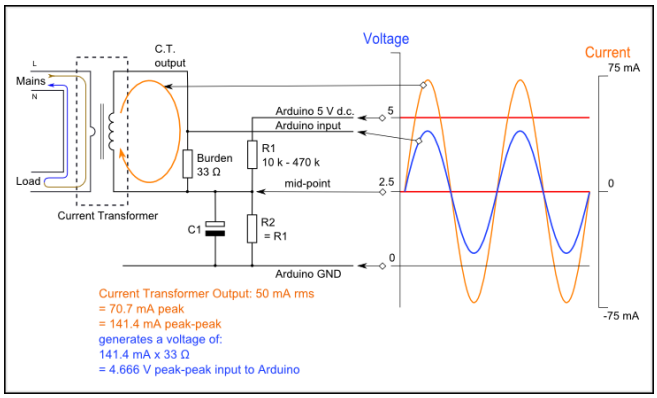


Figura 12. Circuit de lectura de corrent.

4.7 Conversió valors tensió a bits

L'Arduino Mega 2560 disposa d'un ADC de 10 bits, el qual proporciona valors possibles de tensió compresos entre 0 i 5 V. Segons el que hem llegit a l'apartat anterior, s'observa ràpidament que, si no es pot assolir el valor màxim de la tensió, tampoc no podrem assolir els 0 V; sinó que oscil·larem entre 4,8333 V i 0,1666 V.

Per poder convertir un valor en binari obtingut de l'ADC a un corrent determinat, hem de realitzar una sèrie de càlculs matemàtics senzills. En primer lloc, sabem que la relació entre tensió i bits és de:

$$\textbf{(8)} \qquad \frac{5\ V}{1024\ div} = 0{,}00488 \frac{V}{div}$$

Això significa que saltarem d'un valor del rang de valors possibles de l'ADC a l'altre quan existeixi una diferència d'aproximadament 5 mV. Suposem que hem llegint directament de l'ADC hem obtingut el valor 550; això suposa que, restant l'offset (512), tenim un valor de 38 unitats. En a<ltres paraules, per saber la tensió que arriba a l'ADC (restant els 2,5 V d'offset) farem ús de la següent relació:

$$\textbf{(9)} \qquad \frac{38 \cdot 5}{1024} = 0{,}1855\ V$$

Aquesta tensió és en realitat la que hi ha entre els borns del *burden resistor*, així que, coneixent el seu valor, podem determinar la intensitat que està circulant per la malla:

$$\textbf{(10)} \qquad I_{II} = \frac{0{,}1856\ V}{33\ \Omega} = 0{,}00562\ A$$

Finalment, coneixent la relació de transformació, podem establir el valor instantani de corrent que circula pel primari:

$$\textbf{(11)} \qquad I_I = I_{II} \cdot N_{CT} = 0{,}0056 \cdot 2000 = 11{,}24\ A$$

En la pràctica, existeix una certa tolerància en els valors reals dels components. El valor del *burden resistor* té una tolerància d'un 1%; d'altra banda, les pinces amperimètriques tenen una tolerància d'un 1%, a causa de pèrdues en el circuit magnètic, sumant l'error en la tensió de referència, que pot ser de fins a un 9%[1]. Així doncs, podem agrupar la relació entre Rb i la relació de transformació de les pinces com a una constant, anomenada constant de calibratge. En un cas ideal que en un cas ideal seria de:

$$\textbf{(12)} \qquad \frac{2000\,\frac{voltes}{voltes}}{33\Omega} = 60{,}6\Omega^{-1}$$

Així doncs, la fórmula final per calcular el valor eficaç del corrent a partir d'una sèrie de valors de l'ADC obtinguts seria :

$$\textbf{(13)} \qquad I_{I.rms} = \frac{N_{CT}}{R_b}\frac{V_{cc}}{2^{10}-1}\sqrt{ADC_{rms}^2}$$

4.8 High-pass filter

El hardware que incorpora l'EmonTX fa que a cada pin d'entrada analògica tinguem un component continu d'aproximadament 2,5 V per tal que puguem obtenir tant valors positius com negatius dels senyals sinusoïdals de la xarxa. És a dir, com que el valor màxim permès és 5 V i el valor mínim és 0 V, en un cas ideal (ignorant el fenomen de l'apartat anterior degut a utilitzar valors comercials de resistències) els valors positius de l'ona se situaran entre $V_{CC}/2$ i V_{CC}, mentre que els valors negatius se situaran entre $V_{CC}/2$ i 0 V.

És necessari eliminar aquest component continu de $V_{CC}/2$. Per dur-ho a terme, aplicarem un filtre passa-alts que eliminarà qualsevol senyal de molt baixa freqüència. Mitjançant un simple algorisme, podem simular un filtre passa-alts de primer ordre com el del circuit de sota:

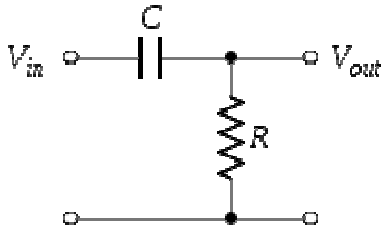


Figura 13. Circuit filtre passa-alts.

L'algorisme consisteix en la següent expressió:

$$\textbf{(14)} \qquad Y[n] = \alpha[Y[n-1] - (X[n] - X[n-1])]$$

On Y[n] és el valor filtrat, X[n] el valor actual obtingut de l'ADC sense filtrar i Y[n-1] i X[n-1] els valors de l'anterior iteració. Observem per tant que és una expressió que ha

[1] http://openenergymonitor.org/emon/buildingblocks/ct-and-ac-power-adaptor-installation-and-calibration-theory [Consulta] 22 abril 2015.

de calcular-se de forma iterativa, ja que quan més iteracions es realitzen més filtrat queda el senyal de molt baixa freqüència.

La constant α es troba íntimament relacionada amb la constant de temps RC del circuit de primer ordre descrit anteriorment. Des d'OpenEnergyMonitor[2] recomanen un valor de 0,996 per tindre una constant de temps suficientment llarga per reduir la distorsió de l'amplitud a la freqüència fonamental del senyal (50/60 Hz).

4.9 Estructures i rutines principals

4.9.1 InputStruct

EmonLib crea una classe anomenada EnergyMonitor on incorpora les rutines i variables associades als canals d'entrada de tensió i corrent de l'EmonTX. Allunyant-se d'una programació orientada a objectes típica del C++ per utilitzar un estil més propi del C, el programa utilitza una estructura anomenada InputStruct que inclou les següents variables:

```
typedef struct{
    float K; //Calibration constant
    float filteredValue,lastFilteredValue;
    float sample, lastSample;
    uint8_t pin; //Pin reference (look up for pin values)
    float Rms, oldRms; //Last known RMS value
    char updated; //Tells whether it has finished reading values
    unsigned int n; //Number of samples obtained
    double sum;
    unsigned long timer; //Used to calculate reading times
    double res_t; //Used to calculate average reading time
    float avg; //Used in long-term storage mode
    uint64_t c; //Used in long-term storage mode
  }InputStruct;
```

Codi 21. Implementació de l'estructura InputStruct.

De manera que les entrades de tensió i corrent queden definides amb un array anomenat EmonTX_InMap de dimensions EMONTX_INPUTS, on EMONTX_INPUTS (igual a 5) és el nombre d'entrades de què diposa el shield.

4.9.2 Button

El nostre prototip incorpora diversos menús amb botons que realitzen diverses funcionalitats. Una manera senzilla d'implementar-los és creant variables de tipus Button

2 http://openenergymonitor.org/emon/buildingblocks/digital-filters-for-offset-removal. [Consulta] 10 febrer 2015.

que tinguin dades sobre posició i mida, que a més poden completar-se amb una sèrie de funcions que requereixen una variable d'aquest tipus com a paràmetre.

```
typedef struct{

  short x,y;
  short w,h;
}Button;
```

Codi 22. Implementació de l'estructura Button.

Les rutines que poden cridar-se passant una variable de tipus Button com a paràmetre permeten realitzar accions com comprovar si un botó ha estat pitjat, dibuixar el botó blau típic de la interfície del prototip o dibuixar text sobre un botó:

```
voidDrawTealRoundButton(Button * b);
boolButtonPressed(Button * b);
voidButtonChar(const __FlashStringHelper * ifsh, Button * b);
Y[n]=α[Y[n-1]-(X[n]-X[n-1])]
```

4.9.3 Lectura valors RMS

Rms.cpp és l'arxiu més gran del Projecte, on s'engloben diverses de les opcions del menú principal. La seva estructura bàsica i funcionament vénen donats en funció dels flags que s'envien com a paràmetres des del menú principal. No obstant, la ISR és comuna per a tots els modes. Els valors que poden enviar-se són:

- sd_flag: permet guardar dades a la targeta externa SD. No defineix el comportament del programa, ja que dependrà de l'estat de la resta de variables. **Si cap més flag es troba activat, envia valors RMS a la SD.**
- lt_flag: mode llarg termini o long-term storage. Processa valors de corrent i tensió en temps real, però només salva a la targeta SD la mitjana aritmètica dels valors RMS transcorreguts durant X minuts, on X és el valor ajustat per l'usuari (per defecte, 10 minuts). **Incompatible amb cal_flag i inst_flag, necessita sd_flag per funcionar.**
- cal_flag: entra en el mode de calibratge. **Incompatible amb els flags sd_flag, lt_flag i inst_flag.**
- inst_flag: envia dades sobre valors instantanis de tensió i corrent en comptes de valors RMS. D'utilitat per mesurar puntes de corrent, sobretensions puntuals... **Incompatible amb cal_flag i lt_flag, necessita sd_flag per funcionar.**

Aquest arxiu es troba dividit en tres grans rutines que actuen de diferents formes en funció dels paràmetres anteriors. Són:

- ISR(ADC_vect): rutina de servei a la interrupció. El programa hi entra quan una conversió analògic-digital ha finalitzat. Emmagatzema el valor de la conversió a una variable anomenada ADCValue, aplica el filtre passa-alts i

afegeix el quadrat d'aquest valor a la variable sum si inst_flag no està activada. En cas contrari, iguala sum al valor obtingut del passa-alts.

