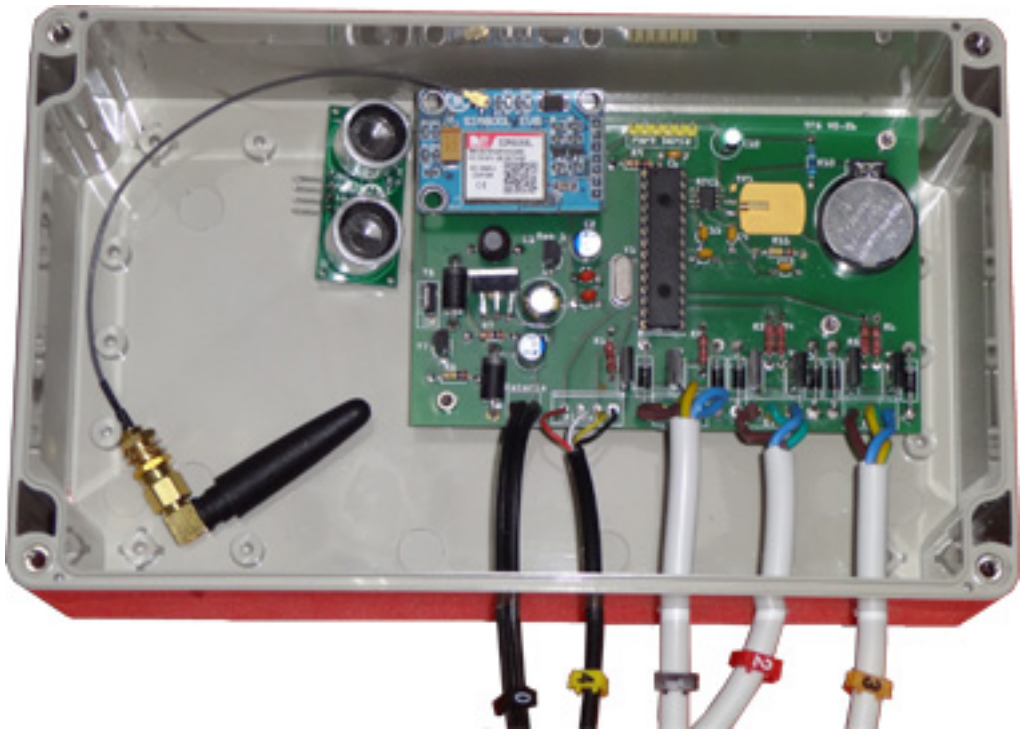
Escola Tècnica Superior d'Enginyeria del Disseny

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria del Disseny

Controlador de reg per degoteig



TREBALL FINAL DE GRAU

D'ENGINYERIA ELECTRÒNICA INDUSTRIAL I AUTOMÀTICA

REALITZAT PER FRANCESC NICOLAU LÓPEZ I NAVARRO

TUTORITZAT PER JAVIER IBÁÑEZ I CIVERA

VALÈNCIA, SETEMBRE 2019

Controlador de reg per degoteig

Índex

Controlador de reg per degoteig

Summary:

This project was created because of the necessity to control a drip irrigation sytem with remote control.

This work is about the design of an electronic device controlled from internet, which allows you to set a schedule to start and stop a drip irrigation system. In addition, the system allows us to measure the level of a tank, as well as the level of the battery that powers the circuit.

The project is based on a microcontroller Atmega328P, which is loaded with the Arduino firmware.

The circuit has been optimized to reduce its consumption and thus extend the battery life.

Keywords: Arduino, irrigation programmer, Atmega328p, low power consumption, GSM, microcontroller.

Controlador de reg per degoteig

Resumen

Este proyecto nace de la necesidad de controlar un sistema de regadío por goteo a distancia.

El trabajo trata sobre el diseño de un dispositivo electrónico controlado desde internet, que permite fijar horario en que se ha de poner en marcha y parar un sistema de riego por goteo. Además el sistema permite medir el nivel de un depósito, así como el nivel de la batería que alimenta el circuito.

El proyecto se basa en un microcontrolador Atmega328P, al cual se le carga el firmware de Arduino.

El circuito se ha optimizado para reducir su consumo y poder alargar así la duración de las baterías.

Palabras clave: Arduino, programador de riego, Atmega328p, bajo consumo, GSM, microcontrolador.

Controlador de reg per degoteig

Resum

Aquest projecte naix de la necessitat de controlar un sistema de regadiu per degoteig a distància.

El treball tracta sobre el disseny d'un dispositiu electrònic controlat des d'internet, que permet fixar l'horari en què s'ha d'engegar i aturar un sistema de reg per degoteig. A més el sistema permet mesurar el nivell d'un depòsit, així com el nivell de la bateria que alimenta el circuit.

El projecte es basa en un microcontrolador Atmega328P, al qual se li carrega el firmware d'Arduino.

El circuit s'ha optimitzat per reduir el seu consum i poder allargar així la duració de les bateries.

Paraules clau: Arduino, programador de reg, controlador, Atmega328p, baix consum, GSM, microcontrolador.

Controlador de reg per degoteig

1- MEMÒRIA.

1.1 Justificació

Cada volta més els sistemes de reg tendeixen a estar automatitzats.
En molts casos quan hi ha canvis climatològics o restriccions de reg, cal desplaçar-se fins a les instal·lacions per modificar la programació de reg, amb la consegüent pèrdua de temps i diners.

Aquestes instal·lacions es troben en llocs normalment aïllats de la xarxa elèctrica i sovint fan servir bateries, per la qual cosa han de ser dispositius de molt baix consum.

Alhora són llocs on es troben desemparats en cas de robatoris, el que suposa un doble problema, per una banda la pèrdua de l'equipament i per l'altra, fins que el propietari s'adona compte el camp no rep l'aigua que cal, el que pot provocar pèrdues.

Amb aquest projecte, es pretén crear un programador de reg, programat de manera remota.

Sovint l'obstrucció dels filtres d'entrada a les instal·lacions de reg, no es detectada a temps, el que provoca una baixada de pressió en d'instal·lació. Aquesta baixada de pressió fa que el sistema que afegís els adobs químics no funcione .

Per això es fa necessari a distància detectar possibles problemes amb el sistema d'adob. Alhora que s'ha de poder detectar problemes amb l'estat de la bateria.

1.2 Objectiu.

Dissenyar un programador de reg, que permeta controlar 3 electrovàlvules, alimentant-se d'una bateria de 12v.
Fer un consum mínim d'energia, dins de les possibilitats d'un Arduino.

Desenvolupar este projecte aprofitant les facilitats que ens aporten els microcontroladors del tipus Arduino.

No es vol que cap pin del microcontrolador subministre corrents molt superiors als 20mA.

El sistema ha de controlar 3 electrovàlvules amb positiu comú, controlades per polsos de 200mS, que els fan canviar d'estat. Segons el fabricant les electrovàlvules per a canviar d'estat han de rebre 12v (±10%) amb una potència de 5w.

A més ha de poder mesurar el nivell d'un dipòsit situat al mateix recinte de reg (la variació de nivell pot anar de 0 a 125cm). S'aprofitarà per a mesurar la tensió de la bateria guardant tota la informació de la tensió i el nivell del depòsit a un servidor remot.

El dispositiu modificarà la seua programació de reg de manera remota a través d'internet, aprofitant un servidor web i una base de dades.

1.3 Descripció de la solució adoptada.

1.3.1 Funcionament general del sistema.

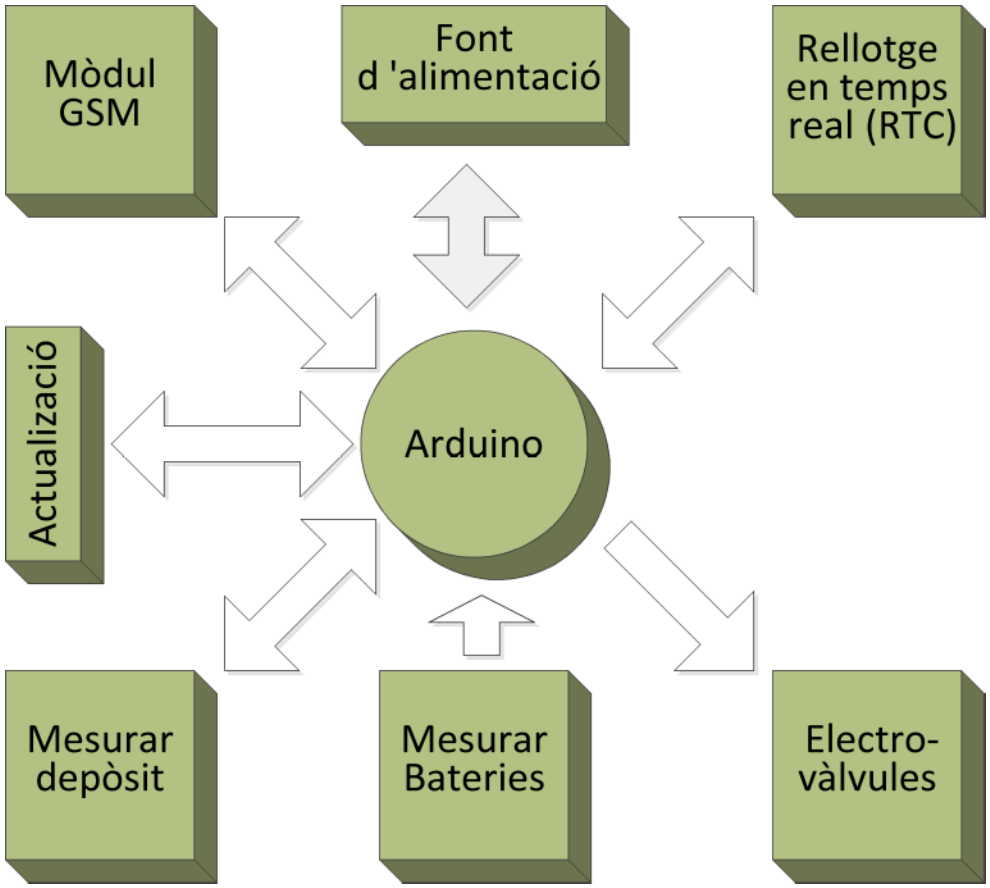


Figura 1.3.1.Diagrama general del sistema.

L'usuari pot accedir per mitjà d'internet a la informació de la base de dades durant les 24 h del dia, els 365 dies, modificant la programació de reg. L'usuari pot consultar les últimes dades rebudes del programador i valorar si cal fer canvis o acudir a la instal·lació per fer alguna actuació.

Una volta al dia el programador de reg intentarà comprovar si ha de fer algun canvi en la seua programació i actualitzarà les dades, si la informació està disponible.

1.3.2 Maquinari.

El programador basat en Arduino està format per diferents mòduls interconnectats.

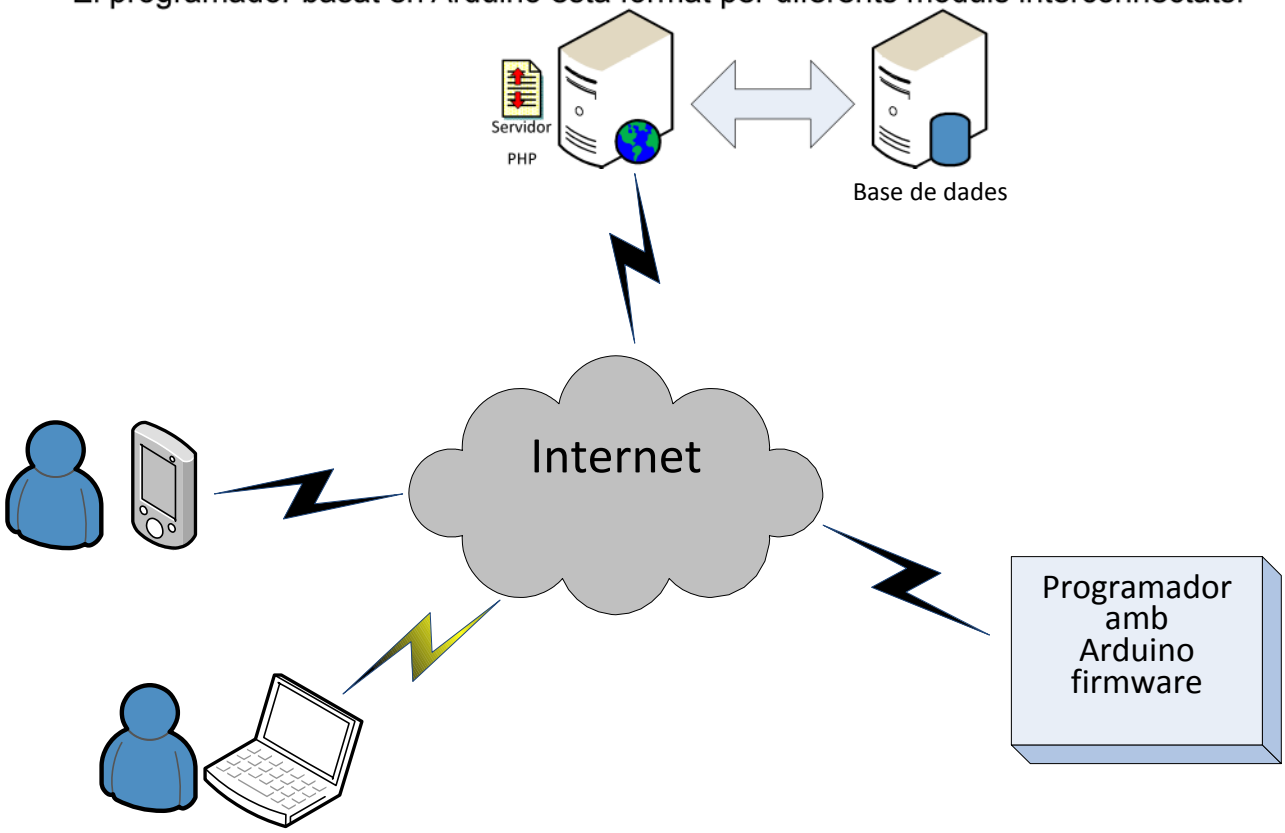


Figura 1.3.2 Diagrama de blocs del maquinari

1.3.2.1 Arduino

El primer problema a què s'ha donat solució en aquest projecte és fer una variant d'Arduino que funcione amb molt baix consum. Per això en lloc de fer servir un Arduino comercial, se fa servir un Atmel328p, amb part dels components de l'Arduino, eliminant alguns components innecessaris, per aquesta aplicació concreta.

Trobem Arduino que treballen a 8Mhz (3,3v) i 16mhz(5v), basant-se en la gràfica de corrent, freqüència i consum de les fulles de característiques del Atmega328p.

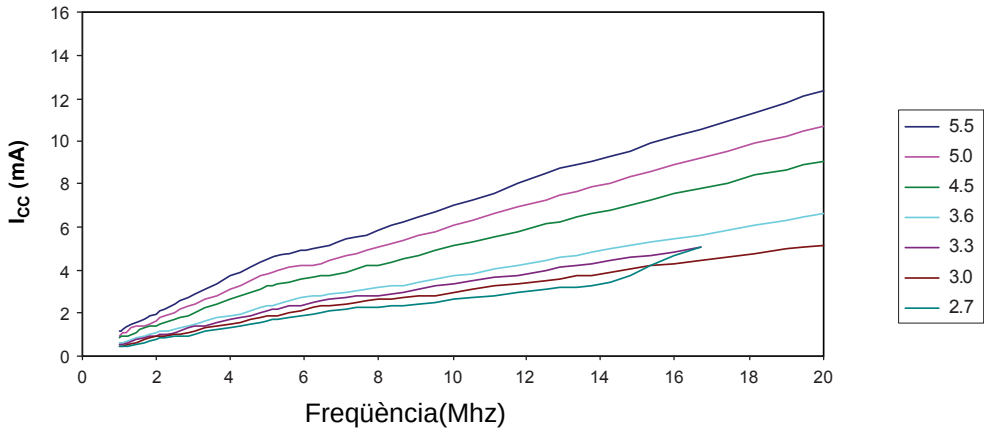


Figura 1.3.2.1.A gràfica de corrent vs freqüència i tensió

S'observa que a menys freqüència i menys tensió, menys consum. Així que aquesta versió particular d'Arduino funcionarà a 3,3v i a 8Mhz. L'esquema de mínims, quedarà com el de la figura 1.3.2.1.B.

Bàsicament s'ha respectat el circuit de comunicació per poder actualitzar el software, el cristall de quars i poc més. Eliminant el led i llevant de moment el circuit regulador de tensió.

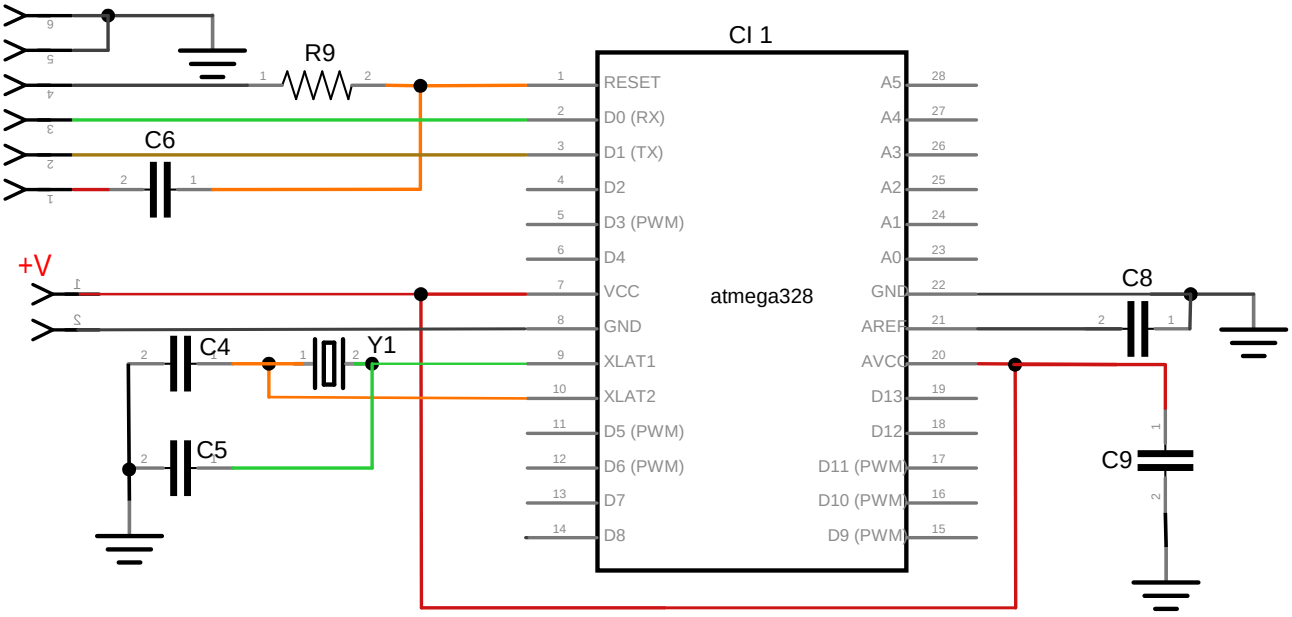


Figura 1.3.2.1.B Circuit base de l'Arduino.

Seguint l'esquema de l'Arduino mini pro, s'han posat R9 de 10kohms, C6,C8 i C9 de 100nF, els condensadors de l'oscil·lador, C4 i C5, que han de ser de valors compresos 10pF a 22pF així que se han triat de 15pF.

Com es va a treballar en un entorn d'Arduino, cal un Atmega328p amb el *frimware* d'Arduino.

Partint d'un Atmel328p nou, se li posa el *firmware* de l'Arduino mini pro en la versió de 3,3v amb una freqüència de rellotge de 8Mhz. Per poder carregar el frimware de l'Arduino, es pot fer amb un Arduino i un xicotet circuit auxiliar, tal com el de l'esquema de la Figura 1.3.2.1C

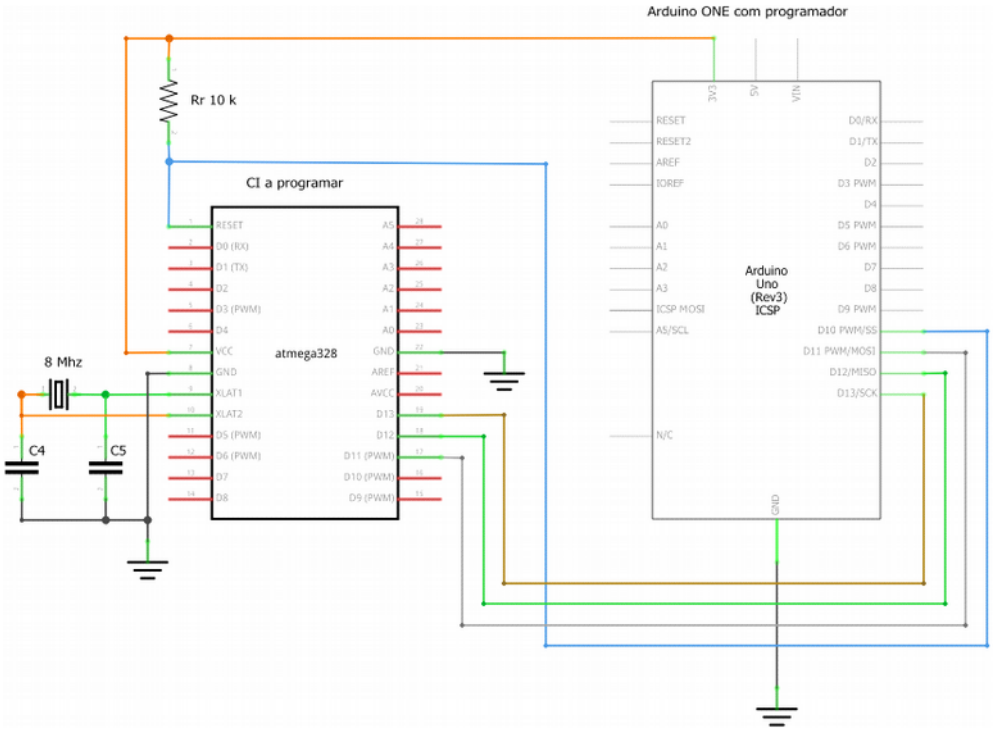


Figura 1.3.2.1.C. Esquema del circuit de càrrega del bootloader d'Arduino.

Amb l'esquema anterior i l'Arduino connectat per USB al PAC, sols cal obrir l'aplicació d'Arduino i triar l'exemple d'aplicació ArduinoISP.

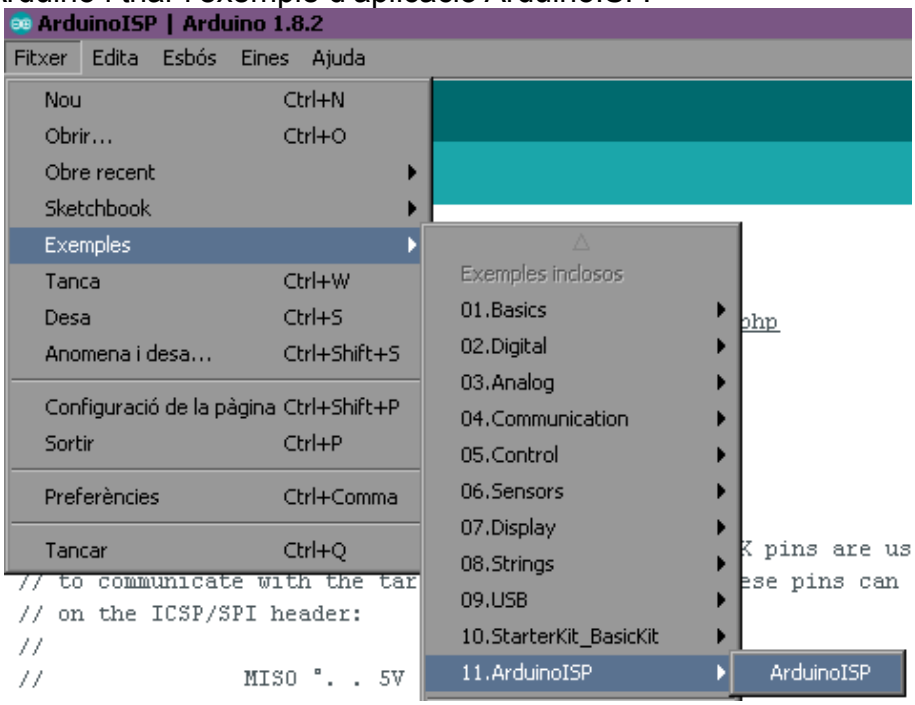


Figura 1.3.2.1D Programa ArduinoISP

Després revisar que està correctament triat el model d'Arduino i el port sèrie, Es carrega a l'Arduino el codi “ArduinoISP.ino”.

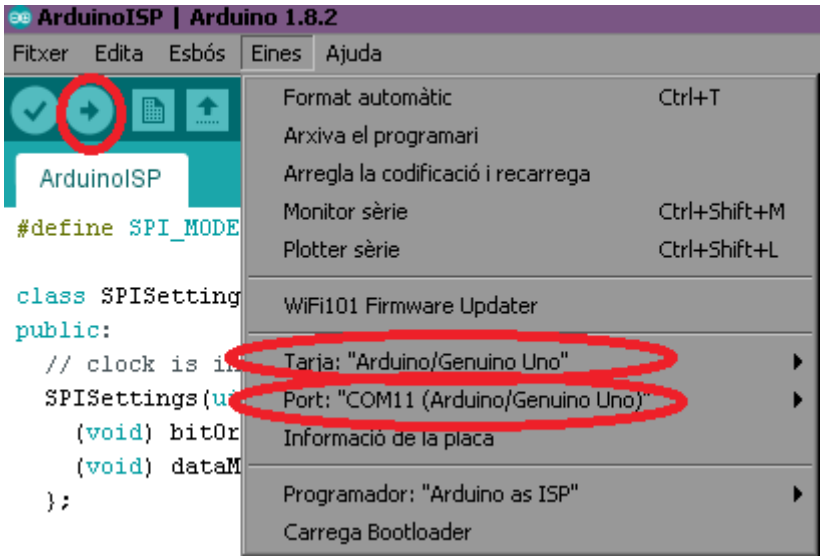


Figura 1.3.2.1.E. Configuració del programador

Una volta programat l'Arduino, ara ja es pot carregar el bootloader al ATmega328p nou, amb el circuit de Figura 1.3.2.1C. I l'ordinador connectat pel port USB a l'Arduino. A continuació es tria com targeta el firmware que interessa, en este cas "Arduino Pro or Pro Mini" amb processador "ATmega328 de (3,3v, 8Mhz). I per últim hi ha que fer clic a "Carrega Bootloader".

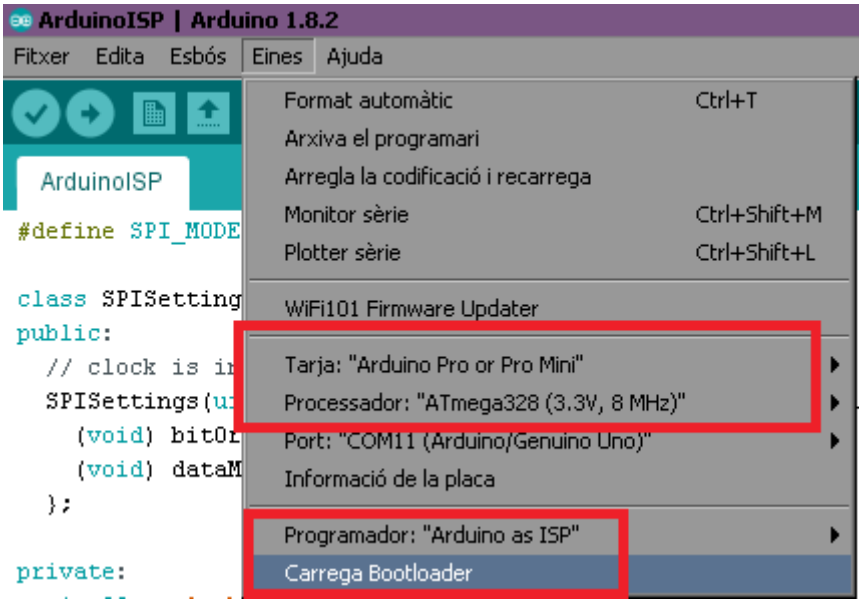


Figura 1.3.2.1.F. Triar i carregar el bootloader d'Arduino

. Una volta carregat el bootloader, el ATmega328 ja està preparat per ser programat com un Arduino Pro mini 3.3v, 8 MHz col·locant-lo al circuit de la figura 1.3.2.1B i fent servir com programador un adaptador d'USB a 232 TTL.

En este projecte, la major part de temps el microcontrolador està "esperant" que arribe el moment de fer alguna cosa, per la qual cosa no té sentit tindre funcionant en manera "Idle" de manera permanent al microcontrolador.

Hi ha diversos modes d'estalvi d'energia , cadascun d'ells permet tindre activades unes parts del circuit intern de l'Arduino i altres no.

	Active Clock Domains					Oscillators		Wake-up Sources							
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other/O	Software BOD Disable
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	
ADC noise Reduction				X	X	X	X(2)	X(3)	X	X(2)	X	X	X		
Power-down								X(3)	X				X		X
Power-save					X		X(2)	X(3)	X	X			X		X
Standby(1)						X		X(3)	X				X		X
Extended Standby					X(2)	X	X(2)	X(3)	X	X			X		X

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

Figura 1.3.2.1.G. Modes de funcionament del Atmega328p

La intenció és deixar esperant el microcontrolador en manera "Power Down" la major part del temps, que és el que menys dispositius té funcionant, i en conseqüència menys consum ens produirà.

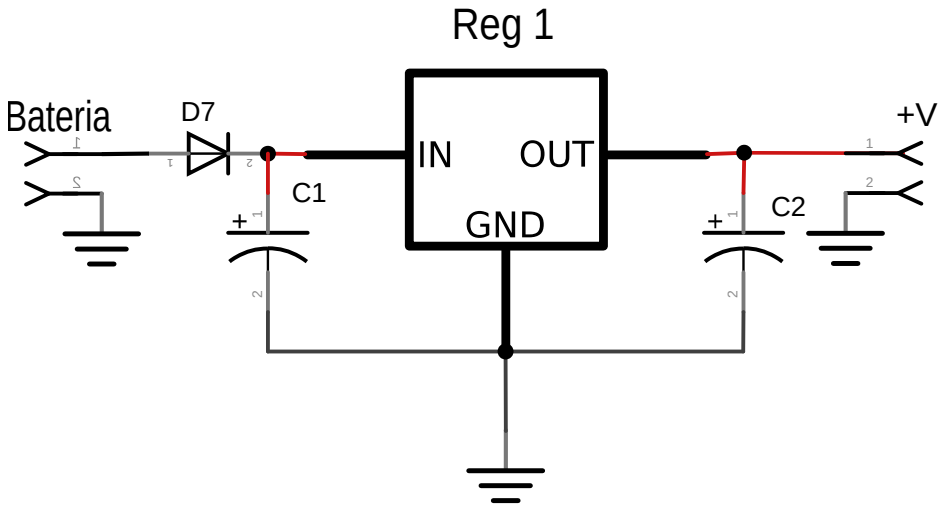
1.3.2.2 Font d'alimentació.

La font d'alimentació està dissenyada per tractar de fer un consum raonable d'energia. És per això, que es fa servir 2 reguladors en lloc d'un, el primer, alimenta de manera permanent el microcontrolador i el RTC sols si és necessari, aconseguint una consum sense càrrega realment baix 2 µA. Per altra banda, el segon regulador té la particularitat de poder subministrar fins a 3A de corrent, que teòricament serà més que suficient per a cobrir la demanda del mòdul GSM.

✗ Font d'alimentació permanent.

Està basada en un MCP1702-33+, que bàsicament és un regulador dels denominats LDO (*Low Quiescent Current*) que tenen la particularitat de tindre una mínima corrent de manteniment. És un regulador que admet tensions d'entrada des d'uns 4v fins a 13,2 V.

Com els programadors de reg a voltes els manipulen persones que no sempre respecten la polaritat, s'ha col·locat a l'entrada un díode de protecció per evitar que un error en la connexió puga fer malbé cap component.



1.3.2.2.1 Font d'alimentació permanent.

El MCP1702-33+ permet subministrar una corrent de fins a 250mA, mes que suficient per a alimentar el Atmega328P, el RTC i el mòdul d'ultrasons (alimentats per mitjà d'una eixida del Atmega328P, quan cal únicament).

En el cas del díode D7, s'ha triat un 1N5822G, ja que és un component compartit entre els dos circuits reguladors i hi ha suportar la corrent màxima dels dos reguladors. En tot cas no aplegarà als 3A que suporta aquest model.

Els condensadors C1 i C2 els han triat de 22µF, cal ficar-los que tinguen una baixa ESR.

✗ Font d'alimentació de potència.

Aquesta font d'alimentació, entra en funcionament, molt ocasionalment, quan activem l'eixida D5 del atmega328p; d'aquesta manera s'estalvia tindre el regulador, el mòdul GSM i el divisor de tensió per mesurar les bateries permanentment connectats. Aquesta part del circuit esta basada en un regulador LM2596-50. L1 s'ha col·locat de 100µH i el D8 s'ha triat un 1N5822G.

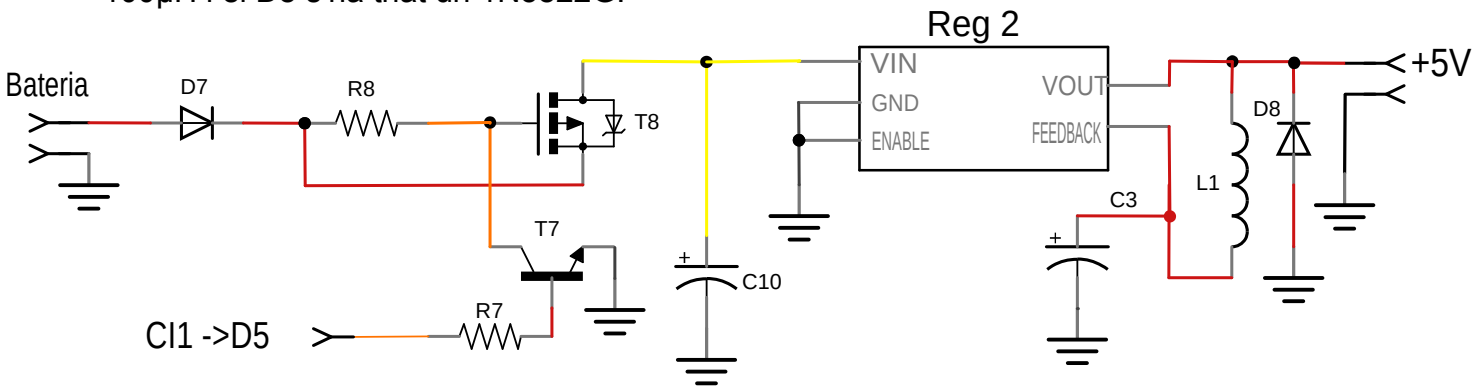


Figura 1.3.2.2.2 Font d'alimentació de potència.

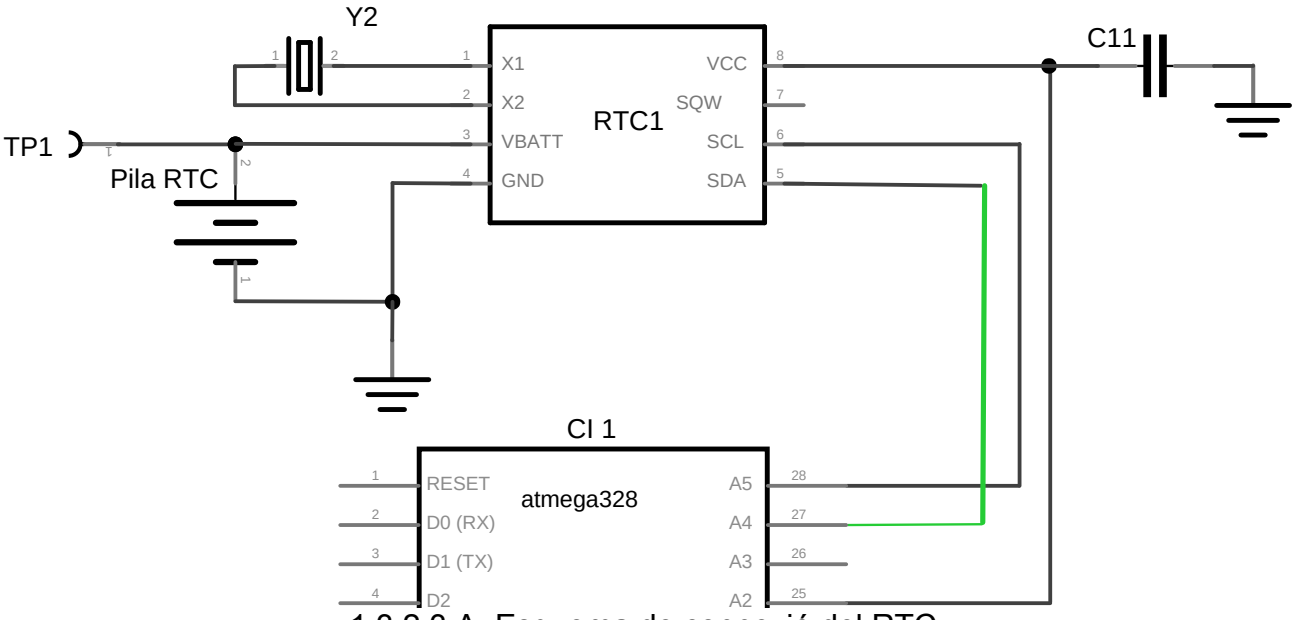
Pel que fa als condensadors, el C10 s'ha ficat de 100µF/25v, el C3 s'ha col·locat de 1000µF/6.3v perquè siga suficient per a absorbir els pics de consum del mòdul GSM.

Per activar el regulador es fa amb un MOSSET de potencia T8 IRFU9024N, que esta controlat amb una etapa de prèvia composta per un transistor T7 NIN BC547 i un parell de resistències.

La R8 (Rs) s'ha ficat de 100k i la R7 (Rb) s'ha triat de 10kohm, perquè limite el corrent d'eixida de l'Arduino.

1.3.2.3 Rellotge en temps real.

Bàsicament, un rellotge en temps real, és un circuit que ens permet guardar i recuperar l'hora actual quan ho necessitem. Hi ha diferents models, s'ha triat el model DS1338-33+, ja que permet treballar a 3,3v, que és la tensió amb què alimenta permanentment l'Atmega328p i admet una bateria auxiliar per seguir funcionat a un molt baix consum, quan no s'alimenta per la bateria principal. A més és compatible amb el DS1307, un circuit molt utilitzat pels usuaris d'Arduino, el que permet fet servir una llibrera i codi dissenyat pel DS1307.



1.3.2.3.A. Esquema de connexió del RTC.

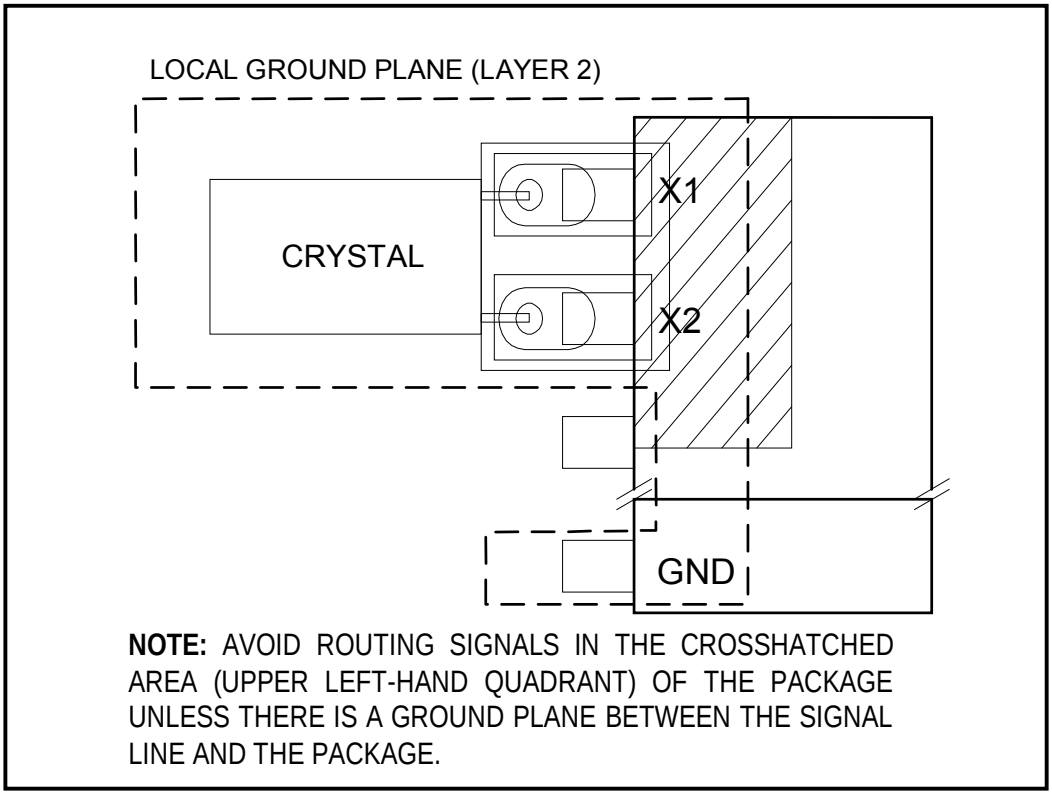
La interconnexió entre el RTC i l'Atmega328p es fa pel bus de comunicacions I2C interconnectant els Pins SCL (Rellotge del bus I2C) del microcontrolador (Màster), amb el pin SCL del DS1338 (Slave).

Per a la comunicació de dades es fa servir el pin SDA en els dos circuits, pin també interconnectat entre ells.

S'alimenta des del A2 del Arduino, estalviant uns 80µA, quan no es fa servir, i passant a consumir 800 nA de la bateria auxiliar, que permetrà una duració de 32 anys, una bateria de 225 mAh. És probable que aquest dispositiu tinga una vida útil inferior a eixos 32 anys, que teòricament tardaria en esgotar-se la bateria.

En aquest tipus de circuits integrats és fonamental a l'hora del seu muntatge tindre certes precaucions amb el cristall de quars per evitar interferències/capacitats que alteren el bon funcionament del rellotge. El cristall s'ha de situar el més pròxim possible al propi integrat per evitar que les pistes puguen acabar influint en el seu bon funcionament.

El mateix fabricant, en les fulles de característiques, aconsella col·locar un pla de massa, en la capa superior de la PCB, per tractar de minimitzar el risc de les interferències.



1.3.2.3.B. Pla de massa suggerit pel fabricant, per evitar interferències.

En el cas concret d'este RTC, si detecta un senyal incorrecta del cristall, seguirà funcionant, però ho farà a una velocitat 10 voltes més a poc a poc, el que pot ser prou perjudicial si no és detectat a temps.

1.3.2.4 Mòdul GSM.

Avui en dia, la millor manera de connectar els microcontroladors instal·lats amb llocs remots (però amb cobertura telefònica mòbil), és fer-ho per mig de les xarxes mòbils existents, amb un SIM i una tarifa de dades "màquina a màquina".

Hi ha mòduls que ofereixen cobertura 2G, 3G o fins i tot 4G.

Pel que fa al seu funcionament, estos dispositius fan servir un protocol molt semblant. En este cas s'ha triat un model basat amb el mòdul SIM800L.

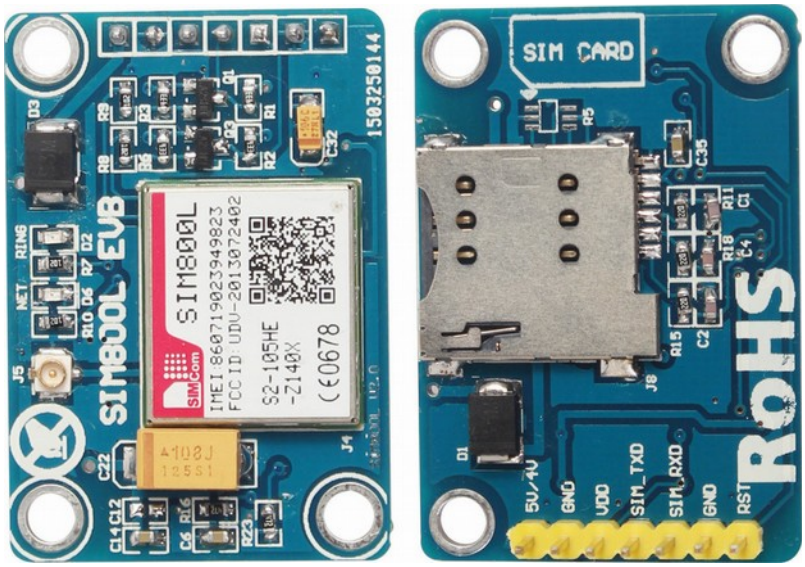


Figura 1.3.2.4.A Mòdul SIM.

No obstant això, queda clar que aquest mòdul GSM, és "temporal" i tard o d'hora s'haurà de substituir quan el tipus de xarxa actual deixe d'estar operativa.

De tota manera eixe canvi és "mínim" a efectes pràctics, ja que tots els mòduls a nivell de hardware necessiten el mateix (tant si són 2g, 3G com 4G), GND, alimentació, TX i RX. Això si, és molt probable que els pins no estiguen al mateix lloc en tots els models que hi ha al mercat, per la qual cosa segurament en el moment del canvi caldrà col·locar una PCB adaptadora, entre el nou mòdul de comunicacions i la placa del nostre mòdul de reg.

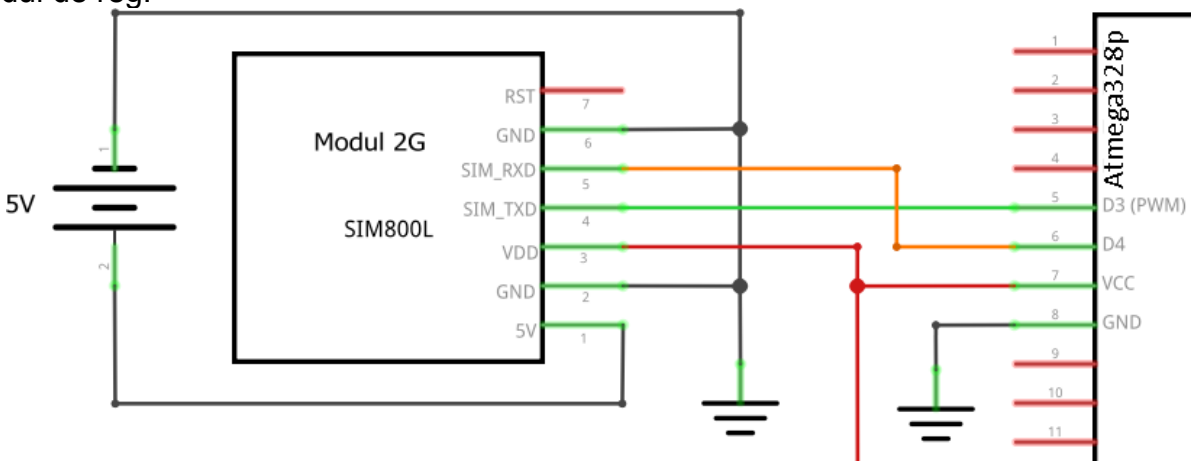


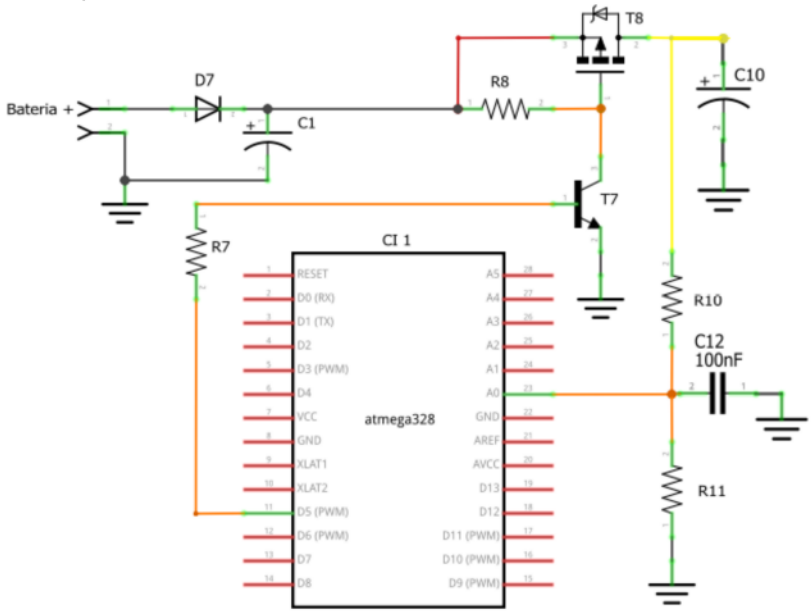
Figura 1.3.2.4.B. Interconnexió del mòdul GSM amb el Atmega328p.

Com es pot observar a la figura1.3.2.4.B, el Mòdul GSM treballa amb una tensió de 5V i està interconnectat amb l'Atmega 328 amb els pins D3 i D4, que en aquest cas funcionaran com si d'un port sèrie es tractara, gràcies a la llibreria "SoftwareSerial", que permet fer servir 2 pins, com si del port sèrie es tractarà, el que permetrà deixar el port sèrie de l'Atmega328p disponible per si cal programar de nou el mateix microcontrolador, sense traure la placa.

1.3.2.5 Mesurar Bateries.

Es fa necessari poder mesurar l'estat de la bateria, per poder previndre una fallada del sistema per manca d'alimentació. Per mesurar el nivell de tensió de la bateria, fa servir el convertidor analògic digital del mateix microcontrolador i un simple divisor de tensió.

Per evitar que aquest divisor de tensió estiga permanentment connectat i en conseqüència consumint energia de la bateria, l'alimenta simultàniament amb el mòdul GSM, que es connecta com a molt durant uns pocs segons al llarg del dia. És per això que no mesura la bateria directament sinó per mitjà del díode d'entrada D7 i el transistor de potencial T8. El que fa que el nivell mesurat, no siga exactament la tensió d'alimentació, així i tot, és suficient.



1.3.2.5. Mesurador de bateria

Col·locant R10 de 1M i R11 de 100k. Per la qual cosa el senyal que aplega al ADC és 1/11 part de la tensió que li arriba el divisor de tensió. C12 tracta de minimitzar el soroll.

El valor de tensió d'eixida la calcula el fitxer PHP, basant-se en l'expressió :

$$V = \left(\frac{Vcc * Factor\_del\_divisor}{1024} \right) * Valorllegit = \left(\frac{3.3 * 11}{1024} \right) * Valorllegit$$

Expressió 1.3.2.5. Valor de tensió equivalent, partint del valor digital.

1.3.2.6 Mesurador de nivell del depòsit.

Per mesurar el nivell de líquids del depòsit s'ha triat un sensor d'ultrasons. Per un costat aquests sensors aporten el seu baix cost i per altre, el fet de no estar en contacte

amb el líquid, fa pensar que la seua vida útil serà suficientment gran com per no haver de preocupar-se en molt de temps.

S'ha triat en concret el model de Adafruit 4007 que té un rang de funcionament que va d'uns 2 cm fins a 450 cm, però obté els millors resultats en el rang de 10cm-250cm, així que en aquest cas treballarà la major part del temps en este rang.

Aquests sensors són ràpids i bastant fàcils d'utilitzar. A diferència de la sèrie HC-SR04 (el més habitual), el "Adafruit 4007" pot alimentar-se a tensions 3V. i és compatible amb el HC-SR04.