- RMS(): rutina principal de l'arxiu. Ajusta els bits dels registres per a configurar l'ADC, imprimeix els strings corresponents en funció dels paràmtres enviats per l'usuari, defineix les variables principals i constitueix el flux del programa.
- CalculateRms(): encarregada de realitzar el càlcul del valor eficaç i les impressions a pantalla/SD en funció dels paràmetres enviats.

4.9.4 Oscil·loscopi

S'ha implementat la visualització de les ones de tensió i corrent que rebem de l'ADC, ja que pot esdevenir d'interès en molts àmbits.

Una de les problemàtiques principals de qualsevol microcontrolador és la quantitat limitada de memòria disponible. En el cas de l'ATMega 2560, disposem d'una memòria flash de 256 kB i una memòria SRAM de 4 kB. La diferència entre ambdues memòries és quin tipus de dades s'emmagatzemen: la memòria flash és la que inclou les instruccions del programa, mentre que a la SRAM s'hi guarden les dades

Existeixen diferents seccions de la memòria anomenades d'una forma concreta[3]:

- `.text`: és on s'inclouen les instruccions que formen el programa en si mateix. Alhora, es troba subdividida per altres seccions anomenades `.initN` i `.finiN`.
- `.data`: és on hi ha les variables ja inicialitzades en el codi. Per exemple:

```
uint8_t numeros[] = {1, 2, 3, 4};
long velocitat = 320;
```

Codi 23. Exemple de dades emmagatzemades a la secció .data.

Són variables que pertanyerien a la secció `.data`.

- `.bss`: és la secció pertanyent a les variables no inicialitzades.
- .initN: executades del zero al nou, són subseccions de .text utilitzades per definir el codi d'engegada des del reset del dispositiu fins a l'inici de la funció `main()`. La seva finalitat és la de permetre una col·locació del codi més específica dins del nostre programa.
- .init0: enllaçada a __init(). Si l'usuari defineix __init(), serà la posició on saltarà tot just després de reiniciar el dispositiu.
- .init1: sense utilitzar. Pot ésser definida per l'usuari.
- .init2: dèbilment enllaçada per inicialitzar la pila i netejar el registre r1, també anomenat zero_reg.
- .init3: sense utilitzar. Pot ésser definida per l'usuari.

[3] http://www.nongnu.org/avr-libc/user-manual/mem_sections.html. [Consulta] 10 gener 2015.

    - .init4: per dispositius amb menys de 64 kB de ROM, aquesta subsecció defineix el codi que s'encarrega de copiar els continguts de .data de la memòria flash a la SRAM. Per a la resta de dispositius, aquest codi és carregat des de libgcc.a.
    - .init5: sense utilitzar. Pot ésser definida per l'usuari.
    - .init6: no utilitzada en C, però utilitzada en C++ per als constructors.
    - .init7: sense utilitzar. Pot ésser definida per l'usuari.
    - .init8: sense utilitzar. Pot ésser definida per l'usuari.
    - .init9: salta a la rutina main().
- .finiN: aquestes subseccions de .text defineixen el codi de sortida un cop s'ha sortit de la rutina principal main() o es fa una crida a exit(). Anàlogament al cas anterior, també existeixen 9 subseccions ordenades de forma descendent.
    - .fini9: sense utilitzar. Pot ésser definida per l'usuari. És on comença _exit().
    - .fini8: sense utilitzar. Pot ésser definida per l'usuari.
    - .fini7: sense utilitzar. Pot ésser definida per l'usuari.
    - .fini6: sense utilitzar en C, però utilitzada per als destructors en C++.
    - .fini5: sense utilitzar. Pot ésser definida per l'usuari.
    - .fini4: sense utilitzar. Pot ésser definida per l'usuari.
    - .fini3: sense utilitzar. Pot ésser definida per l'usuari.
    - .fini2: sense utilitzar. Pot ésser definida per l'usuari.
    - .fini1: sense utilitzar. Pot ésser definida per l'usuari.
    - .fini0: entra en un bucle infinit un cop el programa ha finalitzat i s'ha completat qualsevol codi _exit().

La manera més senzilla és rebre els valors de l'ADC directament un per un i dibuixar un punt a la pantalla en unes coordenades determinades. Dibuixarem en primer lloc un eix de coordenades:

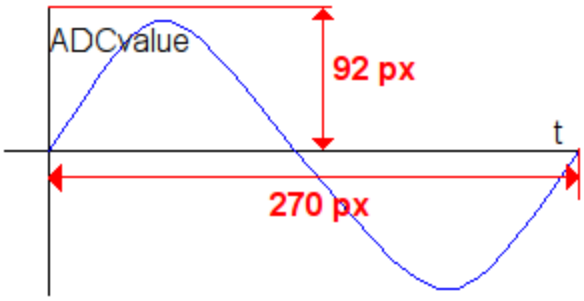


Figura 14. Esquema de representació de les gràfiques a pantalla.

Un petit procediment en pseudocodi seria quelcom així:

```
repetir indefinidament
Numero_samples = 280
desde i=0 a i = Numero_samples
x=i
y=llegirValor(canalADC)
```

```
dibuixarpunt(x,y)
fi bucle
netejarPantalla()
fi algorisme
```

Codi 24. Pseudocodi d'una primera implementació de representació de les gràfiques.

Es pot observar que es tracta d'una implementació molt senzilla que no requereix d'un buffer. No obstant, encara han de tenir-se diferents aspectes:

- Neteja de la pantalla: netejar la pantalla esborrant-ho tot és un procediment molt lent. Això es degut aa la implementació de la llibreria UTFT. A més, l'efecte visual de pampallugues que produeix és desagradable.
- Ara mateix l'òfset Y que obtenim de l'ADC té uns valors massa grans per a la nostra pantalla. Tinguem amb compte que els valors de l'ADC poden moure's entre 0 i 1023 (corresponent a les 2^{10} combinacions possibles), així que podem dir que el valor mig se situa entorn a 511 i 512. Així doncs, en una primera aproximació podem aplicar un òfsetpredefinit mitjançant una relació lineal com la següent:

$$\textbf{(15)} \quad y_o = 92 \cdot 2 = 184$$

$$\textbf{(16)} \quad y = y_o - \frac{184}{1023} \cdot ADC_{Value}$$

Això ens permetrà encaixar els valors que obtinguem de l'ADC dins dels marges de l'oscil·loscopi.

Existeix un altre inconvenient que fa inviable aquest mètode: amb aquest algorisme estem obligant que l'ADC no pot obtindre un nou valor fins que la pantalla no hagi finalitzat de dibuixar el punt. Donat que UTFT triga un temps relativament alt a dibuixar un píxel a la pantalla, a l'hora d'obtindre un nou punt és possible que la diferència de fase entre ambdós punts sigui elevada; de manera que l'ona resultant sortirà prou deformada i lluny de l'aspecte típic d'una ona sinusoïdal. Seria un efecte similar al mostrat a la imatge de sota, on l'ona vermella seria l'ona real procedent de la xarxa i l'ona blava és el resultat de la unió entre els diferents punts on s'ha pres el valor de la tensió a cada instant.

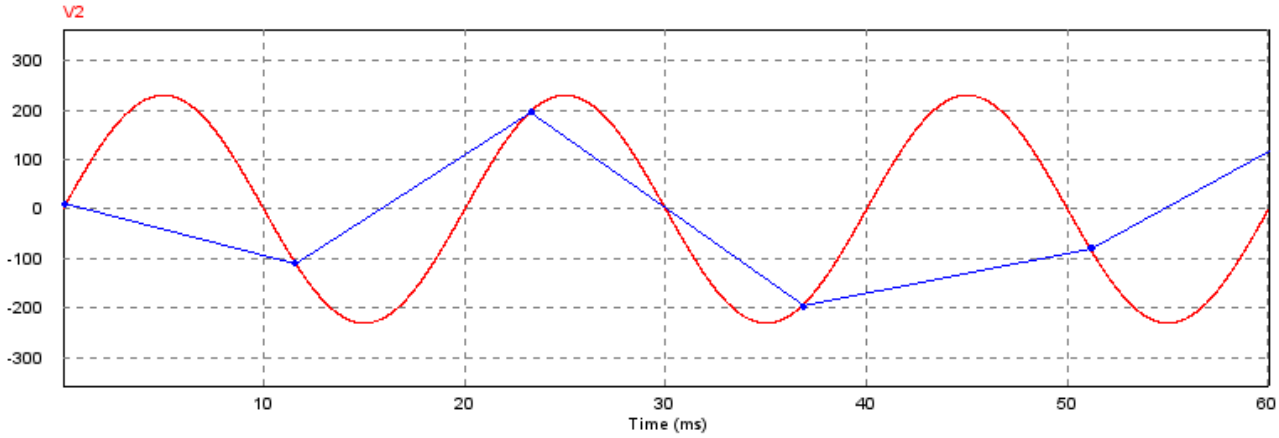


Figura 15. Efecte provocat pel submostreig (blau) d'una ona sinusoïdal (vermell).