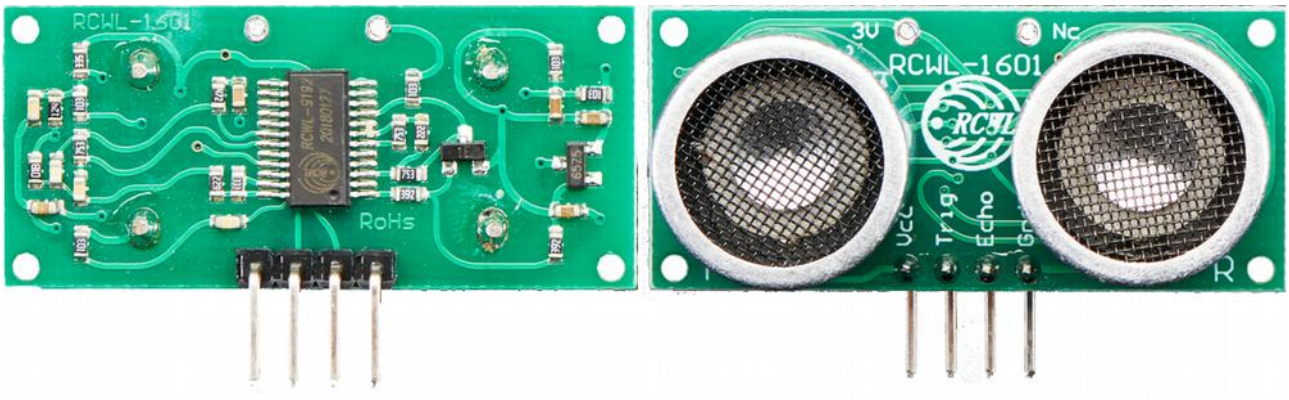


Figura 1.3.2.6.A. Sensor d'Ultrasons

Té un consum de 2.2 mA, per la qual cosa es pot alimentar directament des de l'eixida D5 del Arduino (la mateixa que activa la font d'alimentació de potència).

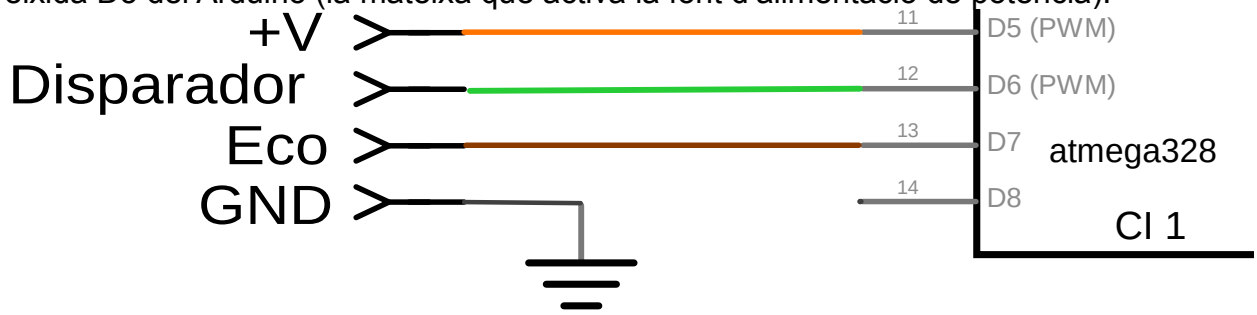


Figura 1.3.2.6.B. Connexió del mòdul d'ultrasons amb el IC 1

La velocitat del so a 20ºC és de 340 m/s , si bé, cada grau centrigrau, la velocitat del so augmenta 0,6m. No obstant això en aquest cas, s'ha assumit l'errada que es produïx per la temperatura, ja que el nivell del dipòsit no és una mesura "vital" pel funcionament del sistema, tan sols aporta informació a l'usuari, que ha de valorar-la de manera àmplia i no basant-se en la última mesura únicament, sinó observant la tendència.

El mòdul d'ultrasons treballa a 40khz. El funcionament del sensor d'ultrasons Adafruit 4007 :

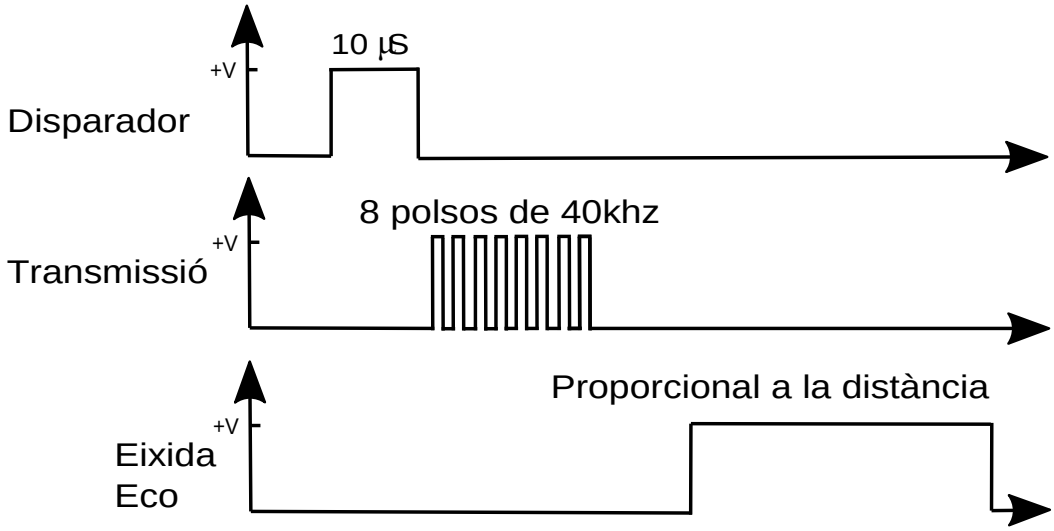


Figura 1.3.2.6.C. Senyals entrada i l'eixida sensor d'ultrasons.

Primer s'envia un pols de 10uS pel pin del disparador. Tot seguit, el sensor enviarà 8 polsos a l'altaveu d'ultrasons i quant rep el rebot, fica la pota d'eco a 0,

Per calcular la distància es calcula fent servir, la velocitat del so i tenint en compte que la velocitat a l'objecte es recorre 2 voltes , d'anada i de tornada.

$$Distancia = temps * (\frac{340}{2}) = temps\,(uS) * 0.170 = mm$$

Expressió.1.3.2.6.C Càlcul de la distància partint del temps

1.3.2.7 Control de les Electrovàlvules.

Imatge. 1.3.2.7 Electrovàlvula burkert.

En este projecte les electrovàlvules que es volen controlar són "biestables de polsos". A efectes elèctrics cada electrovàlvula porta 2 bobines amb un terminal comú (en aquest cas el positiu), que al rebre un pols en una de les bobines canvia l'estat de l'electrovàlvula. Una de les bobines engega el pas d'aigua en rebre un pols i

l'electrovàlvula es manté en eixe estat fins que es rep un altre pols a l'altra bobina, que és l'encarregada d'aturar el pas d'aigua.

A part del circuit de commutació, es col·loca un díode "ràpid" en antiparal·lel en cada una de les bobines de l'electrovàlvula per evitar tensions inverses durant la descàrrega de les bobines.

El circuit de potencia d'una de les tres electrovàlvules quedarà així:

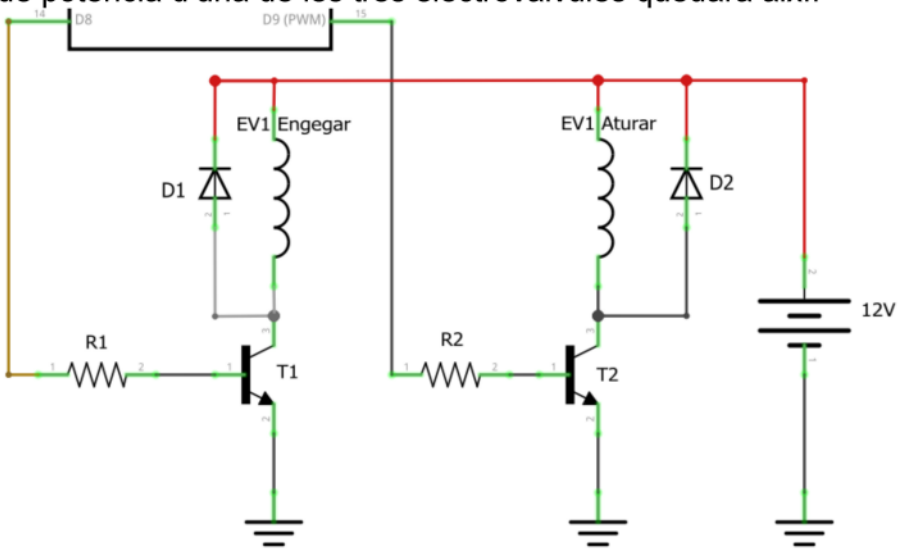


Figura 1.3.2.7.A. Control de les electrovàlvules.

Per triar el model de transistor, s'ha calculat primer quina corrent deu suportar, sabem per informació de un dels fabricants, que la bobina per funcionar correctament rep una potencia de 5W a 12v, pel que la corrent serà:

$$I_{Bobina} = \frac{P_{Bobina}}{V_{Bobina}} = \frac{5}{12} = 0{,}417A$$

Expressió 1.3.2.7.A. Corrent a la bobina.

Tenint en compte que la corrent màxima que volem que suporte el microcontrolador és de 20mA, caldrà un transistor amb una beta mínima.

$$ß_{\text{mínima}} = \frac{I_{Bobina}}{I_{Arduino\_maxima}} = \frac{0{,}417}{0{,}02} = 20{,}85$$

Expressió 1.3.2.7.B. ß mínima.

Amb tot això s'ha triat el transistor BD135G que permet treballar amb tensions i corrents superiors a les que fem servir, alhora que té una beta mínima que és superior a 20,85.

Càlculs de les resistències màxima de base:

$$I_{base} = \frac{I_{Bobina}}{ß} = \frac{0,417}{25} = 0,0167A$$

Expressió 1.3.2.7.C. Corrent de base.

$$R_{Base\_màxima} < \frac{V_{Arduino} - V_{be}}{I_b} = \frac{3,3 - 0,7}{0,0167} = 155,7\ Ohms$$

Expressió 1.3.2.7.D. Resistència de base màxima.

No obstant això, es vol que l'Arduino no entregue més de 20mA per la qual cosa la resistència mínima seria.

$$R_{Base} = \frac{V_{Arduino} - V_{be}}{I_{arduino\ eixida\ màxima}} = \frac{3,3 - 0,7}{0,02} = 130\ Ohms$$

Expressió 1.3.2.7.E. Resistència de base mínima.

Necessita una resistència d'entre 155 a 130 ohms per la qual cosa s'han col·locat unes resistències de base de 130 ohms.

Els díodes s'han col·locat del model 1N5819G que són prou ràpids i potents per absorbir la descàrrega de la bobina. Per comprovar-ho es connecta el prototip a un solenoide, semblants als de la instal·lació i s'ha mesurat el comportament a l'oscil·loscopi.

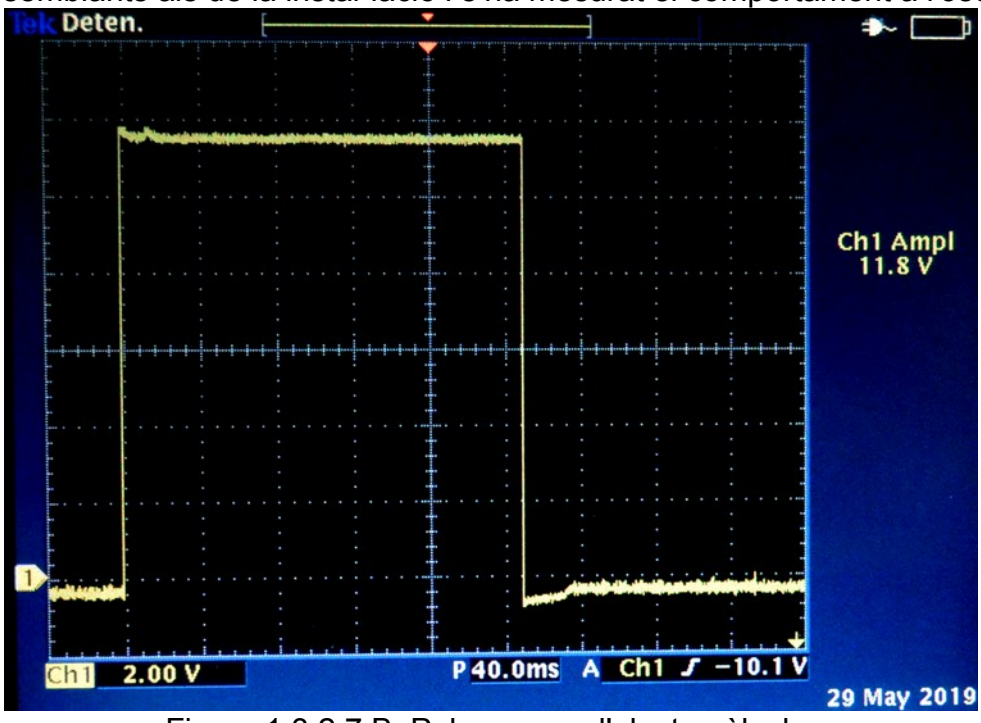
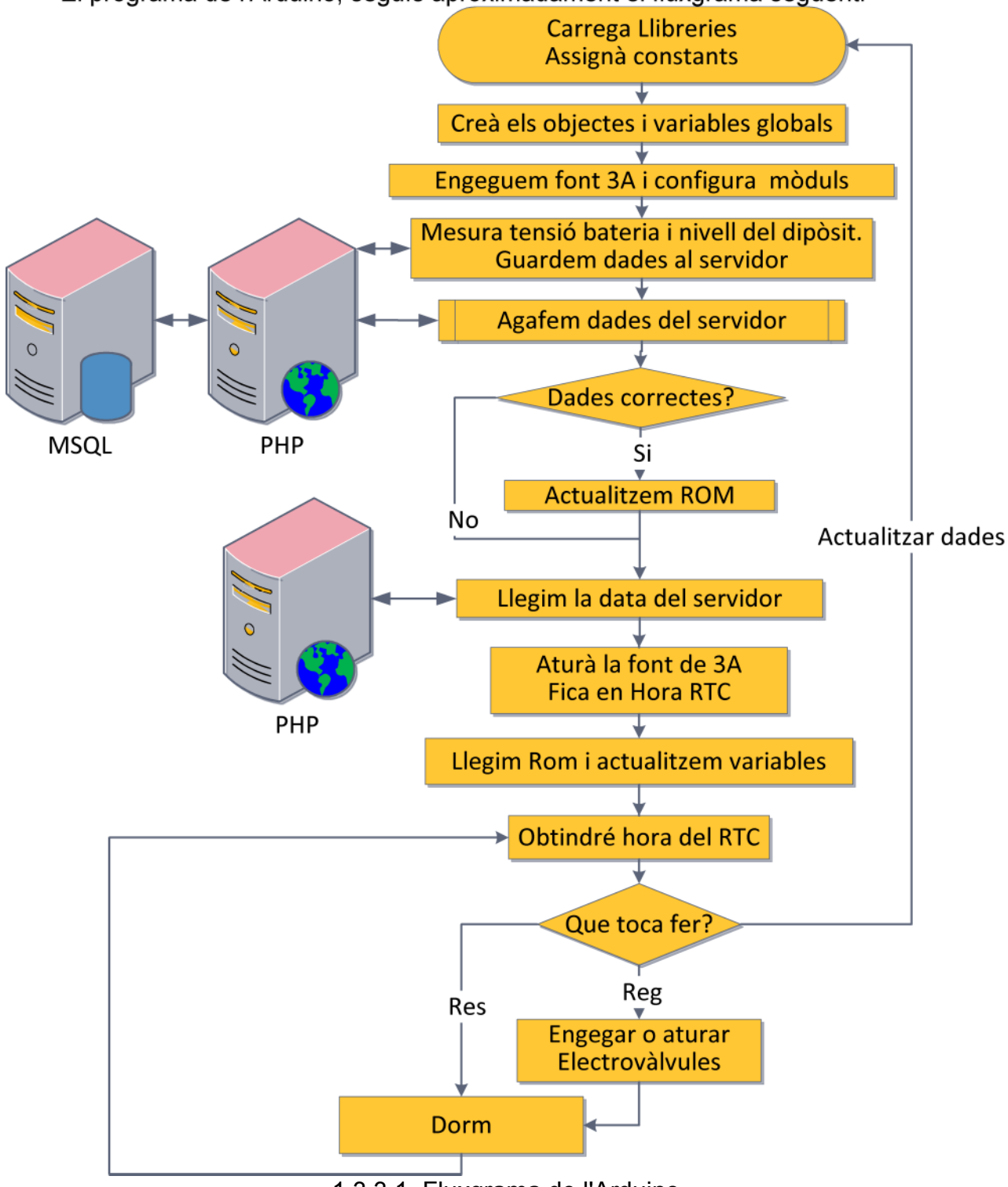


Figura 1.3.2.7.B. Pols que rep l'electrovàlvula.

Com es pot observar en la figura 1.3.2.7.B. s'aconseguix un pols adequat en la bobina, a l'hora que s'elimina suficientment la descàrrega de la mateixa amb el díode corresponent, evitant fer malbé el transistor.

1.3.3 Programari Arduino

El programa de l'Arduino, seguís aproximadament el fluxgrama següent.



1.3.3.1. Fluxgrama de l'Arduino

Per dur a terme aquest projecte i per facilitar la programació s'ha fet servir unes llibreries:

1.3.3.1 La llibreria "PowerPoint".

La llibreria "LowPower" facilita deixar el Atmega328p en un consum mínim.
D'aquesta llibreria únicament es fa servir el mètode:

- **LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);**

Amb això, i el circuit de la figura 1.3.2.1B, s'aconseguís un consum de 4.2 µA, molt inferior als 4 mA que consumiria en estat Idle i molt inferior al consum d'un Arduino Uno funcionant amb tots els seus components.

1.3.3.2 La llibreria "Sim".

S'ha creat per manipular més fàcilment les comunicacions amb l'exterior per mitjà del mòdul GSM. Té 3 mètodes públics:

- **Sim(uint8_t RX, uint8_t TX)**

Permet definir en quins pins està connectat el mòdul GSM

- **void iniciar();**

Permet establir la connexió a la xarxa

- **String obri(String pagina)**

Permet consultar pàgines web i guarda el seu contingut en un String

1.3.3.3 La Llibreria "deposit".

La funció d'aquesta llibreria és facilitar la manipulació del sensor d'ultrasons des del programa principal, tan sols es fan servir dos mètodes:

- **Deposit(int EcoPota, int DesencadenadorPota);**

Este mètode permet definir en quins pins està connectat el mòdul d'ultrasons.

- **String Mesurar_med(int EcoPota, int DesencadenadorPota, int numero_mesures);**

Permet consultar el nivell del depòsit, fent diverses mesures i guarda el valor d'enmig String

1.3.3.4 La llibreria "rellotge".

S'ha creat per manipular més fàcilment el RTC des del programa principal, se bassa principalment en la llibreria "RTClib", però d'una manera més simple i adaptada a les necessitats completes. Té únicament 2 mètodes públics.

✗ **void ficar_en_hora(String tmp, int RTC_Vcc);**

Este mètode permet ficar en hora el RTC a partir d'una hora unix obtinguda prèviament des del servidor web

✗ **word temps_actual(int RTC_Vcc);**

Torna l'hora actual en el format que interessa, “dhhmm” en una variable de tipus Word

d = Ens diu quin dia de la setmana estem del 0 a 6, són els seus possibles valors .
hh = correspon a les hores
mm = correspon als minuts.

1.3.3.5 La llibreria “regar”.

Bàsicament permet manipular les electrovàlvules d'una manera més fàcil.

✗ **void Configurar (int EV1_Engegar, int EV1_Aturar, int EV2_Engegar, int EV2_Aturar, int EV3_Engegar, int EV3_Aturar);**

S’han configurat tots els pins de control de les electrovàlvules com eixides.

✗ **void ara(int Eixida, int Temps);**

Posa a 1 durant un temps determinat, una de les eixides per poder canviar d'estat l'electrovàlvula corresponent , depenent del model d'electrovàlvula caldrà un valor de temps diferent.

1.3.3.6 La llibreria Rom

Fa més fàcil la manipulació de les dades de la EEPROM, té 2 mètodes disponibles.

✗ **Guarda_en_Rom(String text, int adresa_1);**

Guarda en rom, una variable de tipus word.

✗ **Rom_LligWord(int adresa_3);**

Llegeix 2 direccions de ROM. El seu valor compte realment d'una variable word

1.3.4 Programari del servidor web

1.3.4.1 Interfície de web d’usuari.

Bàsicament consta de 3 pàgines principals, una ens facilita informació i les altres dues que permeten modificar la programació de reg en la base de dades.

✗ **La pàgina principal “index.php”:**

Per una part facilita informació sobre l'última vegada que sé connecta el programador així com quin programa de reg hi ha guardat en eixe moment a la base de dades. Per altra part, mostra les dades de tensió i nivell de depòsit dels últims 7 dies.

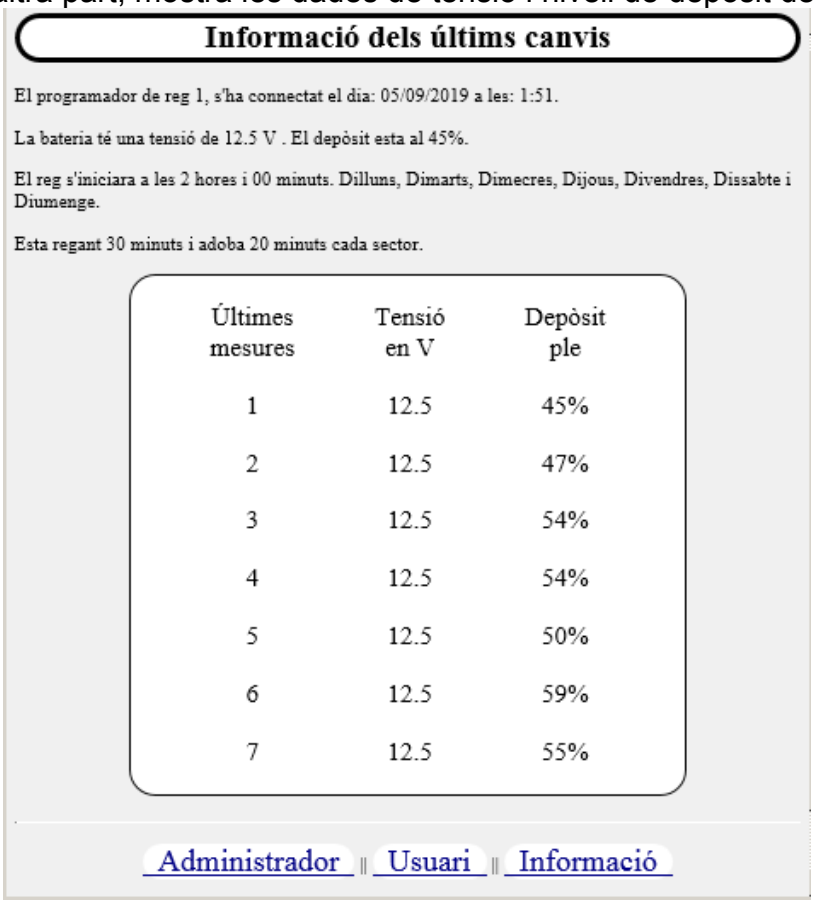


Figura 1.3.4.1.A Pàgina d'informació.

En la part inferior de la pàgina trobem el menú de navegació que ens permet accedir a les pàgines de programació de reg, bé siga com a administrador o usuari.

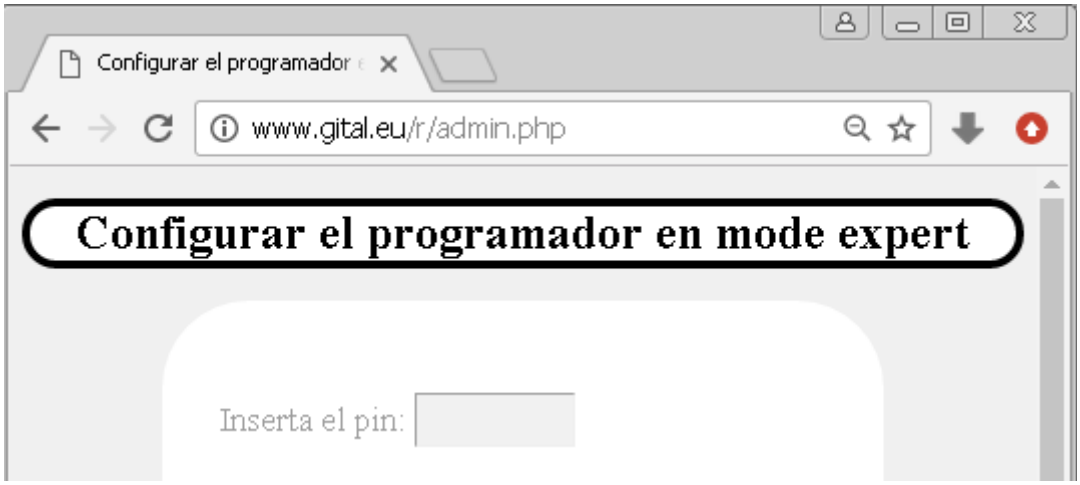
✗ La pàgina "admin.php":

Permet indicar quins dies de la setmana i a quines hores es deu activar o desactivar cada electrovàlvula.

Aquesta pàgina compta amb una primera protecció per evitar que algun robot puga entrar a la web i modificar les dades de la base de dades per accident.

1.3.4.1.B. Control d'usuari humà.

Aquest formulari compta a més amb una segona protecció, un PIN que sols ha de conéixer l'usuari autoritzat per modificar la programació horària.



1.3.4.1.C. Control pin.

També compta el formulari amb un control complet de tots els paràmetres de programació de la base de dades. Podent modificar dia, hora i minut en què s'engega cada electrovàlvula.

Controlador de reg per degoteig

Figura 1.3.4.1.D. Formulari expert.

Aquest formulari, si el PIN és correcte, acabat guardant la informació a la base de dades per mitjà del fitxer “previ.php” que és el que s'encarrega de guardar la informació a la base de dades per mitjançant de la sentència SQL corresponent, els valors de les variables php:

INSERT INTO Taula_programacio (SRV, EV1, AV1, EV2, AV2, E31, A31, E32, A32, Seguretat) VALUES ('$SRV','$EV1','$AV1','$EV2','$AV2','$E31','$A31','$E32','$A32', $Seguretat)

✗ Per últim la pàgina “usuari.php” :

Conté també un nou formulari, també filtrat prèviament per evitar un accés accidental com l'anterior. Això si, aquest formulari és més simple i probablement és l'únic que utilitzen finalment els usuaris.

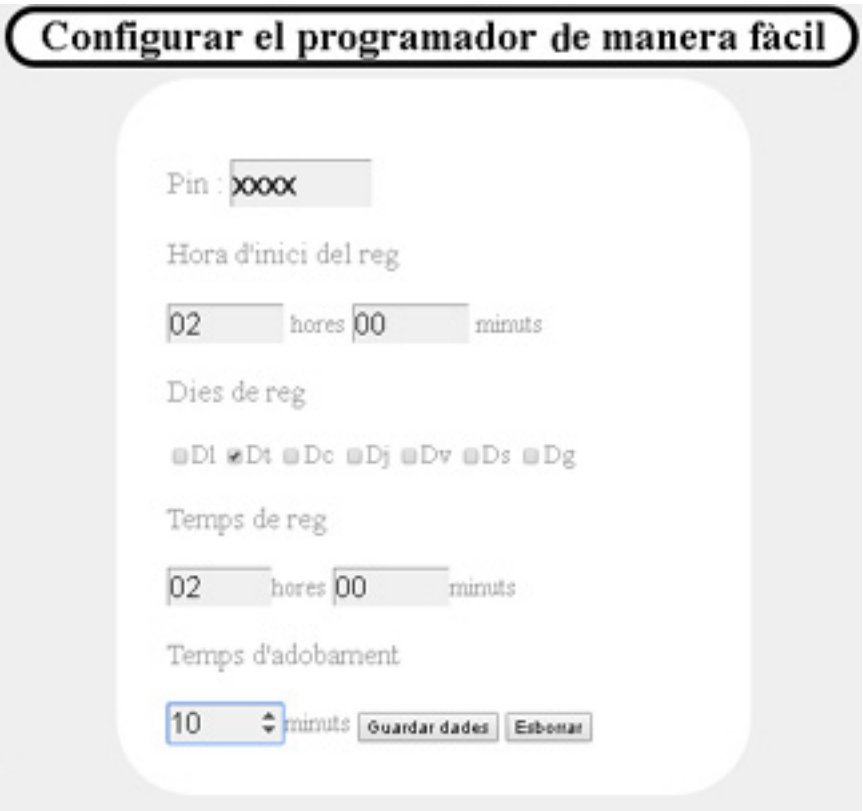


Figura 1.3.4.1.E. Formulari fàcil.

Bàsicament cal indicar-li, hora d'inici, dies que cal regar i duració tant del reg com de l'adobat.

1.3.4.2 Pàgines de comunicació del servidor amb el programador.

Com que l'Arduino té uns recursos limitats, la informació la rep "per parts" en diferents consultes.

La informació que subministra cada fitxer php és inferior al 64 bytes que és la mida màxima de la memòria cau que té configurada de la llibreria "SoftwareSerial" de l'Arduino, que és l'encarregada de permetre la comunicació del mòdul GSM.

Hi ha 7 fitxers de programació horària ("srv.php", "ev1.php", "av1.php", "ev2.php", "av2.php", "e31.php", "a31.php", "e32.php", "a32.php") que consulten a la base de dades diferents valors que necessita el microcontrolador.

✗ El fitxer «srv.php»:

Facilita l'hora, minut i en quins dies de la setmana s'ha de tornar a connectar el controlador de reg per degoteig amb el servidor per actualitzar les dades.

✗ El fitxer «ev1.php»:

Facilita l'hora, minut i dies de la setmana que s'ha d'engegar l'electrovàlvula número 1

✗ El fitxer «av1.php»:

Facilita l'hora, minut i dies de la setmana que s'ha d'aturar l'electrovàlvula número 1

✗ El fitxer «e31.php»:

Facilita l'hora, minut i dies de la setmana que s'ha d'engegar l'electrovàlvula número 3 – Corresponent al programa de reg 1

✗ El fitxer «a31.php»:

Facilita l'hora, minut i dies de la setmana que s'ha d'aturar l'electrovàlvula número 3 – Corresponent al programa de reg 1.

✗ El fitxer «ev2.php»:

Facilita l'hora, minut i dies de la setmana que s'ha d'engegar l'electrovàlvula número 2

✗ El fitxer «av2.php»:

Facilita l'hora, minut i dies de la setmana que s'ha d'aturar l'electrovàlvula número 2.

✗ El fitxer «e32.php»:

Facilita l'hora, minut i dies de la setmana que s'ha d'engegar l'electrovàlvula número 3 – Corresponent al programa de reg 2.

✗ El fitxer «a32.php»:

Facilita l'hora, minut i dies de la setmana que es deu aturar l'electrovàlvula número 3 – Corresponent al programa de reg 2.

Tots els fitxers anteriors mostren el contingut amb la mateixa estructura de dades (obtinguda de la base de dades).

Per exemple el fitxer ev1.php llegeix de la base de dades l'hora, minut i dies de la setmana que s'ha d'engegar l'electrovàlvula 1, podent mostrar la següent informació: "Ev1_0200_0101010_CT_ATZUCAC" ens indica que s'ha engegar el dimarts, el dijous i el dissabte a les 2 de la matinada hora local de la península Ibèrica. Tots els fitxers mostren la mateixa estructura IDD_hhmm_ddddddd_ID_Final, el que seria el ID_final únicament es per assegurar-se que hem llegit fins al final el fitxer i poder fer una "mínima" comprovació de què s'ha fet la lectura.

✗ **El fitxer "ara.php":**

Retorna el temps actual en format "unix time" (segons transcorreguts des d'1 de gener de 1970), a més aplica una correcció per ajustar l'hora a la franja horària peninsular. El farà servir l'Arduino per ajustar l'hora del RTC si és necessari.

✗ **El fitxer "a.php":**

És el que permet que el microcontrolador guarde les lectures del mesurador del nivell del depòsit i del divisor de tensió. Per fer-ho des de l'Arduino consultem la URL http://servidor.eu/camí/a.php?v=valor_sensor&d=Temps_deposit&p=xxxx

1.3.5 Base de dades.

En aquest cas la base de dades, corre en vaig un motor MYSQL 5.5 , i l'administració es fa mitjançant el gestor d'administració de "phpmyAdmin".

La creació es fa seguint uns pocs passos que varien lleugerament depenent del servidor que tenim.

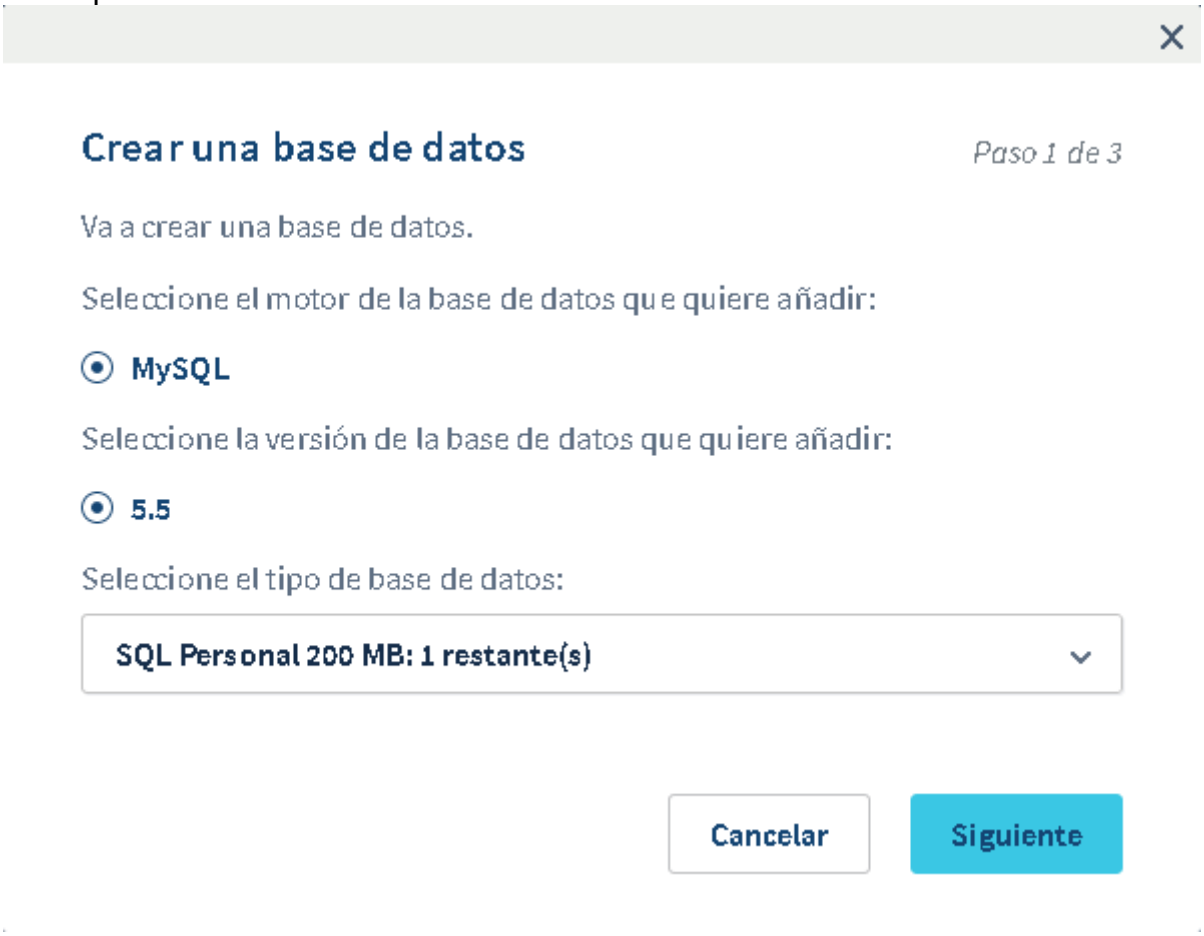


Figura 1.3.5.1 Creació de la base de dades, pas 1.

En aquest cas, 200Mb és més que suficient, ja que després de molts amb anys d'ús només aplegaria a ocupar uns pocs mb.

S'assigna un nom a la base de dades i contrasenya que cal recordar per després poder connectar.

Figura 1.3.5.2 Creació de la base de dades, pas 2.

Finalment es confirma la creació de la base de dades:

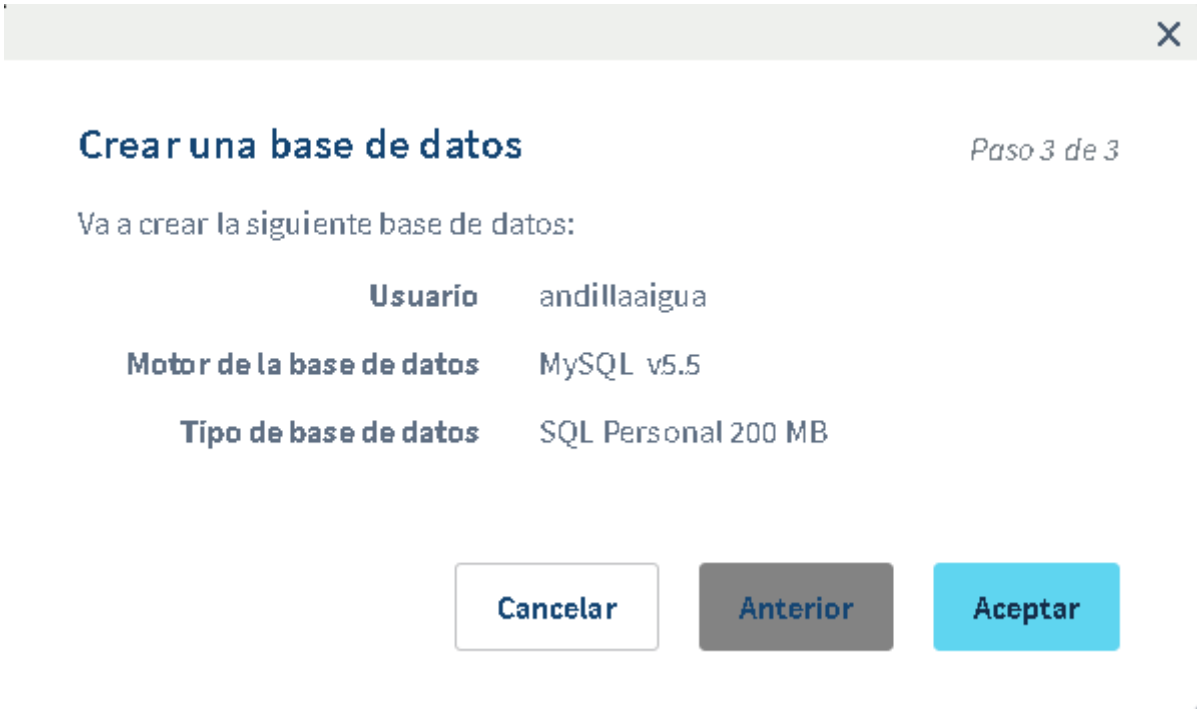


Figura 1.3.5.3 Creació de la base de dades, pas 3 de 3.

Ara ja es pot connectar a la base de dades i crear un parell de taules que fan falta, per cada controlador de reg. En una es guarda la informació generada per l'usuari, i en l'altra, la informació que es rep del controlador de reg.

La “Taula_de_programacio” contindrà la informació que l'usuari guarda perquè el programador puga llegir-la: Consta de dues columnes de control la “ID”, que es va auto incrementant i la de “Seguretat” que reservarem per guardar dades addicionals. La resta de cel·les contindran els dies de la setmana, hores i minuts en què cal fer alguna acció.

Controlador de reg per degoteig

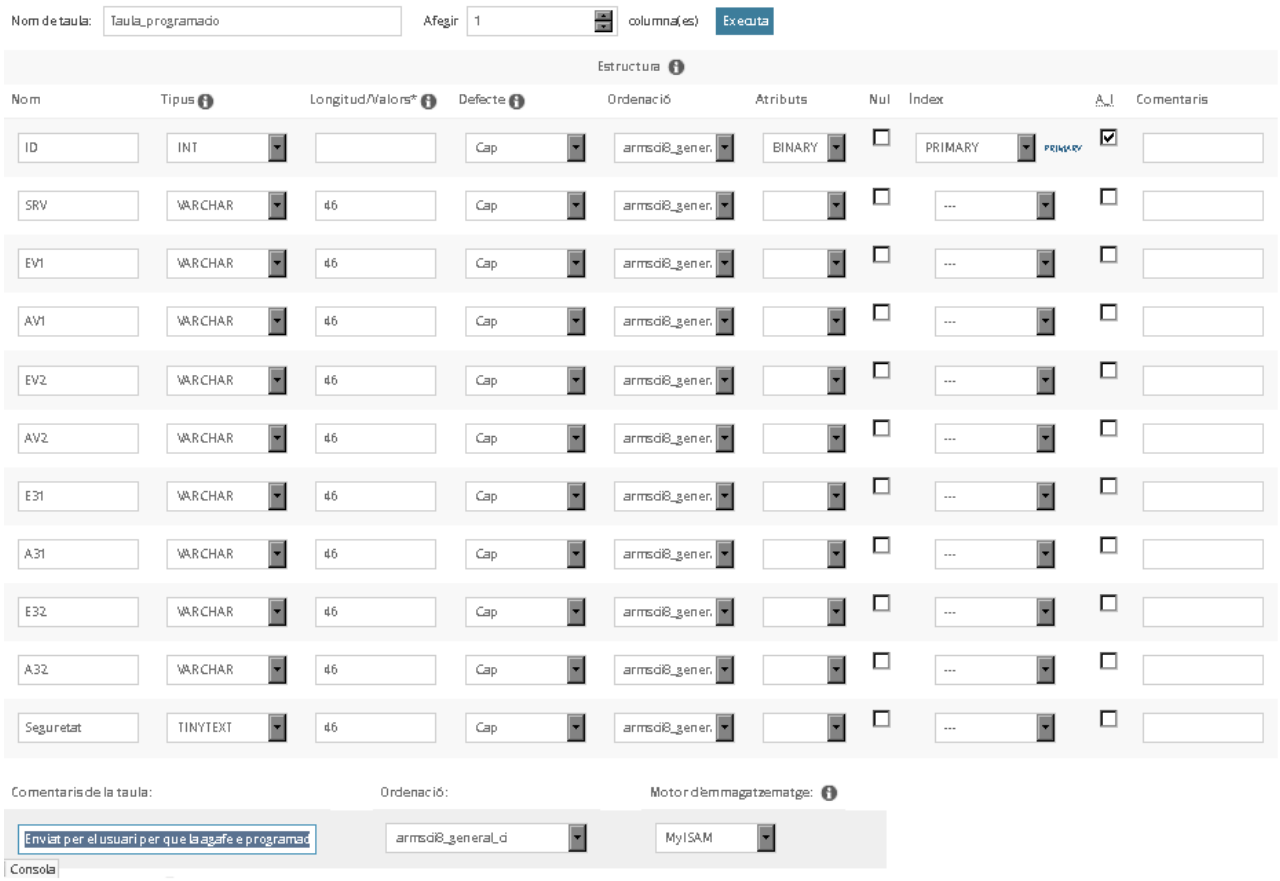


Figura 1.3.5.4 Creació de taula de programació.

A més s'ha creat una segona Taula, "Taula_Arduino" que conté la informació enviada des del controlador de reg. Conté igualment el ID i Seguretat , per motius semblants a més del camp "Bateria" i del camp "deposit" en la que es guarda el nivell del depòsit.

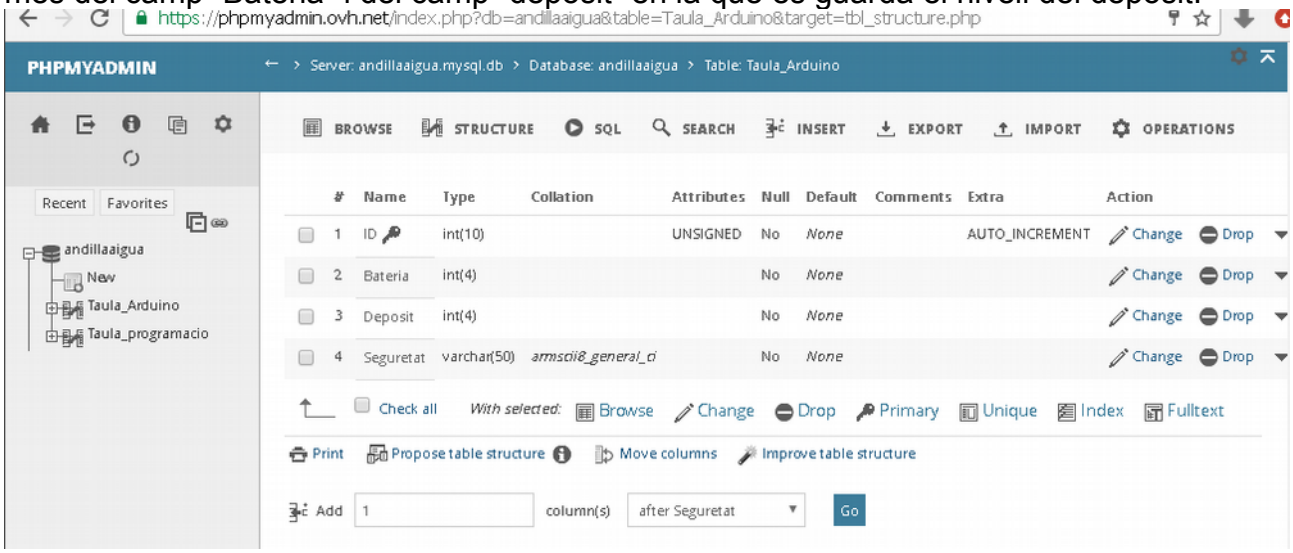


Figura 1.3.5.5 Taula d'Arduino.

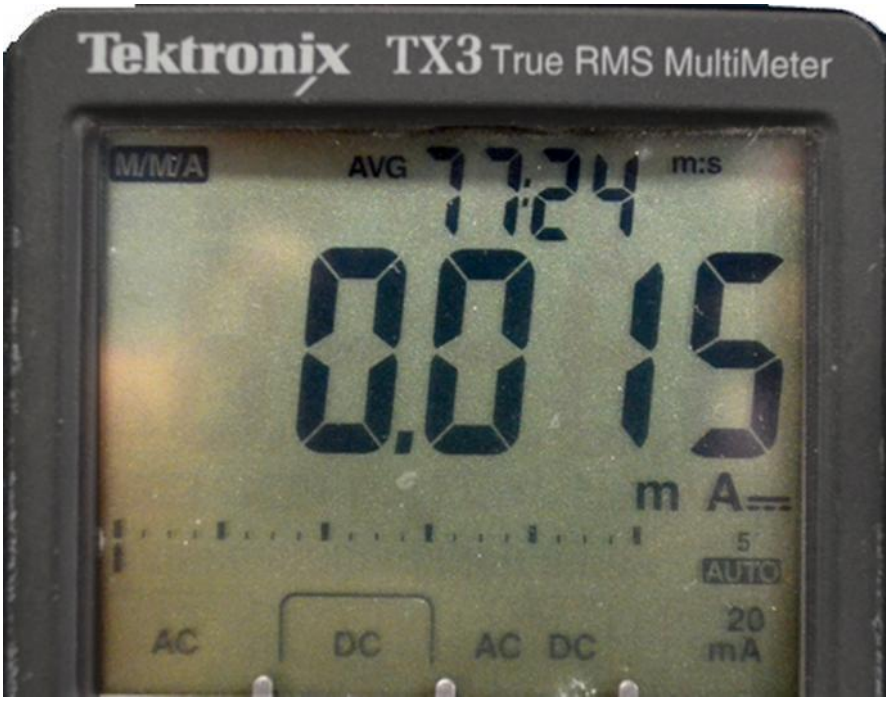
1.3.6 Consums

Aquest programador de reg, té diversos consum depenent del que està fent. Quan s'activen les electrovàlvules aplega a consumir 0,42A, però ho fa en un període molt curt de temps 0,2 segons cada volta (això pot variar depenent el model concret de electrovàlvula).

Quan té el mòdul GSM connectat, té un consum mitjà de 36.5mA, això ho fa durant un curt període de temps també, 3’ 26” una volta al dia com a molt.

Imatge 1.3.6.1 Consum mitjà en el moment d'engegada amb el mòdul GSM actiu

Per últim, la major part del temps el microcontrolador està “dormint” i el RTC sols passa uns instants alimentat, el temps just per poder facilitar l'hora. Aquest consum varia depenent de la tensió de la bateria i la temperatura, no obstant això en el pitjor dels casos el seu consum mitjà no supera els 16 µA



Imatge 1.3.6.2 Consum mitjà al Bucle

	Corrent mitja	Temps total diari	mAh (equivalent en un dia)
Bucle	16µA	24 Hores	0,384
Connexions amb el servidor	36.5mA	206 Segons	2,089
Activació d'electrovàlvules	0,42 A	1,68 Segons	0,196
Total			2,669

Tabla 1.3.6. Resum de consums, en el pitjor dels casos.

En ple estiu, regant 1 volta al dia, cada sector del camp, adobant els dos sectors tots els dies i connectant-se al servidor 1 volta al dia (el pitjor del casos possibles) farà un consum diari total que equivaldria a consumir durant 1 hora al dia una corrent 2,669 mA, l'equivalent de 31 dies 0,083 Ah. El que equival a un consum de 0, 974 Ah el consum de tot un any.

En el cas concret del camp on s'han fet les proves la bateria instal·lada és de tipus AGM de 12v/7ah, aquest tipus de bateries tenen un nivell d'auto-descàrrega aproximat del 2% mensual, per la qual cosa la bateria es descarregaria ella per si soles en un mes 0,14 Ah. Més que ho farà pel consum d'aquest circuit durant el mateix temps 0,083Ah. Per tot això amb una bateria d'este tipus si no es disposa d'algun sistema de recàrrega, s'haurà de recarregar una volta a l'any, com a mesura de precaució.

1.4 Conclusions.

S'ha aconseguit baixar el consum del circuit per baix del 16μA durant la major part del temps.

El sistema registra la tensió de la bateria el que permet tindre un mínim de control sobre l'estat de la mateixa.

El sistema és capaç de registrar cada dia una lectura del depòsit, així que tenint una visió de l'històric permet detectar si hi ha un problema, de pressió.

El sistema, pot comprovar l'última volta que el programador es va connectar, el que permetria saber si el dispositiu està funcionant, o si hem patit un problema d'alimentació o un robatori.

En ser un dispositiu que "depén" d'un servidor web determinat, el fa inservible per als lladres, si no són capaços de reprogramar-lo.

Com futures millores, estaria bé afegir com a complement algun equip amb sensors de saó de la terra, per conéixer el nivell en diferents punts del camp.