Aleshores, l'estratègia havia de ser la següent:

1. En primer lloc, s'emmagatzemaran tots els valors obtinguts de l'ADC de forma consecutiva.
2. En segon lloc, s'imprimirà a pantalla tots els valors anteriors.
3. Repetir algorisme.

Ja pot deduir-se ràpidament que caldrà un buffer on salvar totes aquestes dades. Per motius de resolució de la pantalla que utilitzem, podem dibuixar fins a 280 píxels, on cada píxel representa un punt de mesura. El temps de mesura de l'ADC amb el prescaler és de 13 cicles de rellotge ADC, el qual, ajustat a 125 kHz, suposa un temps de 104 μs. De forma aproximada (si hi sumem el temps que triga a fer altres càlculs com per exemple emmagatzemar cada dada), podem dir que es poden emmagatzemar fins a 190 punts per cicle per a una ona de 50 Hz.

No obstant, malgrat que a la pràctica ens permetrà veure la forma d'ona de tensió, el temps de mostreig serà molt baix, de manera que no podrem visualitzar diverses ones consecutives. Si volem augmentar el temps de mostreig, haurem de salvar més dades al buffer, el qual vol dir que haurem de crear-ne un de nou de major grandària.

En el cas d'un PC domèstic, que disposa d'una quantitat de RAM enorme (entorn a 4 – 8 GB), això no seria un problema en absolut; malauradament, la memòria SRAM d'una Atmega 2560 és tan sols de 4 kB! Una manera indispensable d'estalviar espai és declarant la grandària d'aquestes variables com a int8_t (això és, variables de tipus *signed integer 8-bit*).

S'ha de tenir amb compte, però, que els valors màxims possibles per a una variable de 8 bits se situen entre -127 i 128. Tanmateix, ja hem esmentat en diverses ocasions que el marge de valors possibles de l'ADC se situa entre 0 i 1023; no podrem inserir aquestes dades en el buffer a menys que els multipliquem per un factor reductor.

Però també tenim una altra problemàtica si féssim el següent procediment:

```
bucle principal -> repetir indefinidament
  netejarPantalla()
  bucle secundari -> des de 0 a 280
          obtenirvalorADC()
  fi bucle secundari
  dibuixarPunts()
fi bucle principal
```

Codi 25. Segona implementació en pseudocodi per a la representació de gràfiques.

Aquesta aproximació és senzilla d'implementar perquè en tot moment tenim les dades dels punts a l'hora de dibuixar, així que la diferència entre la rutina netejarPantalla() i dibuixarPunts() és únicament el color en què es dibuixaria; netejarPantalla() utilitzaria el color de fons (blanc) i dibuixarPunts() un altre color, com el nostre cas, el blau. És a dir, ja no hauríem de netejar la pantalla a partir d'un gran rectangle amb les dimensions de la

pantalla, cosa que consumia molt més temps quan al cap i la fi només cal pintar de blanc un conjunt reduït de píxels que són els que formen el dibuix de l'ona.

Tanmateix, a la pràctica resulta molest de veure esborrar un extrem tot just haver acabat de dibuixar l'altre. Una millor implementació seria la següent:

```
bucle principal -> repetir indefinidament
 bucle secundari -> des de 0 a 280
       obtenirvalorADC()
 fi bucle secundari
 bucle secundari -> des de 0 a 280
       esborrarPuntAntic()
       dibuixarPuntAntic()
 fi bucle secundari
fi bucle principal
```

Codi 26. Es creen dos buffers per fer dibuix i esborrat de forma simultània.

L'efecte visual és millor, ja que no ha de netejar la pantalla abans sinó que a mesura que dibuixem punts nous esborrem els antics. No obstant, comporta que haurem de crear un segon buffer on emmagatzemar les dades antigues. El nou algorisme seria el següent:

```
bucle principal -> repetir indefinidament
  copiarDades(buffer_nou ->buffer_antic)
  bucle secundari -> des de 0 a 280
         obtenirvalorADC(buffer_nou)
  fi bucle secundari
  bucle secundari -> des de 0 a 280
         esborrarPuntAntic(buffer_antic)
         dibuixarPuntAntic(buffer_nou)
  fi bucle secundari
fi bucle principal
```

Codi 27. Implementació definitiva en pseudocodi de representació de gràfiques mitjançant dos buffers.

El qual implica una major despesa de SRAM, ja que hem d'emmagatzemar 280 bytes de dades més. Però malgrat que 560 bytes en dades són un percentatge molt important dins d'un total de 4 KB, és l'única forma que tenim d'assegurar un funcionament òptim de l'oscil·loscopi.

Fins ara hem parlat de com emmagatzemar les dades i la forma de dibuixar-les, però no sobre com tractar-les. Recordem que anteriorment hem dit que les dades s'emmagatzemaran en buffers de 8 bits per casella. Els valors que provenen de l'ADC oscil·laran entre 0 i 1023, de manera que per adaptar-los a la resolució de la nostra pantalla, hauran de realitzar-se una regla de tres per adaptar-lo a la resolució de la nostra pantalla. A més, s'haurà d'aplicar un filtre passa-alts per eliminar l'offset en DC que aporta el hardware de l'EmonTXShield.

El temps de càlcul s'ha de reduir el màxim possible, de manera que els càlculs amb coma flotant haurien d'evitar-se perquè l'Arduino no disposa d'una unitat de coma flotant. Podem aplicar l'algorisme següent:

$$\textbf{(17)} \quad Y[i] = \alpha[Y[i-1] - (X[i] - X[i-1])$$

Sempre i quan la variable α no sigui de coma flotant. Com que el valor que utilitzem és de α=0,996, pot aproximar-se a 255/256, el qual, alhora, és 255>>8. Si englobem el següent terme:

$$\textbf{(18)} \quad n = \big(Y[i-1] + (X[i] - X[i-1])\big) \Longrightarrow Y[i] = \propto \cdot n$$

Podem escriure l'anterior fórmula com a:

$$\textbf{(19)} \quad Y[i] = \alpha \cdot n = (255 \gg 8) \cdot n = ((n \gg 8) - n) \ll 8$$

D'aquesta manera, per tant, podem implementar el mateix algorisme utilitzant simplement nombres enters. Però encara tenim xifres que oscil·len entre 0 i 1023. Per poder-ho adaptar a la nostra pantalla, utilitzem un factor reductor en què un valor de ± 512 (màxima desviació del centre possible, ja que 0 representa el menor valor negatiu possible) suposi un increment de Y de ± 80 .

El resultat de $\frac{80}{512}$ és 0,15625. De nou, si volem optimitzar el temps de càlcul, haurem de fer ús de nombres enters; en aquest cas, utilitzarem les llibreries LibFixMath, que fan ús de nombres Q16.16 (on, en un nombre enter QX.Y, X representa el total de bits dedicats a la part entera i Y el total de bits dedicats a la part decimal), de manera que el factor reductor seria de $0{,}15625 \cdot 2^{16} = 10240 = 0x2800$.

Finalment, per evitar que, a causa de l'HPF, els valors se surtin de la pantalla, farem una comprovació del valor obtingut, assegurant-nos que és menor a ± 80 , ja que els buffers només admetran valors entre -128 i 127:

```
  if(temp> -MAX_HEIGHT &&temp< MAX_HEIGHT)
//If value is too big (or small), use previous value as reference
        samples[j]=(int8_t)temp;
  else
        //If value is correct, store it
        samples[j] = samples[j-1];
  j++;
```

Codi 28. Implementació en C++ per representar únicament els valors que no surten de pantalla.

Per fer més flexible l'oscil·loscopi, podem permetre a l'usuari definir el temps de mostreig. Sense entrar en detalls de temporització, la forma més senzilla és simplement prendre 280 valors de l'ADC i dibuixar-ne un cada X iteracions. Per exemple:

```
total_samples = 280 * samples_per_pixel
bucle -> des de 0 a total_samples
 agafarValorADC()
       si i >samples_per_pixel
       copiarValor(ADC ->buffer_nou)
       i = 0
 i++
fi bucle
```

Codi 29. Ajust del temps de mostreig mitjançant comparació d'un comptador *i*.

Lògicament, estem descartant moltes dades que no són dibuixades, però l'objectiu real d'incloure aquest oscil·loscopi és el de tenir una idea ràpida i aproximada de la forma d'ona de la tensió.

D'altra banda, per observar altres canals d'entrada simplement haurem de manipular el bit ADMUX, com ja hem fet anteriorment per al càlcul de la RMS.