1.5 Codi font adjunt.

1.5.1 Codi font del Atmega328P.

1.5.1.1 Codi font de “Programa.ino".

```
const int Temps_Engegat = 200; //mil·lisegons que cal alimentar electrovàlvula .
const byte Id_Maquina = 1 ; // Identificació de la maquina al hora de guardar a la web.

#include "LowPower.h"   // Llibreria de vaig consum

#include "bateria.h"    // Llibreria per mesurar la tensió de la bateria .

#include "Sim.h"        // Llibreria que controla el mòdul 2G.

#include "deposit.h"    // Llibreria per mesurar el dipòsit.

#include "rellotge.h"   // Llibreria per fer servir el rellotge en temps real mes fàcilment.

#include "regar.h"      // Llibreria per activar o aturar el reg.

#include "rom.h"        // Llibreria per manipulació de la ROM.

// Assignant pins.

#define Modul_GSM_RX 3

#define Modul_GSM_TX 4

#define Font_3A 5

#define Deposit_Eco 7

#define Deposit_Desencadenador 6

#define EV1_Engegar 8

#define EV1_Aturar 9

#define EV2_Engegar 10

#define EV2_Aturar 11

#define EV3_Engegar 12

#define EV3_Aturar 13

#define RTC_Vcc A2

#define Voltimetre A0

#define dorm(){  for (byte i = 0 ;  i  <  6 ; i++) {   LowPower.powerDown(SLEEP_8S, ADC_OFF,
BOD_OFF); }}
```

```
word Temps_actual ; // Ací guardarem el número de dia de la setmana junt a l'hora, exemple dimarts a les 22:00 seria "12200", diumenge 23:59 seria 62359.

// Assigne objectes.

Rellotge Modul_rellotge (RTC_Vcc) ;

Sim Modul_SIM(Modul_GSM_RX, Modul_GSM_TX) ;

Deposit Modul_Deposit(Deposit_Eco, Deposit_Desencadenador) ;

Regar Modul_EV ; //

Rom Gestio_Rom ;

Bateria Modul_Bateria(Voltimetre);

// Variables de programació de reg i connexió amb el servidor.

word SRV[7] ;

word EV1E[7] ;

word EV1A[7] ;

word EV2E[7] ;

word EV2A[7] ;

word EV3E1[7] ;

word EV3A1[7] ;

word EV3E2[7] ;

word EV3A2[7] ;

void Sincronitzar ()

{

    digitalWrite(Font_3A, true); // Engegue el divisor de tensió, el mòdul 2G i el mesurador del dipòsit.

    delay(2000); // minim 100ms

    String Mesura_Bateria = Modul_Bateria.Mesurar_med(Voltimetre, 5);

    String deposit = Modul_Deposit.Mesurar_med(Deposit_Eco, Deposit_Desencadenador, 21); // Prenem la mida.

    Modul_SIM.iniciar();

    String tmp = "gital.eu/r/a.php?d=" + deposit + "&v=" + Mesura_Bateria + "&p=XXXXXXXX&m=" + String(Id_Maquina) ;

    Serial.println(tmp);
```

```
        tmp = Modul_SIM.obri(tmp); // Guardem a la base de dades la tensió i la distancia al liquid que hem mesurat

        tmp = Modul_SIM.obri("gital.eu/r/srv.php"); // Llegim en quins horaris ens hem de tornar a connectar al servidor.

        Gestio_Rom.Guarda_en_Rom(tmp, 0) ; // Guarde SRV en ROM

        tmp = Modul_SIM.obri("gital.eu/r/ev1.php"); // Llegim en quins horaris cal engegar la electrovàlvula 1

        Gestio_Rom.Guarda_en_Rom(tmp, 20) ; //Guarde EV1E en ROM

        tmp = Modul_SIM.obri("gital.eu/r/av1.php"); // Llegim en quins horaris cal aturar la electrovàlvula 1

        Gestio_Rom.Guarda_en_Rom(tmp, 40) ; //Guarde EV1A en ROM

        tmp = Modul_SIM.obri("gital.eu/r/ev2.php"); // Llegim en quins horaris cal engegar la electrovàlvula 2

        Gestio_Rom.Guarda_en_Rom(tmp, 60) ; //Guarde EV2E en ROM

        tmp = Modul_SIM.obri("gital.eu/r/av2.php"); // Llegim en quins horaris cal aturar la electrovàlvula 2

        Gestio_Rom.Guarda_en_Rom(tmp, 80) ; //Guarde EV2A en ROM

        tmp = Modul_SIM.obri("gital.eu/r/e31.php"); // Llegim en quins horaris cal engegar la electrovàlvula 3  (Programa 1)

        Gestio_Rom.Guarda_en_Rom(tmp, 100) ; //Guarde EV3E1 en ROM

        tmp = Modul_SIM.obri("gital.eu/r/a31.php"); // Llegim en quins horaris cal aturar la electrovàlvula 3 (Programa 1)

        Gestio_Rom.Guarda_en_Rom(tmp, 120) ; //Guarde EV3A1 en ROM

        tmp = Modul_SIM.obri("gital.eu/r/e32.php"); // Llegim en quins horaris cal engegar la electrovàlvula 3  (Programa 2)

        Gestio_Rom.Guarda_en_Rom(tmp, 140) ; //Guarde EV3E2 en ROM

        tmp = Modul_SIM.obri("gital.eu/r/a32.php"); // Llegim en quins horaris cal aturar la electrovàlvula 3 (Programa 2)

        Gestio_Rom.Guarda_en_Rom(tmp, 160) ; //Guarde EV3A2 en ROM

        tmp = Modul_SIM.obri("gital.eu/r/ara.php"); // Llig l'hora del servidor.

        digitalWrite(Font_3A, false); // Ature el mòdul de 2G, el mesurador de bateries i el del dipòsit

        delay(1000);

        Modul_rellotge.ficar_en_hora(tmp, RTC_Vcc); //Ficar en hora si el hora es correcta.
```

```
int direccio = 0 ;

for (int i = 0 ; i < 7; i++)

{

    SRV[i] = Gestio_Rom.Rom_LligWord(direccio + (i * 2)); // Llig de rom les hores a les que ens tenim que connectar al servidor.

}

direccio = 20 ;

for (int i = 0; i < 7 ; i++)

{

    EV1E[i] = Gestio_Rom.Rom_LligWord(direccio + (i * 2)); // Llig de rom els dies i les hores a les quals cal engegar l'electrovàlvula 1.

}

direccio = 40 ;

for (int i = 0; i < 7 ; i++)

{

    EV1A[i] = Gestio_Rom.Rom_LligWord(direccio + (i * 2)); // Llig de rom els dies i les hores a les quals cal aturar l'electrovàlvula 1.

}

direccio = 60 ;

for (int i = 0; i < 7 ; i++)

{

    EV2E[i] = Gestio_Rom.Rom_LligWord(direccio + (i * 2)); // Llig de rom els dies i les hores a les quals cal engegar l'electrovàlvula 2.

}

direccio = 80 ;

for (int i = 0; i < 7 ; i++)

{

    EV2A[i] = Gestio_Rom.Rom_LligWord(direccio + (i * 2)); // Llig de rom els dies i les hores a les quals cal aturar l'electrovàlvula 2

}

direccio = 100 ;
```

```
        for (int i = 0; i < 7 ; i++)
        {
            EV3E1[i] = Gestio_Rom.Rom_LligWord(direccio + (i * 2)); // Llig de rom els dies i les hores a les quals cal engegar l'electrovàlvula 3
        }
        direccio = 120 ;
        for (int i = 0; i < 7 ; i++)
        {
            EV3A1[i] = Gestio_Rom.Rom_LligWord(direccio + (i * 2)); // Llig de rom els dies i les hores a les quals cal aturar l'electrovàlvula 3
        }
        direccio = 140 ;
        for (int i = 0; i < 7 ; i++)
        {
            EV3E2[i] = Gestio_Rom.Rom_LligWord(direccio + (i * 2)); // Llig de rom els dies i les hores a les quals cal engegar l'electrovàlvula 3
        }
        direccio = 160 ;
        for (int i = 0; i < 7 ; i++)
        {
            EV3A2[i] = Gestio_Rom.Rom_LligWord(direccio + (i * 2)); // Llig de rom els dies i les hores a les quals cal aturar l'electrovàlvula 3
        }
    }

    void setup()
    {

        pinMode(Font_3A, OUTPUT);
        Modul_EV.Configurar(EV1_Engegar, EV1_Aturar, EV2_Engegar, EV2_Aturar, EV3_Engegar, EV3_Aturar);
```

```
    Serial.begin(9600);
    Serial.println("Setup");
    Sincronitzar ();
}

void mostrar_v_globals()
{
    for (int i = 0; i < 7 ; i++)
    {
        Serial.print("Horaris del dia");
        Serial.println(i);
        Serial.println (SRV[i]);
        Serial.println (EV1E[i]);
        Serial.println (EV1A[i]);
        Serial.println (EV2E[i]);
        Serial.println (EV2A[i]);
        Serial.println (EV3E1[i]);
        Serial.println (EV3A1[i]);
        Serial.println (EV3E2[i]);
        Serial.println (EV3A2[i]);
        delay(1000);
    }
}

void loop()
{
    Temps_actual = Modul_rellotge.temps_actual(RTC_Vcc); // Agafa l'hora actual amb dia de la setmana  (d = 0 a 6) + hh (00 a 23) + mm (00 a 59)= dhhmm
```

```
        if ((Temps_actual == SRV[0]) || (Temps_actual == SRV[1]) || (Temps_actual == SRV[2]) ||
(Temps_actual == SRV[3]) || (Temps_actual == SRV[4]) || (Temps_actual == SRV[5]) || (Temps_actual ==
SRV[6])) //Moments en els que se te que engegar la EV1

        {

            Sincronitzar () ; // Sincronitzar les dades del servidor.

        }

        if ((Temps_actual == EV1E[0]) || (Temps_actual == EV1E[1]) || (Temps_actual == EV1E[2]) ||
(Temps_actual == EV1E[3]) || (Temps_actual == EV1E[4]) || (Temps_actual == EV1E[5]) || (Temps_actual
== EV1E[6])) //Moments en els que se te que engegar la EV1

        {

            Modul_EV.ara(EV1_Engegar, Temps_Engegat);  // Electrovàlvula 1 Engegar

        }

        if ((Temps_actual == EV1A[0]) || (Temps_actual == EV1A[1]) || (Temps_actual == EV1A[2]) ||
(Temps_actual == EV1A[3]) || (Temps_actual == EV1A[4]) || (Temps_actual == EV1A[5]) || (Temps_actual
== EV1A[6])) //Moments en els que se te que engegar la EV1

        {

            Modul_EV.ara(EV1_Aturar, Temps_Engegat);  // Electrovàlvula 1 Aturar

        }

        if ((Temps_actual == EV2E[0]) || (Temps_actual == EV2E[1]) || (Temps_actual == EV2E[2]) ||
(Temps_actual == EV2E[3]) || (Temps_actual == EV2E[4]) || (Temps_actual == EV2E[5]) || (Temps_actual
== EV2E[6])) //Moments en els que se te que engegar la EV2

        {

            Modul_EV.ara(EV2_Engegar, Temps_Engegat); // Electrovàlvula 2 Engegar

        }

        if ((Temps_actual == EV2A[0]) || (Temps_actual == EV2A[1]) || (Temps_actual == EV2A[2]) ||
(Temps_actual == EV2A[3]) || (Temps_actual == EV2A[4]) || (Temps_actual == EV2A[5]) || (Temps_actual
== EV2A[6])) //Moments en els que se te que aturar la EV2

        {

            Modul_EV.ara(EV2_Aturar, Temps_Engegat); // Electrovàlvula 2 Aturar

        }

        if ((Temps_actual == EV3E1[0]) || (Temps_actual == EV3E1[1]) || (Temps_actual == EV3E1[2]) ||
(Temps_actual == EV3E1[3]) || (Temps_actual == EV3E1[4]) || (Temps_actual == EV3E1[5]) ||
(Temps_actual == EV3E1[6]) || (Temps_actual == EV3E2[0]) || (Temps_actual == EV3E2[1]) ||
(Temps_actual == EV3E2[2]) || (Temps_actual == EV3E2[3]) || (Temps_actual == EV3E2[4]) ||
```

```
(Temps_actual == EV3E2[5]) || (Temps_actual == EV3E2[6])) //Moments en els que se te que engegar la EV3

        {

            Modul_EV.ara(EV3_Engegar, Temps_Engegat); // Electrovàlvula 3 Engegar

        }

        if ((Temps_actual == EV3A1[0]) || (Temps_actual == EV3A1[1]) || (Temps_actual == EV3A1[2]) ||
(Temps_actual == EV3A1[3]) || (Temps_actual == EV3A1[4]) || (Temps_actual == EV3A1[5]) ||
(Temps_actual == EV3A1[6]) || (Temps_actual == EV3A2[0]) || (Temps_actual == EV3A2[1]) ||
(Temps_actual == EV3A2[2]) || (Temps_actual == EV3A2[3]) || (Temps_actual == EV3A2[4]) ||
(Temps_actual == EV3A2[5]) || (Temps_actual == EV3A2[6])) //Moments en els que se te que aturar la EV3

        {

            Modul_EV.ara(EV3_Aturar, Temps_Engegat); // Electrovàlvula 3 Aturar

        }

        // mostrar_v_globals();

        // Serial.println(Temps_actual);

        dorm();

    }
```

1.5.1.2 Codi font de "bateria.cpp".

```
/*
  Bateria.cpp - Llibreria per llegir el nivell la bateria 01/09/2019
  V 0.2.
*/
#include <Arduino.h>
#include "bateria.h"
#include "calculs.h"

Bateria::Bateria(int Voltimetre)
{
  pinMode(Voltimetre, INPUT);
}

String Bateria::Mesurar_med(int Voltimetre, int numero_mesures)
{
  unsigned int valors[numero_mesures] ;//
  for (int i = 0 ;  i  < numero_mesures ; i++) // fem unes poques mesures
  {
    unsigned int mesura_actual = analogRead(Voltimetre);
    valors[i] = mesura_actual;
    delay (10);
  }
  unsigned int valors_mida = sizeof(valors) / sizeof(unsigned int);
  Estadistica.Ordenar(valors, valors_mida); // Ordenem els valors
  unsigned int posicio_mitjana = (valors_mida / 2);
```

```
  unsigned int Tensio  = ( valors[posicio_mitjana] * 35.44921875 ); // pasem a mV
  String Torne = String(Tensio, DEC) ;
  return Torne;
}
```

1.5.1.3 Codi font de “bateria.h”.

```
#ifndef Bateria_h

#define Bateria_h

#include "Arduino.h"

class Bateria
{
  public :
    Bateria(int Voltimetre);
    String Mesurar_med(int Voltimetre, int numero_mesures);

  private:
};
#endif
```

1.5.1.4 Codi font de “calculs.cpp“.

```
#include <Arduino.h>
#include "calculs.h"

Calculs::Calculs()
{
}

void Calculs::Ordenar(unsigned int* valors, int length)
{
  int i, j, flag = 1;
  int temp;
  for (i = 1; (i <= length) && flag; i++)
  {
    flag = 0;
    for (j = 0; j < (length - 1); j++)
    {
      if (valors[j + 1] < valors[j])
      {
        temp = valors[j];
        valors[j] = valors[j + 1];
        valors[j + 1] = temp;
        flag = 1;
      }
    }
  }
}
Calculs Estadistica;
```

1.5.1.5 Codi font de Calculs.h.

```
#ifndef calculs_h
#define calculs_h

#include "arduino.h"

class Calculs
{
  public :
    Calculs();
    void Ordenar(unsigned int* valors, int length);
  private:

};

extern Calculs Estadistica ;
#endif
```

1.5.1.6 Codi font de deposit.cpp.

```
/*
  Deposit.cpp - Llibreria per llegir el nivell del deposit 01/09/2019
  V 0.6.
*/
#include <Arduino.h>
#include "deposit.h"
#include "calculs.h"

Deposit::Deposit(int EcoPota, int DesencadenadorPota)
{
  pinMode(DesencadenadorPota, OUTPUT);
  pinMode(EcoPota, INPUT);
}

String Deposit::Mesurar_med(int EcoPota, int DesencadenadorPota, int numero_mesures)
{
  unsigned int valors[numero_mesures] ;//
  for (int i = 0 ;  i  < numero_mesures ; i++) // fem unes poques mesures
  {
    unsigned int mesura_actual = Mesurar(EcoPota, DesencadenadorPota);
    valors[i] = mesura_actual;
    delay (10);
  }
  unsigned int valors_mida = sizeof(valors) / sizeof(unsigned int);
  Estadistica.Ordenar(valors, valors_mida); // Ordene els valors

  unsigned int posicio_mitjana = (valors_mida / 2);

  unsigned int Distancia  = ( valors[posicio_mitjana] * (0.17) ); // pase de uS a mm.
  String Torne = String(Distancia, DEC) ;
  return Torne;
}

unsigned int Deposit::Mesurar(int EcoPota, int DesencadenadorPota)
```

```
{
  digitalWrite(DesencadenadorPota, LOW);  // Per generar un pols net. Durant 4us deixe l'eixida a
nivell vaig
  delayMicroseconds(4);
  digitalWrite(DesencadenadorPota, HIGH);  // Genere un pols de 10us
  delayMicroseconds(10);
  digitalWrite(DesencadenadorPota, LOW);
  unsigned int duracio = pulseIn(EcoPota, HIGH);  // Mesure el temps en uS
  return duracio;
}
```

1.5.1.7 Codi font deposit.h

```
  #ifndef Deposit_h
#define Deposit_h

#include "Arduino.h"

class Deposit
{
  public :
    Deposit(int EcoPota, int DesencadenadorPota);
    String Mesurar_med(int EcoPota, int DesencadenadorPota, int numero_mesures);

  private:
    unsigned int Mesurar(int EcoPota, int DesencadenadorPota);

};
#endif
```

1.5.1.8 Codi font LowPower.cpp

```
/*****************************************************************************
LowPower Library
Version: 1.60
Date: 01-04-2016
Author: Lim Phang Moh
Company: Rocket Scream Electronics
Website: www.rocketscream.com

This is a lightweight low power library for Arduino.

This library is licensed under Creative Commons Attribution-ShareAlike 3.0
Unported License.

Revision  Description
========  ===========
1.60      Added support for ATmega256RFR2. Contributed by Rodmg.
1.50      Fixed compiler optimization (Arduino IDE 1.6.x branch) on BOD enable
          function that causes the function to be over optimized.
1.40          Added support for ATSAMD21G18A.
              Library format compliant with Arduino IDE 1.5.x.
1.30          Added support for ATMega168, ATMega2560, ATMega1280 & ATMega32U4.
              Tested to work with Arduino IDE 1.0.1 - 1.0.4.
1.20          Remove typo error in idle method for checking whether Timer 0 was
              turned off.
              Remove dependecy on WProgram.h which is not required.
              Tested to work with Arduino IDE 1.0.
1.10          Added #ifndef for sleep_bod_disable() for compatibility with future
              Arduino IDE release.
1.00      Initial public release.
*****************************************************************************/
#if defined (__AVR__)
#include <avr/sleep.h>
#include <avr/wdt.h>
#include <avr/power.h>
#include <avr/interrupt.h>
#elif defined (__arm__)

#else
#error "Processor architecture is not supported."
#endif

#include "LowPower.h"

#if defined (__AVR__)
// Only Pico Power devices can change BOD settings through software
#if defined __AVR_ATmega328P__
#ifndef sleep_bod_disable
#define sleep_bod_disable()                                                    \
  do {                                                                           \
    unsigned char tempreg;                                                         \
    __asm__ __volatile__("in %[tempreg], %[mcucr]" "\n\t"                  \
                         "ori %[tempreg], %[bods_bodse]" "\n\t"        \
                         "out %[mcucr], %[tempreg]" "\n\t"             \
                         "andi %[tempreg], %[not_bodse]" "\n\t"        \
```

```
                    "out %[mcucr], %[tempreg]"                          \
                    : [tempreg] "=&d" (tempreg)                         \
                    : [mcucr] "I" _SFR_IO_ADDR(MCUCR),                  \
                    [bods_bodse] "i" (_BV(BODS) | _BV(BODSE)), \
                    [not_bodse] "i" (~_BV(BODSE)));                     \
  } while (0)
#endif
#endif

#define lowPowerBodOn(mode)       \
  do {                                    \
    set_sleep_mode(mode); \
    cli();                        \
    sleep_enable();               \
    sei();                        \
    sleep_cpu();                  \
    sleep_disable();              \
    sei();                        \
  } while (0);

// Only Pico Power devices can change BOD settings through software
#if defined __AVR_ATmega328P__
#define lowPowerBodOff(mode)\
  do {                                    \
    set_sleep_mode(mode); \
    cli();                        \
    sleep_enable();               \
    sleep_bod_disable(); \
    sei();                        \
    sleep_cpu();                  \
    sleep_disable();              \
    sei();                        \
  } while (0);
#endif

// Some macros is still missing from AVR GCC distribution for ATmega32U4
#if defined __AVR_ATmega32U4__
// Timer 4 PRR bit is currently not defined in iom32u4.h
#ifndef PRTIM4
#define PRTIM4 4
#endif

// Timer 4 power reduction macro is not defined currently in power.h
#ifndef power_timer4_disable
#define power_timer4_disable()       (PRR1 |= (uint8_t)(1 << PRTIM4))
#endif

#ifndef power_timer4_enable
#define power_timer4_enable()       (PRR1 &= (uint8_t)~(1 << PRTIM4))
#endif
#endif

/************************************************************************
  Name: idle
  Description: Putting ATmega328P/168 into idle state. Please make sure you
                    understand the implication and result of disabling module.
```

```
 Argument   Description
 =========      ===========
 1. period   Duration of low power mode. Use SLEEP_FOREVER to use other wake
                    up resource:
                    (a) SLEEP_15MS - 15 ms sleep
                    (b) SLEEP_30MS - 30 ms sleep
                    (c) SLEEP_60MS - 60 ms sleep
                    (d) SLEEP_120MS - 120 ms sleep
                    (e) SLEEP_250MS - 250 ms sleep
                    (f) SLEEP_500MS - 500 ms sleep
                    (g) SLEEP_1S - 1 s sleep
                    (h) SLEEP_2S - 2 s sleep
                    (i) SLEEP_4S - 4 s sleep
                    (j) SLEEP_8S - 8 s sleep
                    (k) SLEEP_FOREVER - Sleep without waking up through WDT

 2. adc        ADC module disable control:
                    (a) ADC_OFF - Turn off ADC module
                    (b) ADC_ON - Leave ADC module in its default state

 3. timer2          Timer 2 module disable control:
                    (a) TIMER2_OFF - Turn off Timer 2 module
                    (b) TIMER2_ON - Leave Timer 2 module in its default state

 4. timer1          Timer 1 module disable control:
                    (a) TIMER1_OFF - Turn off Timer 1 module
                    (b) TIMER1_ON - Leave Timer 1 module in its default state

 5. timer0          Timer 0 module disable control:
                    (a) TIMER0_OFF - Turn off Timer 0 module
                    (b) TIMER0_ON - Leave Timer 0 module in its default state

 6. spi        SPI module disable control:
                    (a) SPI_OFF - Turn off SPI module
                    (b) SPI_ON - Leave SPI module in its default state

 7. usart0          USART0 module disable control:
                    (a) USART0_OFF - Turn off USART0  module
                    (b) USART0_ON - Leave USART0 module in its default state

 8. twi        TWI module disable control:
                    (a) TWI_OFF - Turn off TWI module
                    (b) TWI_ON - Leave TWI module in its default state

*******************************************************************************/
#if defined (__AVR_ATmega328P__) || defined (__AVR_ATmega168__)
void      LowPowerClass::idle(period_t period, adc_t adc, timer2_t timer2,
                    timer1_t timer1, timer0_t timer0,
                    spi_t spi, usart0_t usart0,       twi_t twi)
{
  // Temporary clock source variable
  unsigned char clockSource = 0;

  if (timer2 == TIMER2_OFF)
  {
    if (TCCR2B & CS22) clockSource |= (1 << CS22);
    if (TCCR2B & CS21) clockSource |= (1 << CS21);
```

```
  if (TCCR2B & CS20) clockSource |= (1 << CS20);

  // Remove the clock source to shutdown Timer2
  TCCR2B &= ~(1 << CS22);
  TCCR2B &= ~(1 << CS21);
  TCCR2B &= ~(1 << CS20);

  power_timer2_disable();
 }

 if (adc == ADC_OFF)
 {
  ADCSRA &= ~(1 << ADEN);
  power_adc_disable();
 }

 if (timer1 == TIMER1_OFF)    power_timer1_disable();
 if (timer0 == TIMER0_OFF)    power_timer0_disable();
 if (spi == SPI_OFF)          power_spi_disable();
 if (usart0 == USART0_OFF)    power_usart0_disable();
 if (twi == TWI_OFF)          power_twi_disable();

 if (period != SLEEP_FOREVER)
 {
  wdt_enable(period);
  WDTCSR |= (1 << WDIE);
 }

 lowPowerBodOn(SLEEP_MODE_IDLE);

 if (adc == ADC_OFF)
 {
  power_adc_enable();
  ADCSRA |= (1 << ADEN);
 }

 if (timer2 == TIMER2_OFF)
 {
  if (clockSource & CS22) TCCR2B |= (1 << CS22);
  if (clockSource & CS21) TCCR2B |= (1 << CS21);
  if (clockSource & CS20) TCCR2B |= (1 << CS20);

  power_timer2_enable();
 }

 if (timer1 == TIMER1_OFF)    power_timer1_enable();
 if (timer0 == TIMER0_OFF)    power_timer0_enable();
 if (spi == SPI_OFF)          power_spi_enable();
 if (usart0 == USART0_OFF)    power_usart0_enable();
 if (twi == TWI_OFF)          power_twi_enable();
}
#endif

/*****************************************************************************
  Name: idle
  Description: Putting ATmega32U4 into idle state. Please make sure you
               understand the implication and result of disabling module.
```

```
                Take note that Timer 2 is not available and USART0 is replaced
                with USART1 on ATmega32U4.

  Argument   Description
  =========      ===========
  1. period     Duration of low power mode. Use SLEEP_FOREVER to use other wake
                    up resource:
                    (a) SLEEP_15MS - 15 ms sleep
                    (b) SLEEP_30MS - 30 ms sleep
                    (c) SLEEP_60MS - 60 ms sleep
                    (d) SLEEP_120MS - 120 ms sleep
                    (e) SLEEP_250MS - 250 ms sleep
                    (f) SLEEP_500MS - 500 ms sleep
                    (g) SLEEP_1S - 1 s sleep
                    (h) SLEEP_2S - 2 s sleep
                    (i) SLEEP_4S - 4 s sleep
                    (j) SLEEP_8S - 8 s sleep
                    (k) SLEEP_FOREVER - Sleep without waking up through WDT

  2. adc        ADC module disable control:
                    (a) ADC_OFF - Turn off ADC module
                    (b) ADC_ON - Leave ADC module in its default state

  3. timer4          Timer 4 module disable control:
                    (a) TIMER4_OFF - Turn off Timer 4 module
                    (b) TIMER4_ON - Leave Timer 4 module in its default state

  4. timer3          Timer 3 module disable control:
                    (a) TIMER3_OFF - Turn off Timer 3 module
                    (b) TIMER3_ON - Leave Timer 3 module in its default state

  5. timer1          Timer 1 module disable control:
                    (a) TIMER1_OFF - Turn off Timer 1 module
                    (b) TIMER1_ON - Leave Timer 1 module in its default state

  6. timer0          Timer 0 module disable control:
                    (a) TIMER0_OFF - Turn off Timer 0 module
                    (b) TIMER0_ON - Leave Timer 0 module in its default state

  7. spi        SPI module disable control:
                    (a) SPI_OFF - Turn off SPI module
                    (b) SPI_ON - Leave SPI module in its default state

  8. usart1          USART1 module disable control:
                    (a) USART1_OFF - Turn off USART1  module
                    (b) USART1_ON - Leave USART1 module in its default state

  9. twi        TWI module disable control:
                    (a) TWI_OFF - Turn off TWI module
                    (b) TWI_ON - Leave TWI module in its default state

  10.usb             USB module disable control:
                    (a) USB_OFF - Turn off USB module
                    (b) USB_ON - Leave USB module in its default state
*******************************************************************************/
#if defined __AVR_ATmega32U4__
void      LowPowerClass::idle(period_t period, adc_t adc,
```

```
                    timer4_t timer4, timer3_t timer3,
                    timer1_t timer1, timer0_t timer0,
                    spi_t spi, usart1_t usart1,      twi_t twi, usb_t usb)
{
  if (adc == ADC_OFF)
  {
    ADCSRA &= ~(1 << ADEN);
    power_adc_disable();
  }

  if (timer4 == TIMER4_OFF)   power_timer4_disable();
  if (timer3 == TIMER3_OFF)   power_timer3_disable();
  if (timer1 == TIMER1_OFF)   power_timer1_disable();
  if (timer0 == TIMER0_OFF)   power_timer0_disable();
  if (spi == SPI_OFF)         power_spi_disable();
  if (usart1 == USART1_OFF)   power_usart1_disable();
  if (twi == TWI_OFF)         power_twi_disable();
  if (usb == USB_OFF)               power_usb_disable();

  if (period != SLEEP_FOREVER)
  {
    wdt_enable(period);
    WDTCSR |= (1 << WDIE);
  }

  lowPowerBodOn(SLEEP_MODE_IDLE);

  if (adc == ADC_OFF)
  {
    power_adc_enable();
    ADCSRA |= (1 << ADEN);
  }

  if (timer4 == TIMER4_OFF)   power_timer4_enable();
  if (timer3 == TIMER3_OFF)   power_timer3_enable();
  if (timer1 == TIMER1_OFF)   power_timer1_enable();
  if (timer0 == TIMER0_OFF)   power_timer0_enable();
  if (spi == SPI_OFF)         power_spi_enable();
  if (usart1 == USART1_OFF)   power_usart1_enable();
  if (twi == TWI_OFF)         power_twi_enable();
  if (usb == USB_OFF)               power_usb_enable();
}
#endif

/*****************************************************************************
  Name: idle
  Description: Putting ATmega2560 & ATmega1280 into idle state. Please make sure
               you understand the implication and result of disabling module.
               Take note that extra Timer 5, 4, 3 compared to an ATmega328P/168.
               Also take note that extra USART 3, 2, 1 compared to an
               ATmega328P/168.

  Argument   Description
  =========     ===========
  1. period     Duration of low power mode. Use SLEEP_FOREVER to use other wake
                up resource:
                (a) SLEEP_15MS - 15 ms sleep
```

(b) SLEEP_30MS - 30 ms sleep
(c) SLEEP_60MS - 60 ms sleep
(d) SLEEP_120MS - 120 ms sleep
(e) SLEEP_250MS - 250 ms sleep
(f) SLEEP_500MS - 500 ms sleep
(g) SLEEP_1S - 1 s sleep
(h) SLEEP_2S - 2 s sleep
(i) SLEEP_4S - 4 s sleep
(j) SLEEP_8S - 8 s sleep
(k) SLEEP_FOREVER - Sleep without waking up through WDT

2. adc ADC module disable control:
(a) ADC_OFF - Turn off ADC module
(b) ADC_ON - Leave ADC module in its default state

3. timer5 Timer 5 module disable control:
(a) TIMER5_OFF - Turn off Timer 5 module
(b) TIMER5_ON - Leave Timer 5 module in its default state

4. timer4 Timer 4 module disable control:
(a) TIMER4_OFF - Turn off Timer 4 module
(b) TIMER4_ON - Leave Timer 4 module in its default state

5. timer3 Timer 3 module disable control:
(a) TIMER3_OFF - Turn off Timer 3 module
(b) TIMER3_ON - Leave Timer 3 module in its default state

6. timer2 Timer 2 module disable control:
(a) TIMER2_OFF - Turn off Timer 2 module
(b) TIMER2_ON - Leave Timer 2 module in its default state

7. timer1 Timer 1 module disable control:
(a) TIMER1_OFF - Turn off Timer 1 module
(b) TIMER1_ON - Leave Timer 1 module in its default state

8. timer0 Timer 0 module disable control:
(a) TIMER0_OFF - Turn off Timer 0 module
(b) TIMER0_ON - Leave Timer 0 module in its default state

9. spi SPI module disable control:
(a) SPI_OFF - Turn off SPI module
(b) SPI_ON - Leave SPI module in its default state

10.usart3 USART3 module disable control:
(a) USART3_OFF - Turn off USART3 module
(b) USART3_ON - Leave USART3 module in its default state

11.usart2 USART2 module disable control:
(a) USART2_OFF - Turn off USART2 module
(b) USART2_ON - Leave USART2 module in its default state

12.usart1 USART1 module disable control:
(a) USART1_OFF - Turn off USART1 module
(b) USART1_ON - Leave USART1 module in its default state

13.usart0 USART0 module disable control:
(a) USART0_OFF - Turn off USART0 module

```
                    (b) USART0_ON - Leave USART0 module in its default state

  14.twi        TWI module disable control:
                    (a) TWI_OFF - Turn off TWI module
                    (b) TWI_ON - Leave TWI module in its default state

*****************************************************************************/
#if defined (__AVR_ATmega2560__) || defined (__AVR_ATmega1280__)
void      LowPowerClass::idle(period_t period, adc_t adc, timer5_t timer5,
                      timer4_t timer4, timer3_t timer3, timer2_t timer2,
                      timer1_t timer1, timer0_t timer0, spi_t spi,
                      usart3_t usart3, usart2_t usart2, usart1_t usart1,
                      usart0_t usart0, twi_t twi)
{
  // Temporary clock source variable
  unsigned char clockSource = 0;

  if (timer2 == TIMER2_OFF)
  {
    if (TCCR2B & CS22) clockSource |= (1 << CS22);
    if (TCCR2B & CS21) clockSource |= (1 << CS21);
    if (TCCR2B & CS20) clockSource |= (1 << CS20);

    // Remove the clock source to shutdown Timer2
    TCCR2B &= ~(1 << CS22);
    TCCR2B &= ~(1 << CS21);
    TCCR2B &= ~(1 << CS20);

    power_timer2_disable();
  }

  if (adc == ADC_OFF)
  {
   ADCSRA &= ~(1 << ADEN);
   power_adc_disable();
  }

  if (timer5 == TIMER5_OFF)    power_timer5_disable();
  if (timer4 == TIMER4_OFF)    power_timer4_disable();
  if (timer3 == TIMER3_OFF)    power_timer3_disable();
  if (timer1 == TIMER1_OFF)    power_timer1_disable();
  if (timer0 == TIMER0_OFF)    power_timer0_disable();
  if (spi == SPI_OFF)          power_spi_disable();
  if (usart3 == USART3_OFF)    power_usart3_disable();
  if (usart2 == USART2_OFF)    power_usart2_disable();
  if (usart1 == USART1_OFF)    power_usart1_disable();
  if (usart0 == USART0_OFF)    power_usart0_disable();
  if (twi == TWI_OFF)          power_twi_disable();

  if (period != SLEEP_FOREVER)
  {
    wdt_enable(period);
    WDTCSR |= (1 << WDIE);
  }

  lowPowerBodOn(SLEEP_MODE_IDLE);
```

```
  if (adc == ADC_OFF)
  {
    power_adc_enable();
    ADCSRA |= (1 << ADEN);
  }

  if (timer2 == TIMER2_OFF)
  {
    if (clockSource & CS22) TCCR2B |= (1 << CS22);
    if (clockSource & CS21) TCCR2B |= (1 << CS21);
    if (clockSource & CS20) TCCR2B |= (1 << CS20);

    power_timer2_enable();
  }

  if (timer5 == TIMER5_OFF)    power_timer5_enable();
  if (timer4 == TIMER4_OFF)    power_timer4_enable();
  if (timer3 == TIMER3_OFF)    power_timer3_enable();
  if (timer1 == TIMER1_OFF)    power_timer1_enable();
  if (timer0 == TIMER0_OFF)    power_timer0_enable();
  if (spi == SPI_OFF)          power_spi_enable();
  if (usart3 == USART3_OFF)    power_usart3_enable();
  if (usart2 == USART2_OFF)    power_usart2_enable();
  if (usart1 == USART1_OFF)    power_usart1_enable();
  if (usart0 == USART0_OFF)    power_usart0_enable();
  if (twi == TWI_OFF)          power_twi_enable();
}
#endif

/*****************************************************************************
  Name: idle
  Description: Putting ATmega256RFR2 into idle state. Please make sure
               you understand the implication and result of disabling module.
               Take note that extra Timer 5, 4, 3 compared to an ATmega328P/168.
               Also take note that extra USART 1 compared to an
               ATmega328P/168.

  Argument   Description
  =========     ===========
  1. period     Duration of low power mode. Use SLEEP_FOREVER to use other wake
                up resource:
                (a) SLEEP_15MS - 15 ms sleep
                (b) SLEEP_30MS - 30 ms sleep
                (c) SLEEP_60MS - 60 ms sleep
                (d) SLEEP_120MS - 120 ms sleep
                (e) SLEEP_250MS - 250 ms sleep
                (f) SLEEP_500MS - 500 ms sleep
                (g) SLEEP_1S - 1 s sleep
                (h) SLEEP_2S - 2 s sleep
                (i) SLEEP_4S - 4 s sleep
                (j) SLEEP_8S - 8 s sleep
                (k) SLEEP_FOREVER - Sleep without waking up through WDT

  2. adc        ADC module disable control:
                (a) ADC_OFF - Turn off ADC module
                (b) ADC_ON - Leave ADC module in its default state
```

```
  3. timer5         Timer 5 module disable control:
                    (a) TIMER5_OFF - Turn off Timer 5 module
                    (b) TIMER5_ON - Leave Timer 5 module in its default state

  4. timer4         Timer 4 module disable control:
                    (a) TIMER4_OFF - Turn off Timer 4 module
                    (b) TIMER4_ON - Leave Timer 4 module in its default state

  5. timer3         Timer 3 module disable control:
                    (a) TIMER3_OFF - Turn off Timer 3 module
                    (b) TIMER3_ON - Leave Timer 3 module in its default state

  6. timer2         Timer 2 module disable control:
                    (a) TIMER2_OFF - Turn off Timer 2 module
                    (b) TIMER2_ON - Leave Timer 2 module in its default state

  7. timer1         Timer 1 module disable control:
                    (a) TIMER1_OFF - Turn off Timer 1 module
                    (b) TIMER1_ON - Leave Timer 1 module in its default state

  8. timer0         Timer 0 module disable control:
                    (a) TIMER0_OFF - Turn off Timer 0 module
                    (b) TIMER0_ON - Leave Timer 0 module in its default state

  9. spi        SPI module disable control:
                    (a) SPI_OFF - Turn off SPI module
                    (b) SPI_ON - Leave SPI module in its default state

  10.usart1         USART1 module disable control:
                    (a) USART1_OFF - Turn off USART1  module
                    (b) USART1_ON - Leave USART1 module in its default state

  11.usart0         USART0 module disable control:
                    (a) USART0_OFF - Turn off USART0  module
                    (b) USART0_ON - Leave USART0 module in its default state

  12.twi        TWI module disable control:
                    (a) TWI_OFF - Turn off TWI module
                    (b) TWI_ON - Leave TWI module in its default state

*****************************************************************************/
#if defined (__AVR_ATmega256RFR2__)
void      LowPowerClass::idle(period_t period, adc_t adc, timer5_t timer5,
                    timer4_t timer4, timer3_t timer3, timer2_t timer2,
                    timer1_t timer1, timer0_t timer0, spi_t spi,
                    usart1_t usart1,
                    usart0_t usart0, twi_t twi)
{
  // Temporary clock source variable
  unsigned char clockSource = 0;

  if (timer2 == TIMER2_OFF)
  {
    if (TCCR2B & CS22) clockSource |= (1 << CS22);
    if (TCCR2B & CS21) clockSource |= (1 << CS21);
    if (TCCR2B & CS20) clockSource |= (1 << CS20);
```

```
  // Remove the clock source to shutdown Timer2
  TCCR2B &= ~(1 << CS22);
  TCCR2B &= ~(1 << CS21);
  TCCR2B &= ~(1 << CS20);

  power_timer2_disable();
}

if (adc == ADC_OFF)
{
  ADCSRA &= ~(1 << ADEN);
  power_adc_disable();
}

if (timer5 == TIMER5_OFF)    power_timer5_disable();
if (timer4 == TIMER4_OFF)    power_timer4_disable();
if (timer3 == TIMER3_OFF)    power_timer3_disable();
if (timer1 == TIMER1_OFF)    power_timer1_disable();
if (timer0 == TIMER0_OFF)    power_timer0_disable();
if (spi == SPI_OFF)            power_spi_disable();
if (usart1 == USART1_OFF)    power_usart1_disable();
if (usart0 == USART0_OFF)    power_usart0_disable();
if (twi == TWI_OFF)            power_twi_disable();

if (period != SLEEP_FOREVER)
{
  wdt_enable(period);
  WDTCSR |= (1 << WDIE);
}

lowPowerBodOn(SLEEP_MODE_IDLE);

if (adc == ADC_OFF)
{
  power_adc_enable();
  ADCSRA |= (1 << ADEN);
}

if (timer2 == TIMER2_OFF)
{
  if (clockSource & CS22) TCCR2B |= (1 << CS22);
  if (clockSource & CS21) TCCR2B |= (1 << CS21);
  if (clockSource & CS20) TCCR2B |= (1 << CS20);

  power_timer2_enable();
}

if (timer5 == TIMER5_OFF)    power_timer5_enable();
if (timer4 == TIMER4_OFF)    power_timer4_enable();
if (timer3 == TIMER3_OFF)    power_timer3_enable();
if (timer1 == TIMER1_OFF)    power_timer1_enable();
if (timer0 == TIMER0_OFF)    power_timer0_enable();
if (spi == SPI_OFF)            power_spi_enable();
if (usart1 == USART1_OFF)    power_usart1_enable();
if (usart0 == USART0_OFF)    power_usart0_enable();
if (twi == TWI_OFF)            power_twi_enable();
}
```

```
#endif


/*****************************************************************************
  Name: adcNoiseReduction
  Description: Putting microcontroller into ADC noise reduction state. This is
                  a very useful state when using the ADC to achieve best and low
               noise signal.

  Argument   Description
  =========      ============
  1. period     Duration of low power mode. Use SLEEP_FOREVER to use other wake
                    up resource:
                    (a) SLEEP_15MS - 15 ms sleep
                    (b) SLEEP_30MS - 30 ms sleep
                    (c) SLEEP_60MS - 60 ms sleep
                    (d) SLEEP_120MS - 120 ms sleep
                    (e) SLEEP_250MS - 250 ms sleep
                    (f) SLEEP_500MS - 500 ms sleep
                    (g) SLEEP_1S - 1 s sleep
                    (h) SLEEP_2S - 2 s sleep
                    (i) SLEEP_4S - 4 s sleep
                    (j) SLEEP_8S - 8 s sleep
                    (k) SLEEP_FOREVER - Sleep without waking up through WDT

  2. adc        ADC module disable control. Turning off the ADC module is
                    basically removing the purpose of this low power mode.
                    (a) ADC_OFF - Turn off ADC module
                    (b) ADC_ON - Leave ADC module in its default state

  3. timer2          Timer 2 module disable control:
                    (a) TIMER2_OFF - Turn off Timer 2 module
                    (b) TIMER2_ON - Leave Timer 2 module in its default state

*****************************************************************************/
void      LowPowerClass::adcNoiseReduction(period_t period, adc_t adc,
                                  timer2_t timer2)
{
  // Temporary clock source variable
  unsigned char clockSource = 0;

#if !defined(__AVR_ATmega32U4__)
  if (timer2 == TIMER2_OFF)
  {
    if (TCCR2B & CS22) clockSource |= (1 << CS22);
    if (TCCR2B & CS21) clockSource |= (1 << CS21);
    if (TCCR2B & CS20) clockSource |= (1 << CS20);

    // Remove the clock source to shutdown Timer2
    TCCR2B &= ~(1 << CS22);
    TCCR2B &= ~(1 << CS21);
    TCCR2B &= ~(1 << CS20);
  }
#endif

  if (adc == ADC_OFF)      ADCSRA &= ~(1 << ADEN);
```

```
  if (period != SLEEP_FOREVER)
  {
    wdt_enable(period);
    WDTCSR |= (1 << WDIE);
  }

  lowPowerBodOn(SLEEP_MODE_ADC);

  if (adc == ADC_OFF) ADCSRA |= (1 << ADEN);

#if !defined(__AVR_ATmega32U4__)
  if (timer2 == TIMER2_OFF)
  {
    if (clockSource & CS22) TCCR2B |= (1 << CS22);
    if (clockSource & CS21) TCCR2B |= (1 << CS21);
    if (clockSource & CS20) TCCR2B |= (1 << CS20);

  }
#endif
}

/*****************************************************************************
  Name: powerDown
  Description: Putting microcontroller into power down state. This is
               the lowest current consumption state. Use this together with
               external pin interrupt to wake up through external event
               triggering (example: RTC clockout pin, SD card detect pin).

  Argument   Description
  =========     ===========
  1. period     Duration of low power mode. Use SLEEP_FOREVER to use other wake
                up resource:
                (a) SLEEP_15MS - 15 ms sleep
                (b) SLEEP_30MS - 30 ms sleep
                (c) SLEEP_60MS - 60 ms sleep
                (d) SLEEP_120MS - 120 ms sleep
                (e) SLEEP_250MS - 250 ms sleep
                (f) SLEEP_500MS - 500 ms sleep
                (g) SLEEP_1S - 1 s sleep
                (h) SLEEP_2S - 2 s sleep
                (i) SLEEP_4S - 4 s sleep
                (j) SLEEP_8S - 8 s sleep
                (k) SLEEP_FOREVER - Sleep without waking up through WDT

  2. adc        ADC module disable control. Turning off the ADC module is
                basically removing the purpose of this low power mode.
                (a) ADC_OFF - Turn off ADC module
                (b) ADC_ON - Leave ADC module in its default state

  3. bod        Brown Out Detector (BOD) module disable control:
                (a) BOD_OFF - Turn off BOD module
                (b) BOD_ON - Leave BOD module in its default state

*****************************************************************************/
void     LowPowerClass::powerDown(period_t period, adc_t adc, bod_t bod)
{
  if (adc == ADC_OFF)      ADCSRA &= ~(1 << ADEN);
```

```
  if (period != SLEEP_FOREVER)
  {
    wdt_enable(period);
    WDTCSR |= (1 << WDIE);
  }
  if (bod == BOD_OFF)
  {
#if defined __AVR_ATmega328P__
    lowPowerBodOff(SLEEP_MODE_PWR_DOWN);
#else
    lowPowerBodOn(SLEEP_MODE_PWR_DOWN);
#endif
  }
  else
  {
    lowPowerBodOn(SLEEP_MODE_PWR_DOWN);
  }

  if (adc == ADC_OFF) ADCSRA |= (1 << ADEN);
}

/*****************************************************************************
  Name: powerSave
  Description: Putting microcontroller into power save state. This is
               the lowest current consumption state after power down.
               Use this state together with an external 32.768 kHz crystal (but
               8/16 MHz crystal/resonator need to be removed) to provide an
               asynchronous clock source to Timer 2. Please take note that
               Timer 2 is also used by the Arduino core for PWM operation.
               Please refer to wiring.c for explanation. Removal of the external
               8/16 MHz crystal/resonator requires the microcontroller to run
               on its internal RC oscillator which is not so accurate for time
               critical operation.

  Argument   Description
  =========     ===========
  1. period   Duration of low power mode. Use SLEEP_FOREVER to use other wake
                 up resource:
                 (a) SLEEP_15MS - 15 ms sleep
                 (b) SLEEP_30MS - 30 ms sleep
                 (c) SLEEP_60MS - 60 ms sleep
                 (d) SLEEP_120MS - 120 ms sleep
                 (e) SLEEP_250MS - 250 ms sleep
                 (f) SLEEP_500MS - 500 ms sleep
                 (g) SLEEP_1S - 1 s sleep
                 (h) SLEEP_2S - 2 s sleep
                 (i) SLEEP_4S - 4 s sleep
                 (j) SLEEP_8S - 8 s sleep
                 (k) SLEEP_FOREVER - Sleep without waking up through WDT

  2. adc      ADC module disable control. Turning off the ADC module is
                 basically removing the purpose of this low power mode.
                 (a) ADC_OFF - Turn off ADC module
                 (b) ADC_ON - Leave ADC module in its default state

  3. bod      Brown Out Detector (BOD) module disable control:
```

```
                    (a) BOD_OFF - Turn off BOD module
                    (b) BOD_ON - Leave BOD module in its default state

  4. timer2         Timer 2 module disable control:
                    (a) TIMER2_OFF - Turn off Timer 2 module
                    (b) TIMER2_ON - Leave Timer 2 module in its default state

*******************************************************************************/
void      LowPowerClass::powerSave(period_t period, adc_t adc, bod_t bod,
                          timer2_t timer2)
{
  // Temporary clock source variable
  unsigned char clockSource = 0;

#if !defined(__AVR_ATmega32U4__)
  if (timer2 == TIMER2_OFF)
  {
    if (TCCR2B & CS22) clockSource |= (1 << CS22);
    if (TCCR2B & CS21) clockSource |= (1 << CS21);
    if (TCCR2B & CS20) clockSource |= (1 << CS20);

    // Remove the clock source to shutdown Timer2
    TCCR2B &= ~(1 << CS22);
    TCCR2B &= ~(1 << CS21);
    TCCR2B &= ~(1 << CS20);
  }
#endif

  if (adc == ADC_OFF)      ADCSRA &= ~(1 << ADEN);

  if (period != SLEEP_FOREVER)
  {
    wdt_enable(period);
    WDTCSR |= (1 << WDIE);
  }

  if (bod == BOD_OFF)
  {
#if defined __AVR_ATmega328P__
    lowPowerBodOff(SLEEP_MODE_PWR_SAVE);
#else
    lowPowerBodOn(SLEEP_MODE_PWR_SAVE);
#endif
  }
  else
  {
    lowPowerBodOn(SLEEP_MODE_PWR_SAVE);
  }

  if (adc == ADC_OFF) ADCSRA |= (1 << ADEN);

#if !defined(__AVR_ATmega32U4__)
  if (timer2 == TIMER2_OFF)
  {
    if (clockSource & CS22) TCCR2B |= (1 << CS22);
    if (clockSource & CS21) TCCR2B |= (1 << CS21);
    if (clockSource & CS20) TCCR2B |= (1 << CS20);
```

```
  }
#endif
}

/*****************************************************************************
  Name: powerStandby
  Description: Putting microcontroller into power standby state.

  Argument   Description
  =========      ===========
  1. period     Duration of low power mode. Use SLEEP_FOREVER to use other wake
                    up resource:
                    (a) SLEEP_15MS - 15 ms sleep
                    (b) SLEEP_30MS - 30 ms sleep
                    (c) SLEEP_60MS - 60 ms sleep
                    (d) SLEEP_120MS - 120 ms sleep
                    (e) SLEEP_250MS - 250 ms sleep
                    (f) SLEEP_500MS - 500 ms sleep
                    (g) SLEEP_1S - 1 s sleep
                    (h) SLEEP_2S - 2 s sleep
                    (i) SLEEP_4S - 4 s sleep
                    (j) SLEEP_8S - 8 s sleep
                    (k) SLEEP_FOREVER - Sleep without waking up through WDT

  2. adc        ADC module disable control. Turning off the ADC module is
                    basically removing the purpose of this low power mode.
                    (a) ADC_OFF - Turn off ADC module
                    (b) ADC_ON - Leave ADC module in its default state

  3. bod        Brown Out Detector (BOD) module disable control:
                    (a) BOD_OFF - Turn off BOD module
                    (b) BOD_ON - Leave BOD module in its default state

*****************************************************************************/
void      LowPowerClass::powerStandby(period_t period, adc_t adc, bod_t bod)
{
  if (adc == ADC_OFF)      ADCSRA &= ~(1 << ADEN);

  if (period != SLEEP_FOREVER)
  {
    wdt_enable(period);
    WDTCSR |= (1 << WDIE);
  }

  if (bod == BOD_OFF)
  {
#if defined __AVR_ATmega328P__
    lowPowerBodOff(SLEEP_MODE_STANDBY);
#else
    lowPowerBodOn(SLEEP_MODE_STANDBY);
#endif
  }
  else
  {
    lowPowerBodOn(SLEEP_MODE_STANDBY);
  }
```

```
  if (adc == ADC_OFF) ADCSRA |= (1 << ADEN);
}

/*******************************************************************************
  Name: powerExtStandby
  Description: Putting microcontroller into power extended standby state. This
               is different from the power standby state as it has the
               capability to run Timer 2 asynchronously.

  Argument   Description
  =========     ============
  1. period     Duration of low power mode. Use SLEEP_FOREVER to use other wake
                up resource:
                (a) SLEEP_15MS - 15 ms sleep
                (b) SLEEP_30MS - 30 ms sleep
                (c) SLEEP_60MS - 60 ms sleep
                (d) SLEEP_120MS - 120 ms sleep
                (e) SLEEP_250MS - 250 ms sleep
                (f) SLEEP_500MS - 500 ms sleep
                (g) SLEEP_1S - 1 s sleep
                (h) SLEEP_2S - 2 s sleep
                (i) SLEEP_4S - 4 s sleep
                (j) SLEEP_8S - 8 s sleep
                (k) SLEEP_FOREVER - Sleep without waking up through WDT

  2. adc        ADC module disable control.
                (a) ADC_OFF - Turn off ADC module
                (b) ADC_ON - Leave ADC module in its default state

  3. bod        Brown Out Detector (BOD) module disable control:
                (a) BOD_OFF - Turn off BOD module
                (b) BOD_ON - Leave BOD module in its default state

  4. timer2     Timer 2 module disable control:
                (a) TIMER2_OFF - Turn off Timer 2 module
                (b) TIMER2_ON - Leave Timer 2 module in its default state

*******************************************************************************/
void     LowPowerClass::powerExtStandby(period_t period, adc_t adc, bod_t bod,
                                        timer2_t timer2)
{
  // Temporary clock source variable
  unsigned char clockSource = 0;

#if !defined(__AVR_ATmega32U4__)
  if (timer2 == TIMER2_OFF)
  {
    if (TCCR2B & CS22) clockSource |= (1 << CS22);
    if (TCCR2B & CS21) clockSource |= (1 << CS21);
    if (TCCR2B & CS20) clockSource |= (1 << CS20);

    // Remove the clock source to shutdown Timer2
    TCCR2B &= ~(1 << CS22);
    TCCR2B &= ~(1 << CS21);
    TCCR2B &= ~(1 << CS20);
  }
#endif
```

```
  if (adc == ADC_OFF)      ADCSRA &= ~(1 << ADEN);

  if (period != SLEEP_FOREVER)
  {
    wdt_enable(period);
    WDTCSR |= (1 << WDIE);
  }
  if (bod == BOD_OFF)
  {
#if defined __AVR_ATmega328P__
    lowPowerBodOff(SLEEP_MODE_EXT_STANDBY);
#else
    lowPowerBodOn(SLEEP_MODE_EXT_STANDBY);
#endif
  }
  else
  {
    lowPowerBodOn(SLEEP_MODE_EXT_STANDBY);
  }

  if (adc == ADC_OFF) ADCSRA |= (1 << ADEN);

#if !defined(__AVR_ATmega32U4__)
  if (timer2 == TIMER2_OFF)
  {
    if (clockSource & CS22) TCCR2B |= (1 << CS22);
    if (clockSource & CS21) TCCR2B |= (1 << CS21);
    if (clockSource & CS20) TCCR2B |= (1 << CS20);
  }
#endif
}

/*****************************************************************************
  Name: ISR (WDT_vect)
  Description: Watchdog Timer interrupt service routine. This routine is
               required to allow automatic WDIF and WDIE bit clearance in
                 hardware.

*****************************************************************************/
ISR (WDT_vect)
{
  // WDIE & WDIF is cleared in hardware upon entering this ISR
  wdt_disable();
}

#elif defined (__arm__)
#if defined (__SAMD21G18A__)
/*****************************************************************************
  Name: standby
  Description: Putting SAMD21G18A into idle mode. This is the lowest current
          consumption mode. Requires separate handling of clock and
               peripheral management (disabling and shutting down) to achieve
               the desired current consumption.

  Argument   Description
  =========      ===========
```

```
  1. idleMode   Idle mode level (0, 1, 2) where IDLE_2 level provide lowest
                current consumption in this mode.

*****************************************************************************/
void      LowPowerClass::idle(idle_t idleMode)
{
  SCB->SCR &= ~SCB_SCR_SLEEPDEEP_Msk;
  PM->SLEEP.reg = idleMode;
  __DSB();
  __WFI();
}

/*****************************************************************************
  Name: standby
  Description: Putting SAMD21G18A into standby mode. This is the lowest current
               consumption mode. Use this together with the built-in RTC (use
               RTCZero library) or external pin interrupt to wake up through
               external event triggering.

  Argument   Description
  =========      ===========
  1. NIL

*****************************************************************************/
void      LowPowerClass::standby()
{
  SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
  __DSB();
  __WFI();
}

#else
#error "Please ensure chosen MCU is ATSAMD21G18A."
#endif
#else
#error "Processor architecture is not supported."
#endif

LowPowerClass LowPower;
```