La visualització simultània de diversos canals seria una opció desitjable, però això implicaria multiplicar per 4 l'espai a reservar en buffers, i si ja en el cas de un únic canal ja estem utilitzant 560 bytes, per a 4 canals seria de 2.240 bytes. Això seria un pes massa elevat, donat que l'ATmega 2560 només compta amb 4 kB de RAM.

5 Pressupost

Component	Quantitat	Preu / unitat (€)	Total (€)
2.8" TFT LCD Touch Shield Module for Arduino - Silver + Blue + Black	1	21,06	**21,06**
Universal Resistive Screen Stylus Pen with Strap for Cellphone/MP3/MP4/GPS/Laptop – Black	1	1,82	**1,82**
Improved Funduino Mega 2560 R3 Module (Compatible w/ Official Arduino Mega 2560 R3) - Blue + Black	1	15,98	**15,98**
100A max clip-on current sensor CT	4	13,29	**53,16**
TOTAL			**92,02**

6 Resultats

6.1 Aspecte del dispositiu

En un començament no era possible acoblar els dos shields a l'Arduino MEGA 2560, ja que físicament es tocaven a l'hora de connectar-los a la placa. Així doncs, es va decidir utilitzar una placa perforada de topos que actués com a connexió intermèdia entre la pantalla i l'EmonTX.

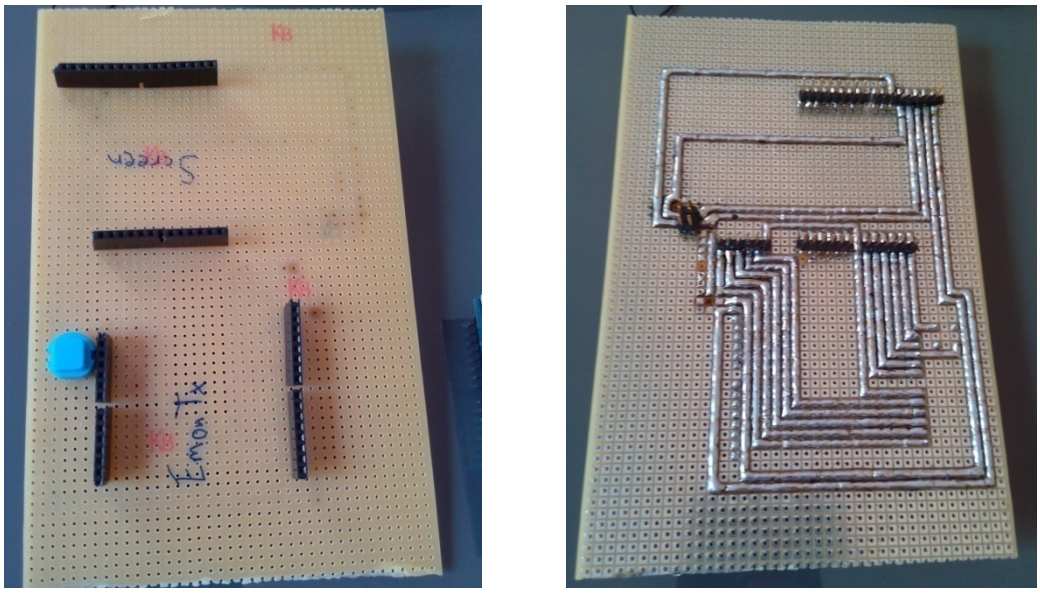


Figura 16. A l'esquerra, vista superior de la placa topos sense cap hardware connectat. A la dreta, vista inferior de la placa mostrant les connexions entre els shields i l'Arduino MEGA 2560.

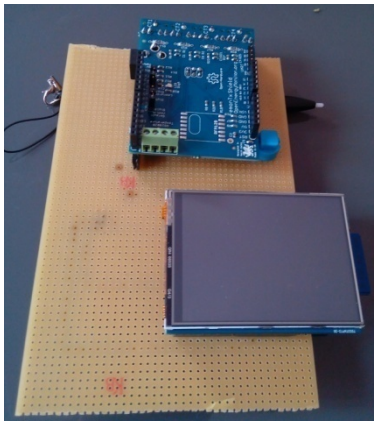


Figura 17. Vista superior de l'analitzador de xarxes.

6.2 Assaigs en laboratori

Per posar a prova la precisió de l'aparell, hem muntat un assaig al laboratori 001 de l'ETSE que ha consistit en la mesura del corrent d'engegada directa d'un motor trifàsic de gàbia d'esquirol sense càrrega, comparant els valors amb altres instruments de mesura i guardant els valors a una targeta SD. L'objectiu era el de veure fins a on podien exprimir-se les capacitats de l'aparell, ja que en un principi es trobava dissenyat per a aplicacions domèstiques i obtenció de dades a llarg termini, en comptes de valors instantanis de l'engegada d'un motor.

Figura 18. El laboratori abans de muntar els aparells de mesura amb els motors a utilitzar.

Figura 19. Placa de característiques del motor trifàsic.

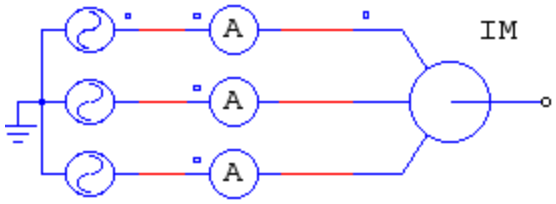


Figura 20. Esquema elèctric de la instal·lació.

6.2.1 Primer assaig. Engegada directa

El primer assaig ha consistit en la lectura de l'engegada directa del motor, salvant les dades a la targeta SD. Els canals que s'han activat han estat els 3 de corrent (CT1 a CT3) i el canal de tensió. Aquests han estat els resultats:

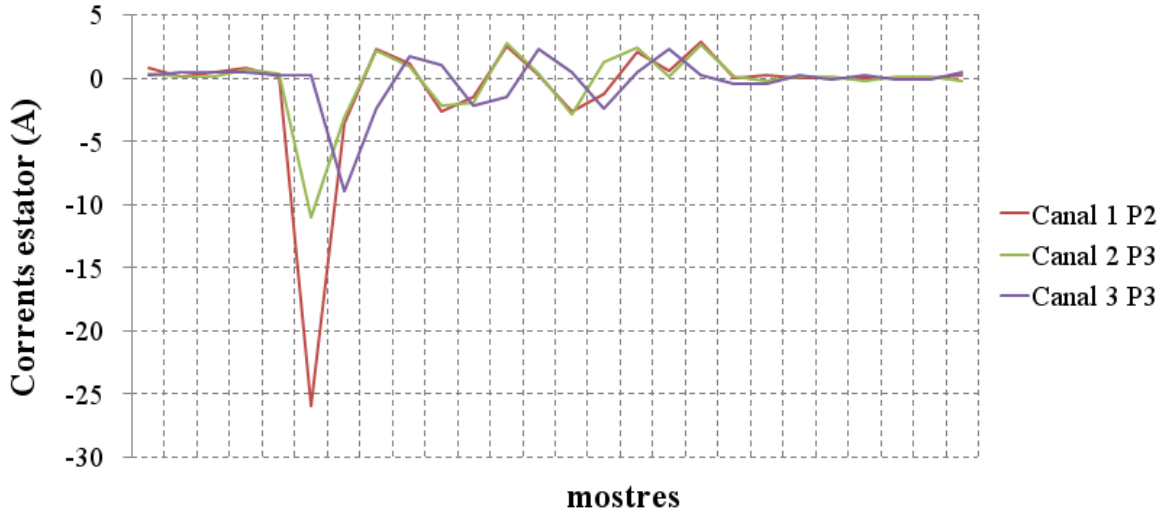


Figura 21. Gràfiques d'evolució temporal dels valors de corrent durant l'engegada en el primer assaig.

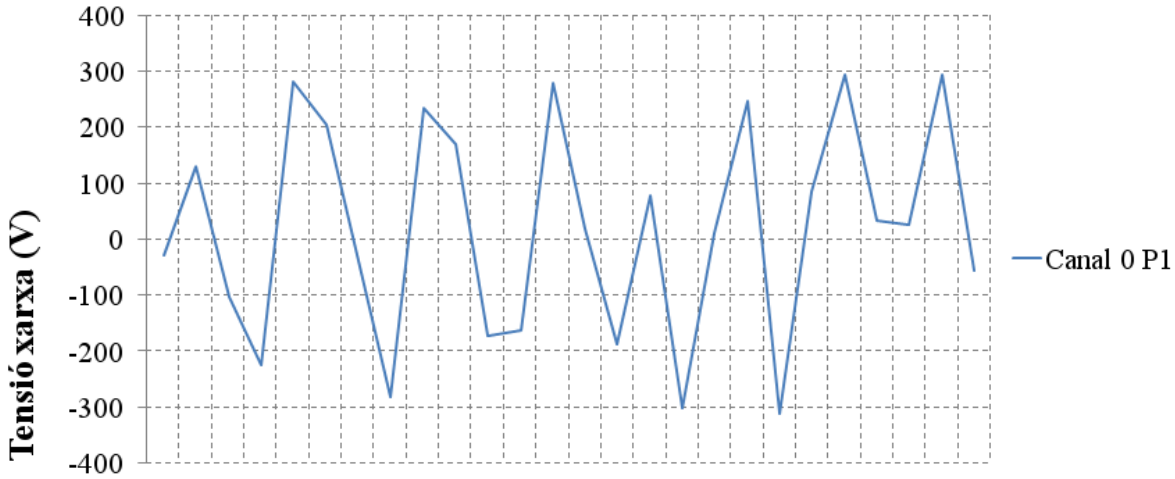


Figura 22. Gràfica d'evolució de tensió a la xarxa elèctrica en el primer assaig.

Clarament s'aprecia en la distorsió dels senyals (molt lluny d'assemblar-se a una ona sinusoïdal) que l'ADC de l'Arduino és massa lent per llegir informació procedent de 4 canals alhora. Si volem llegir valors instantanis de tensió i corrent, serà millor llegir un únic canal, ja que recordem que l'ADC obté els valors de forma seqüencial (tensió – CT1 – CT2 – CT3 – CT4 i comença de nou el cicle).

Amb l'objectiu de poder apreciar millor el pic de corrent produït a l'engegada, configurem el dispositiu perquè només registri valors del canal CT1. Aquest ha estat el resultat:

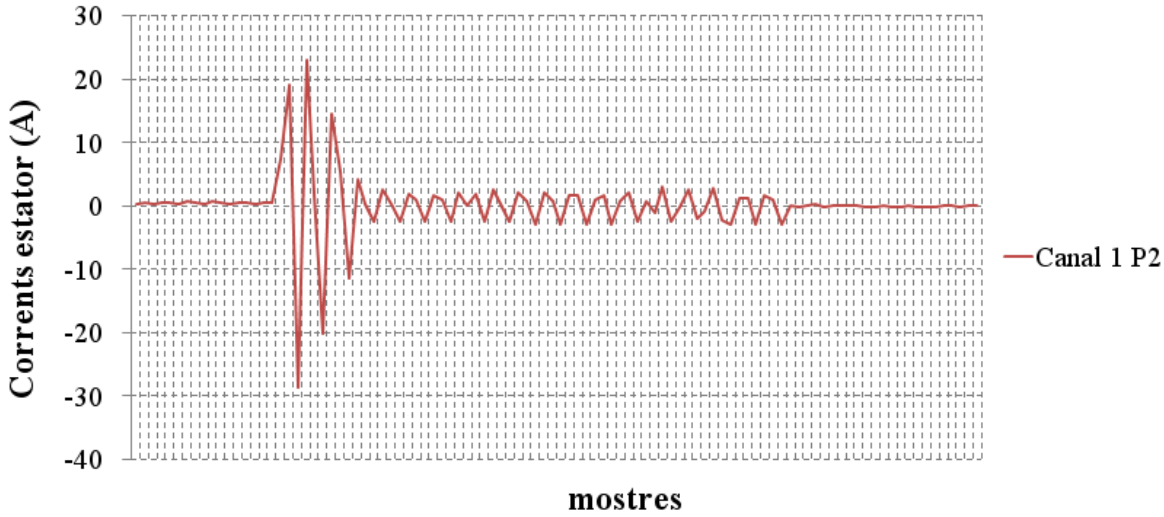


Figura 23. Evolució de corrent mesurant un únic canal (CT 1).

El resultat ha millorat força (en règim permanent es pot apreciar que les ones s'assemblen més al resultat teòric), però continua existint una distorsió considerable.

Una manera d'incrementar la velocitat a què l'ADC obté els valors és augmentant la freqüència f_{ADC}, que es troba ajustada pel prescaler. Per defecte, el nostre programa l'ajusta a 125 kHz (que correspon a un factor divisor de 128), però es pot incrementar ajustant el factor a 64, que farà que f_{ADC} = 250 kHz. Augmentar aquesta freqüència permet, com ja hem dit, la velocitat de l'ADC, però comporta una penalització en la precisió.

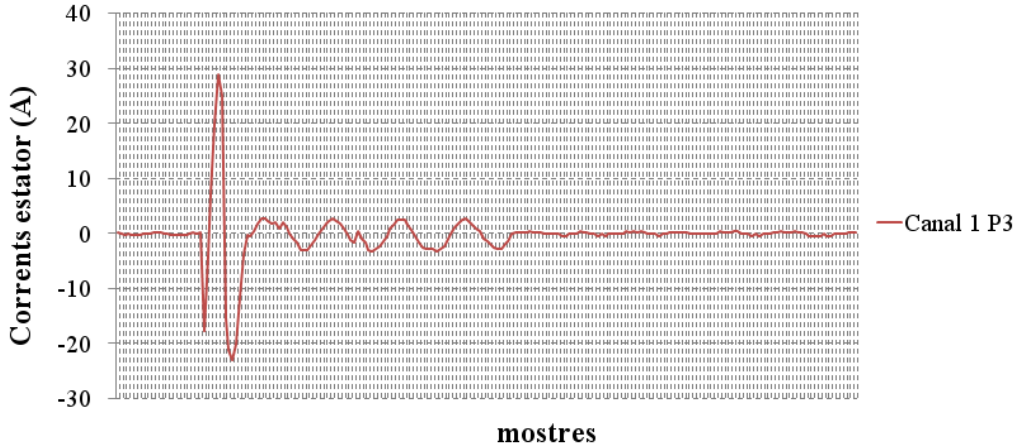


Figura 24. Evolució del corrent amb f_{ADC} = *250 kHz.*

Podem intentar pujar encara més f_{ADC} fins a 500 kHz i observar els resultats:

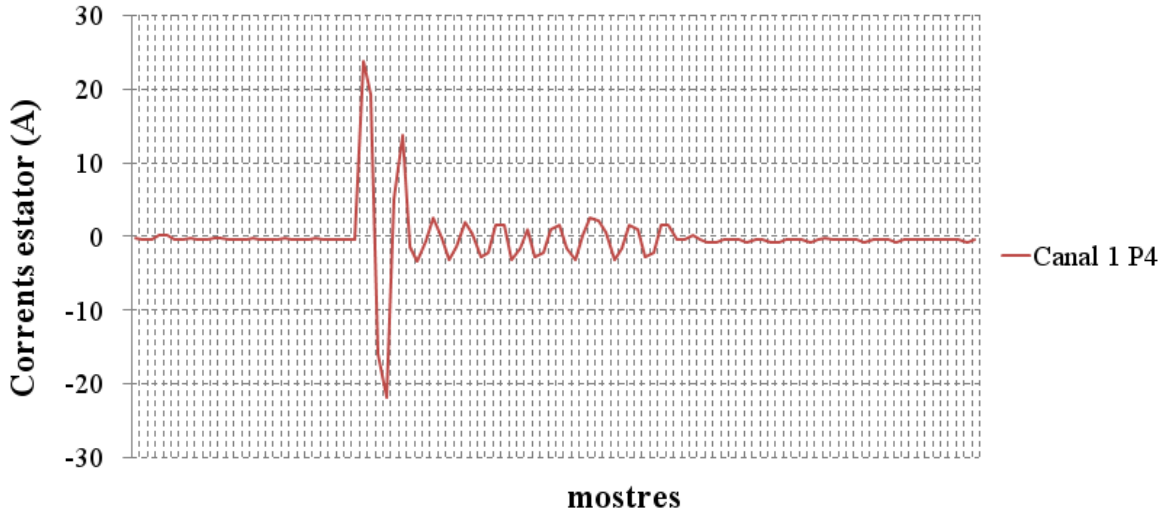


Figura 25. Evolució del corrent de l'engegada directa per a f_{ADC}=500 kHz.

Hi ha certa millora, però la velocitat d'obtenció de punts continua essent massa lenta. El factor limitant no únicament és el temps que triga l'ADC a obtindre un valor, sinó la velocitat de transferència de dades entre l'Arduino i la targeta SD, que depèn de molts factors (implementació de les llibreries de comunicació SPI i SDFatLib, velocitat de transferència de la pròpia targeta SD, etcètera).

6.2.2 Segon assaig. Valors en RPS

Donades les limitacions del sistema, vam desitjar provar a registrar dades del motor asíncron en règim permanent sinusoïdal, llegint els tres corrents. Els resultats van ser els següents:

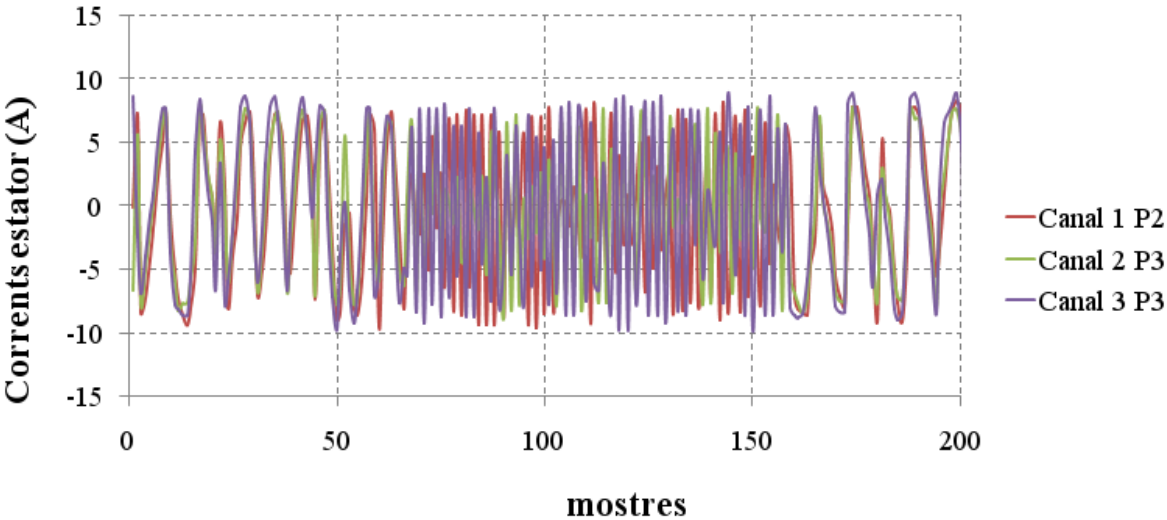


Figura 26. Resultats de l'assaig en regim permanent sinusoïdal d'un motor asíncron de 3,5 kW en buit. S'observa en la part central de la gràfica un augment en la velocitat de transferència de dades entre l'Arduino i la SD.