1.5.1.9 Codi font de LowPower.h

```
#ifndef LowPower_h
#define LowPower_h

#include "Arduino.h"

enum period_t
{
  SLEEP_15MS,
  SLEEP_30MS,
  SLEEP_60MS,
  SLEEP_120MS,
  SLEEP_250MS,
  SLEEP_500MS,
  SLEEP_1S,
  SLEEP_2S,
  SLEEP_4S,
```

```
  SLEEP_8S,
  SLEEP_FOREVER
};

enum bod_t
{
  BOD_OFF,
  BOD_ON
};

enum adc_t
{
  ADC_OFF,
  ADC_ON
};

enum timer5_t
{
  TIMER5_OFF,
  TIMER5_ON
};

enum timer4_t
{
  TIMER4_OFF,
  TIMER4_ON
};

enum timer3_t
{
  TIMER3_OFF,
  TIMER3_ON
};

enum timer2_t
{
  TIMER2_OFF,
  TIMER2_ON
};

enum timer1_t
{
  TIMER1_OFF,
  TIMER1_ON
};

enum timer0_t
{
  TIMER0_OFF,
  TIMER0_ON
};

enum spi_t
{
  SPI_OFF,
  SPI_ON
};
```

```
enum usart0_t
{
  USART0_OFF,
  USART0_ON
};

enum usart1_t
{
  USART1_OFF,
  USART1_ON
};

enum usart2_t
{
  USART2_OFF,
  USART2_ON
};

enum usart3_t
{
  USART3_OFF,
  USART3_ON
};

enum twi_t
{
  TWI_OFF,
  TWI_ON
};

enum usb_t
{
  USB_OFF,
  USB_ON
};

enum idle_t
{
  IDLE_0,
  IDLE_1,
  IDLE_2
};

class LowPowerClass
{
  public:
#if defined (__AVR__)

#if defined (__AVR_ATmega328P__) || defined (__AVR_ATmega168__)
    void  idle(period_t period, adc_t adc, timer2_t timer2,
               timer1_t timer1, timer0_t timer0, spi_t spi,
               usart0_t usart0, twi_t twi);
#elif defined __AVR_ATmega2560__
    void  idle(period_t period, adc_t adc, timer5_t timer5,
               timer4_t timer4, timer3_t timer3, timer2_t timer2,
               timer1_t timer1, timer0_t timer0, spi_t spi,
```

```
                usart3_t usart3, usart2_t usart2, usart1_t usart1,
                usart0_t usart0, twi_t twi);
        #elif defined __AVR_ATmega256RFR2__
            void  idle(period_t period, adc_t adc, timer5_t timer5,
                timer4_t timer4, timer3_t timer3, timer2_t timer2,
                timer1_t timer1, timer0_t timer0, spi_t spi,
                usart1_t usart1,
                usart0_t usart0, twi_t twi);
        #elif defined __AVR_ATmega32U4__
            void  idle(period_t period, adc_t adc, timer4_t timer4,
                timer3_t timer3, timer1_t timer1, timer0_t timer0,
                spi_t spi, usart1_t usart1, twi_t twi, usb_t usb);
        #else
        #error "Please ensure chosen MCU is either 168, 328P, 32U4, 2560 or 256RFR2."
        #endif
            void  adcNoiseReduction(period_t period, adc_t adc, timer2_t timer2) __attribute__((optimize("-
O1")));
            void  powerDown(period_t period, adc_t adc, bod_t bod) __attribute__((optimize("-O1")));
            void  powerSave(period_t period, adc_t adc, bod_t bod, timer2_t timer2) __attribute__((optimize("-
O1")));
            void  powerStandby(period_t period, adc_t adc, bod_t bod) __attribute__((optimize("-O1")));
            void  powerExtStandby(period_t period, adc_t adc, bod_t bod, timer2_t timer2)
__attribute__((optimize("-O1")));

        #elif defined (__arm__)

        #if defined (__SAMD21G18A__)
            void  idle(idle_t idleMode);
            void  standby();
        #else
        #error "Please ensure chosen MCU is ATSAMD21G18A."
        #endif

        #else

        #error "Processor architecture is not supported."

        #endif
        };

        extern LowPowerClass LowPower;
        #endif
```

1.5.1.10 Codi font regar.cpp

```
/*
  regar.cpp - Llibreria per engegar o aturar una electrovàlvula de reg 09/11/2018
  V 0.1.
*/
#include <Arduino.h>
#include "regar.h"

Regar::Regar()
{
}

void Regar::Configurar(int EV1_Engegar, int EV1_Aturar, int EV2_Engegar, int EV2_Aturar, int
EV3_Engegar, int EV3_Aturar)
{
  pinMode(EV1_Engegar, OUTPUT);
  pinMode(EV2_Engegar, OUTPUT);
  pinMode(EV3_Engegar, OUTPUT);
  pinMode(EV1_Aturar, OUTPUT);
  pinMode(EV2_Aturar, OUTPUT);
  pinMode(EV3_Aturar, OUTPUT);
}

void Regar::ara(int Eixida, int Temps)
{
  //lectura de la fecha actual
  digitalWrite(Eixida, HIGH);
  delay(Temps);
  digitalWrite(Eixida, LOW);
}
```

1.5.1.11 Codi font regar.h

```
#ifndef Regar_h
#define Regar_h

#include "Arduino.h"

class Regar
{
  public :
    Regar();

    void Configurar(int EV1_Engegar, int EV1_Aturar, int EV2_Engegar, int EV2_Aturar, int
EV3_Engegar, int EV3_Aturar);
    void ara(int Eixida, int Temps);
};
#endif
```

1.5.1.12 Codi font Rellotge.cpp

```
/*
  rellotge.cpp - Llibreria per llegir l'informació que ens cal del rellotge 10/07/2019
  V 0.3.
  Aquesta llibreria aprofita la llibreria RTClib, i retorna els valors que necessitem únicament.
*/

#include <Arduino.h>
// #include "SPI.h"
// #include "Wire.h"
#include "rellotge.h"
#include "RTClib.h"

RTC_DS1307 rtc;
Rellotge::Rellotge(int RTC_Vcc)
{
  pinMode(RTC_Vcc, OUTPUT);
}

word Rellotge::temps_actual (int Vcc)
{
  digitalWrite(Vcc, true);
  delay(10)  ; // Espere un mínim de 2 ms perquè el RTC estiga estabilitzat després d'alimentar-lo amb
normalitat
  if (! rtc.begin()) {
    Serial.println("No hi ha mòdul RTC");
    while (1);
  }
  DateTime ara = rtc.now();
  digitalWrite(Vcc, false);
  int tmp_dia = (int(ara.dayOfTheWeek())) ;
  if(tmp_dia == 0) // Hui es diumenge
  {
    tmp_dia = 7 ;
  }
  tmp_dia = (tmp_dia - 1); ; // perquè no desborde la variable word, el dilluns li diem dia 0 en lloc de
dia 1 .
  word tmp_hores = (int(ara.hour())) ;
  word tmp_minuts = (int(ara.minute())) ;
  Serial.println(tmp_dia);
  Serial.println(tmp_hores);
  Serial.println(tmp_minuts);
  tmp_dia = tmp_dia * 10000 ;
  tmp_hores = tmp_hores * 100 ;
  word Torne = (tmp_dia + tmp_hores + tmp_minuts) ;
  return (Torne);
}

void Rellotge::ficar_en_hora (String tmp, int Vcc)
{
  //  Serial.println("Vaig a ficar en hora");
  digitalWrite(Vcc, HIGH);
  delay (1000);
  String unix_clock_string = tmp.substring(29, 39);
  //  Serial.println (tmp);
```

```
        //  Serial.println ("Net");
        //  Serial.println (unix_clock_string);
        unsigned long unix_clock_int = unix_clock_string.toInt();   // Guarde en número per poder comparar
        if (unix_clock_int > 1559580181) //Comprove que l'hora pot ser correcta, i esta compresa entre l'any
2019 i el 2038.
        {
          // Serial.println(unix_clock_string);
          // Serial.println("Vaig a ficar el RTC amb el hora del servidor");
          delay(100);
          if (! rtc.begin())
          {
            Serial.println("No hi ha mòdul RTC");
            while (1);
          }
          rtc.adjust(DateTime(unix_clock_int)); //Fique en hora el RTC
           Serial.println(unix_clock_int);
           delay (1000);
        }
        digitalWrite(Vcc, LOW);
}
```

1.5.1.13 Codi font rellotge.h

```
#ifndef Rellotge_h
#define Rellotge_h

#include "Arduino.h"
#include "RTClib.h"

class Rellotge
{
  public :

    Rellotge(int RTC_Vcc);

    void ficar_en_hora(String tmp, int Vcc);
    word temps_actual(int Vcc);
};
#endif
```

1.5.1.14 Codi font rom.cpp

```
/*
  rom.cpp - Llibreria per engegar o aturar una electrovàlvula de reg 17/06/2019
  V 0.1.
*/

#include <Arduino.h>
#include <EEPROM.h>    // Llibreria per accedir a la ROM
#include "rom.h"

Rom::Rom()
{
}

void Rom::Guarda_en_Rom (String text, int direccio_1)
```

```
{
  int P_Control = text.indexOf("CT_ATZUCAC"); //Comprova si tenim la variable de control llegida al final del fitxer.
  if (P_Control > 0)
  {
    String hhmm = text.substring(33, 37); //Retalle les hores i minuts.
    String tmp1 = text.substring(38, 45); //Retalle els dies de la setmana que cal actuar.
    char d[7]; //Cree la matriu per guardar els dies de la setmana
    tmp1.toCharArray(d, 8); // Ja tenim els dies de la setmana guardats en una matriu
    for (int i = 0 ;  i  <  7 ; i++) // Mostrem els valors de la  matriu
    {
      String d_String = String(d[i]) ;
      if (d_String == "1")
      {
        int hores = hhmm.substring(0, 2).toInt();
        int minuts = hhmm.substring(2, 4).toInt();
        word temps_p = ((i * 10000) + (hores * 100) + minuts); // 16 bits (de 0 a 65535) el que ens permet ficar les 2359 del diumenge com 62359.
        ROM_ActualitzaWord(direccio_1, temps_p);
      }
      else
      {
        word temps_p = ((i * 10000) + 2500); // 16 bits (de 0 a 65535) Eixe dia NO cal regar per la qual cosa guarde en rom una hora que no existís com són les 25 hores
        ROM_ActualitzaWord(direccio_1, temps_p);
      }
      direccio_1++ ;  // Incrementem la adreça de memòria següent
      direccio_1++ ;  // Incrementem el segon byte en la adreça de memòria
    }
    delay(10000);
  }
}

void Rom::ROM_ActualitzaWord(int direccio_2, word temps_x)
{
  byte V_2 = (temps_x & 0xFF); // És quedem amb últim byte
  byte V_1 = ((temps_x >> 8) & 0xFF); // Ens quedem amb el de major pes
  EEPROM.update(direccio_2, V_2);
  EEPROM.update(direccio_2 + 1, V_1);
}

word Rom::Rom_LligWord(int direccio_3)
{
  int x_2 = EEPROM.read(direccio_3);
  int x_1 = EEPROM.read(direccio_3 + 1);
  word num = x_1 ; num = num << 8 | x_2 ; // Guarde els 2 bytes en un word.
  return (num);
}
```

1.5.1.15 Codi font rom.h

```
#ifndef Rom_h
#define Rom_h

#include "Arduino.h"
#include <EEPROM.h>    // Llibreria per accedir a la ROM

class Rom
```

```
{
  public :
    Rom();

    void Guarda_en_Rom (String text, int direccio_1);
    word Rom_LligWord(int direccio_3);

  private :
    void ROM_ActualitzaWord(int direccio_2, word temps_x);
};
#endif
```

1.5.1.16 Codi font RTClib.cpp

```
// Code by JeeLabs http://news.jeelabs.org/code/
// Released to the public domain! Enjoy!

#include "Wire.h"
#include "RTClib.h"
#ifdef __AVR__
#include <avr/pgmspace.h>
#elif defined(ESP8266)
#include <pgmspace.h>
#elif defined(ARDUINO_ARCH_SAMD)
// nothing special needed
#elif defined(ARDUINO_SAM_DUE)
#define PROGMEM
#define pgm_read_byte(addr) (*(const unsigned char *)(addr))
#define Wire Wire1
#endif

#if (ARDUINO >= 100)
#include <Arduino.h> // capital A so it is error prone on case-sensitive filesystems
// Macro to deal with the difference in I2C write functions from old and new Arduino versions.
#define _I2C_WRITE write
#define _I2C_READ  read
#else
#include <WProgram.h>
#define _I2C_WRITE send
#define _I2C_READ  receive
#endif


static uint8_t read_i2c_register(uint8_t addr, uint8_t reg) {
  Wire.beginTransmission(addr);
  Wire._I2C_WRITE((byte)reg);
  Wire.endTransmission();

  Wire.requestFrom(addr, (byte)1);
  return Wire._I2C_READ();
}

static void write_i2c_register(uint8_t addr, uint8_t reg, uint8_t val) {
  Wire.beginTransmission(addr);
  Wire._I2C_WRITE((byte)reg);
  Wire._I2C_WRITE((byte)val);
  Wire.endTransmission();
}
```

```
////////////////////////////////////////////////////////////////////////////
// utility code, some of this could be exposed in the DateTime API if needed

const uint8_t daysInMonth [] PROGMEM = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

// number of days since 2000/01/01, valid for 2001..2099
static uint16_t date2days(uint16_t y, uint8_t m, uint8_t d) {
  if (y >= 2000)
    y -= 2000;
  uint16_t days = d;
  for (uint8_t i = 1; i < m; ++i)
    days += pgm_read_byte(daysInMonth + i - 1);
  if (m > 2 && y % 4 == 0)
    ++days;
  return days + 365 * y + (y + 3) / 4 - 1;
}

static long time2long(uint16_t days, uint8_t h, uint8_t m, uint8_t s) {
  return ((days * 24L + h) * 60 + m) * 60 + s;
}

////////////////////////////////////////////////////////////////////////////
// DateTime implementation - ignores time zones and DST changes
// NOTE: also ignores leap seconds, see http://en.wikipedia.org/wiki/Leap_second

DateTime::DateTime (uint32_t t) {
  t -= SECONDS_FROM_1970_TO_2000;    // bring to 2000 timestamp from 1970

  ss = t % 60;
  t /= 60;
  mm = t % 60;
  t /= 60;
  hh = t % 24;
  uint16_t days = t / 24;
  uint8_t leap;
  for (yOff = 0; ; ++yOff) {
    leap = yOff % 4 == 0;
    if (days < 365 + leap)
      break;
    days -= 365 + leap;
  }
  for (m = 1; ; ++m) {
    uint8_t daysPerMonth = pgm_read_byte(daysInMonth + m - 1);
    if (leap && m == 2)
      ++daysPerMonth;
    if (days < daysPerMonth)
      break;
    days -= daysPerMonth;
  }
  d = days + 1;
}

DateTime::DateTime (uint16_t year, uint8_t month, uint8_t day, uint8_t hour, uint8_t min, uint8_t sec) {
  if (year >= 2000)
    year -= 2000;
```

```
  yOff = year;
  m = month;
  d = day;
  hh = hour;
  mm = min;
  ss = sec;
}

DateTime::DateTime (const DateTime& copy):
  yOff(copy.yOff),
  m(copy.m),
  d(copy.d),
  hh(copy.hh),
  mm(copy.mm),
  ss(copy.ss)
{}

static uint8_t conv2d(const char* p) {
  uint8_t v = 0;
  if ('0' <= *p && *p <= '9')
    v = *p - '0';
  return 10 * v + *++p - '0';
}

// A convenient constructor for using "the compiler's time":
//   DateTime now (__DATE__, __TIME__);
// NOTE: using F() would further reduce the RAM footprint, see below.
DateTime::DateTime (const char* date, const char* time) {
  // sample input: date = "Dec 26 2009", time = "12:34:56"
  yOff = conv2d(date + 9);
  // Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
  switch (date[0]) {
    case 'J': m = date[1] == 'a' ? 1 : m = date[2] == 'n' ? 6 : 7; break;
    case 'F': m = 2; break;
    case 'A': m = date[2] == 'r' ? 4 : 8; break;
    case 'M': m = date[2] == 'r' ? 3 : 5; break;
    case 'S': m = 9; break;
    case 'O': m = 10; break;
    case 'N': m = 11; break;
    case 'D': m = 12; break;
  }
  d = conv2d(date + 4);
  hh = conv2d(time);
  mm = conv2d(time + 3);
  ss = conv2d(time + 6);
}

// A convenient constructor for using "the compiler's time":
// This version will save RAM by using PROGMEM to store it by using the F macro.
//   DateTime now (F(__DATE__), F(__TIME__));
DateTime::DateTime (const __FlashStringHelper* date, const __FlashStringHelper* time) {
  // sample input: date = "Dec 26 2009", time = "12:34:56"
  char buff[11];
  memcpy_P(buff, date, 11);
  yOff = conv2d(buff + 9);
  // Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
  switch (buff[0]) {
```

```
    case 'J': m = buff[1] == 'a' ? 1 : m = buff[2] == 'n' ? 6 : 7; break;
    case 'F': m = 2; break;
    case 'A': m = buff[2] == 'r' ? 4 : 8; break;
    case 'M': m = buff[2] == 'r' ? 3 : 5; break;
    case 'S': m = 9; break;
    case 'O': m = 10; break;
    case 'N': m = 11; break;
    case 'D': m = 12; break;
  }
  d = conv2d(buff + 4);
  memcpy_P(buff, time, 8);
  hh = conv2d(buff);
  mm = conv2d(buff + 3);
  ss = conv2d(buff + 6);
}

uint8_t DateTime::dayOfTheWeek() const {
  uint16_t day = date2days(yOff, m, d);
  return (day + 6) % 7; // Jan 1, 2000 is a Saturday, i.e. returns 6
}

uint32_t DateTime::unixtime(void) const {
  uint32_t t;
  uint16_t days = date2days(yOff, m, d);
  t = time2long(days, hh, mm, ss);
  t += SECONDS_FROM_1970_TO_2000;  // seconds from 1970 to 2000

  return t;
}

long DateTime::secondstime(void) const {
  long t;
  uint16_t days = date2days(yOff, m, d);
  t = time2long(days, hh, mm, ss);
  return t;
}

DateTime DateTime::operator+(const TimeSpan& span) {
  return DateTime(unixtime() + span.totalseconds());
}

DateTime DateTime::operator-(const TimeSpan& span) {
  return DateTime(unixtime() - span.totalseconds());
}

TimeSpan DateTime::operator-(const DateTime& right) {
  return TimeSpan(unixtime() - right.unixtime());
}

////////////////////////////////////////////////////////////////////////////
// TimeSpan implementation

TimeSpan::TimeSpan (int32_t seconds):
  _seconds(seconds)
{}

TimeSpan::TimeSpan (int16_t days, int8_t hours, int8_t minutes, int8_t seconds):
```

```
  _seconds((int32_t)days * 86400L + (int32_t)hours * 3600 + (int32_t)minutes * 60 + seconds)
{}

TimeSpan::TimeSpan (const TimeSpan& copy):
  _seconds(copy._seconds)
{}

TimeSpan TimeSpan::operator+(const TimeSpan& right) {
  return TimeSpan(_seconds + right._seconds);
}

TimeSpan TimeSpan::operator-(const TimeSpan& right) {
  return TimeSpan(_seconds - right._seconds);
}

////////////////////////////////////////////////////////////////////////////////
// RTC_DS1307 implementation

static uint8_t bcd2bin (uint8_t val) {
  return val - 6 * (val >> 4);
}
static uint8_t bin2bcd (uint8_t val) {
  return val + 6 * (val / 10);
}

boolean RTC_DS1307::begin(void) {
  Wire.begin();
  return true;
}

uint8_t RTC_DS1307::isrunning(void) {
  Wire.beginTransmission(DS1307_ADDRESS);
  Wire._I2C_WRITE((byte)0);
  Wire.endTransmission();

  Wire.requestFrom(DS1307_ADDRESS, 1);
  uint8_t ss = Wire._I2C_READ();
  return !(ss >> 7);
}

void RTC_DS1307::adjust(const DateTime& dt) {
  Wire.beginTransmission(DS1307_ADDRESS);
  Wire._I2C_WRITE((byte)0); // start at location 0
  Wire._I2C_WRITE(bin2bcd(dt.second()));
  Wire._I2C_WRITE(bin2bcd(dt.minute()));
  Wire._I2C_WRITE(bin2bcd(dt.hour()));
  Wire._I2C_WRITE(bin2bcd(0));
  Wire._I2C_WRITE(bin2bcd(dt.day()));
  Wire._I2C_WRITE(bin2bcd(dt.month()));
  Wire._I2C_WRITE(bin2bcd(dt.year() - 2000));
  Wire.endTransmission();
}

DateTime RTC_DS1307::now() {
  Wire.beginTransmission(DS1307_ADDRESS);
  Wire._I2C_WRITE((byte)0);
  Wire.endTransmission();
```

```
  Wire.requestFrom(DS1307_ADDRESS, 7);
  uint8_t ss = bcd2bin(Wire._I2C_READ() & 0x7F);
  uint8_t mm = bcd2bin(Wire._I2C_READ());
  uint8_t hh = bcd2bin(Wire._I2C_READ());
  Wire._I2C_READ();
  uint8_t d = bcd2bin(Wire._I2C_READ());
  uint8_t m = bcd2bin(Wire._I2C_READ());
  uint16_t y = bcd2bin(Wire._I2C_READ()) + 2000;

  return DateTime (y, m, d, hh, mm, ss);
}

Ds1307SqwPinMode RTC_DS1307::readSqwPinMode() {
  int mode;

  Wire.beginTransmission(DS1307_ADDRESS);
  Wire._I2C_WRITE(DS1307_CONTROL);
  Wire.endTransmission();

  Wire.requestFrom((uint8_t)DS1307_ADDRESS, (uint8_t)1);
  mode = Wire._I2C_READ();

  mode &= 0x93;
  return static_cast<Ds1307SqwPinMode>(mode);
}

void RTC_DS1307::writeSqwPinMode(Ds1307SqwPinMode mode) {
  Wire.beginTransmission(DS1307_ADDRESS);
  Wire._I2C_WRITE(DS1307_CONTROL);
  Wire._I2C_WRITE(mode);
  Wire.endTransmission();
}

void RTC_DS1307::readnvram(uint8_t* buf, uint8_t size, uint8_t address) {
  int addrByte = DS1307_NVRAM + address;
  Wire.beginTransmission(DS1307_ADDRESS);
  Wire._I2C_WRITE(addrByte);
  Wire.endTransmission();

  Wire.requestFrom((uint8_t) DS1307_ADDRESS, size);
  for (uint8_t pos = 0; pos < size; ++pos) {
    buf[pos] = Wire._I2C_READ();
  }
}

void RTC_DS1307::writenvram(uint8_t address, uint8_t* buf, uint8_t size) {
  int addrByte = DS1307_NVRAM + address;
  Wire.beginTransmission(DS1307_ADDRESS);
  Wire._I2C_WRITE(addrByte);
  for (uint8_t pos = 0; pos < size; ++pos) {
    Wire._I2C_WRITE(buf[pos]);
  }
  Wire.endTransmission();
}

uint8_t RTC_DS1307::readnvram(uint8_t address) {
```

```
  uint8_t data;
  readnvram(&data, 1, address);
  return data;
}

void RTC_DS1307::writenvram(uint8_t address, uint8_t data) {
  writenvram(address, &data, 1);
}

//////////////////////////////////////////////////////////////////////
// RTC_Millis implementation

long RTC_Millis::offset = 0;

void RTC_Millis::adjust(const DateTime& dt) {
  offset = dt.unixtime() - millis() / 1000;
}

DateTime RTC_Millis::now() {
  return (uint32_t)(offset + millis() / 1000);
}

//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
// RTC_PCF8563 implementation

boolean RTC_PCF8523::begin(void) {
  Wire.begin();
  return true;
}

boolean RTC_PCF8523::initialized(void) {
  Wire.beginTransmission(PCF8523_ADDRESS);
  Wire._I2C_WRITE((byte)PCF8523_CONTROL_3);
  Wire.endTransmission();

  Wire.requestFrom(PCF8523_ADDRESS, 1);
  uint8_t ss = Wire._I2C_READ();
  return ((ss & 0xE0) != 0xE0);
}

void RTC_PCF8523::adjust(const DateTime& dt) {
  Wire.beginTransmission(PCF8523_ADDRESS);
  Wire._I2C_WRITE((byte)3); // start at location 3
  Wire._I2C_WRITE(bin2bcd(dt.second()));
  Wire._I2C_WRITE(bin2bcd(dt.minute()));
  Wire._I2C_WRITE(bin2bcd(dt.hour()));
  Wire._I2C_WRITE(bin2bcd(dt.day()));
  Wire._I2C_WRITE(bin2bcd(0)); // skip weekdays
  Wire._I2C_WRITE(bin2bcd(dt.month()));
  Wire._I2C_WRITE(bin2bcd(dt.year() - 2000));
  Wire.endTransmission();

  // set to battery switchover mode
  Wire.beginTransmission(PCF8523_ADDRESS);
  Wire._I2C_WRITE((byte)PCF8523_CONTROL_3);
```

```
  Wire._I2C_WRITE((byte)0x00);
  Wire.endTransmission();
}

DateTime RTC_PCF8523::now() {
  Wire.beginTransmission(PCF8523_ADDRESS);
  Wire._I2C_WRITE((byte)3);
  Wire.endTransmission();

  Wire.requestFrom(PCF8523_ADDRESS, 7);
  uint8_t ss = bcd2bin(Wire._I2C_READ() & 0x7F);
  uint8_t mm = bcd2bin(Wire._I2C_READ());
  uint8_t hh = bcd2bin(Wire._I2C_READ());
  uint8_t d = bcd2bin(Wire._I2C_READ());
  Wire._I2C_READ();  // skip 'weekdays'
  uint8_t m = bcd2bin(Wire._I2C_READ());
  uint16_t y = bcd2bin(Wire._I2C_READ()) + 2000;

  return DateTime (y, m, d, hh, mm, ss);
}

Pcf8523SqwPinMode RTC_PCF8523::readSqwPinMode() {
  int mode;

  Wire.beginTransmission(PCF8523_ADDRESS);
  Wire._I2C_WRITE(PCF8523_CLKOUTCONTROL);
  Wire.endTransmission();

  Wire.requestFrom((uint8_t)PCF8523_ADDRESS, (uint8_t)1);
  mode = Wire._I2C_READ();

  mode >>= 3;
  mode &= 0x7;
  return static_cast<Pcf8523SqwPinMode>(mode);
}

void RTC_PCF8523::writeSqwPinMode(Pcf8523SqwPinMode mode) {
  Wire.beginTransmission(PCF8523_ADDRESS);
  Wire._I2C_WRITE(PCF8523_CLKOUTCONTROL);
  Wire._I2C_WRITE(mode << 3);
  Wire.endTransmission();
}



////////////////////////////////////////////////////////////////////////////
// RTC_DS3231 implementation

boolean RTC_DS3231::begin(void) {
  Wire.begin();
  return true;
}

bool RTC_DS3231::lostPower(void) {
  return (read_i2c_register(DS3231_ADDRESS, DS3231_STATUSREG) >> 7);
}
```

```
void RTC_DS3231::adjust(const DateTime& dt) {
  Wire.beginTransmission(DS3231_ADDRESS);
  Wire._I2C_WRITE((byte)0); // start at location 0
  Wire._I2C_WRITE(bin2bcd(dt.second()));
  Wire._I2C_WRITE(bin2bcd(dt.minute()));
  Wire._I2C_WRITE(bin2bcd(dt.hour()));
  Wire._I2C_WRITE(bin2bcd(0));
  Wire._I2C_WRITE(bin2bcd(dt.day()));
  Wire._I2C_WRITE(bin2bcd(dt.month()));
  Wire._I2C_WRITE(bin2bcd(dt.year() - 2000));
  Wire.endTransmission();

  uint8_t statreg = read_i2c_register(DS3231_ADDRESS, DS3231_STATUSREG);
  statreg &= ~0x80; // flip OSF bit
  write_i2c_register(DS3231_ADDRESS, DS3231_STATUSREG, statreg);
}

DateTime RTC_DS3231::now() {
  Wire.beginTransmission(DS3231_ADDRESS);
  Wire._I2C_WRITE((byte)0);
  Wire.endTransmission();

  Wire.requestFrom(DS3231_ADDRESS, 7);
  uint8_t ss = bcd2bin(Wire._I2C_READ() & 0x7F);
  uint8_t mm = bcd2bin(Wire._I2C_READ());
  uint8_t hh = bcd2bin(Wire._I2C_READ());
  Wire._I2C_READ();
  uint8_t d = bcd2bin(Wire._I2C_READ());
  uint8_t m = bcd2bin(Wire._I2C_READ());
  uint16_t y = bcd2bin(Wire._I2C_READ()) + 2000;

  return DateTime (y, m, d, hh, mm, ss);
}

Ds3231SqwPinMode RTC_DS3231::readSqwPinMode() {
  int mode;

  Wire.beginTransmission(DS3231_ADDRESS);
  Wire._I2C_WRITE(DS3231_CONTROL);
  Wire.endTransmission();

  Wire.requestFrom((uint8_t)DS3231_ADDRESS, (uint8_t)1);
  mode = Wire._I2C_READ();

  mode &= 0x93;
  return static_cast<Ds3231SqwPinMode>(mode);
}

void RTC_DS3231::writeSqwPinMode(Ds3231SqwPinMode mode) {
  uint8_t ctrl;
  ctrl = read_i2c_register(DS3231_ADDRESS, DS3231_CONTROL);

  ctrl &= ~0x04; // turn off INTCON
  ctrl &= ~0x18; // set freq bits to 0

  if (mode == DS3231_OFF) {
```

```
    ctrl |= 0x04; // turn on INTCN
  } else {
    ctrl |= mode;
  }
  write_i2c_register(DS3231_ADDRESS, DS3231_CONTROL, ctrl);

  //Serial.println( read_i2c_register(DS3231_ADDRESS, DS3231_CONTROL), HEX);
}
```

1.5.1.17 Codi font RTClib.h

```
// Code by JeeLabs http://news.jeelabs.org/code/
// Released to the public domain! Enjoy!

#ifndef _RTCLIB_H_
#define _RTCLIB_H_

#include <Arduino.h>
class TimeSpan;


#define PCF8523_ADDRESS      0x68
#define PCF8523_CLKOUTCONTROL 0x0F
#define PCF8523_CONTROL_3     0x02

#define DS1307_ADDRESS  0x68   // 68
#define DS1307_CONTROL  0x07
#define DS1307_NVRAM    0x08

#define DS3231_ADDRESS  0x68
#define DS3231_CONTROL  0x0E
#define DS3231_STATUSREG 0x0F

#define SECONDS_PER_DAY 86400L

#define SECONDS_FROM_1970_TO_2000 946684800



// Simple general-purpose date/time class (no TZ / DST / leap second handling!)
class DateTime {
  public:
    DateTime (uint32_t t = 0);
    DateTime (uint16_t year, uint8_t month, uint8_t day,
            uint8_t hour = 0, uint8_t min = 0, uint8_t sec = 0);
    DateTime (const DateTime& copy);
    DateTime (const char* date, const char* time);
    DateTime (const __FlashStringHelper* date, const __FlashStringHelper* time);
    uint16_t year() const       {
      return 2000 + yOff;
    }
    uint8_t month() const       {
      return m;
    }
    uint8_t day() const         {
      return d;
    }
    uint8_t hour() const        {
```

```
    return hh;
  }
  uint8_t minute() const     {
    return mm;
  }
  uint8_t second() const     {
    return ss;
  }
  uint8_t dayOfTheWeek() const;

  // 32-bit times as seconds since 1/1/2000
  long secondstime() const;
  // 32-bit times as seconds since 1/1/1970
  uint32_t unixtime(void) const;

  DateTime operator+(const TimeSpan& span);
  DateTime operator-(const TimeSpan& span);
  TimeSpan operator-(const DateTime& right);

 protected:
  uint8_t yOff, m, d, hh, mm, ss;
};

// Timespan which can represent changes in time with seconds accuracy.
class TimeSpan {
 public:
  TimeSpan (int32_t seconds = 0);
  TimeSpan (int16_t days, int8_t hours, int8_t minutes, int8_t seconds);
  TimeSpan (const TimeSpan& copy);
  int16_t days() const        {
    return _seconds / 86400L;
  }
  int8_t  hours() const       {
    return _seconds / 3600 % 24;
  }
  int8_t  minutes() const     {
    return _seconds / 60 % 60;
  }
  int8_t  seconds() const     {
    return _seconds % 60;
  }
  int32_t totalseconds() const {
    return _seconds;
  }

  TimeSpan operator+(const TimeSpan& right);
  TimeSpan operator-(const TimeSpan& right);

 protected:
  int32_t _seconds;
};

// RTC based on the DS1307 chip connected via I2C and the Wire library
enum Ds1307SqwPinMode { OFF = 0x00, ON = 0x80, SquareWave1HZ = 0x10, SquareWave4kHz =
0x11, SquareWave8kHz = 0x12, SquareWave32kHz = 0x13 };

class RTC_DS1307 {
```

```
  public:
    boolean begin(void);
    static void adjust(const DateTime& dt);
    uint8_t isrunning(void);
    static DateTime now();
    static Ds1307SqwPinMode readSqwPinMode();
    static void writeSqwPinMode(Ds1307SqwPinMode mode);
    uint8_t readnvram(uint8_t address);
    void readnvram(uint8_t* buf, uint8_t size, uint8_t address);
    void writenvram(uint8_t address, uint8_t data);
    void writenvram(uint8_t address, uint8_t* buf, uint8_t size);
};

// RTC based on the DS3231 chip connected via I2C and the Wire library
enum Ds3231SqwPinMode { DS3231_OFF = 0x01, DS3231_SquareWave1Hz = 0x00,
DS3231_SquareWave1kHz = 0x08, DS3231_SquareWave4kHz = 0x10, DS3231_SquareWave8kHz = 0x18
};

class RTC_DS3231 {
  public:
    boolean begin(void);
    static void adjust(const DateTime& dt);
    bool lostPower(void);
    static DateTime now();
    static Ds3231SqwPinMode readSqwPinMode();
    static void writeSqwPinMode(Ds3231SqwPinMode mode);
};


// RTC based on the PCF8523 chip connected via I2C and the Wire library
enum Pcf8523SqwPinMode { PCF8523_OFF = 7, PCF8523_SquareWave1HZ = 6,
PCF8523_SquareWave32HZ = 5, PCF8523_SquareWave1kHz = 4, PCF8523_SquareWave4kHz = 3,
PCF8523_SquareWave8kHz = 2, PCF8523_SquareWave16kHz = 1, PCF8523_SquareWave32kHz = 0 };

class RTC_PCF8523 {
  public:
    boolean begin(void);
    void adjust(const DateTime& dt);
    boolean initialized(void);
    static DateTime now();

    Pcf8523SqwPinMode readSqwPinMode();
    void writeSqwPinMode(Pcf8523SqwPinMode mode);
};

// RTC using the internal millis() clock, has to be initialized before use
// NOTE: this clock won't be correct once the millis() timer rolls over (>49d?)
class RTC_Millis {
  public:
    static void begin(const DateTime& dt) {
      adjust(dt);
    }
    static void adjust(const DateTime& dt);
    static DateTime now();

  protected:
    static long offset;
```

```
};

#endif // _RTCLIB_H_
```

1.5.1.18 Codi font Sim.h

```
/*
  Sim.cpp - 01/09/2019
  V 0.9.
  Llibreria funcionar amb la llibreria SoftwareSerial modificada per poder modificar els pins de RX i TX amb les funcions setTX i setRX.
*/
#include <Arduino.h>
#include "SoftwareSerial.h"
#include "Sim.h"
SoftwareSerial SIM800;

Sim::Sim(uint8_t RX, uint8_t TX)
{
  SIM800.setTX(TX); // TX del port sèrie virtual on col·loque el mòdul de 2G
  SIM800.setRX(RX); // RX del port sèrie virtual on col·loque el mòdul de 2G
  SIM800.begin(1200); // Configure la velocitat del port sèrie del mòdul 2G a 1200 bauds
}

void Sim::Resposta_log(String msg)
{
  String resposta ;
  for (int z = 0; z < 3000; z++) {
    if (SIM800.available())
    {
      char c = SIM800.read();
      resposta = String(resposta + c);
    }
  }
}

String Sim::llegir()
{
  String brut ;
  SIM800.println("AT+HTTPREAD");
  // delay(2000);
  for (int z = 0; z < 4000; z++)  // Espere uns quants cicles de rellotge a obtenir la web en format text.
  {
    if (SIM800.available())
    {
      char c = SIM800.read();
      brut = String(brut + c);
    }
  }
  // El que hem rebut.
  Serial.println(brut);
  return (String(brut));
}

void Sim::borrar()
{
  for (int z = 0; z < 1000; z++) {
    while (SIM800.available()) {
```

```
        SIM800.read(); // Esborre el que hem rebut fins al moment
      }
     }
    }

    void Sim::enviar(String Envie, int Espere, String msg)
    {
     borrar(); // Esborre la memòria cau
     SIM800.println(Envie);
     delay(Espere);
     Resposta_log(msg);
    }

    void Sim::iniciar()
    {
     enviar ("AT", 6000, "Inicie comunicació");
     enviar ("AT+SAPBR=3,1,\"CONTYPE\",\"GPRS\"", 2000, "Connexió GSM");
     enviar ("AT+SAPBR=3,1,\"APN\",\"TM\"", 2000, "Configuració APN");  // Esta configuració depèn de
la companyia triada
     enviar ("AT+SAPBR=1,1", 3000, "Continue iniciant");
    }

    String Sim::obri(String pagina)
    {
     enviar ("AT+HTTPINIT", 1000, "Preparant connexio http");
     String web = "AT+HTTPPARA=URL,\"" + pagina + "\"" ;
     enviar (web, 1000, "Obrint web");
     enviar ("AT+HTTPACTION=0", 2000, "Resposta");
     web = llegir();
     enviar ("AT+HTTPTERM", 10, "Tancant Connexio");
     return (web) ;
    }
```

1.5.1.19 Codi font Sim.h

```
    #ifndef Sim_h
    #define Sim_h

    #include "Arduino.h"
    #include "SoftwareSerial.h"

    class Sim
    {
     public :
       Sim(uint8_t RX, uint8_t TX);
       void iniciar();
       String obri(String pagina);

     private:

       void enviar(String Envie, int Espere, String msg);
       void borrar();
       void Resposta_log(String msg);
       String llegir ();

    };
    #endif
```

1.5.1.20 Codi font SoftwareSerial.cpp

```
/*
  SoftwareSerial.cpp (formerly NewSoftSerial.cpp) -
  Multi-instance software serial library for Arduino/Wiring
  -- Interrupt-driven receive and other improvements by ladyada
   (http://ladyada.net)
  -- Tuning, circular buffer, derivation from class Print/Stream,
   multi-instance support, porting to 8MHz processors,
   various optimizations, PROGMEM delay tables, inverse logic and
   direct port writing by Mikal Hart (http://www.arduiniana.org)
  -- Pin change interrupt macros by Paul Stoffregen (http://www.pjrc.com)
  -- 20MHz processor support by Garrett Mace (http://www.macetech.com)
  -- ATmega1280/2560 support by Brett Hagman (http://www.roguerobotics.com/)

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA

  The latest version of this library can always be found at
  http://arduiniana.org.
*/

// Fichero modificado para poder




// When set, _DEBUG co-opts pins 11 and 13 for debugging with an
// oscilloscope or logic analyzer.  Beware: it also slightly modifies
// the bit times, so don't rely on it too much at high baud rates
#define _DEBUG 0
#define _DEBUG_PIN1 11
#define _DEBUG_PIN2 13
//
// Includes
//
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <Arduino.h>
#include "SoftwareSerial.h"


#if defined(__MK20DX128__) || defined(__MK20DX256__) || defined(__MKL26Z64__) ||
defined(__MK64FX512__) || defined(__MK66FX1M0__)

SoftwareSerial::SoftwareSerial(uint8_t rxPin, uint8_t txPin, bool inverse_logic /* = false */)
```

```
{
  buffer_overflow = false;
  if (rxPin == 0 && txPin == 1) {
    port = &Serial1;
    return;
  } else if (rxPin == 9 && txPin == 10) {
    port = &Serial2;
    return;
  } else if (rxPin == 7 && txPin == 8) {
    port = &Serial3;
    return;
  }
  port = NULL;
  pinMode(txPin, OUTPUT);
  pinMode(rxPin, INPUT_PULLUP);
  txpin = txPin;
  rxpin = rxPin;
  txreg = portOutputRegister(digitalPinToPort(txPin));
  rxreg = portInputRegister(digitalPinToPort(rxPin));
  cycles_per_bit = 0;
}


// Borrar 1


// SoftwareSerial::SoftwareSerial()
// {
// }
void SoftwareSerial::asignar_port(uint8_t rxPin, uint8_t txPin, bool inverse_logic /* = false */)
{
  buffer_overflow = false;
  if (rxPin == 0 && txPin == 1) {
    port = &Serial1;
    return;
  } else if (rxPin == 9 && txPin == 10) {
    port = &Serial2;
    return;
  } else if (rxPin == 7 && txPin == 8) {
    port = &Serial3;
    return;
  }
  port = NULL;
  pinMode(txPin, OUTPUT);
  pinMode(rxPin, INPUT_PULLUP);
  txpin = txPin;
  rxpin = rxPin;
  txreg = portOutputRegister(digitalPinToPort(txPin));
  rxreg = portInputRegister(digitalPinToPort(rxPin));
  cycles_per_bit = 0;
}

void SoftwareSerial::SoftwareSerial() :
  _rx_delay_centering(0),
  _rx_delay_intrabit(0),
  _rx_delay_stopbit(0),
  _tx_delay(0),
```

```
  _buffer_overflow(false),
  _inverse_logic(inverse_logic)
{
  buffer_overflow = false;
  if (rxPin == 0 && txPin == 1) {
    port = &Serial1;
    return;
  } else if (rxPin == 9 && txPin == 10) {
    port = &Serial2;
    return;
  } else if (rxPin == 7 && txPin == 8) {
    port = &Serial3;
    return;
  }
  port = NULL;
  pinMode(txPin, OUTPUT);
  pinMode(rxPin, INPUT_PULLUP);
  txpin = txPin;
  rxpin = rxPin;
  txreg = portOutputRegister(digitalPinToPort(txPin));
  rxreg = portInputRegister(digitalPinToPort(rxPin));
  cycles_per_bit = 0;
}
// Borrar 1 fi


void SoftwareSerial::begin(unsigned long speed)
{
  if (port) {
    port->begin(speed);
  } else {
    cycles_per_bit = (uint32_t)(F_CPU + speed / 2) / speed;
    ARM_DEMCR |= ARM_DEMCR_TRCENA;
    ARM_DWT_CTRL |= ARM_DWT_CTRL_CYCCNTENA;
  }
}

void SoftwareSerial::end()
{
  if (port) {
    port->end();
    port = NULL;
  } else {
    pinMode(txpin, INPUT);
    pinMode(rxpin, INPUT);
  }
  cycles_per_bit = 0;
}

// The worst case expected length of any interrupt routines.  If an
// interrupt runs longer than this number of cycles, it can disrupt
// the transmit waveform.  Increasing this number causes SoftwareSerial
// to hog the CPU longer, delaying all interrupt response for other
// libraries, so this should be made as small as possible but still
// ensure accurate transmit waveforms.
#define WORST_INTERRUPT_CYCLES 360
```