Queda patent en aquest exemple que el dispositiu està programat per obtindre dades de l'ADC tan ràpid com pugui i no de forma isòcrona. Això provoca que la velocitat a què es transfereixen les dades a la SD sigui depenent de factors externs i difícils de controlar, com per exemple el flux d'execució del programa.

6.3 Mesura corrent eficaç d'un motor asíncron en buit en RPS

Una altra de les proves que es van realitzar és la de mesurar el corrent eficaç del motor de l'apartat anterior quan es trobava en règim permanent. Per comprovar la precisió de les mesures, en paral·lel al dispositiu es van mesurar els mateixos valors amb instruments analògics del laboratori.

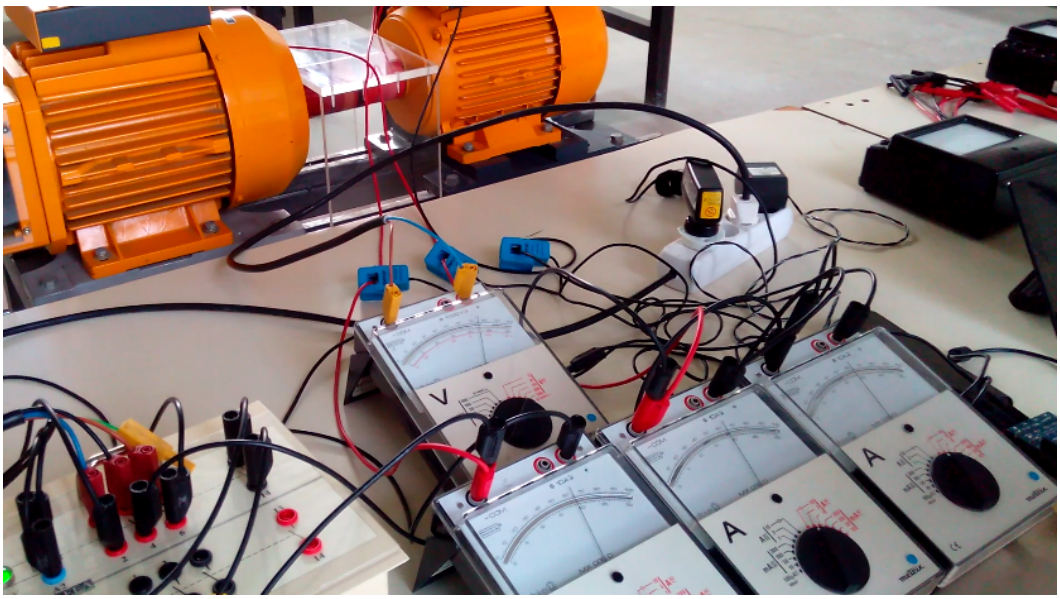


Figura 27. Vista del laboratori amb els aparells de mesura analògics, el quadre de maniobra i el motor.

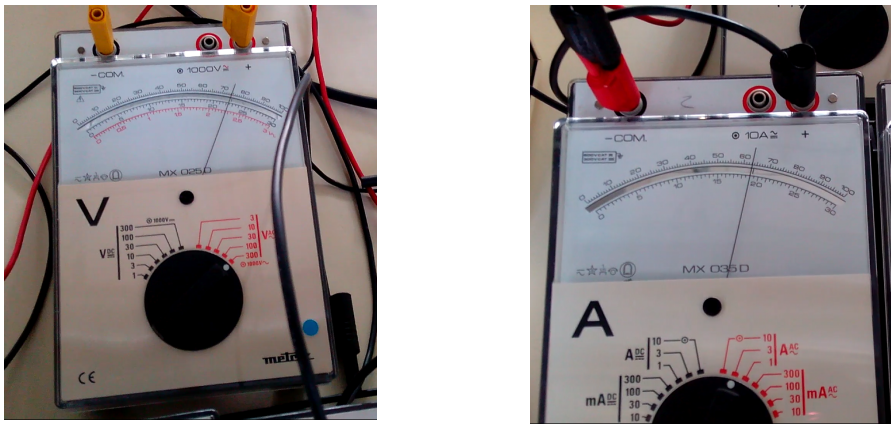


Figura 28. A l'esquerra, voltímetre mesurant la tensió de fase. A la dreta, amperímetre mesurant el corrent que circula per una de les fases del motor.

Com es pot observar a les imatges, el corrent eficaç que circulava pel motor era aproximadament de 2 A. Això quedà també enregistrat pel dispositiu, que mostrà valors similars per a les tres fases:

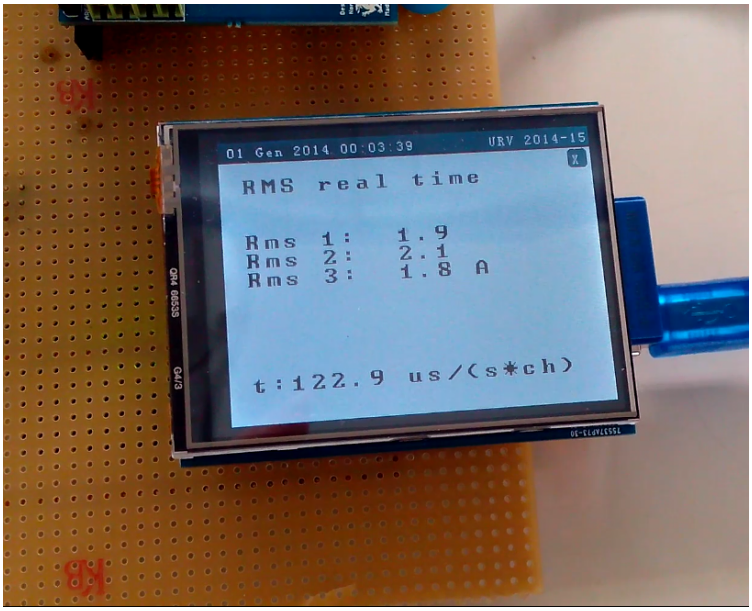


Figura 29. Vista del dispositiu mostrant els valors eficaços de corrent a les fases del motor asíncron.

Amb aquest assaig queda patent la capacitat de l'aparell per mesurar valors de tensió i corrent eficaços en temps real, en comparació al seu comportament limitat com a dispositiu d'enregistrament de dades.

6.4 Assaig del mode oscil·loscopi en un domicili

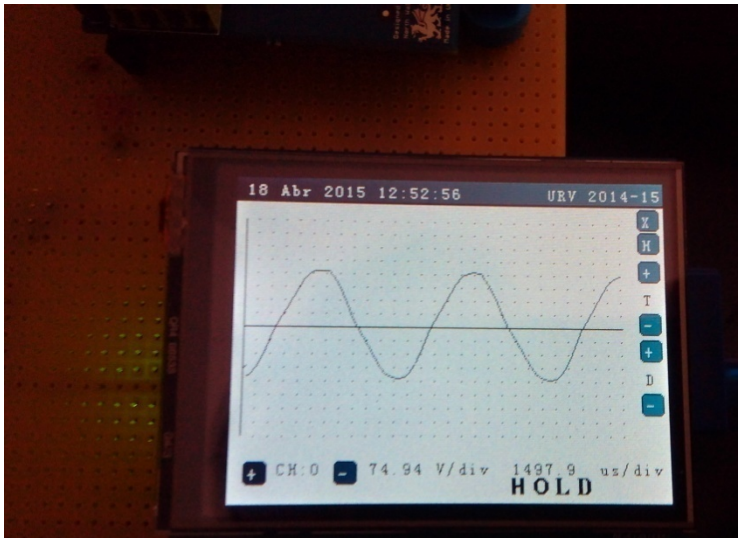


Figura 30. L'aparell funcionant en mode oscil·loscopi.

Una de les altres modalitats de l'aparell, com ja s'ha mencionat abans, és la de visualitzar les ones de tensió i corrent. S'ha realitzat una prova que ha consistit a obtindre la forma de senyal de la xarxa elèctrica en connectar l'entrada 9 V AC de l'EmonTX Shield a l'endoll d'un domicili.

Aquest mode funciona de forma diferent a l'enregistrament de dades dels apartats anteriors, ja que aquí primer s'emmagatzemen els valors de l'ADC en un buffer i posteriorment s'imprimeixen per pantalla. Així doncs, la velocitat a què les dades són guardades és molt més elevada, i la distorsió de la gràfica resultat és molt menor.

Per tant, en aquest assaig queda demostrat que el factor limitant principal és la comunicació SPI entre l'Arduino i la SD. El prescaler en aquest assaig s'ha ajustat a 125 kHz; s'observa que fins i tot amb la menor velocitat de conversió analògic-digital es permet obtenir resultats precisos.

7 Conclusions

Una primera afirmació que podríem fer és que els microcontroladors actuals han evolucionat enormement des dels seus inicis. L'equip d'Arduino ha demostrat que la programació d'un MCU pot ser quelcom intuïtiu i amb una corba d'aprenentatge molt suau, principalment gràcies al disseny senzill i elegant dels xips ATMega desenvolupats per Atmel. En poc menys d'un any, he pogut fer un inici en aquest món des de zero fins aconseguir per fi uns resultats força palpables.

Els resultats de l'apartat anterior han deixat patents que, malgrat de les limitacions del sistema, és possible crear un analitzador de xarxes elèctriques de baix cost amb possibilitat d'emmagatzematge de dades. Òbviament, encara som molt lluny de les prestacions d'un aparell comercial, però ningú no podria negar que es tracta d'un primer pas important. No obstant, s'ha de remarcar que la finalitat del dispositiu no és el de mesurar la intensitat que circula per un motor trifàsic, sinó que es troba més enfocat a un àmbit domèstic, on sí compleix les seves expectatives.

L'equip de OpenEnergyMonitor treballen en un altre camí, més enfocat a obtenir un aparell que enviï dades a un servidor i, a partir d'aquí, interpretar aquestes dades amb un ordinador o fins i tot amb un *smartphone*. La meva intenció des d'un inici va ser desviar-me d'aquesta tendència, a favor del desenvolupament d'un aparell completament autònom que també pogués treballar amb un ordinador a partir d'arxius de text simples.

El primer que vaig sentir en finalitzar aquest Treball és el potencial que podria tindre. El principi de funcionament simple de l'aparell i la seva modularitat em suggereixen una possible segona versió en el futur, on aprendre dels errors comesos durant el desenvolupament del prototip. Idees com utilitzar un MCU amb un ADC més potent, escriure un millor firmware més intuïtiu i funcional, alimentació autònoma amb bateries recarregables o un disseny integrat en una única PCB podrien polir el que considero personalment un petit diamant en brut.

8 Annex I. Manual de l'usuari

8.1 Engegada

En endollar el dispositiu, el primer que mostrarà serà una sèrie de missatges de comprovació. Els passos que realitzarà seran carregar les fonts de text, comprovar la tensió d'alimentació, inicialitzar les variables principals, carregar la targeta SD, comprovar si hi existeixen dades sobre calibratge i carregar-les en cas afirmatiu. **És molt important destacar que és necessari tenir una targeta SD per poder iniciar el programa.** En cas contrari, el programa informarà d'un error i s'aturarà.

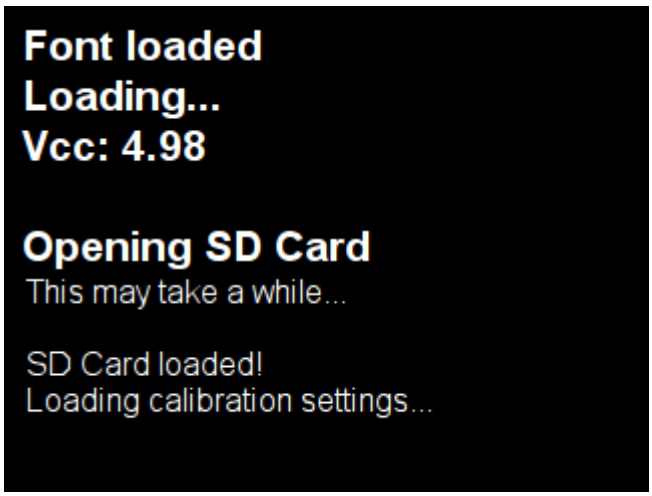


Figura 31. Pantalla d'inici.

8.2 Menú principal

Un cop s'ha realitzat el procediment d'engegada, es mostra a l'usuari el menú principal. Consta de dues pàgines amb quatre botons a cadascuna amb les diverses opcions possibles. A la part superior de tots els apartats es mostra una barra que mostra de forma alterna la data i hora actuals i el títol del Projecte, amb el missatge "URV 2014-15" a la part superior dreta de la pantalla.

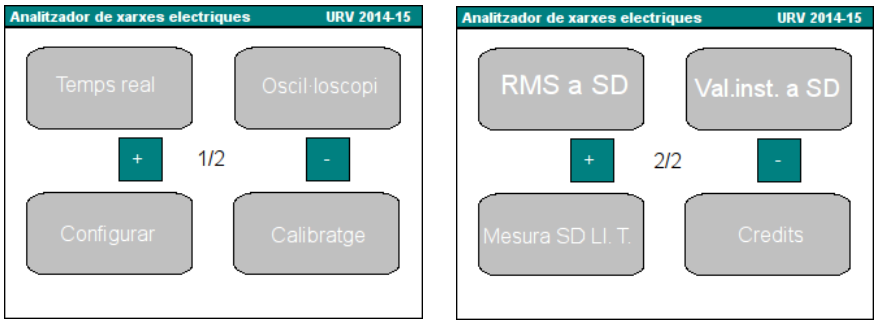


Figura 32. Pantalles del menú principal.

Les opcions disponibles són:

- Pàgina 1:
  - Temps real: impressió per pantalla dels valors eficaços de tensió i corrent en temps real.

- Oscil·loscopi: senzill visualitzador de senyals amb funcionalitats bàsiques.
- Configuració: permet ajustar paràmetres bàsics de l'aparell, com data i hora, canals actius o temps de mesura a llarg termini.
- Calibratge: permet calibrar els diferents canals de tensió i corrent mostrant el valor eficaç en temps real. En sortir del menú salva les dades en un arxiu anomenat K.CAL.

- Pàgina 2:
  - RMS a SD: permet salvar a la SD els valors eficaços de corrent i tensió obtinguts tan ràpid com el xip pugui (aproximadament 0,20 segons).
  - Val. Inst a SD: permet salvar a la SD els valors instantanis de corrent i tensió.
  - Mesura SD. Ll. T: permet salvar a la SD la mitjana aritmètica dels valors eficaços obtinguts durant un llarg període de temps (per defecte, 10 minuts).
  - Crèdits: mostra els crèdits de les persones i empreses que han fet possible aquest Projecte .

8.3 Lectura en temps real

El menú "Temps real" permet saber en temps real valors RMS de tensió i corrent. Si hi entrem dins, se'ns mostraran els diferents canals, on 0 és el canal de tensió i 1-4 els canals de corrent. Cal destacar que només es mostraran els canals que es trobin actius, quelcom que es pot ajustar al menú de configuració.

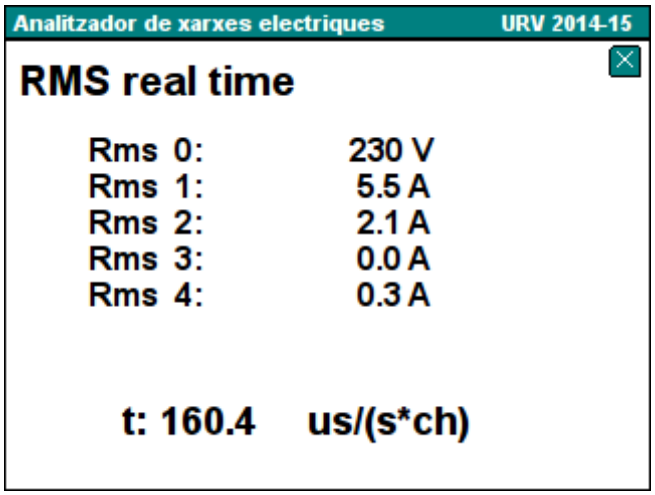


Figura 33. Pantalla de mesura de valors eficaços en temps real.

A la part inferior de la pantalla es calcula el temps, en μs, que el programa triga a processar un sample d'un canal. Per saber el temps total, s'utilitza la següent fórmula:

$$\textbf{(20)} \qquad t_T = 0{,}25t \cdot c\ [ms]$$

8.4 Oscil·loscopi

El menú "Oscil·loscopi" permet visualitzar les ones de tensió i corrent de forma ràpida i senzilla. Al igual que el mode "Temps real", en aquest menú no s'emmagatzemen dades a la targeta SD, sinó que únicament són mostrades per pantalla.