Controlador de reg per degoteig

```
static void wait_for_target(uint32_t begin, uint32_t target)
{
  if (target - (ARM_DWT_CYCCNT - begin) > WORST_INTERRUPT_CYCLES + 20) {
    uint32_t pretarget = target - WORST_INTERRUPT_CYCLES;
    //digitalWriteFast(12, HIGH);
    interrupts();
    while (ARM_DWT_CYCCNT - begin < pretarget) ; // wait
    noInterrupts();
    //digitalWriteFast(12, LOW);
  }
  while (ARM_DWT_CYCCNT - begin < target) ; // wait
}

size_t SoftwareSerial::write(uint8_t b)
{
  elapsedMicros elapsed;
  uint32_t target;
  uint8_t mask;
  uint32_t begin_cycle;

  // use hardware serial, if possible
  if (port) return port->write(b);
  if (cycles_per_bit == 0) return 0;
  ARM_DEMCR |= ARM_DEMCR_TRCENA;
  ARM_DWT_CTRL |= ARM_DWT_CTRL_CYCCNTENA;
  // start bit
  target = cycles_per_bit;
  noInterrupts();
  begin_cycle = ARM_DWT_CYCCNT;
  *txreg = 0;
  wait_for_target(begin_cycle, target);
  // 8 data bits
  for (mask = 1; mask; mask <<= 1) {
    *txreg = (b & mask) ? 1 : 0;
    target += cycles_per_bit;
    wait_for_target(begin_cycle, target);
  }
  // stop bit
  *txreg = 1;
  interrupts();
  target += cycles_per_bit;
  while (ARM_DWT_CYCCNT - begin_cycle < target) ; // wait
  return 1;
}

void SoftwareSerial::flush()
{
  if (port) port->flush();
}

// TODO implement reception using pin change DMA capturing
// ARM_DWT_CYCCNT and the bitband mapped GPIO_PDIR register
// to a circular buffer (8 bytes per event... memory intensive)

int SoftwareSerial::available()
{
  if (port) return port->available();
```

```
  return 0;
}

int SoftwareSerial::peek()
{
  if (port) return port->peek();
  return -1;
}

int SoftwareSerial::read()
{
  if (port) return port->read();
  return -1;
}

#else

//
// Lookup table
//
typedef struct _DELAY_TABLE
{
  long baud;
  unsigned short rx_delay_centering;
  unsigned short rx_delay_intrabit;
  unsigned short rx_delay_stopbit;
  unsigned short tx_delay;
} DELAY_TABLE;

#if F_CPU == 16000000

static const DELAY_TABLE PROGMEM table[] =
{
  //  baud    rxcenter   rxintra    rxstop    tx
  { 115200,   1,         17,        17,       12,    },
  { 57600,    10,        37,        37,       33,    },
  { 38400,    25,        57,        57,       54,    },
  { 31250,    31,        70,        70,       68,    },
  { 28800,    34,        77,        77,       74,    },
  { 19200,    54,        117,       117,      114,   },
  { 14400,    74,        156,       156,      153,   },
  { 9600,     114,       236,       236,      233,   },
  { 4800,     233,       474,       474,      471,   },
  { 2400,     471,       950,       950,      947,   },
  { 1200,     947,       1902,      1902,     1899,  },
  { 600,      1902,      3804,      3804,     3800,  },
  { 300,      3804,      7617,      7617,     7614,  },
};

const int XMIT_START_ADJUSTMENT = 5;

#elif F_CPU == 8000000

static const DELAY_TABLE table[] PROGMEM =
{
  //  baud    rxcenter    rxintra    rxstop  tx
  { 115200,   1,          5,         5,      3,      },
```

```
  { 57600,   1,       15,      15,    13,    },
  { 38400,   2,       25,      26,    23,    },
  { 31250,   7,       32,      33,    29,    },
  { 28800,   11,      35,      35,    32,    },
  { 19200,   20,      55,      55,    52,    },
  { 14400,   30,      75,      75,    72,    },
  { 9600,    50,      114,     114,   112,   },
  { 4800,    110,     233,     233,   230,   },
  { 2400,    229,     472,     472,   469,   },
  { 1200,    467,     948,     948,   945,   },
  { 600,     948,     1895,    1895,  1890,  },
  { 300,     1895,    3805,    3805,  3802,  },
};

const int XMIT_START_ADJUSTMENT = 4;

#elif F_CPU == 20000000

// 20MHz support courtesy of the good people at macegr.com.
// Thanks, Garrett!

static const DELAY_TABLE PROGMEM table[] =
{
  //  baud    rxcenter    rxintra    rxstop  tx
  { 115200,   3,        21,       21,    18,    },
  { 57600,    20,       43,       43,    41,    },
  { 38400,    37,       73,       73,    70,    },
  { 31250,    45,       89,       89,    88,    },
  { 28800,    46,       98,       98,    95,    },
  { 19200,    71,       148,      148,   145,   },
  { 14400,    96,       197,      197,   194,   },
  { 9600,     146,      297,      297,   294,   },
  { 4800,     296,      595,      595,   592,   },
  { 2400,     592,      1189,     1189,  1186,  },
  { 1200,     1187,     2379,     2379,  2376,  },
  { 600,      2379,     4759,     4759,  4755,  },
  { 300,      4759,     9523,     9523,  9520,  },
};

const int XMIT_START_ADJUSTMENT = 6;

#else

#error This version of SoftwareSerial supports only 20, 16 and 8MHz processors

#endif

//
// Statics
//
SoftwareSerial *SoftwareSerial::active_object = 0;
char SoftwareSerial::_receive_buffer[_SS_MAX_RX_BUFF];
volatile uint8_t SoftwareSerial::_receive_buffer_tail = 0;
volatile uint8_t SoftwareSerial::_receive_buffer_head = 0;

//
// Debugging
```

```
//
// This function generates a brief pulse
// for debugging or measuring on an oscilloscope.
inline void DebugPulse(uint8_t pin, uint8_t count)
{
#if _DEBUG
  volatile uint8_t *pport = portOutputRegister(digitalPinToPort(pin));

  uint8_t val = *pport;
  while (count--)
  {
    *pport = val | digitalPinToBitMask(pin);
    *pport = val;
  }
#endif
}

//
// Private methods
//

/* static */
inline void SoftwareSerial::tunedDelay(uint16_t delay) {
  uint8_t tmp = 0;

  asm volatile("sbiw    %0, 0x01 \n\t"
               "ldi %1, 0xFF \n\t"
               "cpi %A0, 0xFF \n\t"
               "cpc %B0, %1 \n\t"
               "brne .-10 \n\t"
               : "+r" (delay), "+a" (tmp)
               : "0" (delay)
              );
}

// This function sets the current object as the "listening"
// one and returns true if it replaces another
bool SoftwareSerial::listen()
{
  if (active_object != this)
  {
    _buffer_overflow = false;
    uint8_t oldSREG = SREG;
    cli();
    _receive_buffer_head = _receive_buffer_tail = 0;
    active_object = this;
    SREG = oldSREG;
    return true;
  }

  return false;
}

//
// The receive routine called by the interrupt handler
//
void SoftwareSerial::recv()
```

```
{

#if GCC_VERSION < 40302
  // Work-around for avr-gcc 4.3.0 OSX version bug
  // Preserve the registers that the compiler misses
  // (courtesy of Arduino forum user *etracer*)
  asm volatile(
    "push r18 \n\t"
    "push r19 \n\t"
    "push r20 \n\t"
    "push r21 \n\t"
    "push r22 \n\t"
    "push r23 \n\t"
    "push r26 \n\t"
    "push r27 \n\t"
    ::);
#endif

  uint8_t d = 0;

  // If RX line is high, then we don't see any start bit
  // so interrupt is probably not for us
  if (_inverse_logic ? rx_pin_read() : !rx_pin_read())
  {
    // Wait approximately 1/2 of a bit width to "center" the sample
    tunedDelay(_rx_delay_centering);
    DebugPulse(_DEBUG_PIN2, 1);

    // Read each of the 8 bits
    for (uint8_t i = 0x1; i; i <<= 1)
    {
      tunedDelay(_rx_delay_intrabit);
      DebugPulse(_DEBUG_PIN2, 1);
      uint8_t noti = ~i;
      if (rx_pin_read())
        d |= i;
      else // else clause added to ensure function timing is ~balanced
        d &= noti;
    }

    // skip the stop bit
    tunedDelay(_rx_delay_stopbit);
    DebugPulse(_DEBUG_PIN2, 1);

    if (_inverse_logic)
      d = ~d;

    // if buffer full, set the overflow flag and return
    if ((_receive_buffer_tail + 1) % _SS_MAX_RX_BUFF != _receive_buffer_head)
    {
      // save new data in buffer: tail points to where byte goes
      _receive_buffer[_receive_buffer_tail] = d; // save new byte
      _receive_buffer_tail = (_receive_buffer_tail + 1) % _SS_MAX_RX_BUFF;
    }
    else
    {
#if _DEBUG // for scope: pulse pin as overflow indictator
```

```
      DebugPulse(_DEBUG_PIN1, 1);
#endif
      _buffer_overflow = true;
    }
  }

#if GCC_VERSION < 40302
  // Work-around for avr-gcc 4.3.0 OSX version bug
  // Restore the registers that the compiler misses
  asm volatile(
    "pop r27 \n\t"
    "pop r26 \n\t"
    "pop r23 \n\t"
    "pop r22 \n\t"
    "pop r21 \n\t"
    "pop r20 \n\t"
    "pop r19 \n\t"
    "pop r18 \n\t"
    ::);
#endif
}

void SoftwareSerial::tx_pin_write(uint8_t pin_state)
{
  if (pin_state == LOW)
    *_transmitPortRegister &= ~_transmitBitMask;
  else
    *_transmitPortRegister |= _transmitBitMask;
}

uint8_t SoftwareSerial::rx_pin_read()
{
  return *_receivePortRegister & _receiveBitMask;
}

//
// Interrupt handling
//

/* static */
inline void SoftwareSerial::handle_interrupt()
{
  if (active_object)
  {
    active_object->recv();
  }
}

#if defined(PCINT0_vect)
ISR(PCINT0_vect)
{
  SoftwareSerial::handle_interrupt();
}
#endif

#if defined(PCINT1_vect)
ISR(PCINT1_vect)
```

```
{
  SoftwareSerial::handle_interrupt();
}
#endif

#if defined(PCINT2_vect)
ISR(PCINT2_vect)
{
  SoftwareSerial::handle_interrupt();
}
#endif

#if defined(PCINT3_vect)
ISR(PCINT3_vect)
{
  SoftwareSerial::handle_interrupt();
}
#endif

//
// Constructor
//
SoftwareSerial::SoftwareSerial(uint8_t receivePin, uint8_t transmitPin, bool inverse_logic /* = false */)
:
  _rx_delay_centering(0),
  _rx_delay_intrabit(0),
  _rx_delay_stopbit(0),
  _tx_delay(0),
  _buffer_overflow(false),
  _inverse_logic(inverse_logic)
{
  setTX(transmitPin);
  setRX(receivePin);
}
// - By Paco, Constructor sense parametres

SoftwareSerial::SoftwareSerial()
{
}

// - Fi by Paco.
//
// Destructor
//
SoftwareSerial::~SoftwareSerial()
{
  end();
}

void SoftwareSerial::setTX(uint8_t tx)
{
  pinMode(tx, OUTPUT);
  digitalWrite(tx, HIGH);
  _transmitBitMask = digitalPinToBitMask(tx);
  uint8_t port = digitalPinToPort(tx);
  _transmitPortRegister = portOutputRegister(port);
}
```

```
void SoftwareSerial::setRX(uint8_t rx)
{
  pinMode(rx, INPUT);
  if (!_inverse_logic)
    digitalWrite(rx, HIGH);  // pullup for normal logic!
  _receivePin = rx;
  _receiveBitMask = digitalPinToBitMask(rx);
  uint8_t port = digitalPinToPort(rx);
  _receivePortRegister = portInputRegister(port);
}

//
// Public methods
//

void SoftwareSerial::begin(long speed)
{
  _rx_delay_centering = _rx_delay_intrabit = _rx_delay_stopbit = _tx_delay = 0;

  for (unsigned i = 0; i < sizeof(table) / sizeof(table[0]); ++i)
  {
    long baud = pgm_read_dword(&table[i].baud);
    if (baud == speed)
    {
      _rx_delay_centering = pgm_read_word(&table[i].rx_delay_centering);
      _rx_delay_intrabit = pgm_read_word(&table[i].rx_delay_intrabit);
      _rx_delay_stopbit = pgm_read_word(&table[i].rx_delay_stopbit);
      _tx_delay = pgm_read_word(&table[i].tx_delay);
      break;
    }
  }

  // Set up RX interrupts, but only if we have a valid RX baud rate
  if (_rx_delay_stopbit)
  {
    if (digitalPinToPCICR(_receivePin))
    {
      *digitalPinToPCICR(_receivePin) |= _BV(digitalPinToPCICRbit(_receivePin));
      *digitalPinToPCMSK(_receivePin) |= _BV(digitalPinToPCMSKbit(_receivePin));
    }
    tunedDelay(_tx_delay); // if we were low this establishes the end
  }

#if _DEBUG
  pinMode(_DEBUG_PIN1, OUTPUT);
  pinMode(_DEBUG_PIN2, OUTPUT);
#endif

  listen();
}

void SoftwareSerial::end()
{
  if (digitalPinToPCMSK(_receivePin))
    *digitalPinToPCMSK(_receivePin) &= ~_BV(digitalPinToPCMSKbit(_receivePin));
}
```

```
// Read data from buffer
int SoftwareSerial::read()
{
  if (!isListening())
    return -1;

  // Empty buffer?
  if (_receive_buffer_head == _receive_buffer_tail)
    return -1;

  // Read from "head"
  uint8_t d = _receive_buffer[_receive_buffer_head]; // grab next byte
  _receive_buffer_head = (_receive_buffer_head + 1) % _SS_MAX_RX_BUFF;
  return d;
}

int SoftwareSerial::available()
{
  if (!isListening())
    return 0;

  return (_receive_buffer_tail + _SS_MAX_RX_BUFF - _receive_buffer_head) %
_SS_MAX_RX_BUFF;
}

size_t SoftwareSerial::write(uint8_t b)
{
  if (_tx_delay == 0) {
    setWriteError();
    return 0;
  }

  uint8_t oldSREG = SREG;
  cli();  // turn off interrupts for a clean txmit

  // Write the start bit
  tx_pin_write(_inverse_logic ? HIGH : LOW);
  tunedDelay(_tx_delay + XMIT_START_ADJUSTMENT);

  // Write each of the 8 bits
  if (_inverse_logic)
  {
    for (byte mask = 0x01; mask; mask <<= 1)
    {
      if (b & mask) // choose bit
        tx_pin_write(LOW); // send 1
      else
        tx_pin_write(HIGH); // send 0

      tunedDelay(_tx_delay);
    }

    tx_pin_write(LOW); // restore pin to natural state
  }
  else
```

```
  {
    for (byte mask = 0x01; mask; mask <<= 1)
    {
      if (b & mask) // choose bit
        tx_pin_write(HIGH); // send 1
      else
        tx_pin_write(LOW); // send 0

      tunedDelay(_tx_delay);
    }

    tx_pin_write(HIGH); // restore pin to natural state
  }

  SREG = oldSREG; // turn interrupts back on
  tunedDelay(_tx_delay);

  return 1;
}

void SoftwareSerial::flush()
{
  if (!isListening())
    return;

  uint8_t oldSREG = SREG;
  cli();
  _receive_buffer_head = _receive_buffer_tail = 0;
  SREG = oldSREG;
}

int SoftwareSerial::peek()
{
  if (!isListening())
    return -1;

  // Empty buffer?
  if (_receive_buffer_head == _receive_buffer_tail)
    return -1;

  // Read from "head"
  return _receive_buffer[_receive_buffer_head];
}

#endif
```

1.5.1.21 Codi font de SoftwareSerial.h

```
/*
  SoftwareSerial.h (formerly NewSoftSerial.h) -
  Multi-instance software serial library for Arduino/Wiring
  -- Interrupt-driven receive and other improvements by ladyada
   (http://ladyada.net)
  -- Tuning, circular buffer, derivation from class Print/Stream,
   multi-instance support, porting to 8MHz processors,
   various optimizations, PROGMEM delay tables, inverse logic and
   direct port writing by Mikal Hart (http://www.arduiniana.org)
  -- Pin change interrupt macros by Paul Stoffregen (http://www.pjrc.com)
```

```
  -- 20MHz processor support by Garrett Mace (http://www.macetech.com)
  -- ATmega1280/2560 support by Brett Hagman (http://www.roguerobotics.com/)

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA

  The latest version of this library can always be found at
  http://arduiniana.org.
*/

#ifndef SoftwareSerial_h
#define SoftwareSerial_h

#include <inttypes.h>
#include <Stream.h>
#include <HardwareSerial.h>

/******************************************************************************
  Definitions
******************************************************************************/

#define _SS_MAX_RX_BUFF 64 // RX buffer size
#ifndef GCC_VERSION
#define GCC_VERSION (__GNUC__ * 10000 + __GNUC_MINOR__ * 100 +
__GNUC_PATCHLEVEL__)
#endif

#if defined(__MK20DX128__) || defined(__MK20DX256__) || defined(__MKL26Z64__) ||
defined(__MK64FX512__) || defined(__MK66FX1M0__)

class SoftwareSerial : public Stream
{
  public:
    // Modificar - Paco SoftwareSerial(uint8_t rxPin, uint8_t txPin, bool inverse_logic = false);
    SoftwareSerial(uint8_t rxPin, uint8_t txPin);
    // SoftwareSerial();
    ~SoftwareSerial() {
      end();
    }
    //  void asignar_port(uint8_t rxPin, uint8_t txPin, bool inverse_logic /* = false */);

    // By Paco Public (desde privat)
    void setTX(uint8_t transmitPin);
    void setRX(uint8_t receivePin);
    // Fi by paco.
    void begin(unsigned long speed);
```

```
    void end();
    bool listen() {
      return true;
    }
    bool isListening() {
      return true;
    }
    bool overflow() {
      bool ret = buffer_overflow;
      buffer_overflow = false;
      return ret;
    }
    virtual int available();
    virtual int read();
    int peek();
    virtual void flush();
    virtual size_t write(uint8_t byte);
    using Print::write;
  private:
    HardwareSerial *port;
    uint32_t cycles_per_bit;
    volatile uint8_t *txreg;
    volatile uint8_t *rxreg;
    bool buffer_overflow;
    uint8_t txpin;
    uint8_t rxpin;
};

#else
class SoftwareSerial : public Stream
{
  private:

    // per object data
    uint8_t _receivePin;
    uint8_t _receiveBitMask;
    volatile uint8_t *_receivePortRegister;
    uint8_t _transmitBitMask;
    volatile uint8_t *_transmitPortRegister;

    uint16_t _rx_delay_centering;
    uint16_t _rx_delay_intrabit;
    uint16_t _rx_delay_stopbit;
    uint16_t _tx_delay;

    uint16_t _buffer_overflow: 1;
    uint16_t _inverse_logic: 1;

    // static data
    static char _receive_buffer[_SS_MAX_RX_BUFF];
    static volatile uint8_t _receive_buffer_tail;
    static volatile uint8_t _receive_buffer_head;
    static SoftwareSerial *active_object;

    // private methods
    void recv();
    uint8_t rx_pin_read();
```

```
    void tx_pin_write(uint8_t pin_state);
    /* By Paco
      void setTX(uint8_t transmitPin);
      void setRX(uint8_t receivePin);
    */
    // private static method for timing
    static inline void tunedDelay(uint16_t delay);


  public:
    // public methods
    SoftwareSerial(uint8_t receivePin, uint8_t transmitPin, bool inverse_logic = false);
    SoftwareSerial();   // By Paco
    ~SoftwareSerial();
    // By Paco Public
    void setTX(uint8_t transmitPin);
    void setRX(uint8_t receivePin);
    //
    void begin(long speed);
    bool listen();
    void end();
    bool isListening() {
      return this == active_object;
    }
    bool overflow() {
      bool ret = _buffer_overflow;
      _buffer_overflow = false;
      return ret;
    }
    int peek();

    virtual size_t write(uint8_t byte);
    virtual int read();
    virtual int available();
    virtual void flush();

    using Print::write;

    // public only for easy access by interrupt handlers
    static inline void handle_interrupt();
};

// Arduino 0012 workaround
#undef int
#undef char
#undef long
#undef byte
#undef float
#undef abs
#undef round

#endif

#endif
```

1.5.1.22 Codi font SPI.cpp

```
/*
  Copyright (c) 2010 by Cristian Maglie <c.maglie@bug.st>
  SPI Master library for arduino.

  This file is free software; you can redistribute it and/or modify
  it under the terms of either the GNU General Public License version 2
  or the GNU Lesser General Public License version 2.1, both as
  published by the Free Software Foundation.
*/

#include "SPI.h"


/*************************************************************/
/*     8 bit AVR-based boards                                */
/*************************************************************/

#if defined(__AVR__)

SPIClass SPI;

uint8_t SPIClass::interruptMode = 0;
uint8_t SPIClass::interruptMask = 0;
uint8_t SPIClass::interruptSave = 0;
#ifdef SPI_TRANSACTION_MISMATCH_LED
uint8_t SPIClass::inTransactionFlag = 0;
#endif
uint8_t SPIClass::_transferWriteFill = 0;


void SPIClass::begin()
{
  // Set SS to high so a connected chip will be "deselected" by default
  digitalWrite(SS, HIGH);

  // When the SS pin is set as OUTPUT, it can be used as
  // a general purpose output port (it doesn't influence
  // SPI operations).
  pinMode(SS, OUTPUT);

  // Warning: if the SS pin ever becomes a LOW INPUT then SPI
  // automatically switches to Slave, so the data direction of
  // the SS pin MUST be kept as OUTPUT.
  SPCR |= _BV(MSTR);
  SPCR |= _BV(SPE);

  // Set direction register for SCK and MOSI pin.
  // MISO pin automatically overrides to INPUT.
  // By doing this AFTER enabling SPI, we avoid accidentally
  // clocking in a single bit since the lines go directly
  // from "input" to SPI control.
  // http://code.google.com/p/arduino/issues/detail?id=888
  pinMode(SCK, OUTPUT);
  pinMode(MOSI, OUTPUT);
}
```

```
void SPIClass::end() {
  SPCR &= ~_BV(SPE);
}

// mapping of interrupt numbers to bits within SPI_AVR_EIMSK
#if defined(__AVR_ATmega32U4__)
#define SPI_INT0_MASK  (1<<INT0)
#define SPI_INT1_MASK  (1<<INT1)
#define SPI_INT2_MASK  (1<<INT2)
#define SPI_INT3_MASK  (1<<INT3)
#define SPI_INT4_MASK  (1<<INT6)
#elif defined(__AVR_AT90USB646__) || defined(__AVR_AT90USB1286__)
#define SPI_INT0_MASK  (1<<INT0)
#define SPI_INT1_MASK  (1<<INT1)
#define SPI_INT2_MASK  (1<<INT2)
#define SPI_INT3_MASK  (1<<INT3)
#define SPI_INT4_MASK  (1<<INT4)
#define SPI_INT5_MASK  (1<<INT5)
#define SPI_INT6_MASK  (1<<INT6)
#define SPI_INT7_MASK  (1<<INT7)
#elif defined(EICRA) && defined(EICRB) && defined(EIMSK)
#define SPI_INT0_MASK  (1<<INT4)
#define SPI_INT1_MASK  (1<<INT5)
#define SPI_INT2_MASK  (1<<INT0)
#define SPI_INT3_MASK  (1<<INT1)
#define SPI_INT4_MASK  (1<<INT2)
#define SPI_INT5_MASK  (1<<INT3)
#define SPI_INT6_MASK  (1<<INT6)
#define SPI_INT7_MASK  (1<<INT7)
#else
#ifdef INT0
#define SPI_INT0_MASK  (1<<INT0)
#endif
#ifdef INT1
#define SPI_INT1_MASK  (1<<INT1)
#endif
#ifdef INT2
#define SPI_INT2_MASK  (1<<INT2)
#endif
#endif

void SPIClass::usingInterrupt(uint8_t interruptNumber)
{
  uint8_t stmp, mask;

  if (interruptMode > 1) return;

  stmp = SREG;
  noInterrupts();
  switch (interruptNumber) {
#ifdef SPI_INT0_MASK
    case 0: mask = SPI_INT0_MASK; break;
#endif
#ifdef SPI_INT1_MASK
    case 1: mask = SPI_INT1_MASK; break;
#endif
```

```
#ifdef SPI_INT2_MASK
   case 2: mask = SPI_INT2_MASK; break;
#endif
#ifdef SPI_INT3_MASK
   case 3: mask = SPI_INT3_MASK; break;
#endif
#ifdef SPI_INT4_MASK
   case 4: mask = SPI_INT4_MASK; break;
#endif
#ifdef SPI_INT5_MASK
   case 5: mask = SPI_INT5_MASK; break;
#endif
#ifdef SPI_INT6_MASK
   case 6: mask = SPI_INT6_MASK; break;
#endif
#ifdef SPI_INT7_MASK
   case 7: mask = SPI_INT7_MASK; break;
#endif
   default:
     interruptMode = 2;
     SREG = stmp;
     return;
  }
  interruptMode = 1;
  interruptMask |= mask;
  SREG = stmp;
}

void SPIClass::transfer(const void * buf, void * retbuf, uint32_t count) {
  if (count == 0) return;

  const uint8_t *p = (const uint8_t *)buf;
  uint8_t *pret = (uint8_t *)retbuf;
  uint8_t in;

  uint8_t out = p ? *p++ : _transferWriteFill;
  SPDR = out;
  while (--count > 0) {
    if (p) {
      out = *p++;
    }
    while (!(SPSR & _BV(SPIF))) ;
    in = SPDR;
    SPDR = out;
    if (pret)*pret++ = in;
  }
  while (!(SPSR & _BV(SPIF))) ;
  in = SPDR;
  if (pret)*pret = in;
}


/*************************************************************/
/*     32 bit Teensy 3.x                                     */
/*************************************************************/

#elif defined(__arm__) && defined(TEENSYDUINO) && defined(KINETISK)
```

```
#if defined(KINETISK) && defined( SPI_HAS_TRANSFER_ASYNC)

#ifndef TRANSFER_COUNT_FIXED
inline void DMAChanneltransferCount(DMAChannel * dmac, unsigned int len) {
  // note does no validation of length...
  DMABaseClass::TCD_t *tcd = dmac->TCD;
  if (!(tcd->BITER & DMA_TCD_BITER_ELINK)) {
    tcd->BITER = len & 0x7fff;
  } else {
    tcd->BITER = (tcd->BITER & 0xFE00) | (len & 0x1ff);
  }
  tcd->CITER = tcd->BITER;
}
#else
inline void DMAChanneltransferCount(DMAChannel * dmac, unsigned int len) {
  dmac->transferCount(len);
}
#endif
#endif


#if defined(__MK20DX128__) || defined(__MK20DX256__)
#ifdef SPI_HAS_TRANSFER_ASYNC
void _spi_dma_rxISR0(void) {
  SPI.dma_rxisr();
}
#else
void _spi_dma_rxISR0(void) {
  ;
}
#endif

const SPIClass::SPI_Hardware_t SPIClass::spi0_hardware = {
  SIM_SCGC6, SIM_SCGC6_SPI0, 4, IRQ_SPI0,
  32767, DMAMUX_SOURCE_SPI0_TX, DMAMUX_SOURCE_SPI0_RX,
  _spi_dma_rxISR0,
  12, 8,
  PORT_PCR_MUX(2), PORT_PCR_MUX(2),
  11, 7,
  PORT_PCR_DSE | PORT_PCR_MUX(2), PORT_PCR_MUX(2),
  13, 14,
  PORT_PCR_DSE | PORT_PCR_MUX(2), PORT_PCR_MUX(2),
  10, 2, 9, 6, 20, 23, 21, 22, 15,
  PORT_PCR_MUX(2),  PORT_PCR_MUX(2), PORT_PCR_MUX(2),  PORT_PCR_MUX(2),
PORT_PCR_MUX(2),  PORT_PCR_MUX(2),  PORT_PCR_MUX(2),  PORT_PCR_MUX(2),
PORT_PCR_MUX(2),
  0x1, 0x1, 0x2, 0x2, 0x4, 0x4, 0x8, 0x8, 0x10
};
SPIClass SPI((uintptr_t)&KINETISK_SPI0, (uintptr_t)&SPIClass::spi0_hardware);

#elif defined(__MK64FX512__) || defined(__MK66FX1M0__)
#ifdef SPI_HAS_TRANSFER_ASYNC
void _spi_dma_rxISR0(void) {
  SPI.dma_rxisr();
}
void _spi_dma_rxISR1(void) {
  SPI1.dma_rxisr();
```

```
}
void _spi_dma_rxISR2(void) {
  SPI2.dma_rxisr();
}
#else
void _spi_dma_rxISR0(void) {
  ;
}
void _spi_dma_rxISR1(void) {
  ;
}
void _spi_dma_rxISR2(void) {
  ;
}
#endif
const SPIClass::SPI_Hardware_t SPIClass::spi0_hardware = {
  SIM_SCGC6, SIM_SCGC6_SPI0, 4, IRQ_SPI0,
  32767, DMAMUX_SOURCE_SPI0_TX, DMAMUX_SOURCE_SPI0_RX,
  _spi_dma_rxISR0,
  12, 8, 39, 255,
  PORT_PCR_MUX(2), PORT_PCR_MUX(2), PORT_PCR_MUX(2), 0,
  11, 7, 28, 255,
  PORT_PCR_MUX(2), PORT_PCR_MUX(2), PORT_PCR_MUX(2), 0,
  13, 14, 27,
  PORT_PCR_MUX(2), PORT_PCR_MUX(2), PORT_PCR_MUX(2),
  10, 2, 9, 6, 20, 23, 21, 22, 15, 26, 45,
  PORT_PCR_MUX(2),  PORT_PCR_MUX(2), PORT_PCR_MUX(2),  PORT_PCR_MUX(2),
PORT_PCR_MUX(2),  PORT_PCR_MUX(2),  PORT_PCR_MUX(2),  PORT_PCR_MUX(2),
PORT_PCR_MUX(2),   PORT_PCR_MUX(2),   PORT_PCR_MUX(3),
  0x1, 0x1, 0x2, 0x2, 0x4, 0x4, 0x8, 0x8, 0x10, 0x1, 0x20
};
const SPIClass::SPI_Hardware_t SPIClass::spi1_hardware = {
  SIM_SCGC6, SIM_SCGC6_SPI1, 1, IRQ_SPI1,
#if defined(__MK66FX1M0__)
  32767, DMAMUX_SOURCE_SPI1_TX, DMAMUX_SOURCE_SPI1_RX,
#else
  // T3.5 does not have good DMA support on 1 and 2
  511, 0, DMAMUX_SOURCE_SPI1,
#endif
  _spi_dma_rxISR1,
  1, 5, 61, 59,
  PORT_PCR_MUX(2), PORT_PCR_MUX(7), PORT_PCR_MUX(2), PORT_PCR_MUX(7),
  0, 21, 61, 59,
  PORT_PCR_MUX(2), PORT_PCR_MUX(7), PORT_PCR_MUX(7), PORT_PCR_MUX(2),
  32, 20, 60,
  PORT_PCR_MUX(2), PORT_PCR_MUX(7), PORT_PCR_MUX(2),
  6, 31, 58, 62, 63, 255, 255, 255, 255, 255, 255,
  PORT_PCR_MUX(7),  PORT_PCR_MUX(2),  PORT_PCR_MUX(2),  PORT_PCR_MUX(2),
PORT_PCR_MUX(2),  0,  0,  0,  0,   0,   0,
  0x1, 0x1, 0x2, 0x1, 0x4, 0, 0, 0, 0, 0, 0
};
const SPIClass::SPI_Hardware_t SPIClass::spi2_hardware = {
  SIM_SCGC3, SIM_SCGC3_SPI2, 1, IRQ_SPI2,
#if defined(__MK66FX1M0__)
  32767, DMAMUX_SOURCE_SPI2_TX, DMAMUX_SOURCE_SPI2_RX,
#else
  // T3.5 does not have good DMA support on 1 and 2
```

```
  511, 0, DMAMUX_SOURCE_SPI2,
#endif
  _spi_dma_rxISR2,
  45, 51, 255, 255,
  PORT_PCR_MUX(2), PORT_PCR_MUX(2), 0, 0,
  44, 52, 255, 255,
  PORT_PCR_MUX(2), PORT_PCR_MUX(2), 0, 0,
  46, 53, 255,
  PORT_PCR_MUX(2), PORT_PCR_MUX(2), 0,
  43, 54, 55, 255, 255, 255, 255, 255, 255, 255, 255,
  PORT_PCR_MUX(2),  PORT_PCR_MUX(2),  PORT_PCR_MUX(2),  0,  0,  0,  0,  0,  0,  0,  0,
  0x1, 0x2, 0x1, 0, 0, 0, 0, 0, 0, 0, 0
};
SPIClass SPI((uintptr_t)&KINETISK_SPI0, (uintptr_t)&SPIClass::spi0_hardware);
SPIClass SPI1((uintptr_t)&KINETISK_SPI1, (uintptr_t)&SPIClass::spi1_hardware);
SPIClass SPI2((uintptr_t)&KINETISK_SPI2, (uintptr_t)&SPIClass::spi2_hardware);
#endif


void SPIClass::begin()
{
  volatile uint32_t *reg;

  hardware().clock_gate_register |= hardware().clock_gate_mask;
  port().MCR = SPI_MCR_MDIS | SPI_MCR_HALT | SPI_MCR_PCSIS(0x1F);
  port().CTAR0 = SPI_CTAR_FMSZ(7) | SPI_CTAR_PBR(0) | SPI_CTAR_BR(1) |
SPI_CTAR_CSSCK(1);
  port().CTAR1 = SPI_CTAR_FMSZ(15) | SPI_CTAR_PBR(0) | SPI_CTAR_BR(1) |
SPI_CTAR_CSSCK(1);
  port().MCR = SPI_MCR_MSTR | SPI_MCR_PCSIS(0x1F);
  reg = portConfigRegister(hardware().mosi_pin[mosi_pin_index]);
  *reg = hardware().mosi_mux[mosi_pin_index];
  reg = portConfigRegister(hardware().miso_pin[miso_pin_index]);
  *reg = hardware().miso_mux[miso_pin_index];
  reg = portConfigRegister(hardware().sck_pin[sck_pin_index]);
  *reg = hardware().sck_mux[sck_pin_index];
}

void SPIClass::end()
{
  volatile uint32_t *reg;

  reg = portConfigRegister(hardware().mosi_pin[mosi_pin_index]);
  *reg = 0;
  reg = portConfigRegister(hardware().miso_pin[miso_pin_index]);
  *reg = 0;
  reg = portConfigRegister(hardware().sck_pin[sck_pin_index]);
  *reg = 0;
  port().MCR = SPI_MCR_MDIS | SPI_MCR_HALT | SPI_MCR_PCSIS(0x1F);
}

void SPIClass::usingInterrupt(IRQ_NUMBER_t interruptName)
{
  uint32_t n = (uint32_t)interruptName;

  if (n >= NVIC_NUM_INTERRUPTS) return;
```

```
  //Serial.print("usingInterrupt ");
  //Serial.println(n);
  interruptMasksUsed |= (1 << (n >> 5));
  interruptMask[n >> 5] |= (1 << (n & 0x1F));
  //Serial.printf("interruptMasksUsed = %d\n", interruptMasksUsed);
  //Serial.printf("interruptMask[0] = %08X\n", interruptMask[0]);
  //Serial.printf("interruptMask[1] = %08X\n", interruptMask[1]);
  //Serial.printf("interruptMask[2] = %08X\n", interruptMask[2]);
}

void SPIClass::notUsingInterrupt(IRQ_NUMBER_t interruptName)
{
  uint32_t n = (uint32_t)interruptName;
  if (n >= NVIC_NUM_INTERRUPTS) return;
  interruptMask[n >> 5] &= ~(1 << (n & 0x1F));
  if (interruptMask[n >> 5] == 0) {
    interruptMasksUsed &= ~(1 << (n >> 5));
  }
}

const uint16_t SPISettings::ctar_div_table[23] = {
  2, 3, 4, 5, 6, 8, 10, 12, 16, 20, 24, 32, 40,
  56, 64, 96, 128, 192, 256, 384, 512, 640, 768
};
const uint32_t SPISettings::ctar_clock_table[23] = {
  SPI_CTAR_PBR(0) | SPI_CTAR_BR(0) | SPI_CTAR_DBR | SPI_CTAR_CSSCK(0),
  SPI_CTAR_PBR(1) | SPI_CTAR_BR(0) | SPI_CTAR_DBR | SPI_CTAR_CSSCK(0),
  SPI_CTAR_PBR(0) | SPI_CTAR_BR(0) | SPI_CTAR_CSSCK(0),
  SPI_CTAR_PBR(2) | SPI_CTAR_BR(0) | SPI_CTAR_DBR | SPI_CTAR_CSSCK(0),
  SPI_CTAR_PBR(1) | SPI_CTAR_BR(0) | SPI_CTAR_CSSCK(0),
  SPI_CTAR_PBR(0) | SPI_CTAR_BR(1) | SPI_CTAR_CSSCK(1),
  SPI_CTAR_PBR(2) | SPI_CTAR_BR(0) | SPI_CTAR_CSSCK(0),
  SPI_CTAR_PBR(1) | SPI_CTAR_BR(1) | SPI_CTAR_CSSCK(1),
  SPI_CTAR_PBR(0) | SPI_CTAR_BR(3) | SPI_CTAR_CSSCK(2),
  SPI_CTAR_PBR(2) | SPI_CTAR_BR(1) | SPI_CTAR_CSSCK(0),
  SPI_CTAR_PBR(1) | SPI_CTAR_BR(3) | SPI_CTAR_CSSCK(2),
  SPI_CTAR_PBR(0) | SPI_CTAR_BR(4) | SPI_CTAR_CSSCK(3),
  SPI_CTAR_PBR(2) | SPI_CTAR_BR(3) | SPI_CTAR_CSSCK(2),
  SPI_CTAR_PBR(3) | SPI_CTAR_BR(3) | SPI_CTAR_CSSCK(2),
  SPI_CTAR_PBR(0) | SPI_CTAR_BR(5) | SPI_CTAR_CSSCK(4),
  SPI_CTAR_PBR(1) | SPI_CTAR_BR(5) | SPI_CTAR_CSSCK(4),
  SPI_CTAR_PBR(0) | SPI_CTAR_BR(6) | SPI_CTAR_CSSCK(5),
  SPI_CTAR_PBR(1) | SPI_CTAR_BR(6) | SPI_CTAR_CSSCK(5),
  SPI_CTAR_PBR(0) | SPI_CTAR_BR(7) | SPI_CTAR_CSSCK(6),
  SPI_CTAR_PBR(1) | SPI_CTAR_BR(7) | SPI_CTAR_CSSCK(6),
  SPI_CTAR_PBR(0) | SPI_CTAR_BR(8) | SPI_CTAR_CSSCK(7),
  SPI_CTAR_PBR(2) | SPI_CTAR_BR(7) | SPI_CTAR_CSSCK(6),
  SPI_CTAR_PBR(1) | SPI_CTAR_BR(8) | SPI_CTAR_CSSCK(7)
};

void SPIClass::updateCTAR(uint32_t ctar)
{
  if (port().CTAR0 != ctar) {
    uint32_t mcr = port().MCR;
    if (mcr & SPI_MCR_MDIS) {
      port().CTAR0 = ctar;
      port().CTAR1 = ctar | SPI_CTAR_FMSZ(8);
```

```
    } else {
      port().MCR = SPI_MCR_MDIS | SPI_MCR_HALT | SPI_MCR_PCSIS(0x1F);
      port().CTAR0 = ctar;
      port().CTAR1 = ctar | SPI_CTAR_FMSZ(8);
      port().MCR = mcr;
    }
  }
}

void SPIClass::setBitOrder(uint8_t bitOrder)
{
  hardware().clock_gate_register |= hardware().clock_gate_mask;
  uint32_t ctar = port().CTAR0;
  if (bitOrder == LSBFIRST) {
    ctar |= SPI_CTAR_LSBFE;
  } else {
    ctar &= ~SPI_CTAR_LSBFE;
  }
  updateCTAR(ctar);
}

void SPIClass::setDataMode(uint8_t dataMode)
{
  hardware().clock_gate_register |= hardware().clock_gate_mask;
  //uint32_t ctar = port().CTAR0;

  // TODO: implement with native code
  //SPCR = (SPCR & ~SPI_MODE_MASK) | dataMode;
}

void SPIClass::setClockDivider_noInline(uint32_t clk)
{
  hardware().clock_gate_register |= hardware().clock_gate_mask;
  uint32_t ctar = port().CTAR0;
  ctar &= (SPI_CTAR_CPOL | SPI_CTAR_CPHA | SPI_CTAR_LSBFE);
  if (ctar & SPI_CTAR_CPHA) {
    clk = (clk & 0xFFFF0FFF) | ((clk & 0xF000) >> 4);
  }
  ctar |= clk;
  updateCTAR(ctar);
}

uint8_t SPIClass::pinIsChipSelect(uint8_t pin)
{
  for (unsigned int i = 0; i < sizeof(hardware().cs_pin); i++) {
    if (pin == hardware().cs_pin[i]) return hardware().cs_mask[i];
  }
  return 0;
}

bool SPIClass::pinIsChipSelect(uint8_t pin1, uint8_t pin2)
{
  uint8_t pin1_mask, pin2_mask;
  if ((pin1_mask = (uint8_t)pinIsChipSelect(pin1)) == 0) return false;
  if ((pin2_mask = (uint8_t)pinIsChipSelect(pin2)) == 0) return false;
  //Serial.printf("pinIsChipSelect %d %d %x %x\n\r", pin1, pin2, pin1_mask, pin2_mask);
  if ((pin1_mask & pin2_mask) != 0) return false;
```

```
  return true;
}

bool SPIClass::pinIsMOSI(uint8_t pin)
{
  for (unsigned int i = 0; i < sizeof(hardware().mosi_pin); i++) {
    if (pin == hardware().mosi_pin[i]) return true;
  }
  return false;
}

bool SPIClass::pinIsMISO(uint8_t pin)
{
  for (unsigned int i = 0; i < sizeof(hardware().miso_pin); i++) {
    if (pin == hardware().miso_pin[i]) return true;
  }
  return false;
}

bool SPIClass::pinIsSCK(uint8_t pin)
{
  for (unsigned int i = 0; i < sizeof(hardware().sck_pin); i++) {
    if (pin == hardware().sck_pin[i]) return true;
  }
  return false;
}

// setCS() is not intended for use from normal Arduino programs/sketches.
uint8_t SPIClass::setCS(uint8_t pin)
{
  for (unsigned int i = 0; i < sizeof(hardware().cs_pin); i++) {
    if (pin == hardware().cs_pin[i]) {
      volatile uint32_t *reg = portConfigRegister(pin);
      *reg = hardware().cs_mux[i];
      return hardware().cs_mask[i];
    }
  }
  return 0;
}

void SPIClass::setMOSI(uint8_t pin)
{
  if (hardware_addr == (uintptr_t)&spi0_hardware) {
    SPCR.setMOSI_soft(pin);
  }
  if (pin != hardware().mosi_pin[mosi_pin_index]) {
    for (unsigned int i = 0; i < sizeof(hardware().mosi_pin); i++) {
      if  (pin == hardware().mosi_pin[i]) {
        if (hardware().clock_gate_register & hardware().clock_gate_mask) {
          volatile uint32_t *reg;
          reg = portConfigRegister(hardware().mosi_pin[mosi_pin_index]);
          *reg = 0;
          reg = portConfigRegister(hardware().mosi_pin[i]);
          *reg = hardware().mosi_mux[i];
        }
        mosi_pin_index = i;
        return;
```

```
    }
   }
  }
}

void SPIClass::setMISO(uint8_t pin)
{
  if (hardware_addr == (uintptr_t)&spi0_hardware) {
    SPCR.setMISO_soft(pin);
  }
  if (pin != hardware().miso_pin[miso_pin_index]) {
    for (unsigned int i = 0; i < sizeof(hardware().miso_pin); i++) {
      if  (pin == hardware().miso_pin[i]) {
        if (hardware().clock_gate_register & hardware().clock_gate_mask) {
          volatile uint32_t *reg;
          reg = portConfigRegister(hardware().miso_pin[miso_pin_index]);
          *reg = 0;
          reg = portConfigRegister(hardware().miso_pin[i]);
          *reg = hardware().miso_mux[i];
        }
        miso_pin_index = i;
        return;
      }
    }
  }
}

void SPIClass::setSCK(uint8_t pin)
{
  if (hardware_addr == (uintptr_t)&spi0_hardware) {
    SPCR.setSCK_soft(pin);
  }
  if (pin != hardware().sck_pin[sck_pin_index]) {
    for (unsigned int i = 0; i < sizeof(hardware().sck_pin); i++) {
      if  (pin == hardware().sck_pin[i]) {
        if (hardware().clock_gate_register & hardware().clock_gate_mask) {
          volatile uint32_t *reg;
          reg = portConfigRegister(hardware().sck_pin[sck_pin_index]);
          *reg = 0;
          reg = portConfigRegister(hardware().sck_pin[i]);
          *reg = hardware().sck_mux[i];
        }
        sck_pin_index = i;
        return;
      }
    }
  }
}

void SPIClass::transfer(const void * buf, void * retbuf, size_t count)
{

  if (count == 0) return;
  if (!(port().CTAR0 & SPI_CTAR_LSBFE)) {
    // We are doing the standard MSB order
    const uint8_t *p_write = (const uint8_t *)buf;
    uint8_t *p_read = (uint8_t *)retbuf;
```

Controlador de reg per degoteig

```
size_t count_read = count;

// Lets clear the reader queue
port().MCR = SPI_MCR_MSTR | SPI_MCR_CLR_RXF | SPI_MCR_PCSIS(0x1F);

uint32_t sr;

// Now lets loop while we still have data to output
if (count & 1) {
  if (p_write) {
    if (count > 1)
      port().PUSHR = *p_write++ | SPI_PUSHR_CONT | SPI_PUSHR_CTAS(0);
    else
      port().PUSHR = *p_write++ | SPI_PUSHR_CTAS(0);
  } else {
    if (count > 1)
      port().PUSHR = _transferWriteFill | SPI_PUSHR_CONT | SPI_PUSHR_CTAS(0);
    else
      port().PUSHR = _transferWriteFill | SPI_PUSHR_CTAS(0);
  }
  count--;
}

uint16_t w =  (uint16_t)(_transferWriteFill << 8) | _transferWriteFill;

while (count > 0) {
  // Push out the next byte;
  if (p_write) {
    w = (*p_write++) << 8;
    w |= *p_write++;
  }
  uint16_t queue_full_status_mask = (hardware().queue_size - 1) << 12;
  if (count == 2)
    port().PUSHR = w | SPI_PUSHR_CTAS(1);
  else
    port().PUSHR = w | SPI_PUSHR_CONT | SPI_PUSHR_CTAS(1);
  count -= 2; // how many bytes to output.
  // Make sure queue is not full before pushing next byte out
  do {
    sr = port().SR;
    if (sr & 0xF0)  {
      uint16_t w = port().POPR;  // Read any pending RX bytes in
      if (count_read & 1) {
        if (p_read) {
          *p_read++ = w;  // Read any pending RX bytes in
        }
        count_read--;
      } else {
        if (p_read) {
          *p_read++ = w >> 8;
          *p_read++ = (w & 0xff);
        }
        count_read -= 2;
      }
    }
  } while ((sr & (15 << 12)) > queue_full_status_mask);
```

```
  }

  // now lets wait for all of the read bytes to be returned...
  while (count_read) {
    sr = port().SR;
    if (sr & 0xF0)  {
      uint16_t w = port().POPR;  // Read any pending RX bytes in
      if (count_read & 1) {
        if (p_read)
          *p_read++ = w;  // Read any pending RX bytes in
        count_read--;
      } else {
        if (p_read) {
          *p_read++ = w >> 8;
          *p_read++ = (w & 0xff);
        }
        count_read -= 2;
      }
    }
  }
} else {
  // We are doing the less ofen LSB mode
  const uint8_t *p_write = (const uint8_t *)buf;
  uint8_t *p_read = (uint8_t *)retbuf;
  size_t count_read = count;

  // Lets clear the reader queue
  port().MCR = SPI_MCR_MSTR | SPI_MCR_CLR_RXF | SPI_MCR_PCSIS(0x1F);

  uint32_t sr;

  // Now lets loop while we still have data to output
  if (count & 1) {
    if (p_write) {
      if (count > 1)
        port().PUSHR = *p_write++ | SPI_PUSHR_CONT | SPI_PUSHR_CTAS(0);
      else
        port().PUSHR = *p_write++ | SPI_PUSHR_CTAS(0);
    } else {
      if (count > 1)
        port().PUSHR = _transferWriteFill | SPI_PUSHR_CONT | SPI_PUSHR_CTAS(0);
      else
        port().PUSHR = _transferWriteFill | SPI_PUSHR_CTAS(0);
    }
    count--;
  }

  uint16_t w = _transferWriteFill;

  while (count > 0) {
    // Push out the next byte;
    if (p_write) {
      w = *p_write++;
      w |= ((*p_write++) << 8);
    }
    uint16_t queue_full_status_mask = (hardware().queue_size - 1) << 12;
    if (count == 2)
```

```
            port().PUSHR = w | SPI_PUSHR_CTAS(1);
          else
            port().PUSHR = w | SPI_PUSHR_CONT | SPI_PUSHR_CTAS(1);
          count -= 2; // how many bytes to output.
          // Make sure queue is not full before pushing next byte out
          do {
            sr = port().SR;
            if (sr & 0xF0)  {
              uint16_t w = port().POPR;  // Read any pending RX bytes in
              if (count_read & 1) {
                if (p_read) {
                  *p_read++ = w;  // Read any pending RX bytes in
                }
                count_read--;
              } else {
                if (p_read) {
                  *p_read++ = (w & 0xff);
                  *p_read++ = w >> 8;
                }
                count_read -= 2;
              }
            }
          } while ((sr & (15 << 12)) > queue_full_status_mask);

        }

        // now lets wait for all of the read bytes to be returned...
        while (count_read) {
          sr = port().SR;
          if (sr & 0xF0)  {
            uint16_t w = port().POPR;  // Read any pending RX bytes in
            if (count_read & 1) {
              if (p_read)
                *p_read++ = w;  // Read any pending RX bytes in
              count_read--;
            } else {
              if (p_read) {
                *p_read++ = (w & 0xff);
                *p_read++ = w >> 8;
              }
              count_read -= 2;
            }
          }
        }
      }
      //================================================================================
=
      // ASYNCH Support
      //================================================================================
=
      //================================================================
      // Try Transfer using DMA.
      //================================================================
      #ifdef SPI_HAS_TRANSFER_ASYNC
      static uint8_t bit_bucket;
      #define dontInterruptAtCompletion(dmac) (dmac)->TCD->CSR &= ~DMA_TCD_CSR_INTMAJOR
```

```
//=============================================================================
// Init the DMA channels
//=============================================================================
bool SPIClass::initDMAChannels() {
  // Allocate our channels.
  _dmaTX = new DMAChannel();
  if (_dmaTX == nullptr) {
    return false;
  }

  _dmaRX = new DMAChannel();
  if (_dmaRX == nullptr) {
    delete _dmaTX; // release it
    _dmaTX = nullptr;
    return false;
  }

  // Let's setup the RX chain
  _dmaRX->disable();
  _dmaRX->source((volatile uint8_t&)port().POPR);
  _dmaRX->disableOnCompletion();
  _dmaRX->triggerAtHardwareEvent(hardware().rx_dma_channel);
  _dmaRX->attachInterrupt(hardware().dma_rxisr);
  _dmaRX->interruptAtCompletion();

  // We may be using settings chain here so lets set it up.
  // Now lets setup TX chain.  Note if trigger TX is not set
  // we need to have the RX do it for us.
  _dmaTX->disable();
  _dmaTX->destination((volatile uint8_t&)port().PUSHR);
  _dmaTX->disableOnCompletion();

  if (hardware().tx_dma_channel) {
    _dmaTX->triggerAtHardwareEvent(hardware().tx_dma_channel);
  } else {
    //            Serial.printf("SPI InitDMA tx triger by RX: %x\n", (uint32_t)_dmaRX);
    _dmaTX->triggerAtTransfersOf(*_dmaRX);
  }


  _dma_state = DMAState::idle;  // Should be first thing set!
  return true;
}

//=============================================================================
// Main Async Transfer function
//=============================================================================

bool SPIClass::transfer(const void *buf, void *retbuf, size_t count, EventResponderRef
event_responder) {
  uint8_t dma_first_byte;
  if (_dma_state == DMAState::notAllocated) {
    if (!initDMAChannels())
      return false;
  }
```

```
        if (_dma_state == DMAState::active)
          return false; // already active

        event_responder.clearEvent();       // Make sure it is not set yet
        if (count < 2) {
          // Use non-async version to simplify cases...
          transfer(buf, retbuf, count);
          event_responder.triggerEvent();
          return true;
        }

        // Now handle the cases where the count > then how many we can output in one DMA request
        if (count > hardware().max_dma_count) {
          _dma_count_remaining = count - hardware().max_dma_count;
          count = hardware().max_dma_count;
        } else {
          _dma_count_remaining = 0;
        }

        // Now See if caller passed in a source buffer.
        _dmaTX->TCD->ATTR_DST = 0;          // Make sure set for 8 bit mode
        uint8_t *write_data = (uint8_t*) buf;
        if (buf) {
          dma_first_byte = *write_data;
          _dmaTX->sourceBuffer((uint8_t*)write_data + 1, count - 1);
          _dmaTX->TCD->SLAST = 0;       // Finish with it pointing to next location
        } else {
          dma_first_byte = _transferWriteFill;
          _dmaTX->source((uint8_t&)_transferWriteFill);   // maybe have setable value
          DMAChanneltransferCount(_dmaTX, count - 1);
        }
        if (retbuf) {
          // On T3.5 must handle SPI1/2 differently as only one DMA channel
          _dmaRX->TCD->ATTR_SRC = 0;              //Make sure set for 8 bit mode...
          _dmaRX->destinationBuffer((uint8_t*)retbuf, count);
          _dmaRX->TCD->DLASTSGA = 0;              // At end point after our bufffer
        } else {
          // Write  only mode
          _dmaRX->TCD->ATTR_SRC = 0;              //Make sure set for 8 bit mode...
          _dmaRX->destination((uint8_t&)bit_bucket);
          DMAChanneltransferCount(_dmaRX, count);
        }

        _dma_event_responder = &event_responder;
        // Now try to start it?
        // Setup DMA main object
        yield();
        port().MCR = SPI_MCR_MSTR | SPI_MCR_CLR_RXF | SPI_MCR_CLR_TXF |
SPI_MCR_PCSIS(0x1F);

        port().SR = 0xFF0F0000;

        // Lets try to output the first byte to make sure that we are in 8 bit mode...
        port().PUSHR = dma_first_byte | SPI_PUSHR_CTAS(0) | SPI_PUSHR_CONT;

        if (hardware().tx_dma_channel) {
```

```
        port().RSER =  SPI_RSER_RFDF_RE | SPI_RSER_RFDF_DIRS | SPI_RSER_TFFF_RE |
SPI_RSER_TFFF_DIRS;
        _dmaRX->enable();
        // Get the initial settings.
        _dmaTX->enable();
       } else {
        //T3.5 SP1 and SPI2 - TX is not triggered by SPI but by RX...
        port().RSER =  SPI_RSER_RFDF_RE | SPI_RSER_RFDF_DIRS ;
        _dmaTX->triggerAtTransfersOf(*_dmaRX);
        _dmaTX->enable();
        _dmaRX->enable();
       }

       _dma_state = DMAState::active;
       return true;
      }


      //-----------------------------------------------------------------------
      // DMA RX ISR
      //-----------------------------------------------------------------------
      void SPIClass::dma_rxisr(void) {
       _dmaRX->clearInterrupt();
       _dmaTX->clearComplete();
       _dmaRX->clearComplete();

       uint8_t should_reenable_tx = true; // should we re-enable TX maybe not if count will be 0...
       if (_dma_count_remaining) {
        // What do I need to do to start it back up again...
        // We will use the BITR/CITR from RX as TX may have prefed some stuff
        if (_dma_count_remaining > hardware().max_dma_count) {
         _dma_count_remaining -= hardware().max_dma_count;
        } else {
         DMAChanneltransferCount(_dmaTX, _dma_count_remaining - 1);
         DMAChanneltransferCount(_dmaRX, _dma_count_remaining);
         if (_dma_count_remaining == 1) should_reenable_tx = false;

         _dma_count_remaining = 0;
        }
        // In some cases we need to again start the TX manually to get it to work...
        if (_dmaTX->TCD->SADDR == &_transferWriteFill) {
         if (port().CTAR0  & SPI_CTAR_FMSZ(8)) {
          port().PUSHR = (_transferWriteFill | SPI_PUSHR_CTAS(0) | SPI_PUSHR_CONT);
         } else  {
          port().PUSHR = (_transferWriteFill | SPI_PUSHR_CTAS(0) | SPI_PUSHR_CONT);
         }
        } else {
         if (port().CTAR0  & SPI_CTAR_FMSZ(8)) {
          // 16 bit mode
          uint16_t w = *((uint16_t*)_dmaTX->TCD->SADDR);
          _dmaTX->TCD->SADDR = (volatile uint8_t*)(_dmaTX->TCD->SADDR) + 2;
          port().PUSHR = (w | SPI_PUSHR_CTAS(0) | SPI_PUSHR_CONT);
         } else  {
          uint8_t w = *((uint8_t*)_dmaTX->TCD->SADDR);
          _dmaTX->TCD->SADDR = (volatile uint8_t*)(_dmaTX->TCD->SADDR) + 1;
          port().PUSHR = (w | SPI_PUSHR_CTAS(0) | SPI_PUSHR_CONT);
         }
```

```
        }
        _dmaRX->enable();
        if (should_reenable_tx)
          _dmaTX->enable();
      } else {

        port().RSER = 0;
        //port().MCR = SPI_MCR_MSTR | SPI_MCR_CLR_RXF | SPI_MCR_PCSIS(0x1F);  // clear out the
queue
        port().SR = 0xFF0F0000;
        port().CTAR0  &= ~(SPI_CTAR_FMSZ(8));       // Hack restore back to 8 bits

        _dma_state = DMAState::completed;   // set back to 1 in case our call wants to start up dma again
        _dma_event_responder->triggerEvent();

      }
    }
    #endif // SPI_HAS_TRANSFER_ASYNC


    /*****************************************************/
    /*     32 bit Teensy-LC                              */
    /*****************************************************/

    #elif defined(__arm__) && defined(TEENSYDUINO) && defined(KINETISL)

    #ifdef SPI_HAS_TRANSFER_ASYNC
    void _spi_dma_rxISR0(void) {
      SPI.dma_isr();
    }
    void _spi_dma_rxISR1(void) {
      SPI1.dma_isr();
    }
    #else
    void _spi_dma_rxISR0(void) {
      ;
    }
    void _spi_dma_rxISR1(void) {
      ;
    }
    #endif

    const SPIClass::SPI_Hardware_t SPIClass::spi0_hardware = {
      SIM_SCGC4, SIM_SCGC4_SPI0,
      0, // BR index 0
      DMAMUX_SOURCE_SPI0_TX, DMAMUX_SOURCE_SPI0_RX, _spi_dma_rxISR0,
      12, 8,
      PORT_PCR_MUX(2), PORT_PCR_MUX(2),
      11, 7,
      PORT_PCR_DSE | PORT_PCR_MUX(2), PORT_PCR_MUX(2),
      13, 14,
      PORT_PCR_DSE | PORT_PCR_MUX(2), PORT_PCR_MUX(2),
      10, 2,
      PORT_PCR_MUX(2),  PORT_PCR_MUX(2),
      0x1, 0x1
    };
    SPIClass SPI((uintptr_t)&KINETISL_SPI0, (uintptr_t)&SPIClass::spi0_hardware);
```

Controlador de reg per degoteig

```
const SPIClass::SPI_Hardware_t SPIClass::spi1_hardware = {
  SIM_SCGC4, SIM_SCGC4_SPI1,
  1, // BR index 1 in SPI Settings
  DMAMUX_SOURCE_SPI1_TX, DMAMUX_SOURCE_SPI1_RX, _spi_dma_rxISR1,
  1, 5,
  PORT_PCR_MUX(2), PORT_PCR_MUX(2),
  0, 21,
  PORT_PCR_MUX(2), PORT_PCR_MUX(2),
  20, 255,
  PORT_PCR_MUX(2), 0,
  6, 255,
  PORT_PCR_MUX(2),  0,
  0x1, 0
};
SPIClass SPI1((uintptr_t)&KINETISL_SPI1, (uintptr_t)&SPIClass::spi1_hardware);


void SPIClass::begin()
{
  volatile uint32_t *reg;

  hardware().clock_gate_register |= hardware().clock_gate_mask;
  port().C1 = SPI_C1_SPE | SPI_C1_MSTR;
  port().C2 = 0;
  uint8_t tmp __attribute__((unused)) = port().S;
  reg = portConfigRegister(hardware().mosi_pin[mosi_pin_index]);
  *reg = hardware().mosi_mux[mosi_pin_index];
  reg = portConfigRegister(hardware().miso_pin[miso_pin_index]);
  *reg = hardware().miso_mux[miso_pin_index];
  reg = portConfigRegister(hardware().sck_pin[sck_pin_index]);
  *reg = hardware().sck_mux[sck_pin_index];
}

void SPIClass::end() {
  volatile uint32_t *reg;

  reg = portConfigRegister(hardware().mosi_pin[mosi_pin_index]);
  *reg = 0;
  reg = portConfigRegister(hardware().miso_pin[miso_pin_index]);
  *reg = 0;
  reg = portConfigRegister(hardware().sck_pin[sck_pin_index]);
  *reg = 0;
  port().C1 = 0;
}

const uint16_t SPISettings::br_div_table[30] = {
  2, 4, 6, 8, 10, 12, 14, 16, 20, 24,
  28, 32, 40, 48, 56, 64, 80, 96, 112, 128,
  160, 192, 224, 256, 320, 384, 448, 512, 640, 768,
};

const uint8_t SPISettings::br_clock_table[30] = {
  SPI_BR_SPPR(0) | SPI_BR_SPR(0),
  SPI_BR_SPPR(1) | SPI_BR_SPR(0),
  SPI_BR_SPPR(2) | SPI_BR_SPR(0),
  SPI_BR_SPPR(3) | SPI_BR_SPR(0),
```

```
  SPI_BR_SPPR(4) | SPI_BR_SPR(0),
  SPI_BR_SPPR(5) | SPI_BR_SPR(0),
  SPI_BR_SPPR(6) | SPI_BR_SPR(0),
  SPI_BR_SPPR(7) | SPI_BR_SPR(0),
  SPI_BR_SPPR(4) | SPI_BR_SPR(1),
  SPI_BR_SPPR(5) | SPI_BR_SPR(1),
  SPI_BR_SPPR(6) | SPI_BR_SPR(1),
  SPI_BR_SPPR(7) | SPI_BR_SPR(1),
  SPI_BR_SPPR(4) | SPI_BR_SPR(2),
  SPI_BR_SPPR(5) | SPI_BR_SPR(2),
  SPI_BR_SPPR(6) | SPI_BR_SPR(2),
  SPI_BR_SPPR(7) | SPI_BR_SPR(2),
  SPI_BR_SPPR(4) | SPI_BR_SPR(3),
  SPI_BR_SPPR(5) | SPI_BR_SPR(3),
  SPI_BR_SPPR(6) | SPI_BR_SPR(3),
  SPI_BR_SPPR(7) | SPI_BR_SPR(3),
  SPI_BR_SPPR(4) | SPI_BR_SPR(4),
  SPI_BR_SPPR(5) | SPI_BR_SPR(4),
  SPI_BR_SPPR(6) | SPI_BR_SPR(4),
  SPI_BR_SPPR(7) | SPI_BR_SPR(4),
  SPI_BR_SPPR(4) | SPI_BR_SPR(5),
  SPI_BR_SPPR(5) | SPI_BR_SPR(5),
  SPI_BR_SPPR(6) | SPI_BR_SPR(5),
  SPI_BR_SPPR(7) | SPI_BR_SPR(5),
  SPI_BR_SPPR(4) | SPI_BR_SPR(6),
  SPI_BR_SPPR(5) | SPI_BR_SPR(6)
};

void SPIClass::setMOSI(uint8_t pin)
{
  if (pin != hardware().mosi_pin[mosi_pin_index]) {
    for (unsigned int i = 0; i < sizeof(hardware().mosi_pin); i++) {
      if (pin == hardware().mosi_pin[i] ) {
        if (hardware().clock_gate_register & hardware().clock_gate_mask) {
          volatile uint32_t *reg;
          reg = portConfigRegister(hardware().mosi_pin[mosi_pin_index]);
          *reg = 0;
          reg = portConfigRegister(hardware().mosi_pin[i]);
          *reg = hardware().mosi_mux[i];
        }
        mosi_pin_index = i;
        return;
      }
    }
  }
}

void SPIClass::setMISO(uint8_t pin)
{
  if (pin != hardware().miso_pin[miso_pin_index]) {
    for (unsigned int i = 0; i < sizeof(hardware().miso_pin); i++) {
      if (pin == hardware().miso_pin[i] ) {
        if (hardware().clock_gate_register & hardware().clock_gate_mask) {
          volatile uint32_t *reg;
          reg = portConfigRegister(hardware().miso_pin[miso_pin_index]);
          *reg = 0;
          reg = portConfigRegister(hardware().miso_pin[i]);
```

```
        *reg = hardware().miso_mux[i];
       }
       miso_pin_index = i;
       return;
     }
    }
   }
}

void SPIClass::setSCK(uint8_t pin)
{
  if (pin != hardware().sck_pin[sck_pin_index]) {
    for (unsigned int i = 0; i < sizeof(hardware().sck_pin); i++) {
      if (pin == hardware().sck_pin[i] ) {
        if (hardware().clock_gate_register & hardware().clock_gate_mask) {
          volatile uint32_t *reg;
          reg = portConfigRegister(hardware().sck_pin[sck_pin_index]);
          *reg = 0;
          reg = portConfigRegister(hardware().sck_pin[i]);
          *reg = hardware().sck_mux[i];
        }
        sck_pin_index = i;
        return;
      }
    }
  }
}

bool SPIClass::pinIsChipSelect(uint8_t pin)
{
  for (unsigned int i = 0; i < sizeof(hardware().cs_pin); i++) {
    if (pin == hardware().cs_pin[i]) return hardware().cs_mask[i];
  }
  return 0;
}

bool SPIClass::pinIsMOSI(uint8_t pin)
{
  for (unsigned int i = 0; i < sizeof(hardware().mosi_pin); i++) {
    if (pin == hardware().mosi_pin[i]) return true;
  }
  return false;
}

bool SPIClass::pinIsMISO(uint8_t pin)
{
  for (unsigned int i = 0; i < sizeof(hardware().miso_pin); i++) {
    if (pin == hardware().miso_pin[i]) return true;
  }
  return false;
}

bool SPIClass::pinIsSCK(uint8_t pin)
{
  for (unsigned int i = 0; i < sizeof(hardware().sck_pin); i++) {
    if (pin == hardware().sck_pin[i]) return true;
  }
```

```
  return false;
}

// setCS() is not intended for use from normal Arduino programs/sketches.
uint8_t SPIClass::setCS(uint8_t pin)
{
  for (unsigned int i = 0; i < sizeof(hardware().cs_pin); i++) {
    if  (pin == hardware().cs_pin[i]) {
      volatile uint32_t *reg = portConfigRegister(pin);
      *reg = hardware().cs_mux[i];
      return hardware().cs_mask[i];
    }
  }
  return 0;
}

void SPIClass::transfer(const void * buf, void * retbuf, size_t count) {
  if (count == 0) return;
  const uint8_t *p = (const uint8_t *)buf;
  uint8_t *pret = (uint8_t *)retbuf;
  uint8_t in;

  while (!(port().S & SPI_S_SPTEF)) ; // wait
  uint8_t out = p ? *p++ : _transferWriteFill;
  port().DL = out;
  while (--count > 0) {
    if (p) {
      out = *p++;
    }
    while (!(port().S & SPI_S_SPTEF)) ; // wait
    __disable_irq();
    port().DL = out;
    while (!(port().S & SPI_S_SPRF)) ; // wait
    in = port().DL;
    __enable_irq();
    if (pret)*pret++ = in;
  }
  while (!(port().S & SPI_S_SPRF)) ; // wait
  in = port().DL;
  if (pret)*pret = in;
}
//================================================================================
=
// ASYNCH Support
//================================================================================
=
//================================================================================
// Try Transfer using DMA.
//================================================================================
#ifdef SPI_HAS_TRANSFER_ASYNC
static uint8_t      _dma_dummy_rx;

void SPIClass::dma_isr(void) {
  //  Serial.println("_spi_dma_rxISR");
  _dmaRX->clearInterrupt();
  port().C2 = 0;
  uint8_t tmp __attribute__((unused)) = port().S;
```

```
  _dmaTX->clearComplete();
  _dmaRX->clearComplete();

  _dma_state = DMAState::completed;   // set back to 1 in case our call wants to start up dma again
  _dma_event_responder->triggerEvent();
}

bool SPIClass::initDMAChannels() {
  //Serial.println("First dma call"); Serial.flush();
  _dmaTX = new DMAChannel();
  if (_dmaTX == nullptr) {
    return false;
  }

  _dmaTX->disable();
  _dmaTX->destination((volatile uint8_t&)port().DL);
  _dmaTX->disableOnCompletion();
  _dmaTX->triggerAtHardwareEvent(hardware().tx_dma_channel);


  _dmaRX = new DMAChannel();
  if (_dmaRX == NULL) {
    delete _dmaTX;
    _dmaRX = nullptr;
    return false;
  }
  _dmaRX->disable();
  _dmaRX->source((volatile uint8_t&)port().DL);
  _dmaRX->disableOnCompletion();
  _dmaRX->triggerAtHardwareEvent(hardware().rx_dma_channel);
  _dmaRX->attachInterrupt(hardware().dma_isr);
  _dmaRX->interruptAtCompletion();

  _dma_state = DMAState::idle;  // Should be first thing set!
  //Serial.println("end First dma call");
  return true;
}

//=============================================================================
// Main Async Transfer function
//=============================================================================
bool SPIClass::transfer(const void *buf, void *retbuf, size_t count, EventResponderRef
event_responder) {
  if (_dma_state == DMAState::notAllocated) {
    if (!initDMAChannels()) {
      return false;
    }
  }

  if (_dma_state == DMAState::active)
    return false; // already active

  event_responder.clearEvent();       // Make sure it is not set yet

  if (count < 2) {
    // Use non-async version to simplify cases...
    transfer(buf, retbuf, count);
```

```
        event_responder.triggerEvent();
        return true;
      }
      //_dmaTX->destination((volatile uint8_t&)port().DL);
      //_dmaRX->source((volatile uint8_t&)port().DL);
      _dmaTX->CFG->DCR = (_dmaTX->CFG->DCR & ~DMA_DCR_DSIZE(3)) | DMA_DCR_DSIZE(1);
      _dmaRX->CFG->DCR = (_dmaRX->CFG->DCR & ~DMA_DCR_SSIZE(3)) | DMA_DCR_SSIZE(1);
// 8 bit transfer

      // Now see if the user passed in TX buffer to send.
      uint8_t first_char;
      if (buf) {
        uint8_t *data_out = (uint8_t*)buf;
        first_char = *data_out++;
        _dmaTX->sourceBuffer(data_out, count - 1);
      } else {
        first_char = (_transferWriteFill & 0xff);
        _dmaTX->source((uint8_t&)_transferWriteFill);   // maybe have setable value
        _dmaTX->transferCount(count - 1);
      }

      if (retbuf) {
        _dmaRX->destinationBuffer((uint8_t*)retbuf, count);
      } else {
        _dmaRX->destination(_dma_dummy_rx);    // NULL ?
        _dmaRX->transferCount(count);
      }

      _dma_event_responder = &event_responder;

      //Serial.println("Before DMA C2");
      // Try pushing the first character
      while (!(port().S & SPI_S_SPTEF));
      port().DL = first_char;

      port().C2 |= SPI_C2_TXDMAE | SPI_C2_RXDMAE;

      // Now  make sure SPI is enabled.
      port().C1 |= SPI_C1_SPE;

      _dmaRX->enable();
      _dmaTX->enable();
      _dma_state = DMAState::active;
      return true;
    }
    #endif //SPI_HAS_TRANSFER_ASYNC

    #endif
```

1.5.1.23 Codi font SPI.h

```
    /*
      Copyright (c) 2010 by Cristian Maglie <c.maglie@bug.st>
      Copyright (c) 2014 by Paul Stoffregen <paul@pjrc.com> (Transaction API)
      Copyright (c) 2014 by Matthijs Kooijman <matthijs@stdin.nl> (SPISettings AVR)
      SPI Master library for arduino.

      This file is free software; you can redistribute it and/or modify
```

```
   it under the terms of either the GNU General Public License version 2
   or the GNU Lesser General Public License version 2.1, both as
   published by the Free Software Foundation.
*/

#ifndef _SPI_H_INCLUDED
#define _SPI_H_INCLUDED

#include <Arduino.h>

#if defined(__arm__) && defined(TEENSYDUINO)
#if defined(__has_include) && __has_include(<EventResponder.h>)
// SPI_HAS_TRANSFER_ASYNC - Defined to say that the SPI supports an ASYNC version
// of the SPI_HAS_TRANSFER_BUF
#define SPI_HAS_TRANSFER_ASYNC 1
#include <DMAChannel.h>
#include <EventResponder.h>
#endif
#endif

// SPI_HAS_TRANSACTION means SPI has beginTransaction(), endTransaction(),
// usingInterrupt(), and SPISetting(clock, bitOrder, dataMode)
#define SPI_HAS_TRANSACTION 1

// Uncomment this line to add detection of mismatched begin/end transactions.
// A mismatch occurs if other libraries fail to use SPI.endTransaction() for
// each SPI.beginTransaction().  Connect a LED to this pin.  The LED will turn
// on if any mismatch is ever detected.
//#define SPI_TRANSACTION_MISMATCH_LED 5

// SPI_HAS_TRANSFER_BUF - is defined to signify that this library supports
// a version of transfer which allows you to pass in both TX and RX buffer
// pointers, either of which could be NULL
#define SPI_HAS_TRANSFER_BUF 1


#ifndef LSBFIRST
#define LSBFIRST 0
#endif
#ifndef MSBFIRST
#define MSBFIRST 1
#endif

#define SPI_MODE0 0x00
#define SPI_MODE1 0x04
#define SPI_MODE2 0x08
#define SPI_MODE3 0x0C

#define SPI_CLOCK_DIV4 0x00
#define SPI_CLOCK_DIV16 0x01
#define SPI_CLOCK_DIV64 0x02
#define SPI_CLOCK_DIV128 0x03
#define SPI_CLOCK_DIV2 0x04
#define SPI_CLOCK_DIV8 0x05
#define SPI_CLOCK_DIV32 0x06

#define SPI_MODE_MASK 0x0C  // CPOL = bit 3, CPHA = bit 2 on SPCR
```

Controlador de reg per degoteig

```
#define SPI_CLOCK_MASK 0x03  // SPR1 = bit 1, SPR0 = bit 0 on SPCR
#define SPI_2XCLOCK_MASK 0x01  // SPI2X = bit 0 on SPSR


/*************************************************************/
/*     8 bit AVR-based boards                                */
/*************************************************************/

#if defined(__AVR__)

// define SPI_AVR_EIMSK for AVR boards with external interrupt pins
#if defined(EIMSK)
#define SPI_AVR_EIMSK   EIMSK
#elif defined(GICR)
#define SPI_AVR_EIMSK   GICR
#elif defined(GIMSK)
#define SPI_AVR_EIMSK   GIMSK
#endif

class SPISettings {
  public:
    SPISettings(uint32_t clock, uint8_t bitOrder, uint8_t dataMode) {
      if (__builtin_constant_p(clock)) {
        init_AlwaysInline(clock, bitOrder, dataMode);
      } else {
        init_MightInline(clock, bitOrder, dataMode);
      }
    }
    SPISettings() {
      init_AlwaysInline(4000000, MSBFIRST, SPI_MODE0);
    }
  private:
    void init_MightInline(uint32_t clock, uint8_t bitOrder, uint8_t dataMode) {
      init_AlwaysInline(clock, bitOrder, dataMode);
    }
    void init_AlwaysInline(uint32_t clock, uint8_t bitOrder, uint8_t dataMode)
    __attribute__((__always_inline__)) {
      // Clock settings are defined as follows. Note that this shows SPI2X
      // inverted, so the bits form increasing numbers. Also note that
      // fosc/64 appears twice
      // SPR1 SPR0 ~SPI2X Freq
      //   0    0     0   fosc/2
      //   0    0     1   fosc/4
      //   0    1     0   fosc/8
      //   0    1     1   fosc/16
      //   1    0     0   fosc/32
      //   1    0     1   fosc/64
      //   1    1     0   fosc/64
      //   1    1     1   fosc/128

      // We find the fastest clock that is less than or equal to the
      // given clock rate. The clock divider that results in clock_setting
      // is 2 ^^ (clock_div + 1). If nothing is slow enough, we'll use the
      // slowest (128 == 2 ^^ 7, so clock_div = 6).
      uint8_t clockDiv;

      // When the clock is known at compiletime, use this if-then-else
```

```
      // cascade, which the compiler knows how to completely optimize
      // away. When clock is not known, use a loop instead, which generates
      // shorter code.
      if (__builtin_constant_p(clock)) {
        if (clock >= F_CPU / 2) {
          clockDiv = 0;
        } else if (clock >= F_CPU / 4) {
          clockDiv = 1;
        } else if (clock >= F_CPU / 8) {
          clockDiv = 2;
        } else if (clock >= F_CPU / 16) {
          clockDiv = 3;
        } else if (clock >= F_CPU / 32) {
          clockDiv = 4;
        } else if (clock >= F_CPU / 64) {
          clockDiv = 5;
        } else {
          clockDiv = 6;
        }
      } else {
        uint32_t clockSetting = F_CPU / 2;
        clockDiv = 0;
        while (clockDiv < 6 && clock < clockSetting) {
          clockSetting /= 2;
          clockDiv++;
        }
      }

      // Compensate for the duplicate fosc/64
      if (clockDiv == 6)
        clockDiv = 7;

      // Invert the SPI2X bit
      clockDiv ^= 0x1;

      // Pack into the SPISettings class
      spcr = _BV(SPE) | _BV(MSTR) | ((bitOrder == LSBFIRST) ? _BV(DORD) : 0) |
             (dataMode & SPI_MODE_MASK) | ((clockDiv >> 1) & SPI_CLOCK_MASK);
      spsr = clockDiv & SPI_2XCLOCK_MASK;
    }
    uint8_t spcr;
    uint8_t spsr;
    friend class SPIClass;
};



class SPIClass { // AVR
  public:
    // Initialize the SPI library
    static void begin();

    // If SPI is used from within an interrupt, this function registers
    // that interrupt with the SPI library, so beginTransaction() can
    // prevent conflicts.  The input interruptNumber is the number used
    // with attachInterrupt.  If SPI is used from a different interrupt
    // (eg, a timer), interruptNumber should be 255.
```

```
  static void usingInterrupt(uint8_t interruptNumber);

  // Before using SPI.transfer() or asserting chip select pins,
  // this function is used to gain exclusive access to the SPI bus
  // and configure the correct settings.
  inline static void beginTransaction(SPISettings settings) {
    if (interruptMode > 0) {
#ifdef SPI_AVR_EIMSK
      if (interruptMode == 1) {
        interruptSave = SPI_AVR_EIMSK;
        SPI_AVR_EIMSK &= ~interruptMask;
      } else
#endif
      {
        uint8_t tmp = SREG;
        cli();
        interruptSave = tmp;
      }
    }
#ifdef SPI_TRANSACTION_MISMATCH_LED
    if (inTransactionFlag) {
      pinMode(SPI_TRANSACTION_MISMATCH_LED, OUTPUT);
      digitalWrite(SPI_TRANSACTION_MISMATCH_LED, HIGH);
    }
    inTransactionFlag = 1;
#endif
    SPCR = settings.spcr;
    SPSR = settings.spsr;
  }

  // Write to the SPI bus (MOSI pin) and also receive (MISO pin)
  inline static uint8_t transfer(uint8_t data) {
    SPDR = data;
    asm volatile("nop");
    while (!(SPSR & _BV(SPIF))) ; // wait
    return SPDR;
  }
  inline static uint16_t transfer16(uint16_t data) {
    union {
      uint16_t val;
      struct {
        uint8_t lsb;
        uint8_t msb;
      };
    } in, out;
    in.val = data;
    if ((SPCR & _BV(DORD))) {
      SPDR = in.lsb;
      asm volatile("nop");
      while (!(SPSR & _BV(SPIF))) ;
      out.lsb = SPDR;
      SPDR = in.msb;
      asm volatile("nop");
      while (!(SPSR & _BV(SPIF))) ;
      out.msb = SPDR;
    } else {
      SPDR = in.msb;
```

```
      asm volatile("nop");
      while (!(SPSR & _BV(SPIF))) ;
      out.msb = SPDR;
      SPDR = in.lsb;
      asm volatile("nop");
      while (!(SPSR & _BV(SPIF))) ;
      out.lsb = SPDR;
    }
    return out.val;
  }
  inline static void transfer(void *buf, size_t count) {
    if (count == 0) return;
    uint8_t *p = (uint8_t *)buf;
    SPDR = *p;
    while (--count > 0) {
      uint8_t out = *(p + 1);
      while (!(SPSR & _BV(SPIF))) ;
      uint8_t in = SPDR;
      SPDR = out;
      *p++ = in;
    }
    while (!(SPSR & _BV(SPIF))) ;
    *p = SPDR;
  }
  static void setTransferWriteFill(uint8_t ch ) {
    _transferWriteFill = ch;
  }
  static void transfer(const void * buf, void * retbuf, uint32_t count);

  // After performing a group of transfers and releasing the chip select
  // signal, this function allows others to access the SPI bus
  inline static void endTransaction(void) {
#ifdef SPI_TRANSACTION_MISMATCH_LED
    if (!inTransactionFlag) {
      pinMode(SPI_TRANSACTION_MISMATCH_LED, OUTPUT);
      digitalWrite(SPI_TRANSACTION_MISMATCH_LED, HIGH);
    }
    inTransactionFlag = 0;
#endif
    if (interruptMode > 0) {
#ifdef SPI_AVR_EIMSK
      if (interruptMode == 1) {
        SPI_AVR_EIMSK = interruptSave;
      } else
#endif
      {
        SREG = interruptSave;
      }
    }
  }

  // Disable the SPI bus
  static void end();

  // This function is deprecated.       New applications should use
  // beginTransaction() to configure SPI settings.
  inline static void setBitOrder(uint8_t bitOrder) {
```

```
    if (bitOrder == LSBFIRST) SPCR |= _BV(DORD);
    else SPCR &= ~(_BV(DORD));
  }
  // This function is deprecated.  New applications should use
  // beginTransaction() to configure SPI settings.
  inline static void setDataMode(uint8_t dataMode) {
    SPCR = (SPCR & ~SPI_MODE_MASK) | dataMode;
  }
  // This function is deprecated.  New applications should use
  // beginTransaction() to configure SPI settings.
  inline static void setClockDivider(uint8_t clockDiv) {
    SPCR = (SPCR & ~SPI_CLOCK_MASK) | (clockDiv & SPI_CLOCK_MASK);
    SPSR = (SPSR & ~SPI_2XCLOCK_MASK) | ((clockDiv >> 2) & SPI_2XCLOCK_MASK);
  }
  // These undocumented functions should not be used.  SPI.transfer()
  // polls the hardware flag which is automatically cleared as the
  // AVR responds to SPI's interrupt
  inline static void attachInterrupt() {
    SPCR |= _BV(SPIE);
  }
  inline static void detachInterrupt() {
    SPCR &= ~_BV(SPIE);
  }

 private:
  static uint8_t interruptMode; // 0=none, 1=mask, 2=global
  static uint8_t interruptMask; // which interrupts to mask
  static uint8_t interruptSave; // temp storage, to restore state
#ifdef SPI_TRANSACTION_MISMATCH_LED
  static uint8_t inTransactionFlag;
#endif
  static uint8_t _transferWriteFill;
};



/*************************************************************/
/*     32 bit Teensy 3.x                                     */
/*************************************************************/

#elif defined(__arm__) && defined(TEENSYDUINO) && defined(KINETISK)

#define SPI_HAS_NOTUSINGINTERRUPT 1

class SPISettings {
 public:
  SPISettings(uint32_t clock, uint8_t bitOrder, uint8_t dataMode) {
    if (__builtin_constant_p(clock)) {
      init_AlwaysInline(clock, bitOrder, dataMode);
    } else {
      init_MightInline(clock, bitOrder, dataMode);
    }
  }
  SPISettings() {
    init_AlwaysInline(4000000, MSBFIRST, SPI_MODE0);
  }
 private:
```

Controlador de reg per degoteig

```
void init_MightInline(uint32_t clock, uint8_t bitOrder, uint8_t dataMode) {
  init_AlwaysInline(clock, bitOrder, dataMode);
}
void init_AlwaysInline(uint32_t clock, uint8_t bitOrder, uint8_t dataMode)
__attribute__((__always_inline__)) {
  uint32_t t, c = SPI_CTAR_FMSZ(7);
  if (bitOrder == LSBFIRST) c |= SPI_CTAR_LSBFE;
  if (__builtin_constant_p(clock)) {
    if    (clock >= F_BUS / 2) {
      t = SPI_CTAR_PBR(0) | SPI_CTAR_BR(0) | SPI_CTAR_DBR
          | SPI_CTAR_CSSCK(0);
    } else if (clock >= F_BUS / 3) {
      t = SPI_CTAR_PBR(1) | SPI_CTAR_BR(0) | SPI_CTAR_DBR
          | SPI_CTAR_CSSCK(0);
    } else if (clock >= F_BUS / 4) {
      t = SPI_CTAR_PBR(0) | SPI_CTAR_BR(0) | SPI_CTAR_CSSCK(0);
    } else if (clock >= F_BUS / 5) {
      t = SPI_CTAR_PBR(2) | SPI_CTAR_BR(0) | SPI_CTAR_DBR
          | SPI_CTAR_CSSCK(0);
    } else if (clock >= F_BUS / 6) {
      t = SPI_CTAR_PBR(1) | SPI_CTAR_BR(0) | SPI_CTAR_CSSCK(0);
    } else if (clock >= F_BUS / 8) {
      t = SPI_CTAR_PBR(0) | SPI_CTAR_BR(1) | SPI_CTAR_CSSCK(1);
    } else if (clock >= F_BUS / 10) {
      t = SPI_CTAR_PBR(2) | SPI_CTAR_BR(0) | SPI_CTAR_CSSCK(0);
    } else if (clock >= F_BUS / 12) {
      t = SPI_CTAR_PBR(1) | SPI_CTAR_BR(1) | SPI_CTAR_CSSCK(1);
    } else if (clock >= F_BUS / 16) {
      t = SPI_CTAR_PBR(0) | SPI_CTAR_BR(3) | SPI_CTAR_CSSCK(2);
    } else if (clock >= F_BUS / 20) {
      t = SPI_CTAR_PBR(2) | SPI_CTAR_BR(1) | SPI_CTAR_CSSCK(0);
    } else if (clock >= F_BUS / 24) {
      t = SPI_CTAR_PBR(1) | SPI_CTAR_BR(3) | SPI_CTAR_CSSCK(2);
    } else if (clock >= F_BUS / 32) {
      t = SPI_CTAR_PBR(0) | SPI_CTAR_BR(4) | SPI_CTAR_CSSCK(3);
    } else if (clock >= F_BUS / 40) {
      t = SPI_CTAR_PBR(2) | SPI_CTAR_BR(3) | SPI_CTAR_CSSCK(2);
    } else if (clock >= F_BUS / 56) {
      t = SPI_CTAR_PBR(3) | SPI_CTAR_BR(3) | SPI_CTAR_CSSCK(2);
    } else if (clock >= F_BUS / 64) {
      t = SPI_CTAR_PBR(0) | SPI_CTAR_BR(5) | SPI_CTAR_CSSCK(4);
    } else if (clock >= F_BUS / 96) {
      t = SPI_CTAR_PBR(1) | SPI_CTAR_BR(5) | SPI_CTAR_CSSCK(4);
    } else if (clock >= F_BUS / 128) {
      t = SPI_CTAR_PBR(0) | SPI_CTAR_BR(6) | SPI_CTAR_CSSCK(5);
    } else if (clock >= F_BUS / 192) {
      t = SPI_CTAR_PBR(1) | SPI_CTAR_BR(6) | SPI_CTAR_CSSCK(5);
    } else if (clock >= F_BUS / 256) {
      t = SPI_CTAR_PBR(0) | SPI_CTAR_BR(7) | SPI_CTAR_CSSCK(6);
    } else if (clock >= F_BUS / 384) {
      t = SPI_CTAR_PBR(1) | SPI_CTAR_BR(7) | SPI_CTAR_CSSCK(6);
    } else if (clock >= F_BUS / 512) {
      t = SPI_CTAR_PBR(0) | SPI_CTAR_BR(8) | SPI_CTAR_CSSCK(7);
    } else if (clock >= F_BUS / 640) {
      t = SPI_CTAR_PBR(2) | SPI_CTAR_BR(7) | SPI_CTAR_CSSCK(6);
    } else { /* F_BUS / 768 */
      t = SPI_CTAR_PBR(1) | SPI_CTAR_BR(8) | SPI_CTAR_CSSCK(7);
```

```
      }
     } else {
      for (uint32_t i = 0; i < 23; i++) {
       t = ctar_clock_table[i];
       if (clock >= F_BUS / ctar_div_table[i]) break;
      }
     }
     if (dataMode & 0x08) {
      c |= SPI_CTAR_CPOL;
     }
     if (dataMode & 0x04) {
      c |= SPI_CTAR_CPHA;
      t = (t & 0xFFFF0FFF) | ((t & 0xF000) >> 4);
     }
     ctar = c | t;
    }
    static const uint16_t ctar_div_table[23];
    static const uint32_t ctar_clock_table[23];
    uint32_t ctar;
    friend class SPIClass;
};



class SPIClass { // Teensy 3.x
  public:
#if defined(__MK20DX128__) || defined(__MK20DX256__)
    static const uint8_t CNT_MISO_PINS = 2;
    static const uint8_t CNT_MOSI_PINS = 2;
    static const uint8_t CNT_SCK_PINS = 2;
    static const uint8_t CNT_CS_PINS = 9;
#elif defined(__MK64FX512__) || defined(__MK66FX1M0__)
    static const uint8_t CNT_MISO_PINS = 4;
    static const uint8_t CNT_MOSI_PINS = 4;
    static const uint8_t CNT_SCK_PINS = 3;
    static const uint8_t CNT_CS_PINS = 11;
#endif
    typedef struct {
      volatile uint32_t &clock_gate_register;
      uint32_t clock_gate_mask;
      uint8_t  queue_size;
      uint8_t  spi_irq;
      uint32_t max_dma_count;
      uint8_t  tx_dma_channel;
      uint8_t  rx_dma_channel;
      void     (*dma_rxisr)();
      uint8_t  miso_pin[CNT_MISO_PINS];
      uint32_t  miso_mux[CNT_MISO_PINS];
      uint8_t  mosi_pin[CNT_MOSI_PINS];
      uint32_t  mosi_mux[CNT_MOSI_PINS];
      uint8_t  sck_pin[CNT_SCK_PINS];
      uint32_t  sck_mux[CNT_SCK_PINS];
      uint8_t  cs_pin[CNT_CS_PINS];
      uint32_t  cs_mux[CNT_CS_PINS];
      uint8_t  cs_mask[CNT_CS_PINS];
    } SPI_Hardware_t;
    static const SPI_Hardware_t spi0_hardware;
```

```
  static const SPI_Hardware_t spi1_hardware;
  static const SPI_Hardware_t spi2_hardware;

  enum DMAState { notAllocated, idle, active, completed};
 public:
  constexpr SPIClass(uintptr_t myport, uintptr_t myhardware)
    : port_addr(myport), hardware_addr(myhardware) {
  }
  // Initialize the SPI library
  void begin();

  // If SPI is to used from within an interrupt, this function registers
  // that interrupt with the SPI library, so beginTransaction() can
  // prevent conflicts.  The input interruptNumber is the number used
  // with attachInterrupt.  If SPI is used from a different interrupt
  // (eg, a timer), interruptNumber should be 255.
  void usingInterrupt(uint8_t n) {
    if (n == 3 || n == 4 || n == 24 || n == 33) {
      usingInterrupt(IRQ_PORTA);
    } else if (n == 0 || n == 1 || (n >= 16 && n <= 19) || n == 25 || n == 32) {
      usingInterrupt(IRQ_PORTB);
    } else if ((n >= 9 && n <= 13) || n == 15 || n == 22 || n == 23
            || (n >= 27 && n <= 30)) {
      usingInterrupt(IRQ_PORTC);
    } else if (n == 2 || (n >= 5 && n <= 8) || n == 14 || n == 20 || n == 21) {
      usingInterrupt(IRQ_PORTD);
    } else if (n == 26 || n == 31) {
      usingInterrupt(IRQ_PORTE);
    }
  }
  void usingInterrupt(IRQ_NUMBER_t interruptName);
  void notUsingInterrupt(IRQ_NUMBER_t interruptName);

  // Before using SPI.transfer() or asserting chip select pins,
  // this function is used to gain exclusive access to the SPI bus
  // and configure the correct settings.
  void beginTransaction(SPISettings settings) {
    if (interruptMasksUsed) {
      __disable_irq();
      if (interruptMasksUsed & 0x01) {
        interruptSave[0] = NVIC_ICER0 & interruptMask[0];
        NVIC_ICER0 = interruptSave[0];
      }
#if NVIC_NUM_INTERRUPTS > 32
      if (interruptMasksUsed & 0x02) {
        interruptSave[1] = NVIC_ICER1 & interruptMask[1];
        NVIC_ICER1 = interruptSave[1];
      }
#endif
#if NVIC_NUM_INTERRUPTS > 64 && defined(NVIC_ISER2)
      if (interruptMasksUsed & 0x04) {
        interruptSave[2] = NVIC_ICER2 & interruptMask[2];
        NVIC_ICER2 = interruptSave[2];
      }
#endif
#if NVIC_NUM_INTERRUPTS > 96 && defined(NVIC_ISER3)
      if (interruptMasksUsed & 0x08) {
```

```
            interruptSave[3] = NVIC_ICER3 & interruptMask[3];
            NVIC_ICER3 = interruptSave[3];
          }
        #endif
          __enable_irq();
        }
        #ifdef SPI_TRANSACTION_MISMATCH_LED
        if (inTransactionFlag) {
          pinMode(SPI_TRANSACTION_MISMATCH_LED, OUTPUT);
          digitalWrite(SPI_TRANSACTION_MISMATCH_LED, HIGH);
        }
        inTransactionFlag = 1;
        #endif
        if (port().CTAR0 != settings.ctar) {
          port().MCR = SPI_MCR_MDIS | SPI_MCR_HALT | SPI_MCR_PCSIS(0x3F);
          port().CTAR0 = settings.ctar;
          port().CTAR1 = settings.ctar | SPI_CTAR_FMSZ(8);
          port().MCR = SPI_MCR_MSTR | SPI_MCR_PCSIS(0x3F);
        }
      }

      // Write to the SPI bus (MOSI pin) and also receive (MISO pin)
      uint8_t transfer(uint8_t data) {
        port().SR = SPI_SR_TCF;
        port().PUSHR = data;
        while (!(port().SR & SPI_SR_TCF)) ; // wait
        return port().POPR;
      }
      uint16_t transfer16(uint16_t data) {
        port().SR = SPI_SR_TCF;
        port().PUSHR = data | SPI_PUSHR_CTAS(1);
        while (!(port().SR & SPI_SR_TCF)) ; // wait
        return port().POPR;
      }

      void inline transfer(void *buf, size_t count) {
        transfer(buf, buf, count);
      }
      void setTransferWriteFill(uint8_t ch ) {
        _transferWriteFill = ch;
      }
      void transfer(const void * buf, void * retbuf, size_t count);

      // Asynch support (DMA )
    #ifdef SPI_HAS_TRANSFER_ASYNC
      bool transfer(const void *txBuffer, void *rxBuffer, size_t count,  EventResponderRef
event_responder);

      friend void _spi_dma_rxISR0(void);
      friend void _spi_dma_rxISR1(void);
      friend void _spi_dma_rxISR2(void);

      inline void dma_rxisr(void);
    #endif


      // After performing a group of transfers and releasing the chip select
```

```
    // signal, this function allows others to access the SPI bus
    void endTransaction(void) {
#ifdef SPI_TRANSACTION_MISMATCH_LED
      if (!inTransactionFlag) {
        pinMode(SPI_TRANSACTION_MISMATCH_LED, OUTPUT);
        digitalWrite(SPI_TRANSACTION_MISMATCH_LED, HIGH);
      }
      inTransactionFlag = 0;
#endif
      if (interruptMasksUsed) {
        if (interruptMasksUsed & 0x01) {
          NVIC_ISER0 = interruptSave[0];
        }
#if NVIC_NUM_INTERRUPTS > 32
        if (interruptMasksUsed & 0x02) {
          NVIC_ISER1 = interruptSave[1];
        }
#endif
#if NVIC_NUM_INTERRUPTS > 64 && defined(NVIC_ISER2)
        if (interruptMasksUsed & 0x04) {
          NVIC_ISER2 = interruptSave[2];
        }
#endif
#if NVIC_NUM_INTERRUPTS > 96 && defined(NVIC_ISER3)
        if (interruptMasksUsed & 0x08) {
          NVIC_ISER3 = interruptSave[3];
        }
#endif
      }
    }

    // Disable the SPI bus
    void end();

    // This function is deprecated.       New applications should use
    // beginTransaction() to configure SPI settings.
    void setBitOrder(uint8_t bitOrder);

    // This function is deprecated.       New applications should use
    // beginTransaction() to configure SPI settings.
    void setDataMode(uint8_t dataMode);

    // This function is deprecated.       New applications should use
    // beginTransaction() to configure SPI settings.
    void setClockDivider(uint8_t clockDiv) {
      if (clockDiv == SPI_CLOCK_DIV2) {
        setClockDivider_noInline(SPISettings(12000000, MSBFIRST, SPI_MODE0).ctar);
      } else if (clockDiv == SPI_CLOCK_DIV4) {
        setClockDivider_noInline(SPISettings(4000000, MSBFIRST, SPI_MODE0).ctar);
      } else if (clockDiv == SPI_CLOCK_DIV8) {
        setClockDivider_noInline(SPISettings(2000000, MSBFIRST, SPI_MODE0).ctar);
      } else if (clockDiv == SPI_CLOCK_DIV16) {
        setClockDivider_noInline(SPISettings(1000000, MSBFIRST, SPI_MODE0).ctar);
      } else if (clockDiv == SPI_CLOCK_DIV32) {
        setClockDivider_noInline(SPISettings(500000, MSBFIRST, SPI_MODE0).ctar);
      } else if (clockDiv == SPI_CLOCK_DIV64) {
        setClockDivider_noInline(SPISettings(250000, MSBFIRST, SPI_MODE0).ctar);
```

```
    } else { /* clockDiv == SPI_CLOCK_DIV128 */
      setClockDivider_noInline(SPISettings(125000, MSBFIRST, SPI_MODE0).ctar);
    }
  }
  void setClockDivider_noInline(uint32_t clk);

  // These undocumented functions should not be used.  SPI.transfer()
  // polls the hardware flag which is automatically cleared as the
  // AVR responds to SPI's interrupt
  void attachInterrupt() { }
  void detachInterrupt() { }

  // Teensy 3.x can use alternate pins for these 3 SPI signals.
  void setMOSI(uint8_t pin);
  void setMISO(uint8_t pin);
  void setSCK(uint8_t pin);

  // return true if "pin" has special chip select capability
  uint8_t pinIsChipSelect(uint8_t pin);
  bool pinIsMOSI(uint8_t pin);
  bool pinIsMISO(uint8_t pin);
  bool pinIsSCK(uint8_t pin);
  // return true if both pin1 and pin2 have independent chip select capability
  bool pinIsChipSelect(uint8_t pin1, uint8_t pin2);
  // configure a pin for chip select and return its SPI_MCR_PCSIS bitmask
  // setCS() is a special function, not intended for use from normal Arduino
  // programs/sketches.  See the ILI3941_t3 library for an example.
  uint8_t setCS(uint8_t pin);

 private:
  KINETISK_SPI_t & port() {
    return *(KINETISK_SPI_t *)port_addr;
  }
  const SPI_Hardware_t & hardware() {
    return *(const SPI_Hardware_t *)hardware_addr;
  }
  void updateCTAR(uint32_t ctar);
  uintptr_t port_addr;
  uintptr_t hardware_addr;
  uint8_t miso_pin_index = 0;
  uint8_t mosi_pin_index = 0;
  uint8_t sck_pin_index = 0;
  uint8_t interruptMasksUsed = 0;
  uint32_t interruptMask[(NVIC_NUM_INTERRUPTS + 31) / 32] = {};
  uint32_t interruptSave[(NVIC_NUM_INTERRUPTS + 31) / 32] = {};
#ifdef SPI_TRANSACTION_MISMATCH_LED
  uint8_t inTransactionFlag = 0;
#endif

  uint8_t _transferWriteFill = 0;

  // DMA Support
#ifdef SPI_HAS_TRANSFER_ASYNC
  bool initDMAChannels();
  DMAState     _dma_state = DMAState::notAllocated;
  uint32_t   _dma_count_remaining = 0; // How many bytes left to output after current DMA completes
  DMAChannel   *_dmaTX = nullptr;
```

```
    DMAChannel    *_dmaRX = nullptr;
    EventResponder *_dma_event_responder = nullptr;
#endif
};



/**********************************************************/
/*     32 bit Teensy-LC                                   */
/**********************************************************/

#elif defined(__arm__) && defined(TEENSYDUINO) && defined(KINETISL)

class SPISettings {
  public:
    SPISettings(uint32_t clock, uint8_t bitOrder, uint8_t dataMode) {
      if (__builtin_constant_p(clock)) {
        init_AlwaysInline(clock, bitOrder, dataMode);
      } else {
        init_MightInline(clock, bitOrder, dataMode);
      }
    }
    SPISettings() {
      init_AlwaysInline(4000000, MSBFIRST, SPI_MODE0);
    }
  private:
    void init_MightInline(uint32_t clock, uint8_t bitOrder, uint8_t dataMode) {
      init_AlwaysInline(clock, bitOrder, dataMode);
    }
    void init_AlwaysInline(uint32_t clock, uint8_t bitOrder, uint8_t dataMode)
    __attribute__((__always_inline__)) {
      uint8_t c = SPI_C1_MSTR | SPI_C1_SPE;
      if (dataMode & 0x04) c |= SPI_C1_CPHA;
      if (dataMode & 0x08) c |= SPI_C1_CPOL;
      if (bitOrder == LSBFIRST) c |= SPI_C1_LSBFE;
      c1 = c;
      if (__builtin_constant_p(clock)) {
        if    (clock >= F_BUS /   2) {
          c = SPI_BR_SPPR(0) | SPI_BR_SPR(0);
        } else if (clock >= F_BUS /   4) {
          c = SPI_BR_SPPR(1) | SPI_BR_SPR(0);
        } else if (clock >= F_BUS /   6) {
          c = SPI_BR_SPPR(2) | SPI_BR_SPR(0);
        } else if (clock >= F_BUS /   8) {
          c = SPI_BR_SPPR(3) | SPI_BR_SPR(0);
        } else if (clock >= F_BUS /  10) {
          c = SPI_BR_SPPR(4) | SPI_BR_SPR(0);
        } else if (clock >= F_BUS /  12) {
          c = SPI_BR_SPPR(5) | SPI_BR_SPR(0);
        } else if (clock >= F_BUS /  14) {
          c = SPI_BR_SPPR(6) | SPI_BR_SPR(0);
        } else if (clock >= F_BUS /  16) {
          c = SPI_BR_SPPR(7) | SPI_BR_SPR(0);
        } else if (clock >= F_BUS /  20) {
          c = SPI_BR_SPPR(4) | SPI_BR_SPR(1);
        } else if (clock >= F_BUS /  24) {
          c = SPI_BR_SPPR(5) | SPI_BR_SPR(1);
```

```
  } else if (clock >= F_BUS /  28) {
    c = SPI_BR_SPPR(6) | SPI_BR_SPR(1);
  } else if (clock >= F_BUS /  32) {
    c = SPI_BR_SPPR(7) | SPI_BR_SPR(1);
  } else if (clock >= F_BUS /  40) {
    c = SPI_BR_SPPR(4) | SPI_BR_SPR(2);
  } else if (clock >= F_BUS /  48) {
    c = SPI_BR_SPPR(5) | SPI_BR_SPR(2);
  } else if (clock >= F_BUS /  56) {
    c = SPI_BR_SPPR(6) | SPI_BR_SPR(2);
  } else if (clock >= F_BUS /  64) {
    c = SPI_BR_SPPR(7) | SPI_BR_SPR(2);
  } else if (clock >= F_BUS /  80) {
    c = SPI_BR_SPPR(4) | SPI_BR_SPR(3);
  } else if (clock >= F_BUS /  96) {
    c = SPI_BR_SPPR(5) | SPI_BR_SPR(3);
  } else if (clock >= F_BUS / 112) {
    c = SPI_BR_SPPR(6) | SPI_BR_SPR(3);
  } else if (clock >= F_BUS / 128) {
    c = SPI_BR_SPPR(7) | SPI_BR_SPR(3);
  } else if (clock >= F_BUS / 160) {
    c = SPI_BR_SPPR(4) | SPI_BR_SPR(4);
  } else if (clock >= F_BUS / 192) {
    c = SPI_BR_SPPR(5) | SPI_BR_SPR(4);
  } else if (clock >= F_BUS / 224) {
    c = SPI_BR_SPPR(6) | SPI_BR_SPR(4);
  } else if (clock >= F_BUS / 256) {
    c = SPI_BR_SPPR(7) | SPI_BR_SPR(4);
  } else if (clock >= F_BUS / 320) {
    c = SPI_BR_SPPR(4) | SPI_BR_SPR(5);
  } else if (clock >= F_BUS / 384) {
    c = SPI_BR_SPPR(5) | SPI_BR_SPR(5);
  } else if (clock >= F_BUS / 448) {
    c = SPI_BR_SPPR(6) | SPI_BR_SPR(5);
  } else if (clock >= F_BUS / 512) {
    c = SPI_BR_SPPR(7) | SPI_BR_SPR(5);
  } else if (clock >= F_BUS / 640) {
    c = SPI_BR_SPPR(4) | SPI_BR_SPR(6);
  } else {    /* F_BUS / 768 */
    c = SPI_BR_SPPR(5) | SPI_BR_SPR(6);
  }
} else {
  for (uint32_t i = 0; i < 30; i++) {
    c = br_clock_table[i];
    if (clock >= F_BUS / br_div_table[i]) break;
  }
}
br[0] = c;
if (__builtin_constant_p(clock)) {
  if    (clock >= (F_PLL / 2) /   2) {
    c = SPI_BR_SPPR(0) | SPI_BR_SPR(0);
  } else if (clock >= (F_PLL / 2) /   4) {
    c = SPI_BR_SPPR(1) | SPI_BR_SPR(0);
  } else if (clock >= (F_PLL / 2) /   6) {
    c = SPI_BR_SPPR(2) | SPI_BR_SPR(0);
  } else if (clock >= (F_PLL / 2) /   8) {
    c = SPI_BR_SPPR(3) | SPI_BR_SPR(0);
```

```
  } else if (clock >= (F_PLL / 2) /  10) {
    c = SPI_BR_SPPR(4) | SPI_BR_SPR(0);
  } else if (clock >= (F_PLL / 2) /  12) {
    c = SPI_BR_SPPR(5) | SPI_BR_SPR(0);
  } else if (clock >= (F_PLL / 2) /  14) {
    c = SPI_BR_SPPR(6) | SPI_BR_SPR(0);
  } else if (clock >= (F_PLL / 2) /  16) {
    c = SPI_BR_SPPR(7) | SPI_BR_SPR(0);
  } else if (clock >= (F_PLL / 2) /  20) {
    c = SPI_BR_SPPR(4) | SPI_BR_SPR(1);
  } else if (clock >= (F_PLL / 2) /  24) {
    c = SPI_BR_SPPR(5) | SPI_BR_SPR(1);
  } else if (clock >= (F_PLL / 2) /  28) {
    c = SPI_BR_SPPR(6) | SPI_BR_SPR(1);
  } else if (clock >= (F_PLL / 2) /  32) {
    c = SPI_BR_SPPR(7) | SPI_BR_SPR(1);
  } else if (clock >= (F_PLL / 2) /  40) {
    c = SPI_BR_SPPR(4) | SPI_BR_SPR(2);
  } else if (clock >= (F_PLL / 2) /  48) {
    c = SPI_BR_SPPR(5) | SPI_BR_SPR(2);
  } else if (clock >= (F_PLL / 2) /  56) {
    c = SPI_BR_SPPR(6) | SPI_BR_SPR(2);
  } else if (clock >= (F_PLL / 2) /  64) {
    c = SPI_BR_SPPR(7) | SPI_BR_SPR(2);
  } else if (clock >= (F_PLL / 2) /  80) {
    c = SPI_BR_SPPR(4) | SPI_BR_SPR(3);
  } else if (clock >= (F_PLL / 2) /  96) {
    c = SPI_BR_SPPR(5) | SPI_BR_SPR(3);
  } else if (clock >= (F_PLL / 2) / 112) {
    c = SPI_BR_SPPR(6) | SPI_BR_SPR(3);
  } else if (clock >= (F_PLL / 2) / 128) {
    c = SPI_BR_SPPR(7) | SPI_BR_SPR(3);
  } else if (clock >= (F_PLL / 2) / 160) {
    c = SPI_BR_SPPR(4) | SPI_BR_SPR(4);
  } else if (clock >= (F_PLL / 2) / 192) {
    c = SPI_BR_SPPR(5) | SPI_BR_SPR(4);
  } else if (clock >= (F_PLL / 2) / 224) {
    c = SPI_BR_SPPR(6) | SPI_BR_SPR(4);
  } else if (clock >= (F_PLL / 2) / 256) {
    c = SPI_BR_SPPR(7) | SPI_BR_SPR(4);
  } else if (clock >= (F_PLL / 2) / 320) {
    c = SPI_BR_SPPR(4) | SPI_BR_SPR(5);
  } else if (clock >= (F_PLL / 2) / 384) {
    c = SPI_BR_SPPR(5) | SPI_BR_SPR(5);
  } else if (clock >= (F_PLL / 2) / 448) {
    c = SPI_BR_SPPR(6) | SPI_BR_SPR(5);
  } else if (clock >= (F_PLL / 2) / 512) {
    c = SPI_BR_SPPR(7) | SPI_BR_SPR(5);
  } else if (clock >= (F_PLL / 2) / 640) {
    c = SPI_BR_SPPR(4) | SPI_BR_SPR(6);
  } else {    /* (F_PLL/2) / 768 */
    c = SPI_BR_SPPR(5) | SPI_BR_SPR(6);
  }
 } else {
  for (uint32_t i = 0; i < 30; i++) {
    c = br_clock_table[i];
    if (clock >= (F_PLL / 2) / br_div_table[i]) break;
```

```
      }
     }
     br[1] = c;
   }
   static const uint8_t  br_clock_table[30];
   static const uint16_t br_div_table[30];
   uint8_t c1, br[2];
   friend class SPIClass;
};


class SPIClass { // Teensy-LC
  public:
   static const uint8_t CNT_MISO_PINS = 2;
   static const uint8_t CNT_MMOSI_PINS = 2;
   static const uint8_t CNT_SCK_PINS = 2;
   static const uint8_t CNT_CS_PINS = 2;
   typedef struct {
    volatile uint32_t &clock_gate_register;
    uint32_t clock_gate_mask;
    uint8_t  br_index;
    uint8_t  tx_dma_channel;
    uint8_t  rx_dma_channel;
    void     (*dma_isr)();
    uint8_t  miso_pin[CNT_MISO_PINS];
    uint32_t  miso_mux[CNT_MISO_PINS];
    uint8_t  mosi_pin[CNT_MMOSI_PINS];
    uint32_t  mosi_mux[CNT_MMOSI_PINS];
    uint8_t  sck_pin[CNT_SCK_PINS];
    uint32_t  sck_mux[CNT_SCK_PINS];
    uint8_t  cs_pin[CNT_CS_PINS];
    uint32_t  cs_mux[CNT_CS_PINS];
    uint8_t  cs_mask[CNT_CS_PINS];
   } SPI_Hardware_t;
   static const SPI_Hardware_t spi0_hardware;
   static const SPI_Hardware_t spi1_hardware;
   enum DMAState { notAllocated, idle, active, completed};
  public:
   constexpr SPIClass(uintptr_t myport, uintptr_t myhardware)
    : port_addr(myport), hardware_addr(myhardware) {
   }
   // Initialize the SPI library
   void begin();

   // If SPI is to used from within an interrupt, this function registers
   // that interrupt with the SPI library, so beginTransaction() can
   // prevent conflicts.  The input interruptNumber is the number used
   // with attachInterrupt.  If SPI is used from a different interrupt
   // (eg, a timer), interruptNumber should be 255.
   void usingInterrupt(uint8_t n) {
    if (n == 3 || n == 4) {
     usingInterrupt(IRQ_PORTA);
    } else if ((n >= 2 && n <= 15) || (n >= 20 && n <= 23)) {
     usingInterrupt(IRQ_PORTCD);
    }
   }
   void usingInterrupt(IRQ_NUMBER_t interruptName) {
```

```
    uint32_t n = (uint32_t)interruptName;
    if (n < NVIC_NUM_INTERRUPTS) interruptMask |= (1 << n);
  }
  void notUsingInterrupt(IRQ_NUMBER_t interruptName) {
    uint32_t n = (uint32_t)interruptName;
    if (n < NVIC_NUM_INTERRUPTS) interruptMask &= ~(1 << n);
  }

  // Before using SPI.transfer() or asserting chip select pins,
  // this function is used to gain exclusive access to the SPI bus
  // and configure the correct settings.
  void beginTransaction(SPISettings settings) {
    if (interruptMask) {
      __disable_irq();
      interruptSave = NVIC_ICER0 & interruptMask;
      NVIC_ICER0 = interruptSave;
      __enable_irq();
    }
#ifdef SPI_TRANSACTION_MISMATCH_LED
    if (inTransactionFlag) {
      pinMode(SPI_TRANSACTION_MISMATCH_LED, OUTPUT);
      digitalWrite(SPI_TRANSACTION_MISMATCH_LED, HIGH);
    }
    inTransactionFlag = 1;
#endif
    port().C1 = settings.c1;
    port().BR = settings.br[hardware().br_index];
  }

  // Write to the SPI bus (MOSI pin) and also receive (MISO pin)
  uint8_t transfer(uint8_t data) {
    port().DL = data;
    while (!(port().S & SPI_S_SPRF)) ; // wait
    return port().DL;
  }
  uint16_t transfer16(uint16_t data) {
    port().C2 = SPI_C2_SPIMODE;
    port().S;
    port().DL = data;
    port().DH = data >> 8;
    while (!(port().S & SPI_S_SPRF)) ; // wait
    uint16_t r = port().DL | (port().DH << 8);
    port().C2 = 0;
    port().S;
    return r;
  }
  void transfer(void *buf, size_t count) {
    if (count == 0) return;
    uint8_t *p = (uint8_t *)buf;
    while (!(port().S & SPI_S_SPTEF)) ; // wait
    port().DL = *p;
    while (--count > 0) {
      uint8_t out = *(p + 1);
      while (!(port().S & SPI_S_SPTEF)) ; // wait
      __disable_irq();
      port().DL = out;
      while (!(port().S & SPI_S_SPRF)) ; // wait
```

```
            uint8_t in = port().DL;
            __enable_irq();
            *p++ = in;
          }
          while (!(port().S & SPI_S_SPRF)) ; // wait
          *p = port().DL;
        }

        void setTransferWriteFill(uint8_t ch ) {
          _transferWriteFill = ch;
        }
        void transfer(const void * buf, void * retbuf, size_t count);

        // Asynch support (DMA )
      #ifdef SPI_HAS_TRANSFER_ASYNC
        bool transfer(const void *txBuffer, void *rxBuffer, size_t count,  EventResponderRef
event_responder);

        friend void _spi_dma_rxISR0(void);
        friend void _spi_dma_rxISR1(void);
        inline void dma_isr(void);
      #endif

        // After performing a group of transfers and releasing the chip select
        // signal, this function allows others to access the SPI bus
        void endTransaction(void) {
      #ifdef SPI_TRANSACTION_MISMATCH_LED
          if (!inTransactionFlag) {
            pinMode(SPI_TRANSACTION_MISMATCH_LED, OUTPUT);
            digitalWrite(SPI_TRANSACTION_MISMATCH_LED, HIGH);
          }
          inTransactionFlag = 0;
      #endif
          if (interruptMask) {
            NVIC_ISER0 = interruptSave;
          }
        }

        // Disable the SPI bus
        void end();

        // This function is deprecated.       New applications should use
        // beginTransaction() to configure SPI settings.
        void setBitOrder(uint8_t bitOrder) {
          uint8_t c = port().C1 | SPI_C1_SPE;
          if (bitOrder == LSBFIRST) c |= SPI_C1_LSBFE;
          else c &= ~SPI_C1_LSBFE;
          port().C1 = c;
        }

        // This function is deprecated.       New applications should use
        // beginTransaction() to configure SPI settings.
        void setDataMode(uint8_t dataMode) {
          uint8_t c = port().C1 | SPI_C1_SPE;
          if (dataMode & 0x04) c |= SPI_C1_CPHA;
          else c &= ~SPI_C1_CPHA;
          if (dataMode & 0x08) c |= SPI_C1_CPOL;
```

```
  else c &= ~SPI_C1_CPOL;
  port().C1 = c;
}

// This function is deprecated.       New applications should use
// beginTransaction() to configure SPI settings.
void setClockDivider(uint8_t clockDiv) {
  unsigned int i = hardware().br_index;
  if (clockDiv == SPI_CLOCK_DIV2) {
    port().BR = (SPISettings(12000000, MSBFIRST, SPI_MODE0).br[i]);
  } else if (clockDiv == SPI_CLOCK_DIV4) {
    port().BR = (SPISettings(4000000, MSBFIRST, SPI_MODE0).br[i]);
  } else if (clockDiv == SPI_CLOCK_DIV8) {
    port().BR = (SPISettings(2000000, MSBFIRST, SPI_MODE0).br[i]);
  } else if (clockDiv == SPI_CLOCK_DIV16) {
    port().BR = (SPISettings(1000000, MSBFIRST, SPI_MODE0).br[i]);
  } else if (clockDiv == SPI_CLOCK_DIV32) {
    port().BR = (SPISettings(500000, MSBFIRST, SPI_MODE0).br[i]);
  } else if (clockDiv == SPI_CLOCK_DIV64) {
    port().BR = (SPISettings(250000, MSBFIRST, SPI_MODE0).br[i]);
  } else { /* clockDiv == SPI_CLOCK_DIV128 */
    port().BR = (SPISettings(125000, MSBFIRST, SPI_MODE0).br[i]);
  }
}

// These undocumented functions should not be used.  SPI.transfer()
// polls the hardware flag which is automatically cleared as the
// AVR responds to SPI's interrupt
void attachInterrupt() { }
void detachInterrupt() { }

// Teensy LC can use alternate pins for these 3 SPI signals.
void setMOSI(uint8_t pin);
void setMISO(uint8_t pin);
void setSCK(uint8_t pin);
// return true if "pin" has special chip select capability
bool pinIsChipSelect(uint8_t pin);
bool pinIsMOSI(uint8_t pin);
bool pinIsMISO(uint8_t pin);
bool pinIsSCK(uint8_t pin);
// return true if both pin1 and pin2 have independent chip select capability
bool pinIsChipSelect(uint8_t pin1, uint8_t pin2) {
  return false;
}
// configure a pin for chip select and return its SPI_MCR_PCSIS bitmask
// setCS() is a special function, not intended for use from normal Arduino
// programs/sketches.  See the ILI3941_t3 library for an example.
uint8_t setCS(uint8_t pin);

private:
  KINETISL_SPI_t & port() {
    return *(KINETISL_SPI_t *)port_addr;
  }
  const SPI_Hardware_t & hardware() {
    return *(const SPI_Hardware_t *)hardware_addr;
  }
  uintptr_t port_addr;
```

```
    uintptr_t hardware_addr;
    uint32_t interruptMask = 0;
    uint32_t interruptSave = 0;
    uint8_t mosi_pin_index = 0;
    uint8_t miso_pin_index = 0;
    uint8_t sck_pin_index = 0;
#ifdef SPI_TRANSACTION_MISMATCH_LED
    uint8_t inTransactionFlag = 0;
#endif
    uint8_t _transferWriteFill = 0;
#ifdef SPI_HAS_TRANSFER_ASYNC
    // DMA Support
    bool initDMAChannels();

    DMAState _dma_state = DMAState::notAllocated;
    uint32_t _dma_count_remaining = 0;  // How many bytes left to output after current DMA completes
    DMAChannel *_dmaTX = nullptr;
    DMAChannel *_dmaRX = nullptr;
    EventResponder *_dma_event_responder = nullptr;
#endif
};


#endif



extern SPIClass SPI;
#if defined(__MKL26Z64__)
extern SPIClass SPI1;
#endif
#if defined(__MK64FX512__) || defined(__MK66FX1M0__)
extern SPIClass SPI1;
extern SPIClass SPI2;
#endif
#endif
```