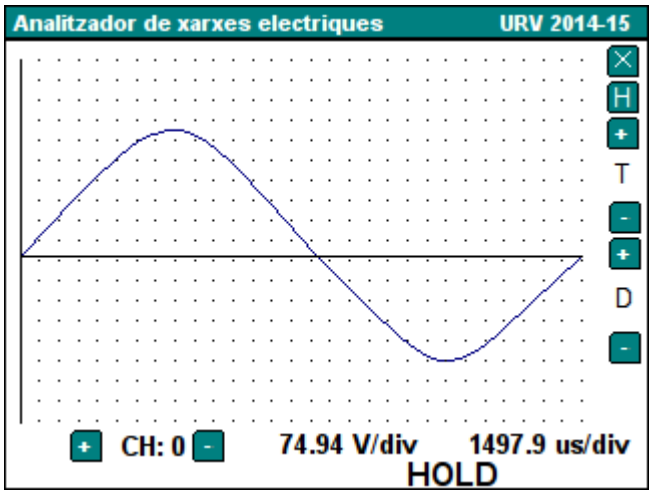


Figura 34. Mode oscil·loscopi.

A la part inferior esquerra de la pantalla podem seleccionar el canal, on 0 és tensió i 1-4 els canals de corrent. També es mostren els V/div (o A/div), que es calculen de la següent forma:

$$\frac{u}{div} = K_{cal} \cdot \frac{V_{CC}}{2^{10}-1} \cdot x \cdot \frac{512}{h} \cdot \frac{1}{m} \quad \textbf{(21)}$$

On:

- K_{cal}: constant de calibratge del canal.
- V_{CC}: tensió d'alimentació d'Arduino.
- x: píxels per divisió (x=10).
- 512/h: relació entre valor màxim positiu possible de l'ADC i alçada del semieix (h=80).
- m: factor d'escala de l'oscil·loscopi.

8.5 Calibratge

A fi de millorar la precisió de les mesures, l'usuari pot ajustar la constant de calibratge de cada canal a partir de la comparació dels valors obtinguts amb un altre instrument. Quan s'entra en aquest menú, es mostra en temps real el valor eficaç del canal seleccionat i la susdita constant, la qual podrà modificar-se a través dels botons del menú.

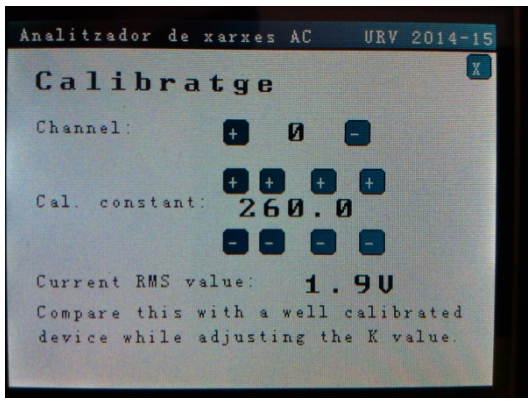


Figura 35. Menú de calibratge.

En tancar el menú, les constants són guardades a un arxiu dins de la targeta SD per a la pròxima vegada que l'usuari iniciï el dispositiu.

8.6 Menú de configuració

A part de les constants de calibratge, l'usuari pot realitzar altres ajustos al dispositiu, com data i hora, velocitat del prescaler, etcètera.

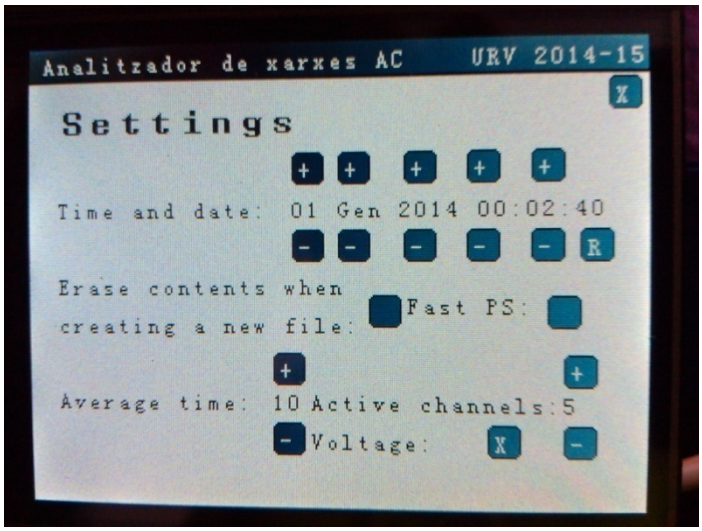


Figura 36. Menú de configuració.

Detall de cada opció:

- *Time and date*: permet ajustar dia i hora.
- *Erase contents when creating a new file*: a l'hora d'emmagatzemar dades a la SD, el dispositiu sempre crea un arxiu amb el format DDMMMYY.TXT (Day/Month/Year). En el cas que l'arxiu ja existeixi, si l'opció es troba desactivada guardarà les dades a sota de les dades antigues; en cas contrari, esborrarà els continguts antics per bolcar-hi les noves dades.
- *Fast PS*: deshabilita el rellotge de la part superior i canvia la freqüència f_{ADC} de 125 kHz a 500 kHz. La finalitat és poder reduir les distorsions a l'hora de guardar valors a la targeta SD, tal i com s'ha vist a l'apartat de resultats.
- *Average time*: estableix el temps, en minuts, per al mode de mesura en llarg termini. Per defecte, es troba a ajustat a 10 minuts.
- *Active channels*: permet ajustar els canals actius per mesurar. Depèn de l'opció *Voltage*, que activarà o desactivarà l'entrada de tensió.

8.7 Enregistrament de dades a la SD

Totes les variants relacionades amb guardar dades a la targeta SD tenen un aspecte molt similar. El primer que el programa requerirà a l'usuari és que la data i hora estiguin correctament ajustats, ja que contràriament es llançarà un missatge d'error:

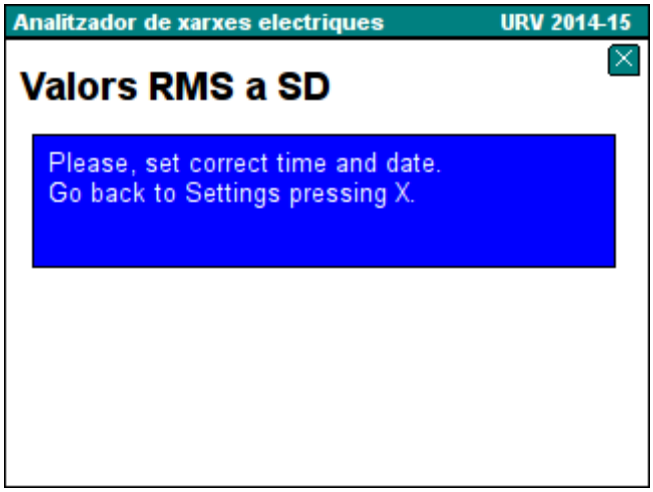


Figura 37. Error a l'hora de comprovar l'hora i data.

Si l'usuari ha realitzat l'ajust d'hora i data correctament, començarà l'enregistrament de dades a la SD. Depenent del mode que s'hagi escollit, es mostraran diferents missatges a pantalla. **És molt important no treure l'alimentació del dispositiu en cap moment o totes les dades es perdran!** Per salvar les dades registrades, només cal prémer el botó de sortida a la part superior dreta de la pantalla.

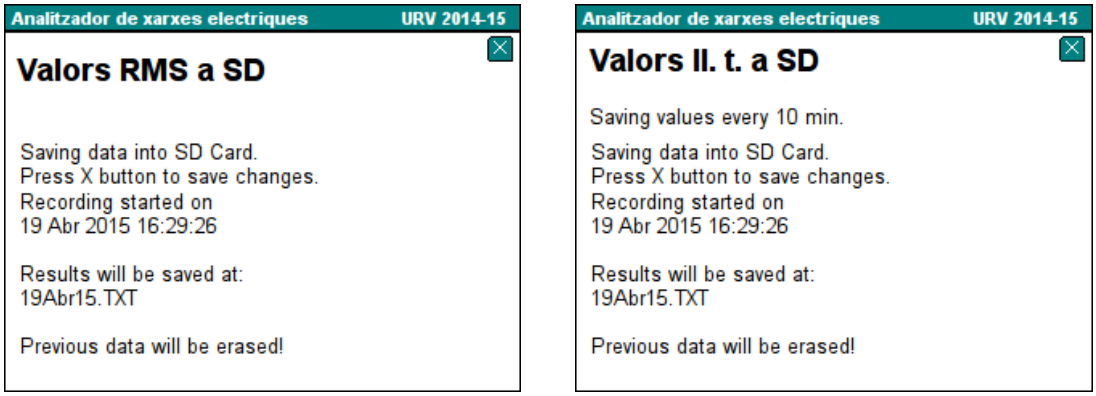


Figura 38. Enregistrament de valors eficaços a la SD.

8.8 Crèdits

El dispositiu també compta amb un menú de crèdits en reconeixement a les persones i equips que han fet possible aquest Treball.

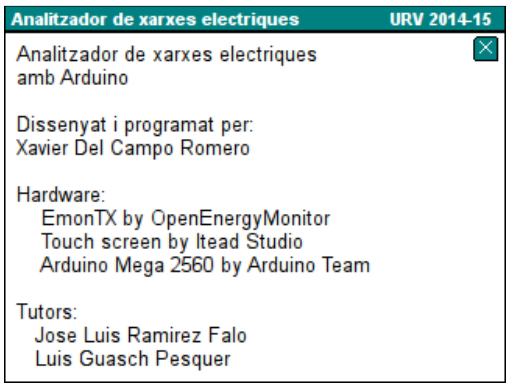


Figura 39. Pantalla de crèdits.