1.5.1.24 Codi font twi.c

```
/*
  twi.c - TWI/I2C library for Wiring & Arduino
  Copyright (c) 2006 Nicholas Zambetti.  All right reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA

  Modified 2012 by Todd Krein (todd@krein.org) to implement repeated starts
```

```
*/

#include <math.h>
#include <stdlib.h>
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <compat/twi.h>
#include "Arduino.h" // for digitalWrite

#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif

#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

#include "pins_arduino.h"
#include "twi.h"

static volatile uint8_t twi_state;
static volatile uint8_t twi_slarw;
static volatile uint8_t twi_sendStop;           // should the transaction end with a stop
static volatile uint8_t twi_inRepStart;         // in the middle of a repeated start

static void (*twi_onSlaveTransmit)(void);
static void (*twi_onSlaveReceive)(uint8_t*, int);

static uint8_t twi_masterBuffer[TWI_BUFFER_LENGTH];
static volatile uint8_t twi_masterBufferIndex;
static volatile uint8_t twi_masterBufferLength;

static uint8_t twi_txBuffer[TWI_BUFFER_LENGTH];
static volatile uint8_t twi_txBufferIndex;
static volatile uint8_t twi_txBufferLength;

static uint8_t twi_rxBuffer[TWI_BUFFER_LENGTH];
static volatile uint8_t twi_rxBufferIndex;

static volatile uint8_t twi_error;

/*
  Function twi_init
  Desc     readys twi pins and sets twi bitrate
  Input    none
  Output   none
*/
void twi_init(void)
{
  // initialize state
  twi_state = TWI_READY;
  twi_sendStop = true;       // default value
  twi_inRepStart = false;

  // activate internal pullups for twi.
  digitalWrite(SDA, 1);
```

```
  digitalWrite(SCL, 1);

  // initialize twi prescaler and bit rate
  cbi(TWSR, TWPS0);
  cbi(TWSR, TWPS1);
  TWBR = ((F_CPU / TWI_FREQ) - 16) / 2;

  /* twi bit rate formula from atmega128 manual pg 204
    SCL Frequency = CPU Clock Frequency / (16 + (2 * TWBR))
    note: TWBR should be 10 or higher for master mode
    It is 72 for a 16mhz Wiring board with 100kHz TWI */

  // enable twi module, acks, and twi interrupt
  TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA);
}

/*
   Function twi_slaveInit
   Desc     sets slave address and enables interrupt
   Input    none
   Output   none
*/
void twi_setAddress(uint8_t address)
{
  // set twi slave address (skip over TWGCE bit)
  TWAR = address << 1;
}

/*
   Function twi_readFrom
   Desc     attempts to become twi bus master and read a
            series of bytes from a device on the bus
   Input    address: 7bit i2c device address
            data: pointer to byte array
            length: number of bytes to read into array
            sendStop: Boolean indicating whether to send a stop at the end
   Output   number of bytes read
*/
uint8_t twi_readFrom(uint8_t address, uint8_t* data, uint8_t length, uint8_t sendStop)
{
  uint8_t i;

  // ensure data will fit into buffer
  if (TWI_BUFFER_LENGTH < length) {
    return 0;
  }

  // wait until twi is ready, become master receiver
  while (TWI_READY != twi_state) {
    continue;
  }
  twi_state = TWI_MRX;
  twi_sendStop = sendStop;
  // reset error state (0xFF.. no error occured)
  twi_error = 0xFF;

  // initialize buffer iteration vars
```

```
  twi_masterBufferIndex = 0;
  twi_masterBufferLength = length - 1; // This is not intuitive, read on...
  // On receive, the previously configured ACK/NACK setting is transmitted in
  // response to the received byte before the interrupt is signalled.
  // Therefor we must actually set NACK when the _next_ to last byte is
  // received, causing that NACK to be sent in response to receiving the last
  // expected byte of data.

  // build sla+w, slave device address + w bit
  twi_slarw = TW_READ;
  twi_slarw |= address << 1;

  if (true == twi_inRepStart) {
    // if we're in the repeated start state, then we've already sent the start,
    // (@@@ we hope), and the TWI statemachine is just waiting for the address byte.
    // We need to remove ourselves from the repeated start state before we enable interrupts,
    // since the ISR is ASYNC, and we could get confused if we hit the ISR before cleaning
    // up. Also, don't enable the START interrupt. There may be one pending from the
    // repeated start that we sent outselves, and that would really confuse things.
    twi_inRepStart = false;                 // remember, we're dealing with an ASYNC ISR
    TWDR = twi_slarw;
    TWCR = _BV(TWINT) | _BV(TWEA) | _BV(TWEN) | _BV(TWIE);    // enable INTs, but not START
  }
  else
    // send start condition
    TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT) | _BV(TWSTA);

  // wait for read operation to complete
  while (TWI_MRX == twi_state) {
    continue;
  }

  if (twi_masterBufferIndex < length)
    length = twi_masterBufferIndex;

  // copy twi buffer to data
  for (i = 0; i < length; ++i) {
    data[i] = twi_masterBuffer[i];
  }

  return length;
}

/*
    Function twi_writeTo
    Desc     attempts to become twi bus master and write a
             series of bytes to a device on the bus
    Input    address: 7bit i2c device address
             data: pointer to byte array
             length: number of bytes in array
             wait: boolean indicating to wait for write or not
             sendStop: boolean indicating whether or not to send a stop at the end
    Output   0 .. success
             1 .. length to long for buffer
             2 .. address send, NACK received
             3 .. data send, NACK received
             4 .. other twi error (lost bus arbitration, bus error, ..)
```

```
*/
uint8_t twi_writeTo(uint8_t address, uint8_t* data, uint8_t length, uint8_t wait, uint8_t sendStop)
{
  uint8_t i;

  // ensure data will fit into buffer
  if (TWI_BUFFER_LENGTH < length) {
    return 1;
  }

  // wait until twi is ready, become master transmitter
  while (TWI_READY != twi_state) {
    continue;
  }
  twi_state = TWI_MTX;
  twi_sendStop = sendStop;
  // reset error state (0xFF.. no error occured)
  twi_error = 0xFF;

  // initialize buffer iteration vars
  twi_masterBufferIndex = 0;
  twi_masterBufferLength = length;

  // copy data to twi buffer
  for (i = 0; i < length; ++i) {
    twi_masterBuffer[i] = data[i];
  }

  // build sla+w, slave device address + w bit
  twi_slarw = TW_WRITE;
  twi_slarw |= address << 1;

  // if we're in a repeated start, then we've already sent the START
  // in the ISR. Don't do it again.
  //
  if (true == twi_inRepStart) {
    // if we're in the repeated start state, then we've already sent the start,
    // (@@@ we hope), and the TWI statemachine is just waiting for the address byte.
    // We need to remove ourselves from the repeated start state before we enable interrupts,
    // since the ISR is ASYNC, and we could get confused if we hit the ISR before cleaning
    // up. Also, don't enable the START interrupt. There may be one pending from the
    // repeated start that we sent outselves, and that would really confuse things.
    twi_inRepStart = false;                 // remember, we're dealing with an ASYNC ISR
    TWDR = twi_slarw;
    TWCR = _BV(TWINT) | _BV(TWEA) | _BV(TWEN) | _BV(TWIE);    // enable INTs, but not START
  }
  else
    // send start condition
    TWCR = _BV(TWINT) | _BV(TWEA) | _BV(TWEN) | _BV(TWIE) | _BV(TWSTA);  // enable INTs

  // wait for write operation to complete
  while (wait && (TWI_MTX == twi_state)) {
    continue;
  }

  if (twi_error == 0xFF)
    return 0;  // success
```

```
  else if (twi_error == TW_MT_SLA_NACK)
    return 2; // error: address send, nack received
  else if (twi_error == TW_MT_DATA_NACK)
    return 3; // error: data send, nack received
  else
    return 4; // other twi error
}

/*
   Function twi_transmit
   Desc     fills slave tx buffer with data
            must be called in slave tx event callback
   Input    data: pointer to byte array
            length: number of bytes in array
   Output   1 length too long for buffer
            2 not slave transmitter
            0 ok
*/
uint8_t twi_transmit(const uint8_t* data, uint8_t length)
{
  uint8_t i;

  // ensure data will fit into buffer
  if (TWI_BUFFER_LENGTH < length) {
    return 1;
  }

  // ensure we are currently a slave transmitter
  if (TWI_STX != twi_state) {
    return 2;
  }

  // set length and copy data into tx buffer
  twi_txBufferLength = length;
  for (i = 0; i < length; ++i) {
    twi_txBuffer[i] = data[i];
  }

  return 0;
}

/*
   Function twi_attachSlaveRxEvent
   Desc     sets function called before a slave read operation
   Input    function: callback function to use
   Output   none
*/
void twi_attachSlaveRxEvent( void (*function)(uint8_t*, int) )
{
  twi_onSlaveReceive = function;
}

/*
   Function twi_attachSlaveTxEvent
   Desc     sets function called before a slave write operation
   Input    function: callback function to use
   Output   none
```

```
*/
void twi_attachSlaveTxEvent( void (*function)(void) )
{
  twi_onSlaveTransmit = function;
}

/*
   Function twi_reply
   Desc     sends byte or readys receive line
   Input    ack: byte indicating to ack or to nack
   Output   none
*/
void twi_reply(uint8_t ack)
{
  // transmit master read ready signal, with or without ack
  if (ack) {
    TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWINT) | _BV(TWEA);
  } else {
    TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWINT);
  }
}

/*
   Function twi_stop
   Desc     relinquishes bus master status
   Input    none
   Output   none
*/
void twi_stop(void)
{
  // send stop condition
  TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT) | _BV(TWSTO);

  // wait for stop condition to be exectued on bus
  // TWINT is not set after a stop condition!
  while (TWCR & _BV(TWSTO)) {
    continue;
  }

  // update twi state
  twi_state = TWI_READY;
}

/*
   Function twi_releaseBus
   Desc     releases bus control
   Input    none
   Output   none
*/
void twi_releaseBus(void)
{
  // release bus
  TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT);

  // update twi state
  twi_state = TWI_READY;
}
```

Controlador de reg per degoteig

```
ISR(TWI_vect)
{
  switch (TW_STATUS) {
    // All Master
    case TW_START:     // sent start condition
    case TW_REP_START: // sent repeated start condition
      // copy device address and r/w bit to output register and ack
      TWDR = twi_slarw;
      twi_reply(1);
      break;

    // Master Transmitter
    case TW_MT_SLA_ACK:  // slave receiver acked address
    case TW_MT_DATA_ACK: // slave receiver acked data
      // if there is data to send, send it, otherwise stop
      if (twi_masterBufferIndex < twi_masterBufferLength) {
        // copy data to output register and ack
        TWDR = twi_masterBuffer[twi_masterBufferIndex++];
        twi_reply(1);
      } else {
        if (twi_sendStop)
          twi_stop();
        else {
          twi_inRepStart = true;     // we're gonna send the START
          // don't enable the interrupt. We'll generate the start, but we
          // avoid handling the interrupt until we're in the next transaction,
          // at the point where we would normally issue the start.
          TWCR = _BV(TWINT) | _BV(TWSTA) | _BV(TWEN) ;
          twi_state = TWI_READY;
        }
      }
      break;
    case TW_MT_SLA_NACK:  // address sent, nack received
      twi_error = TW_MT_SLA_NACK;
      twi_stop();
      break;
    case TW_MT_DATA_NACK: // data sent, nack received
      twi_error = TW_MT_DATA_NACK;
      twi_stop();
      break;
    case TW_MT_ARB_LOST: // lost bus arbitration
      twi_error = TW_MT_ARB_LOST;
      twi_releaseBus();
      break;

    // Master Receiver
    case TW_MR_DATA_ACK: // data received, ack sent
      // put byte into buffer
      twi_masterBuffer[twi_masterBufferIndex++] = TWDR;
    case TW_MR_SLA_ACK:  // address sent, ack received
      // ack if more bytes are expected, otherwise nack
      if (twi_masterBufferIndex < twi_masterBufferLength) {
        twi_reply(1);
      } else {
        twi_reply(0);
      }
```

```
  break;
case TW_MR_DATA_NACK: // data received, nack sent
  // put final byte into buffer
  twi_masterBuffer[twi_masterBufferIndex++] = TWDR;
  if (twi_sendStop)
    twi_stop();
  else {
    twi_inRepStart = true;       // we're gonna send the START
    // don't enable the interrupt. We'll generate the start, but we
    // avoid handling the interrupt until we're in the next transaction,
    // at the point where we would normally issue the start.
    TWCR = _BV(TWINT) | _BV(TWSTA) | _BV(TWEN) ;
    twi_state = TWI_READY;
  }
  break;
case TW_MR_SLA_NACK: // address sent, nack received
  twi_stop();
  break;
// TW_MR_ARB_LOST handled by TW_MT_ARB_LOST case

// Slave Receiver
case TW_SR_SLA_ACK:   // addressed, returned ack
case TW_SR_GCALL_ACK: // addressed generally, returned ack
case TW_SR_ARB_LOST_SLA_ACK:   // lost arbitration, returned ack
case TW_SR_ARB_LOST_GCALL_ACK: // lost arbitration, returned ack
  // enter slave receiver mode
  twi_state = TWI_SRX;
  // indicate that rx buffer can be overwritten and ack
  twi_rxBufferIndex = 0;
  twi_reply(1);
  break;
case TW_SR_DATA_ACK:       // data received, returned ack
case TW_SR_GCALL_DATA_ACK: // data received generally, returned ack
  // if there is still room in the rx buffer
  if (twi_rxBufferIndex < TWI_BUFFER_LENGTH) {
    // put byte in buffer and ack
    twi_rxBuffer[twi_rxBufferIndex++] = TWDR;
    twi_reply(1);
  } else {
    // otherwise nack
    twi_reply(0);
  }
  break;
case TW_SR_STOP: // stop or repeated start condition received
  // put a null char after data if there's room
  if (twi_rxBufferIndex < TWI_BUFFER_LENGTH) {
    twi_rxBuffer[twi_rxBufferIndex] = '\0';
  }
  // sends ack and stops interface for clock stretching
  twi_stop();
  // callback to user defined callback
  twi_onSlaveReceive(twi_rxBuffer, twi_rxBufferIndex);
  // since we submit rx buffer to "wire" library, we can reset it
  twi_rxBufferIndex = 0;
  // ack future responses and leave slave receiver state
  twi_releaseBus();
  break;
```

```
    case TW_SR_DATA_NACK:       // data received, returned nack
    case TW_SR_GCALL_DATA_NACK: // data received generally, returned nack
      // nack back at master
      twi_reply(0);
      break;

    // Slave Transmitter
    case TW_ST_SLA_ACK:         // addressed, returned ack
    case TW_ST_ARB_LOST_SLA_ACK: // arbitration lost, returned ack
      // enter slave transmitter mode
      twi_state = TWI_STX;
      // ready the tx buffer index for iteration
      twi_txBufferIndex = 0;
      // set tx buffer length to be zero, to verify if user changes it
      twi_txBufferLength = 0;
      // request for txBuffer to be filled and length to be set
      // note: user must call twi_transmit(bytes, length) to do this
      twi_onSlaveTransmit();
      // if they didn't change buffer & length, initialize it
      if (0 == twi_txBufferLength) {
        twi_txBufferLength = 1;
        twi_txBuffer[0] = 0x00;
      }
    // transmit first byte from buffer, fall
    case TW_ST_DATA_ACK: // byte sent, ack returned
      // copy data to output register
      TWDR = twi_txBuffer[twi_txBufferIndex++];
      // if there is more to send, ack, otherwise nack
      if (twi_txBufferIndex < twi_txBufferLength) {
        twi_reply(1);
      } else {
        twi_reply(0);
      }
      break;
    case TW_ST_DATA_NACK: // received nack, we are done
    case TW_ST_LAST_DATA: // received ack, but we are done already!
      // ack future responses
      twi_reply(1);
      // leave slave receiver state
      twi_state = TWI_READY;
      break;

    // All
    case TW_NO_INFO:   // no state information
      break;
    case TW_BUS_ERROR: // bus error, illegal stop/start
      twi_error = TW_BUS_ERROR;
      twi_stop();
      break;
  }
}
```

1.5.1.25 Codi font twi.h

```
/*
  twi.h - TWI/I2C library for Wiring & Arduino
  Copyright (c) 2006 Nicholas Zambetti.  All right reserved.
```

```
  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/

#ifndef twi_h
#define twi_h

#include <inttypes.h>

//#define ATMEGA8

#ifndef TWI_FREQ
#define TWI_FREQ 100000L
#endif

#ifndef TWI_BUFFER_LENGTH
#define TWI_BUFFER_LENGTH 32
#endif

#define TWI_READY 0
#define TWI_MRX   1
#define TWI_MTX   2
#define TWI_SRX   3
#define TWI_STX   4

void twi_init(void);
void twi_setAddress(uint8_t);
uint8_t twi_readFrom(uint8_t, uint8_t*, uint8_t, uint8_t);
uint8_t twi_writeTo(uint8_t, uint8_t*, uint8_t, uint8_t, uint8_t);
uint8_t twi_transmit(const uint8_t*, uint8_t);
void twi_attachSlaveRxEvent( void (*)(uint8_t*, int) );
void twi_attachSlaveTxEvent( void (*)(void) );
void twi_reply(uint8_t);
void twi_stop(void);
void twi_releaseBus(void);

#endif
```

1.5.1.26 Codi font Wire.cpp

```
/*
  TwoWire.cpp - TWI/I2C library for Wiring & Arduino
  Copyright (c) 2006 Nicholas Zambetti.  All right reserved.

  This library is free software; you can redistribute it and/or
```

```
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA

  Modified 2012 by Todd Krein (todd@krein.org) to implement repeated starts
*/

extern "C" {
#include <stdlib.h>
#include <string.h>
#include <inttypes.h>
#include "twi.h"
}

#include "Wire.h"

// Initialize Class Variables /////////////////////////////////////////////////

uint8_t TwoWire::rxBuffer[BUFFER_LENGTH];
uint8_t TwoWire::rxBufferIndex = 0;
uint8_t TwoWire::rxBufferLength = 0;

uint8_t TwoWire::txAddress = 0;
uint8_t TwoWire::txBuffer[BUFFER_LENGTH];
uint8_t TwoWire::txBufferIndex = 0;
uint8_t TwoWire::txBufferLength = 0;

uint8_t TwoWire::transmitting = 0;
void (*TwoWire::user_onRequest)(void);
void (*TwoWire::user_onReceive)(int);

// Constructors ////////////////////////////////////////////////////////////////

TwoWire::TwoWire()
{
}

// Public Methods //////////////////////////////////////////////////////////////

void TwoWire::begin(void)
{
  rxBufferIndex = 0;
  rxBufferLength = 0;

  txBufferIndex = 0;
  txBufferLength = 0;

  twi_init();
```

```
}

void TwoWire::begin(uint8_t address)
{
  twi_setAddress(address);
  twi_attachSlaveTxEvent(onRequestService);
  twi_attachSlaveRxEvent(onReceiveService);
  begin();
}

void TwoWire::begin(int address)
{
  begin((uint8_t)address);
}

uint8_t TwoWire::requestFrom(uint8_t address, uint8_t quantity, uint8_t sendStop)
{
  // clamp to buffer length
  if (quantity > BUFFER_LENGTH) {
    quantity = BUFFER_LENGTH;
  }
  // perform blocking read into buffer
  uint8_t read = twi_readFrom(address, rxBuffer, quantity, sendStop);
  // set rx buffer iterator vars
  rxBufferIndex = 0;
  rxBufferLength = read;

  return read;
}

uint8_t TwoWire::requestFrom(uint8_t address, uint8_t quantity)
{
  return requestFrom((uint8_t)address, (uint8_t)quantity, (uint8_t)true);
}

uint8_t TwoWire::requestFrom(int address, int quantity)
{
  return requestFrom((uint8_t)address, (uint8_t)quantity, (uint8_t)true);
}

uint8_t TwoWire::requestFrom(int address, int quantity, int sendStop)
{
  return requestFrom((uint8_t)address, (uint8_t)quantity, (uint8_t)sendStop);
}

void TwoWire::beginTransmission(uint8_t address)
{
  // indicate that we are transmitting
  transmitting = 1;
  // set address of targeted slave
  txAddress = address;
  // reset tx buffer iterator vars
  txBufferIndex = 0;
  txBufferLength = 0;
}

void TwoWire::beginTransmission(int address)
```

```
{
  beginTransmission((uint8_t)address);
}

//
//   Originally, 'endTransmission' was an f(void) function.
//   It has been modified to take one parameter indicating
//   whether or not a STOP should be performed on the bus.
//   Calling endTransmission(false) allows a sketch to
//   perform a repeated start.
//
//   WARNING: Nothing in the library keeps track of whether
//   the bus tenure has been properly ended with a STOP. It
//   is very possible to leave the bus in a hung state if
//   no call to endTransmission(true) is made. Some I2C
//   devices will behave oddly if they do not see a STOP.
//
uint8_t TwoWire::endTransmission(uint8_t sendStop)
{
  // transmit buffer (blocking)
  int8_t ret = twi_writeTo(txAddress, txBuffer, txBufferLength, 1, sendStop);
  // reset tx buffer iterator vars
  txBufferIndex = 0;
  txBufferLength = 0;
  // indicate that we are done transmitting
  transmitting = 0;
  return ret;
}

//   This provides backwards compatibility with the original
//   definition, and expected behaviour, of endTransmission
//
uint8_t TwoWire::endTransmission(void)
{
  return endTransmission(true);
}

// must be called in:
// slave tx event callback
// or after beginTransmission(address)
size_t TwoWire::write(uint8_t data)
{
  if (transmitting) {
    // in master transmitter mode
    // don't bother if buffer is full
    if (txBufferLength >= BUFFER_LENGTH) {
      setWriteError();
      return 0;
    }
    // put byte in tx buffer
    txBuffer[txBufferIndex] = data;
    ++txBufferIndex;
    // update amount in buffer
    txBufferLength = txBufferIndex;
  } else {
    // in slave send mode
    // reply to master
```

```
    twi_transmit(&data, 1);
  }
  return 1;
}

// must be called in:
// slave tx event callback
// or after beginTransmission(address)
size_t TwoWire::write(const uint8_t *data, size_t quantity)
{
  if (transmitting) {
    // in master transmitter mode
    for (size_t i = 0; i < quantity; ++i) {
      write(data[i]);
    }
  } else {
    // in slave send mode
    // reply to master
    twi_transmit(data, quantity);
  }
  return quantity;
}

// must be called in:
// slave rx event callback
// or after requestFrom(address, numBytes)
int TwoWire::available(void)
{
  return rxBufferLength - rxBufferIndex;
}

// must be called in:
// slave rx event callback
// or after requestFrom(address, numBytes)
int TwoWire::read(void)
{
  int value = -1;

  // get each successive byte on each call
  if (rxBufferIndex < rxBufferLength) {
    value = rxBuffer[rxBufferIndex];
    ++rxBufferIndex;
  }

  return value;
}

// must be called in:
// slave rx event callback
// or after requestFrom(address, numBytes)
int TwoWire::peek(void)
{
  int value = -1;

  if (rxBufferIndex < rxBufferLength) {
    value = rxBuffer[rxBufferIndex];
  }
```

```
  return value;
}

void TwoWire::flush(void)
{
  // XXX: to be implemented.
}

// behind the scenes function that is called when data is received
void TwoWire::onReceiveService(uint8_t* inBytes, int numBytes)
{
  // don't bother if user hasn't registered a callback
  if (!user_onReceive) {
    return;
  }
  // don't bother if rx buffer is in use by a master requestFrom() op
  // i know this drops data, but it allows for slight stupidity
  // meaning, they may not have read all the master requestFrom() data yet
  if (rxBufferIndex < rxBufferLength) {
    return;
  }
  // copy twi rx buffer into local read buffer
  // this enables new reads to happen in parallel
  for (uint8_t i = 0; i < numBytes; ++i) {
    rxBuffer[i] = inBytes[i];
  }
  // set rx iterator vars
  rxBufferIndex = 0;
  rxBufferLength = numBytes;
  // alert user program
  user_onReceive(numBytes);
}

// behind the scenes function that is called when data is requested
void TwoWire::onRequestService(void)
{
  // don't bother if user hasn't registered a callback
  if (!user_onRequest) {
    return;
  }
  // reset tx buffer iterator vars
  // !!! this will kill any pending pre-master sendTo() activity
  txBufferIndex = 0;
  txBufferLength = 0;
  // alert user program
  user_onRequest();
}

// sets function called on slave write
void TwoWire::onReceive( void (*function)(int) )
{
  user_onReceive = function;
}

// sets function called on slave read
void TwoWire::onRequest( void (*function)(void) )
```

```
{
  user_onRequest = function;
}

// Preinstantiate Objects //////////////////////////////////////////////

TwoWire Wire = TwoWire();
```

1.5.1.27 Codi font wire.h

```
/*
  TwoWire.h - TWI/I2C library for Arduino & Wiring
  Copyright (c) 2006 Nicholas Zambetti.  All right reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA

  Modified 2012 by Todd Krein (todd@krein.org) to implement repeated starts
*/

#ifndef TwoWire_h
#define TwoWire_h

#include <inttypes.h>
#include "Stream.h"

#define BUFFER_LENGTH 32

class TwoWire : public Stream
{
  private:
    static uint8_t rxBuffer[];
    static uint8_t rxBufferIndex;
    static uint8_t rxBufferLength;

    static uint8_t txAddress;
    static uint8_t txBuffer[];
    static uint8_t txBufferIndex;
    static uint8_t txBufferLength;

    static uint8_t transmitting;
    static void (*user_onRequest)(void);
    static void (*user_onReceive)(int);
    static void onRequestService(void);
    static void onReceiveService(uint8_t*, int);
  public:
    TwoWire();
```

```
    void begin();
    void begin(uint8_t);
    void begin(int);
    void beginTransmission(uint8_t);
    void beginTransmission(int);
    uint8_t endTransmission(void);
    uint8_t endTransmission(uint8_t);
    uint8_t requestFrom(uint8_t, uint8_t);
    uint8_t requestFrom(uint8_t, uint8_t, uint8_t);
    uint8_t requestFrom(int, int);
    uint8_t requestFrom(int, int, int);
    virtual size_t write(uint8_t);
    virtual size_t write(const uint8_t *, size_t);
    virtual int available(void);
    virtual int read(void);
    virtual int peek(void);
    virtual void flush(void);
    void onReceive( void (*)(int) );
    void onRequest( void (*)(void) );

    inline size_t write(unsigned long n) {
      return write((uint8_t)n);
    }
    inline size_t write(long n) {
      return write((uint8_t)n);
    }
    inline size_t write(unsigned int n) {
      return write((uint8_t)n);
    }
    inline size_t write(int n) {
      return write((uint8_t)n);
    }
    using Print::write;
};

extern TwoWire Wire;

#endif
```

1.5.2 Codi font servidor Web .

1.5.2.1 Codi font a.php.

```
<?php
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);
// Desde el arduino deurem  "http://gital.eu/r/b.php?d=1500&v=12700&PIN=XXXX&m=1"

$PIN=$_GET["p"];
$Deposit =$_GET["d"];
$tensio = $_GET["v"] ;
$Maquina = $_GET["m"] ;

$ara = getdate(); //Array de dades que en posicio 0 te la hora unix
$timezone = new DateTimeZone('Europe/Madrid');
```

```
$diferencia = $timezone->getOffset(new DateTime()); // Segons de diferencia entre l'hora peninsular i la del servidor.
$hora_local = $ara[0] + $diferencia ;
$Valor_id = trie ("ID");
$Seguretat = $hora_local."000".$Valor_id."000".$Maquina  ;
$sql = "INSERT INTO Taula_Arduino (Bateria, Deposit, Seguretat) VALUES ( $tensio, $Deposit, $Seguretat ) ";

// Comprobe si el pin es correcte.
if ($PIN == "XXXXXXXX")
{
// echo $sql ;
guardar ($sql);
}
else
{
echo "Pin incorrecte";
}

function guardar($sql)
{
if ($conexio = mysqli_connect("andillaaigua.mysql.db","andillaaigua", "XXXXXXXX", "andillaaigua"))
{
if($resultat = mysqli_query ($conexio, $sql))
{  echo "ok"; }
else
{ echo "Error consulta"; }
} else { echo "Error bd"; }
mysqli_close($conexio);
}

function trie($trie)
{
if ($conexio = mysqli_connect("andillaaigua.mysql.db","andillaaigua", "XXXXXXXX", "andillaaigua"))
{
$consulta = "SELECT `".$trie."` FROM `Taula_programacio` ORDER BY `ID` DESC LIMIT 1";
if($resultat = mysqli_query ($conexio, $consulta))
{
 while ($fila = mysqli_fetch_assoc($resultat))
{    return $fila[$trie];}
}
else
{
echo "Error consulta";
}
}
 else
{
echo "Error bd";
}
mysqli_close($conexio);
}
?>
```

1.5.2.2 Codi font a31.php

```
<?
```

```
echo trie("A31");
 function trie($trie)
{
if ($conexio = mysqli_connect("andillaaigua.mysql.db","andillaaigua", "XXXXXXXX", "andillaaigua"))
{
$consulta = "SELECT `".$trie."` FROM `Taula_programacio` ORDER BY `ID` DESC LIMIT 1";
if($resultat = mysqli_query ($conexio, $consulta))
{
 while ($fila = mysqli_fetch_assoc($resultat))
{
 return $fila[$trie];}
}
else
{
echo "Error consulta";
}
}
 else
{
echo "Error bd";
}
mysqli_close($conexio);
}
?>
```

1.5.2.3 Codi font a32.php

```
<?php
echo trie("A32");
function trie($trie)
{
if ($conexio = mysqli_connect("andillaaigua.mysql.db","andillaaigua", "XXXXXXXX", "andillaaigua"))
{
$consulta = "SELECT `".$trie."` FROM `Taula_programacio` ORDER BY `ID` DESC LIMIT 1";
if($resultat = mysqli_query ($conexio, $consulta))
{
 while ($fila = mysqli_fetch_assoc($resultat))
{
 return $fila[$trie];}
}
else
{
echo "Error consulta";
}
}
 else
{
echo "Error bd";
}
mysqli_close($conexio);
}
?>
```

1.5.2.4 Codi font admin.php

```
<?
session_start();
$titol = "Configurar el programador de manera experta";
```

```
include("inici.inc");

if ($_POST['Puntal'] == "dels_llops")
      {
    if ($_SESSION['numeret'] == $_POST['numeret'])
      {
    include("formulari.inc");
    }
       else
       {
    echo "Error has introduit una clau incorrecta" ;
    }
  include("fi.inc");
    exit;
      }
else
{
   include("formulari_anti.inc");
   include("fi.inc");
}
?>
```

1.5.2.5 Codi font ara.php

```
<?php
$ara = getdate();
$timezone = new DateTimeZone('Europe/Madrid');
$diferencia = $timezone->getOffset(new DateTime());
$hora_local = $ara[0] + $diferencia ;
echo $hora_local ;
?>
```

1.5.2.6 Codi font av1.php

```
<?php
echo trie("AV1");
 function trie($trie)
{
if ($conexio = mysqli_connect("andillaaigua.mysql.db","andillaaigua", "XXXXXXXX", "andillaaigua"))
{
$consulta = "SELECT `".$trie."` FROM `Taula_programacio` ORDER BY `ID` DESC LIMIT 1";
if($resultat = mysqli_query ($conexio, $consulta))
{
 while ($fila = mysqli_fetch_assoc($resultat))
{
 return $fila[$trie];}
}
else
{
echo "Error consulta";
}
}
 else
{
echo "Error bd";
}
mysqli_close($conexio);
}
?>
```

1.5.2.7 Codi font av2.php

```
<?php
echo trie("AV2");
 function trie($trie)
{
if ($conexio = mysqli_connect("andillaaigua.mysql.db","andillaaigua", "XXXXXXXX", "andillaaigua"))
{
$consulta = "SELECT `".$trie."` FROM `Taula_programacio` ORDER BY `ID` DESC LIMIT 1";
if($resultat = mysqli_query ($conexio, $consulta))
{
 while ($fila = mysqli_fetch_assoc($resultat))
{
 return $fila[$trie];}
}
else
{
echo "Error consulta";
}
}
 else
{
echo "Error bd";
}
mysqli_close($conexio);
}
?>
```

1.5.2.8 Codi font e31.php

```
<?php
echo trie("E31");
 function trie($trie)
{
if ($conexio = mysqli_connect("andillaaigua.mysql.db","andillaaigua", "XXXXXXXX", "andillaaigua"))
{
$consulta = "SELECT `".$trie."` FROM `Taula_programacio` ORDER BY `ID` DESC LIMIT 1";
if($resultat = mysqli_query ($conexio, $consulta))
{
 while ($fila = mysqli_fetch_assoc($resultat))
{
 return $fila[$trie];}
}
else
{
echo "Error consulta";
}
}
 else
{
echo "Error bd";
}
mysqli_close($conexio);
}
?>
```

1.5.2.9 Codi font e32.php

```php
<?php
echo trie("E32");
 function trie($trie)
{
if ($conexio = mysqli_connect("andillaaigua.mysql.db","andillaaigua", "XXXXXXXX", "andillaaigua"))
{
$consulta = "SELECT `".$trie."` FROM `Taula_programacio` ORDER BY `ID` DESC LIMIT 1";
if($resultat = mysqli_query ($conexio, $consulta))
{
 while ($fila = mysqli_fetch_assoc($resultat))
{
 return $fila[$trie];}
}
else
{
echo "Error consulta";
}
}
 else
{
echo "Error bd";
}
mysqli_close($conexio);
}
?>
```

1.5.2.10 Codi font ev1.php

```php
<?php
echo trie("EV1");
 function trie($trie)
{
if ($conexio = mysqli_connect("andillaaigua.mysql.db","andillaaigua", "XXXXXXXX", "andillaaigua"))
{
$consulta = "SELECT `".$trie."` FROM `Taula_programacio` ORDER BY `ID` DESC LIMIT 1";
if($resultat = mysqli_query ($conexio, $consulta))
{
 while ($fila = mysqli_fetch_assoc($resultat))
{
 return $fila[$trie];}
}
else
{
echo "Error consulta";
}
}
 else
{
echo "Error bd";
}
mysqli_close($conexio);
}
?>
```

1.5.2.11 Codi font ev2.php

```
<?php
echo trie("EV2");
 function trie($trie)
{
if ($conexio = mysqli_connect("andillaaigua.mysql.db","andillaaigua", "XXXXXXXX", "andillaaigua"))
{
$consulta = "SELECT `".$trie."` FROM `Taula_programacio` ORDER BY `ID` DESC LIMIT 1";
if($resultat = mysqli_query ($conexio, $consulta))
{
 while ($fila = mysqli_fetch_assoc($resultat))
{
 return $fila[$trie];}
}
else
{
echo "Error consulta";
}
}
 else
{
echo "Error bd";
}
mysqli_close($conexio);
}
?>
```

1.5.2.12 Codi font fi.inc

```
<hr>
<p class="fi">
<a href="admin.php">  Administrador  </a>
||
<a href="usuari.php">  Usuari  </a>
||
<a href="index.php">  Informació  </a>
</p>
<hr>
<p class="fi">gital.eu 2019 &copy;</p>
</body>
</html>
```

1.5.2.13 Codi font formulari.inc

```
<?
function Genera_form ($ID, $Titol)
{
echo '<p>'.$Titol.'</p>' ;
echo '<p>Hora <input value="00" type="number" name="'.$ID.'_hh" min="0" max="23" required>';
echo ' <input value="00" type="number" name="'.$ID.'_mm" min="0" max="59" required >minuts</p>';
echo '<p>';
echo '    <input type="checkbox" name="'.$ID.'_Dilluns" value="1">Dl';
echo '    <input type="checkbox" name="'.$ID.'_Dimarts" value="1">Dt';
echo '    <input type="checkbox" name="'.$ID.'_Dimecres" value="1">Dc';
echo '    <input type="checkbox" name="'.$ID.'_Dijous" value="1">Dj';
echo '    <input type="checkbox" name="'.$ID.'_Divendres" value="1">Dv';
echo '    <input type="checkbox" name="'.$ID.'_Disabte" value="1">Ds';
echo '    <input type="checkbox" name="'.$ID.'_Diumenge" value="1">Dg';
```

```
echo '</p>';
}
?>
<form action="previ.php" method="post">
   <p>Inserta el pin: <input name="PIN" size="4" min="0" max="9999" required> </p>
   <?
   Genera_form ("Srv","Horari en que se actualitzen les dades del servidor") ;
   Genera_form ("Ev1","Engega electrovàlvula 1") ;
   Genera_form ("Av1","Atura electrovàlvula 1") ;
   Genera_form ("E31","Engega electrovàlvula 3 programa 1") ;
   Genera_form ("A31","Atura electrovàlvula 3 programa 1") ;
   Genera_form ("Ev2","Engega electrovàlvula 2") ;
   Genera_form ("Av2","Atura electrovàlvula 2") ;
   Genera_form ("E32","Engega electrovàlvula 3 programa 2") ;
   Genera_form ("A32","Atura electrovàlvula 3 programa 2") ;
   ?>
   <button type="submit" value="Submit">Guardar dades</button>
   <button type="reset" value="Reset">Esborrar formulari</button>
</form>
```

1.5.2.14 Codi font formulari_anti.inc

```
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
        Escriu les xifres que apareixen a la imatge i fes clic a "continuar" per accedir al
formulari.
         <br>
         <img src="imatge.php" width="80" height="30" Alt="Llegir este numero per seguir
endavant">
        <br>
         <input name="numeret" type="text" size="4">
         <input name="Botonet" type="submit" class="boton" value="Continuar">
         <input name="Puntal" type="hidden" value="dels_llops">
</form>
```

1.5.2.15 Codi font formulari_usuari.inc

```
<html>
   <form action="previ_usuari.php" method="post">
        <p>Pin : <input name="PIN" size="4" min="0" max="9999" required> </p>
        <p>Hora d'inici del reg</p>
        <input value="02" type="number" name="hh_inici" min="0" max="23" required> hores
        <input value="00" type="number" name="mm_inici" min="0" max="59" required> minuts
        <p>Dies de reg</p>
        <input type="checkbox" name="Dilluns" value="1">Dl
        <input type="checkbox" name="Dimarts" value="1">Dt
        <input type="checkbox" name="Dimecres" value="1">Dc
        <input type="checkbox" name="Dijous" value="1">Dj
        <input type="checkbox" name="Divendres" value="1">Dv
        <input type="checkbox" name="Disabte" value="1">Ds
        <input type="checkbox" name="Diumenge" value="1">Dg
        <p>Temps de reg</p>
        <input value="02" type="number" name="hh_reg" min="0" max="3" required>hores
        <input value="00" type="number" name="mm_reg" min="0" max="59" required >minuts
        <p>Temps d'adonament</p>
        <input value="00" type="number" name="mm_adobament" min="0" max="59" required
>minuts
        <button type="submit" value="Submit">Guardar dades</button>
        <button type="reset" value="Reset">Esborrar</button>
   </form>
```

```
</html>
```

1.5.2.16 Codi font imatge.php

```
<?php
session_start();
function randomText($length) {
   $patro = "1234567890";
   for($i=0;$i<$length;$i++) {
     $Clau .= $patro{rand(0,8)};
   }
   return $Clau;
}

$_SESSION['numeret'] = randomText(4);
$Fons = imagecreatefromgif("fons.gif");
$Text = imagecolorallocate($Fons, 0, 0, 0);
imagestring($Fons, 5, 16, 7, $_SESSION['numeret'], $Text);

header("Content-type: image/gif");
imagegif($Fons);
?>
```

1.5.2.17 Codi font index.php

```
<?php
$titol = "Informació dels últims canvis";
include("inici.inc");

$Seguretat = consultar("Seguretat", "Taula_Arduino");

$temps_unix_local = substr($Seguretat, 0, 10);

echo "<p>";
echo "El programador de reg ";
echo substr($Seguretat, 18, 1);
echo ", s'ha connectat el dia: ";
echo gmstrftime('%d/%m/%Y', $temps_unix_local);
echo " a les: ";
$hores = gmstrftime('%H', $temps_unix_local);
$hores = $hores + 0 ;
echo $hores ;
echo gmstrftime(':%M', $temps_unix_local);
echo ".</p>";

$Volts = round(consultar("Bateria", "Taula_Arduino")/1000,1) ;
echo "<p> La bateria té una tensió de " . $Volts . " V. " ;
$Deposit_buit_actual = 0 ;
$Deposit_buit_actual += consultar("Deposit", "Taula_Arduino");
$Deposit_buit = 1250 ;
$Deposit_ple_actual_mm = $Deposit_buit - $Deposit_buit_actual ;
$Deposit_ple_en_percentatge = round(((100 * $Deposit_ple_actual_mm)/
$Deposit_buit),0) ;
echo "  El depòsit esta al " . $Deposit_ple_en_percentatge . "%.</p>" ;

$Ev1 = consultar("Ev1", "Taula_programacio");
$h_inici = substr($Ev1, 4, 2);
$m_inici = substr($Ev1, 6, 2);
$Rega_els = Dies_de_reg (substr($Ev1, 9, 7)) ;
```

Controlador de reg per degoteig

```
     echo "<p> El reg s'iniciara a les " . Round($h_inici,0) . " hores i ".
$m_inici ." minuts. " . $Rega_els . "</p>";
     $minuts_regant = minuts("Ev1", "Av1") ;
     $minuts_deposit = minuts("E31", "A31") ;

     echo "<p> Esta regant " .$minuts_regant. " minuts i adoba ".
$minuts_deposit. " minuts cada sector.";

     echo ultims_dies (); // Informació dels últims 7 dies

     function ultims_dies ()
     {
     if ($conexio = mysqli_connect("andillaaigua.mysql.db","andillaaigua",
"TFGraser90", "andillaaigua")){
     $consulta = "SELECT * FROM `Taula_Arduino` ORDER BY `ID` DESC LIMIT 7";
     if($resultat = mysqli_query ($conexio, $consulta)){
     echo "<table>\n";
       echo '<tr>
                 <td> Últimes mesures </td>
                 <td> Tensió en V  </td>
              <td> Depòsit ple </td>
              </tr>';
     $suma = 0 ;
     while($fila = mysqli_fetch_array($resultat)){
        $suma=$suma+1;
        $pos_array = ($suma-1);
        echo "<TR>";
        echo "<TD>". $suma;
        echo "</TD>";
            echo "<TD>". round(($fila["Bateria"])/1000,1) ;
                      echo "</TD>";
            echo "<TD>".  round(Deposit_per_cent($fila["Deposit"]),0);
              echo "% </TD>";
        echo "</TR>" ;
     }
     }else{
     echo "Error consulta";
     }
     } else{
     echo "Error bd";
     }
      echo "</table>";
      mysqli_close($conexio);
     }

     function consultar($trie, $taula)
     {
     if ($conexio = mysqli_connect("andillaaigua.mysql.db","andillaaigua",
"xxxxxxxxx", "andillaaigua"))
     {
     $consulta = "SELECT `".$trie."` FROM `".$taula."` ORDER BY `ID` DESC LIMIT
1";
     if($resultat = mysqli_query ($conexio, $consulta))
     {
      while ($fila = mysqli_fetch_assoc($resultat))
     {     return $fila[$trie];}
     }
```

```
      else
      {
      echo "Error consulta";
      }
      }
       else
      {
      echo "Error bd";
      }
      mysqli_close($conexio);
      }

      Function Deposit_per_cent($Deposit_buit_actual)
      {
      $Deposit_buit = 1250 ;
      $Deposit_ple_actual_mm = $Deposit_buit - $Deposit_buit_actual ;
      $Deposit_ple_en_percentatge = (100 * $Deposit_ple_actual_mm)/$Deposit_buit
;
      return $Deposit_ple_en_percentatge ;
      }

      Function Dies_de_reg ($string_binari)
      {
      if ((substr($string_binari, 0, 1) == 1)||(substr($string_binari, 1, 1) ==
1)||(substr($string_binari, 2, 1) == 1)||(substr($string_binari, 3, 1) == 1)||
(substr($string_binari, 4, 1) == 1) ||(substr($string_binari, 5, 1) == 1)||
(substr($string_binari, 6, 1) == 1) ) // rega algun dia ?
      {
      $Torne = "Regara " ;
      if ( substr($string_binari, 0, 1) == 1)
      {
      $Torne = "Dilluns, ";
      }
      if ( substr($string_binari, 1, 1) == 1)
      {
      $Torne = $Torne . "Dimarts, " ;
      }
      if ( substr($string_binari, 2, 1) == 1)
      {
      $Torne = $Torne . "Dimecres, " ;
      }
      if ( substr($string_binari, 3, 1) == 1)
      {
      $Torne = $Torne . "Dijous, " ;
      }
      if ( substr($string_binari, 4, 1) == 1)
      {
      $Torne = $Torne . "Divendres, " ;
      }
      if ( substr($string_binari, 5, 1) == 1)
      {
      $Torne = $Torne . "Dissabte" ;
      }
      if ( substr($string_binari, 6, 1) == 1)
      {
      $Torne = $Torne . " i Diumenge" ;
      }
      $Torne = $Torne . ".";
```

```
}
else
{
$Torne = "No rega cap dia" ;
}
Return $Torne ;
}

Function minuts($Engegar, $Aturar)
{
$En = consultar($Engegar, "Taula_programacio");
$h_inici = substr($En, 4, 2);
$m_inici = substr($En, 6, 2);
$At = consultar($Aturar, "Taula_programacio");
$h_fi = substr($At, 4, 2);
$m_fi = substr($At, 6, 2);
$torna = 0 ;
if ($h_inici <= $h_fi )
{
$torna = (($h_fi*60)+$m_fi)-(($h_inici * 60)+$m_inici) ;
}
else
{
$torna = (($h_fi*60)+$m_fi)-(($h_inici * 60)+$m_inici)+(24*60) ;
}
Return $torna ;
}

include("fi.inc");
?>
```

1.5.2.18 Codi font inici.inc

```
<?PHP
/*
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);
*/
?>

<!DOCTYPE html>
<html lang="ca">
<head>
<title>
<? echo $titol ?>
</title>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<style>
        body  {
            background-color: #EFEFEF;
            box-sizing: border-box;
              }

        form{
```

```
            background-color: white;
               border-radius: 3em;
               color: #999;
               font-size: 1.2em;
               padding: 2em;
               margin: 0 auto;
               width: 21em;
            }

            input { background-color: #F0F0F0 ; font-size: 1.1em; }

            a { border-radius: 2em; background-color: #ffffff ; font-size: 1.6em; }

            h1 {
            text-align: center;
            border-radius: 2em;
            background-color: #ffffff;
         border: 0.2em solid black;
            margin-left: auto;
            margin-right: auto;
            border-spacing: 6em 1em;
            display: block;
            }

            img
            {
            display: block;
            margin-left: auto;
            margin-right: auto;
            width: 50%;
            }

            p { font-size: 1.2em; }

            table
            {
            font-size: 1.6em;
            text-align: center;
            border-radius: 1em;
            background-color: #ffffff;
         border: 0.1em solid black;
            margin-left: auto;
            margin-right: auto;
            width: 60%;
            border-spacing: 3em 1em;
            display: block;
            }

            .fi
            {
            text-align: center;
            }

      </style>
      </head>
      <body>
```

```
<h1><? echo $titol ?> </h1>
```

1.5.2.19 Codi font previ.php

```
<?php
// Pendent de modificar el mode de acces a la base de dades.
$titol = "Previsualitza el resultat del formulari";
include("inici.inc");

$PIN=$_POST[PIN];
$SRV = ID ("Srv") ;
$EV1 = ID ("Ev1") ;
$AV1 = ID ("Av1") ;
$EV2 = ID ("Ev2") ;
$AV2 = ID ("Av2") ;
$E31 = ID ("E31") ;
$A31 = ID ("A31") ;
$E32 = ID ("E32") ;
$A32 = ID ("A32") ;

echo $PIN ;
echo "<br>";
echo $SRV;
echo "<br>";
echo $EV1;
echo "<br>";
echo $AV1;
echo "<br>";
echo $E31;
echo "<br>";
echo $A31;
echo "<br>";
echo $EV2;
echo "<br>";
echo $AV2;
echo "<br>";
echo $E32;
echo "<br>";
echo $A32;
echo "<br>";

$Seguretat = $PIN ;

$sql = "INSERT INTO `Taula_programacio` (SRV, EV1, AV1, EV2, AV2, E31, A31, E32, A32,
Seguretat) VALUES ('$SRV','$EV1','$AV1','$EV2','$AV2','$E31','$A31','$E32','$A32', $Seguretat ) ";

// Comprobe si el pin es correcte.
if ($PIN == "XXXXXXXX")
{
guardar ($sql);
}
else
{
echo "El pin introduit es incorrecte";
}

//=========== Afegim dades a la base de dades
```

```
function guardar($sql)
{
if ($conexio = mysqli_connect("andillaaigua.mysql.db","andillaaigua", "XXXXXXXX", "andillaaigua"))
{
if($resultat = mysqli_query ($conexio, $sql))
{  echo "ok"; }
else
{ echo "Error consulta"; }
} else { echo "Error bd"; }
mysqli_close($conexio);
}

// FUNCIO NECESARIA PER GENERAR ELS STRINGS, D'ACTUALITZAR EL SISTEMA, DE ENGEGAR I ATURAR LES DIFERENTS ELECTROVALVULES  :

function ID($id)
{
 $hh=$_POST[$id._hh];            // Agafe valor del formulari
 $hh=sprintf("%02d", $hh);       // Me asegure que asigne 2 digits
 $mm=$_POST[$id._mm];
 $mm=sprintf("%02d", $mm);
 $d0=sprintf("%01d",$_POST[$id._Dilluns]);
 $d1=sprintf("%01d",$_POST[$id._Dimarts]);
 $d2=sprintf("%01d",$_POST[$id._Dimecres]);
 $d3=sprintf("%01d",$_POST[$id._Dijous]);
 $d4=sprintf("%01d",$_POST[$id._Divendres]);
 $d5=sprintf("%01d",$_POST[$id._Disabte]);
 $d6=sprintf("%01d",$_POST[$id._Diumenge]);
 $text1= $id.$text1."_".$hh.$mm."_".$d0.$d1.$d2.$d3.$d4.$d5.$d6."_CT_ATZUCAC";          // Genera el codic tipo IDD_HHMM_DDDDDDD_CT_ATZUCAC per cada id
 return $text1 ;
}

include("fi.inc");
?>
```

1.5.2.20 Codi font previ_usuari.php

```
<?php
// Pendent de modificar el mode de acces a la base de dades.
$titol = "Previsualitza el resultat del formulari";
include("inici.inc");

$Hora_inici = new DateTime('2019-07-15 00:00:00');  // Trie les 00 de un dilluns com el meu 0.

$PIN=$_POST['PIN'];

$Hora_reg_inici =$_POST['hh_inici'];            // Agafe valor del formulari
$Hora_reg_inici =sprintf("%02d", $Hora_reg_inici);       // Me asegure que assigne 2 digits
$Minuts_reg_inici =$_POST['mm_inici'];
$Minuts_reg_inici =sprintf("%02d", $Minuts_reg_inici);

$Hores_duracio_regar =$_POST['hh_reg'];
$Hores_duracio_regar =sprintf("%02d", $Hores_duracio_regar);

$Minuts_duracio_regar =$_POST['mm_reg'];
$Minuts_duracio_regar =sprintf("%02d", $Minuts_duracio_regar);
```

```
        $Minuts_duracio_adobament =$_POST['mm_adobament'];
        $Minuts_duracio_adobament =sprintf("%02d", $Minuts_duracio_adobament);

        $Dilluns=$_POST['Dilluns'];
        $Dimarts=$_POST['Dimarts'];
        $Dimecres=$_POST['Dimecres'];
        $Dijous=$_POST['Dijous'];
        $Divendres=$_POST['Divendres'];
        $Disabte=$_POST['Disabte'];
        $Diumenge=$_POST['Diumenge'];

        $Dies_de_la_setmana = [$Dilluns, $Dimarts, $Dimecres, $Dijous, $Divendres, $Disabte, $Diumenge]
;

        // Calculem el hora a la que deu connectarse al servidor

        $Ev1 = clone($Hora_inici);  // Clone el objecte de referencia
        $Ev1_complet = recalcular($Ev1, $Hora_reg_inici, $Minuts_reg_inici, $Dies_de_la_setmana) ;
        $Ev1_complet = 'Ev1_'.$Ev1_complet ;

        $SRV = clone($Ev1); // Faig un clon del objecte hora inici per poder restarli 10 minuts.
        $SRV = resta($SRV, 0, 10);  // Comproba si hi ha noves dades al servidor 10 minuts abans del hora
del primer reg.
        $SRV_complet = $SRV->format('Hi');
        $SRV_complet = 'Srv_'.$SRV_complet.'_1111111_CT_ATZUCAC' ;

        $Av1 = clone($Ev1);  // Clone el objecte de referencia
        $Av1_complet = recalcular($Av1 , $Hores_duracio_regar, $Minuts_duracio_regar ,
$Dies_de_la_setmana) ;
        $Av1_complet = 'Av1_'.$Av1_complet ;

        $Ev2 = clone($Av1);  // Clone el objecte de referencia
        $Ev2_complet = recalcular($Ev2 , 0, 5, $Dies_de_la_setmana) ;
        $Ev2_complet = 'Ev2_'.$Ev2_complet ;

        $Av2 = clone($Ev2);  // Clone el objecte de referencia
        $Av2_complet = recalcular($Av2 , $Hores_duracio_regar, $Minuts_duracio_regar ,
$Dies_de_la_setmana) ;
        $Av2_complet = 'Av2_'.$Av2_complet ;


        if ($Minuts_duracio_adobament == 0)

        {
        $E31_complet = 'E31_2500_0000000_CT_ATZUCAC' ;
        $A31_complet = 'A31_2500_0000000_CT_ATZUCAC' ;
        $E32_complet = 'E32_2500_0000000_CT_ATZUCAC' ;
        $A32_complet = 'A32_2500_0000000_CT_ATZUCAC' ;
        }

        else
        {
        $E31 = clone($Ev1);  // Clone el objecte de referencia
        $E31_complet = recalcular($E31 , 0, 5, $Dies_de_la_setmana) ;
        $E31_complet = 'E31_'.$E31_complet ;

        $A31 = clone($E31);  // Clone el objecte de referencia
```

```
$A31_complet = recalcular($A31 , 0, $Minuts_duracio_adobament, $Dies_de_la_setmana) ;
$A31_complet = 'A31_'.$A31_complet ;

$E32 = clone($Ev2);  // Clone el objecte de referencia
$E32_complet = recalcular($E32 , 0, 5, $Dies_de_la_setmana) ;
$E32_complet = 'E32_'.$E32_complet ;

$A32 = clone($E32);  // Clone el objecte de referencia
$A32_complet = recalcular($A32 , 0, $Minuts_duracio_adobament, $Dies_de_la_setmana) ;
$A32_complet = 'A32_'.$A32_complet ;
}

// Mostrem els resultats.
 echo $PIN ;
 echo "<br/>";
 echo $SRV_complet ; // Ja he generat el que cal guardar a la base de dades.
 echo "<br/>";
 echo $Ev1_complet ; // Ja he generat el que cal guardar a la base de dades.
 echo "<br/>";
 echo $E31_complet ; // Ja he generat el que cal guardar a la base de dades.
 echo "<br/>";
 echo $A31_complet ; // Ja he generat el que cal guardar a la base de dades.
 echo "<br/>";
 echo $Av1_complet ; // Ja he generat el que cal guardar a la base de dades.
 echo "<br/>";
 echo $Ev2_complet ; // Ja he generat el que cal guardar a la base de dades.
 echo "<br/>";
 echo $E32_complet ; // Ja he generat el que cal guardar a la base de dades.
 echo "<br/>";
 echo $A32_complet ; // Ja he generat el que cal guardar a la base de dades.
 echo "<br/>";
 echo $Av2_complet ; // Ja he generat el que cal guardar a la base de dades.
 echo "<br/>";

$Seguretat = $PIN ;

$bd = "andillaaigua";
$sql = "INSERT INTO Taula_programacio (SRV, EV1, AV1, EV2, AV2, E31, A31, E32, A32, Seguretat)
VALUES
('$SRV_complet','$Ev1_complet','$Av1_complet','$Ev2_complet','$Av2_complet','$E31_complet','$A31_
complet','$E32_complet','$A32_complet', $Seguretat ) ";

// Comprobe si el pin es correcte.
if ($PIN == "XXXXXXXX")
{
guardar ($sql);
}
else
{
echo "El pin introduit es incorrecte";
}
//=========== Afegim dades a la base de dades
function guardar($sql)
{
if ($conexio = mysqli_connect("andillaaigua.mysql.db","andillaaigua", "XXXXXXXX", "andillaaigua"))
{
if($resultat = mysqli_query ($conexio, $sql))
```

```
{  echo "ok"; }
else
{ echo "Error consulta"; }
} else { echo "Error bd"; }
mysqli_close($conexio);
}

// Declare funcions auxiliars.
function Suma($T_inici, $hores, $minuts)
{
$Interval = 'PT'.$hores.'H'.$minuts.'M'  ;
$T_inici->add(new DateInterval($Interval)) ;
return $T_inici ;
}

function resta($T_inici, $hores, $minuts)
{
$Interval = 'PT'.$hores.'H'.$minuts.'M'  ;
$T_inici->sub(new DateInterval($Interval)) ;
return $T_inici ;
}

function fica_0($var)
{
if ($var == true )
{
return '1';
}
else
{
return '0';
}
}

 function recalcular($hora_previa, $hh, $mm, $array_dies)

{
 $hora_sumada = suma($hora_previa, $hh, $mm);

 $complet = $hora_sumada->format('Hi');
 $Dies_tmp = '';

 $Control = $hora_sumada->format('N') ;
 if ( $Control == 2)
 {
 $Dies_tmp = fica_0($array_dies[6]) ; //Assigne lo del diumenge al dilluns
for ($i = 0; $i <= 5; $i++)
{
 $Dies_tmp = $Dies_tmp . fica_0($array_dies[$i]) ;
}
 $complet = $complet.'_'.$Dies_tmp.'_CT_ATZUCAC' ;
 return $complet ;
 }
 else
 {
 $Dies_tmp = "";
 for ($i = 0; $i <= 6; $i++)
```

```
        {
        $Dies_tmp = $Dies_tmp . fica_0($array_dies[$i]) ;
        }
        $complet = $complet.'_'.$Dies_tmp.'_CT_ATZUCAC' ;
        return $complet ;
        }
       }
       include("fi.inc");
?>
```

1.5.2.21 Codi font srv.php

```
       <?php
       echo trie("SRV");
        function trie($trie)
       {
       if ($conexio = mysqli_connect("andillaaigua.mysql.db","andillaaigua", "XXXXXXXX", "andillaaigua"))
       {
       $consulta = "SELECT `".$trie."` FROM `Taula_programacio` ORDER BY `ID` DESC LIMIT 1";
       if($resultat = mysqli_query ($conexio, $consulta))
       {
        while ($fila = mysqli_fetch_assoc($resultat))
       {
        return $fila[$trie];}
       }
       else
       {
       echo "Error consulta";
       }
       }
        else
       {
       echo "Error bd";
       }
       mysqli_close($conexio);
       }
    ?>
```

1.5.2.22 Codi font usuari.php

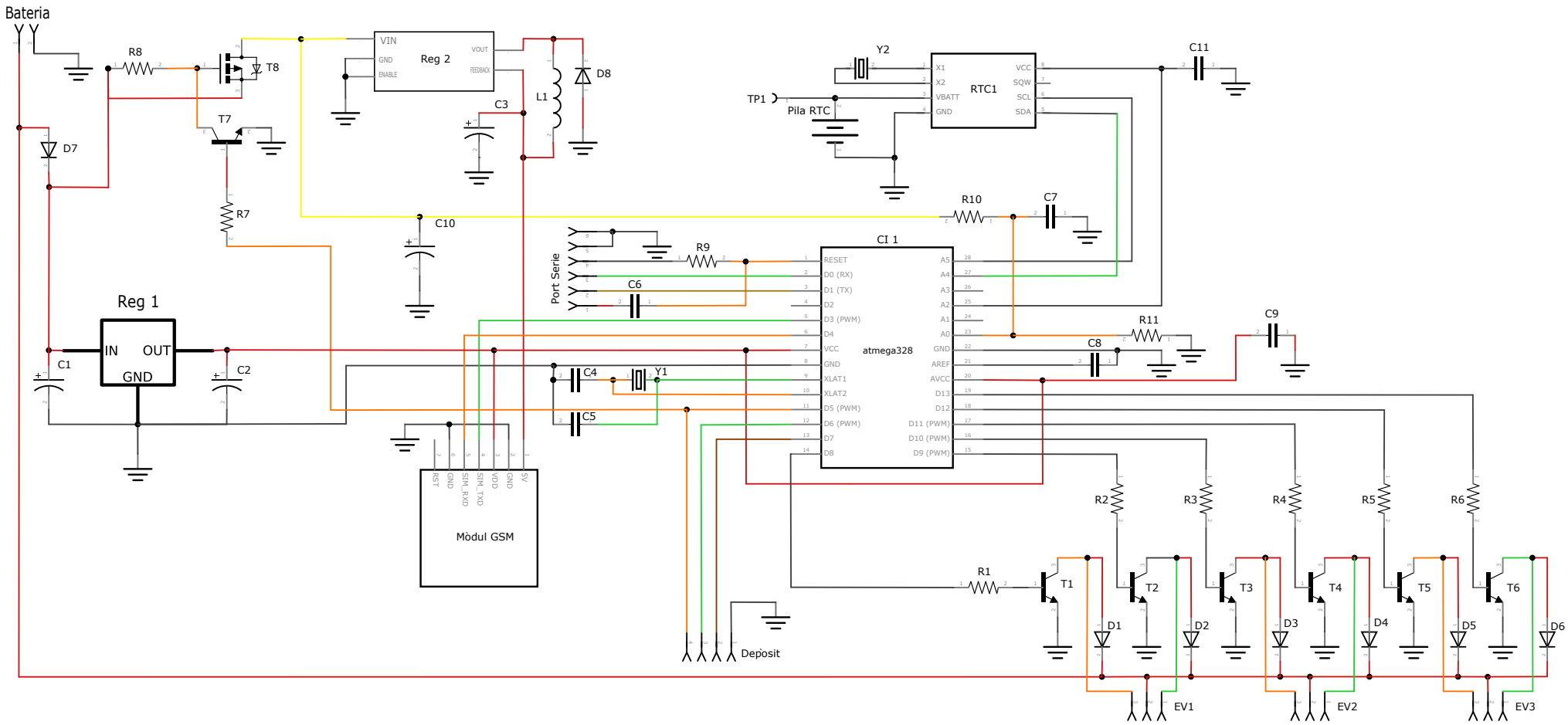
```
       <?php
       session_start();
       $titol = "Configurar el programador de manera fàcil";
       include("inici.inc");

       if ($_POST['Puntal'] == "dels_llops")
           {
        if ($_SESSION['numeret'] == $_POST['numeret'])
           {
        include("formulari_usuari.inc");
        include("fi.inc");
        } else {
        echo "Error has introduït una clau incorrecta" ;
        include("fi.inc");
        }
        exit;
       }
       else
       {
```

```
    include("formulari_anti.inc");
    include("fi.inc");
}
?>
```

1.6 Altres fonts d'informació.

- http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.
- https://www.luisllamas.es/arduino-bubble-sort/
- https://www.elecrow.com/wiki/images/2/20/SIM800_Series_AT_Command_Manual_V1.09.pdf
- https://img.filipeflop.com/files/download/Datasheet_SIM800L.pdf

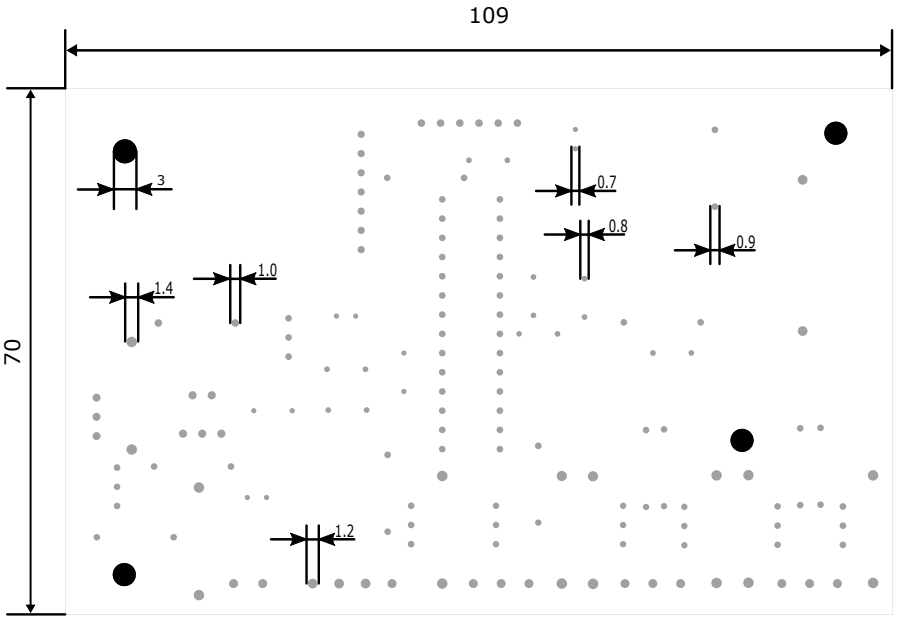


	Data	Nom
Dibuixat	10-9-19	Francesc López i Navarro
Revisat		
Escala		

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Escola Tècnica Superior d'Enginyeria del Disseny

Esquema

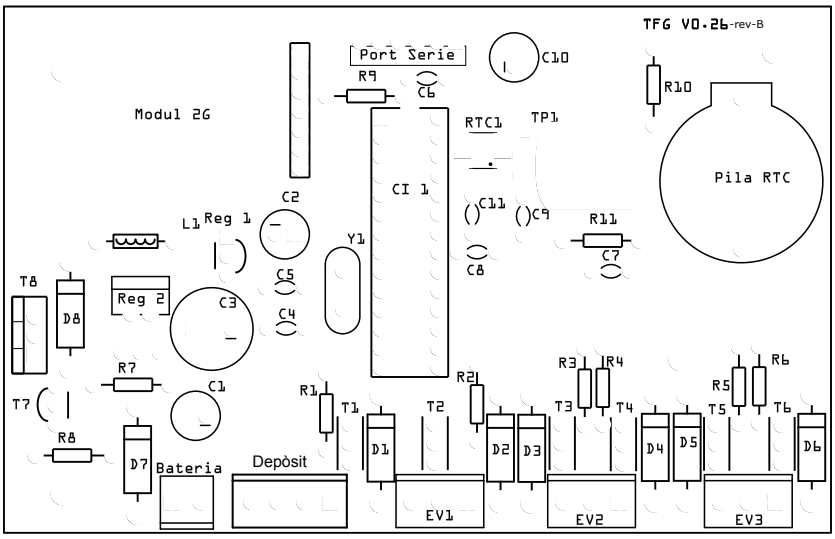
Plànol 2.1



	Data	Nom
Dibuixat	10-9-19	Francesc López i Navarro
Revisat		

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Escola Tècnica Superior d'Enginyeria del Disseny

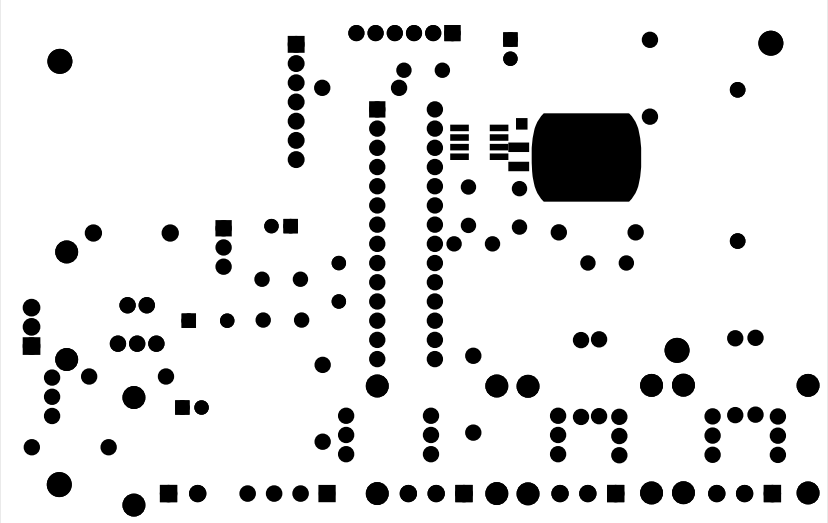
Escala		Plànol
1:1	PCB, Mecanitzat	2.2



	Data	Nom
Dibuixat	10-9-19	Francesc López i Navarro
Revisat		

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Escola Tècnica Superior d'Enginyeria del Disseny

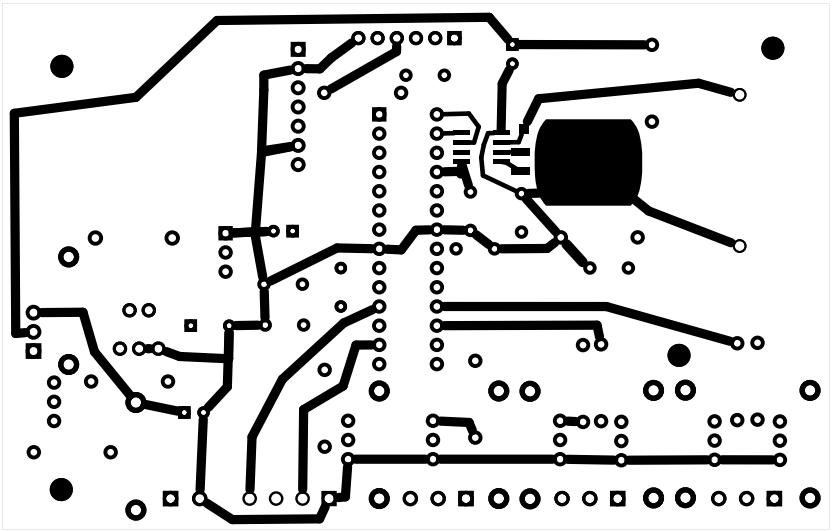
Escala		Plànol
1:1	PCB, Components	2.3



	Data	Nom
Dibuixat	10-9-19	Francesc López i Navarro
Revisat		

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Escola Tècnica Superior d'Enginyeria del Disseny

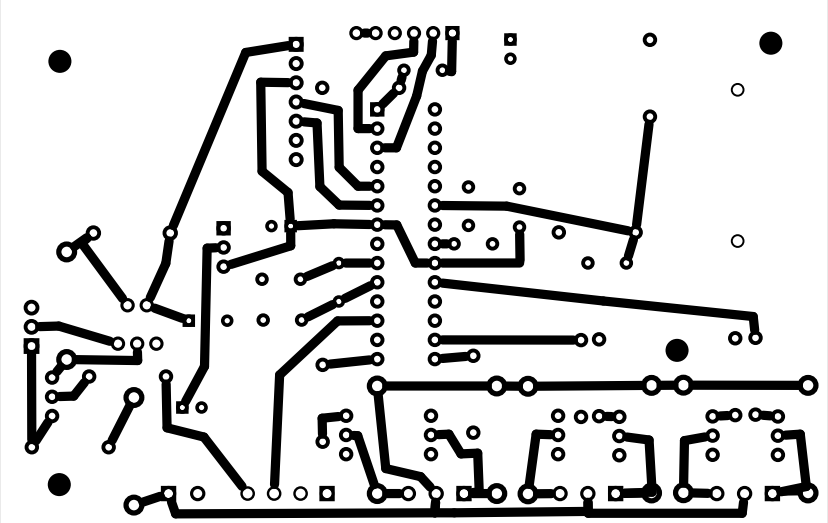
Escala		Plànol
1:1	PCB, Mascara de gravat superior	2.4



	Data	Nom
Dibuixat	10-9-19	Francesc López i Navarro
Revisat		

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Escola Tècnica Superior d'Enginyeria del Disseny

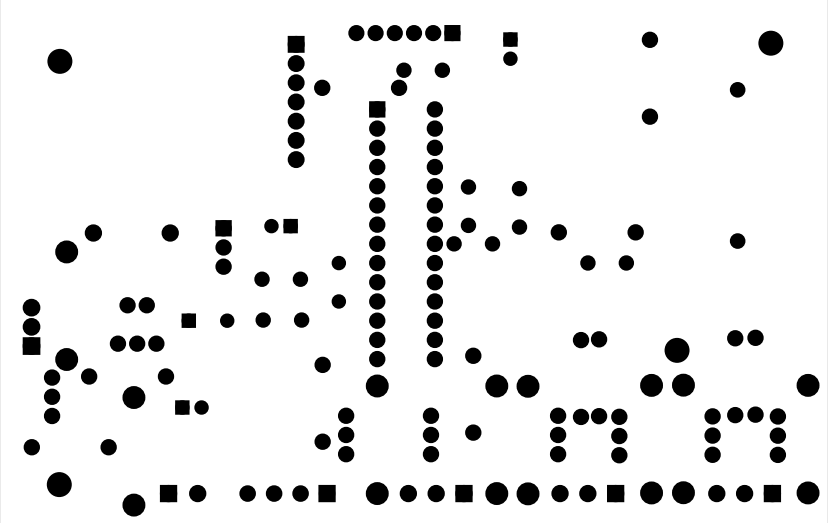
Escala		Plànol
1:1	PCB, Pistes superiors	2.5



	Data	Nom
Dibuixat	10-9-19	Francesc López i Navarro
Revisat		

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Escola Tècnica Superior d'Enginyeria del Disseny

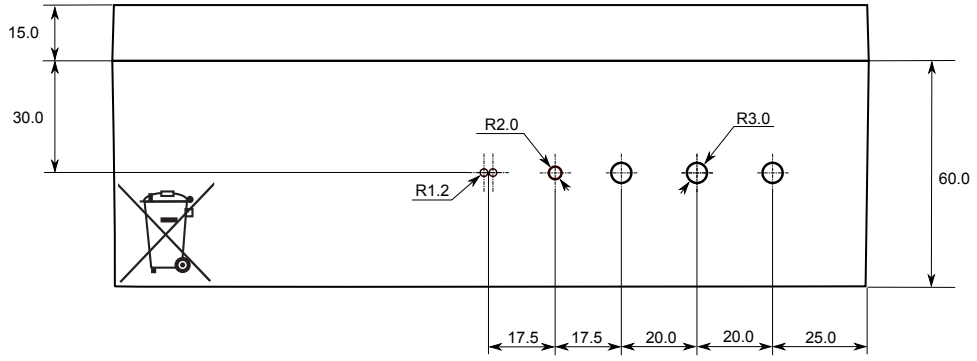
Escala		Plànol
1:1	PCB, Pistes inferiors	2.6



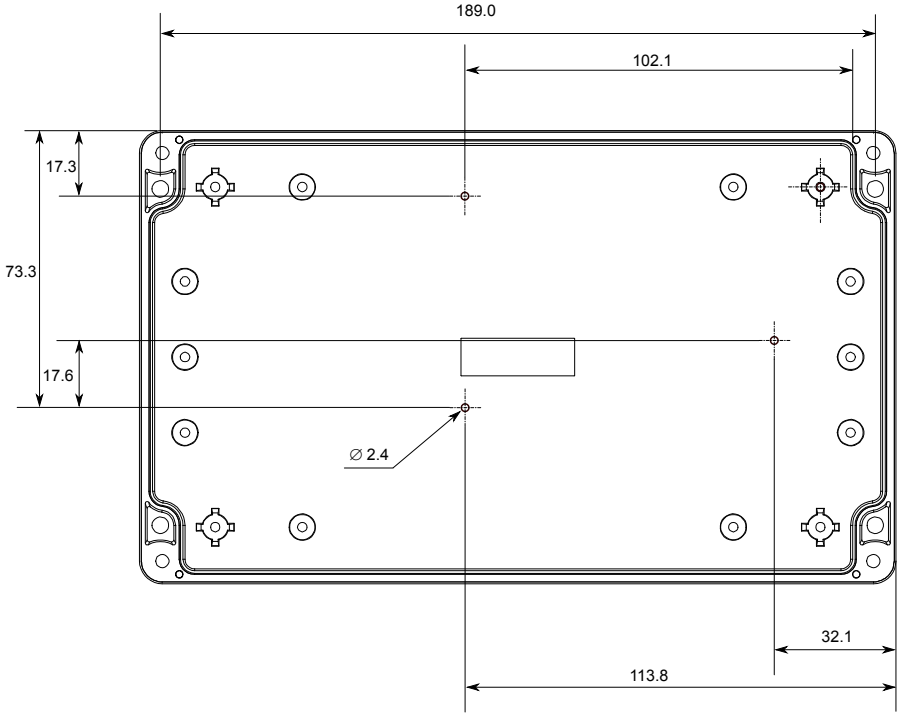
	Data	Nom
Dibuixat	10-9-19	Francesc López i Navarro
Revisat		

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Escola Tècnica Superior d'Enginyeria del Disseny

Escala		Plànol
1:1	PCB, Mascara de gravat inferior	2.7



Vista inferior



Vista interior de la caixa

	Data	Nom
Dibuixat	10-9-19	Francesc López i Navarro
Revisat		

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Escola Tècnica Superior d'Enginyeria del Disseny

Escala		Plànol
1:2	Mecanitzat caixa	2.8

3 Plec de condicions.

3.1 Abast d' aquest plec de condicions

L' objectiu d' este document és assenyalar les condicions tècniques per fixar la implementació del controlador de reg, especificant els requeriments d' utilitat, funcionament i seguretat.

L'objectiu d'este projecte és dissenyar un programador de reg controlat a distància.

L'àmbit d'aplicació d' este document s'estén als àmbits electrònics i mecànics del controlador de reg per degoteig.

El microcontrolador utilitzat es un Atrega 328P, amb una primera capa de *firmware* d'Arduino i se programa amb la aplicació pròpia d' Arduino.

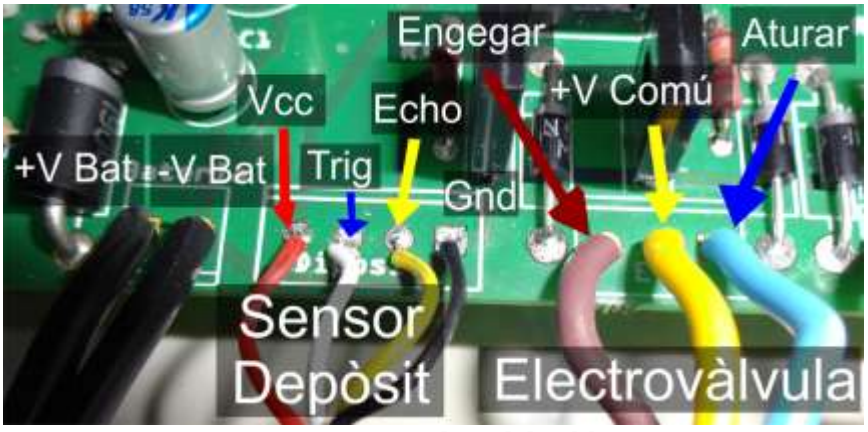
3.2 Condicions tècniques durant el muntatge.

La font d'alimentació, de laboratori utilitzada durant les proves, haurà de tindre un limitador de corrent i amperímetre; per poder limitar la corrent i detectar possibles anomalies en el seu funcionament.

3.3 Muntatge:

3.3.1 Premuntatge:

- Realitzar les modificacions al servidor, per incloure el nou equip.
- Carregar el bootloader d' Arduino al Atmega328p
- Soldar tots els component a la placa, fent servir estany que complisca la normativa Rosh.
- A la eixides de les electrovàlvules soldem 1 metre de cable a cada una, deixant-lo a puntes lliures, però sense pelar.



Imatge 3.3.1 Colors del cables.

- Fem una revisió ocular de tot el circuit amb deteniment.

- Si tot és correcte, s'alimentarà a 12V i carreguarà el programa personalitzat amb el identificador de màquina, URL assignada per cada equip.
- Si tot funciona bé, s'apaga i ja es pot instal·lar el mòdul GSM amb una targeta SIM.
- Es torna a engegar i es comprova que se connecta correctament a la URL corresponent.
- Fer els mecanitzats de la caixa, per fixar la PCB i deixar eixir els cables d'alimentació, electrovàlvules i sensor d' ultrasons.
- Es passen els cables pels seus orificis corresponents, els col·loquem la numeració adequada 1, 2, 3, per les electrovàlvules, el número 0 per la entrada d'alimentació i per últim fixem amb els separadors la PCB a la caixa amb caragols de m3.
- Realitzar ja una comprovació del funcionament del sistema amb unes electrovàlvules al taller. (El sensor d' ultrasons, es col·locarà i provarà al propi camp amb un cablejat fet a la mida de l'instal·lació.)
- Guardar l'equip, la tapa i els caragols adequadament identificats. Guardar una còpia del programari personalitzat al servidor i ja el tenim apunt per instal·lar sobre el terreny.

3.3.2 Muntatge sobre el terreny.

- Abans d' anar a instal·lar comprovarem que la companyia de telefonia té cobertura, teòricament en el punt on anem a instal·lar el equip.
- Prepararem totes les ferramentes, incloent un soldador autònom i resta de material necessari per poder treballar, assegurant-nos que totes complisquen les normatives de seguretat vigents.
- Abans de començar cal ficar-se l'equip de protecció corresponent, i assegurar-se que se donen les condiciones de seguretat i disposem de totes les ferramentes per poder fer la instal·lació amb total seguretat.
- Observar detingudament i buscar quin és el millor punt per ubicar el programador de reg, així com per fixar el cablejat. Evitar, en la mesura del possible aquells punts on és més fàcil que es puguen produir fuites d'aigua, o aquells que puguen dificultar el desmuntatge de filtres o altres parts del sistema que requereixen un cert manteniment.
- Una volta triada la ubicació, mesurar i connectar el mòdul d' ultrasons amb el controlador de reg per degoteig.
- Tallem un llistó de fusta, que ajuste adequadament a la part superior del depòsit, per poder fixar el sensor d'ultrasons en la part central del mateix. Depenent de la instal·lació en particular és possible que tinguem que fer algun mecanitzat més complexe.
- Fixar el sensor amb brides al llistó i col·locar el cablejat del sensor d'ultrasons. Màxima precaució si el depòsit conté líquid, doncs podria ser corrosiu.
- Interconnectar cada electrovàlvula amb els seus cables corresponents, procedents del programador.
- Col·locar la goma de protecció i la tapa a la caixa.

- Soldar el cablejat de la bateria, alimentar, amb un programa de reg curt precarregat al servidor i comprovar que funciona adequadament.

3.4 Precaucions durant el muntatge.

La persona encarregada del muntatge deu ser personal qualificat. Haurà d'aplicar les lleis de prevenció de riscos laborals i seguretat e higiene en el treball.

3.4.1 Per evitar colps i talls en les mans durant la manipulació de les ferramentes :

- No fer servir, ferramentes deteriorades.
- Recollir i guardar les ferramentes en llocs destinats a tal finalitat, quan no se utilitzen.
- Mantindre les zones de pas i de treball lliures d' obstacles.
- Mantindre les ferramentes de manuals , en perfecte estat de conservació i netes de grassa.

3.4.2 Precaucions en la utilització del soldador.

- Comprovar l'estat del soldador abans de la seua utilització. Mai utilitzar un soldador que no es trobe en perfecte estat.

3.4.3 Precaució per risc químic:

- Durant la instal·lació , fer servir equip de protecció individual front a risc químic, per la probable presència de líquids corrosius al depòsit.

3.4.4 Compliment de la normativa RoHs.

- Fer servir en tot moment materials que complisquen amb la normativa RoHs.

3.4.5 Sistema de gestió de residus a aplicar.

- Tot el material sobrant i no reutilitzable després de fer la instal·lació, que no siga re-aprofitable, es gestionarà el seu adequat tractament en una empresa o deixalleria autoritzada, seguint les normes locals aplicables.

4 Pressupost.

Totes les quantitats, estan en euros i sense impostos al menys que s'indique el contrari. Els càlculs s'han fet tenint en compte una producció estimada de 100 equips.

Els preus dels components han sigut obtinguts de la pàgina web del proveïdors. El preu de la mà d'obra i la duració són fruit d'una estimació..

4.1 Cost d'enginyeria.

Els costos d'enginyeria es valora en uns 3000 Euros. Suposant una producció de 100 equips ,el cost a repercutir unitari seria de 30 euros per equip.

4.2 Costos d'amortització d'equips.

Equip	Preu unitari	Quantitat	Preu €
Arduino Uno Genuino	7,93	1/100	0,08
Adaptador USB a Serie	4,25	1/100	0,05
Kit Bootloader	3	1/100	0,03
Cost per equip.			0,16

Taula 4.2. Costos d'amortització d'equips

La resta d'equips electrònics, informàtics, utilitzats per aquest projecte tenen més de 10 anys per la qual cosa es poden considerar el seu cost d'amortització pràcticament 0.

Pel que fa al programari actual utilitzat, he fet servir programari lliure, Fritzing, Arduino, Inkscape i Notepad++, per la qual cosa el seu cost d'amortització també és 0.

4.3 Costos de serveis externs anuals.

Mòdul	Valor	Preu unitari	Quantitat	Preu €
Cost servidor	Domini+Servidor php+BD	60€ / any	0,01	0,6
Sim	Tràfic de dades	0,01 € / setmana	52	0,52
Total cost anual				1,12

Taula 4.4.1. Components de la placa

4.4 Components de cada equip.

4.4.1 Components de la placa.

Id Component	Descripció	Referència distribuïdor	Preu 1	Unitats	Preu €
PCB1	Circuit Classe 3	Doble Cara 109x70mm	11.65	1	11,65
CI1	Atmega328P-PU	556-ATMEGA328-PU	1.70	1	1,70
Zocal CI1	Zocal DIL	RS:674-2498	0.512	1	0,52
RTC1	RTC DS1338Z-33+	RS:732-7368	3.15	1	3,15
R_1,R_2,R_3,R_4,R_5,R_6	130 Ohms 1/4W	RS:683-3001	0,044	6	0,27
R7,R9	10 kOhms 1/4w	Mouser 756-MFR3-10KFC	0.087	2	0,18
R8,R11	100 kOhms 1/4w	756-MFR3-100KFC	0.087	2	0,18
R10	1Mohms 1/4w	RS:707-7903	0.133	1	0,14
C1,C2	22uF / 25V	RS:173-9454	0.333	2	0,67
C3	1000uF / 6.3v	80-A750KK108M0JAAE15	0.48	1	0,48
C4,C5	15pF / 50V	594-K150J15C0GF5TH5	0.157	2	0,32
C6,C7,C8,C9,C11	100nF/50v	594-K104K15X7RF53H5	0.087	5	0,44
C10	100uF / 16v	140-RGA101M1CBK0611G	0.14	1	0,14
Y1	Cristall 8Mhz	RS:547-6216	0,294	1	0,30
Y2	Cristall 32.768kHz	RS:672-7584	1.132	1	1,14
D1,D2,D3,D4,D5,D6	1N5819G	RS :774-3347	0,25	6	1,50
D7,D8	1N5822G	RS:774-3353	0,284	2	0,57
T1,T2,T3,T4,T5,T6	BD135G	Mouser 863-BD135G	0.498	6	2,99
T7	BC547BTA	Mouser 512-BC547BTA	0.183	1	0,19
T8	IRFU9024NPBF	942-IRFU9024NPBF	0.72	1	0,72
Pila zocal	Zocal 20mm Rohs	-	1.26	1	1,26
Pila 2032	3v 190mah	RS 597-419	3.06	1	3,06
L1	47uH / 3,4A	RS 715-7213	1,34	1	1,34
Reg 1	MCP1702-33+	RS:403-850	0,37	1	0,37
Reg 2	LM2596-5.0/NOPB	RS:533-3743	4.12	1	4,12
Total Components					37,40

Taula 4.4.1. Components de la placa

4.4.2 Mòduls prefabricats.

Mòdul	Valor	Referència distribuïdor	Preu unitari	Quantitat	Preu €
Mòdul ultrasons	Adafruit 4007	Mouser 485-4007	3,45	1	3,45
Mòdul SIM	SIM800l	Sim800L V2 GPRS	10,10	1	10,10
Subtotal mòduls					13,55

Taula 4.4.2. Mòduls prefabricats

4.4.3 Altres components.

Component	Descripció	Referència distribuïdor	Preu 1	Preu €
Caixa	Caixa IP65	Mouser 546-RZ0373	14,88	14,88
SIM	Targeta SIM prepagament	SIM ThingMobile	2,50	2,50
Resta material	Cablejat, estany, bries...	Referència distribuïdor	8,00	8,00
Total Altres				25,34

Taula 4.4.3. Altres components

4.5 Mà d'obra.

Descripció	Hores	Preu unitari	Preu €
Muntatge i programació	5	20	100,00
Instal·lació i posada en marxa	2	20	40,00
Modificacions servidor	1	20	20,00
Total.	-	-	180,00

Taula 4.5. Mà d'obra

4.6 Pressupost total.

Concepte	Preu €
Mòduls prefabricats	13,55
Components de placa	37,40
Altres components	25,34
Mà d'obra	180,00
Costos d'enginyeria	30,00
Amortització d'equips	0,16
Serveis externs 2 Anys	2,24
Despeses generals, energètiques i RAEE	30,00
Benefici industrial 10%	31,87
IVA 21%	73,62
Total per unitat amb iva	349,32

Taula 4.6 Pressupost total