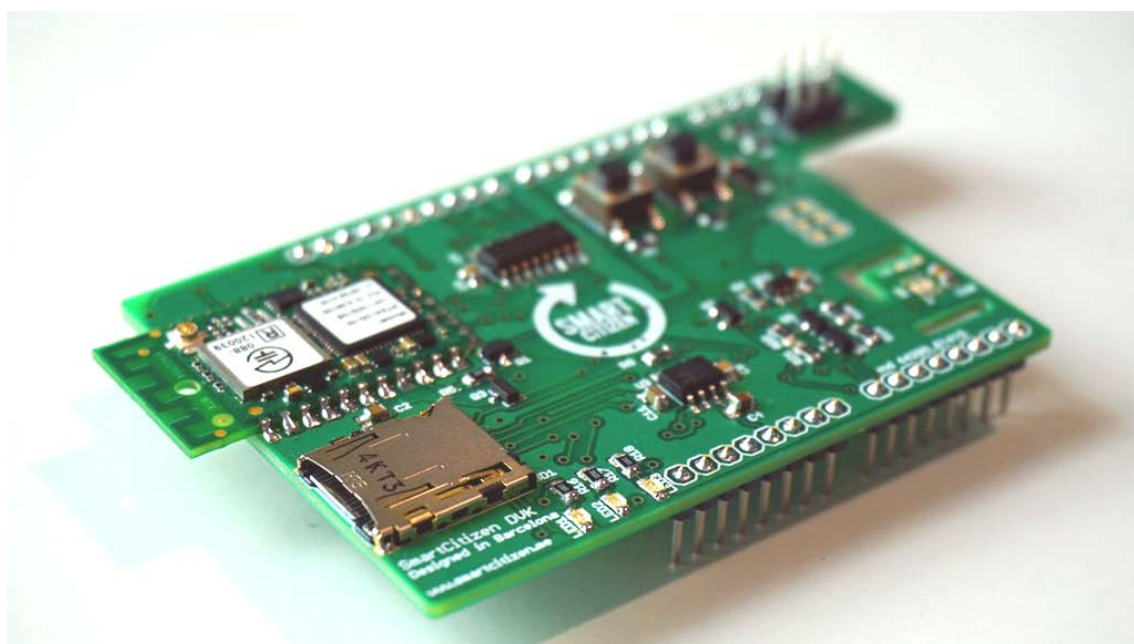


UNIVERSITAT OBERTA DE CATALUNYA



TFG

**DISSENY I IMPLEMENTACIÓ D'UN SERVIDOR WEB PER EL MÒDUL
WiFi RTX4100**



GRAU DE TECNOLOGIA DE TELECOMUNICACIÓ

Autor: Sara Álvarez Garcia
Consultor: Pere Tuset Peiró
Barcelona Juny 2015

Dedicatòria i agraïments

Dedico aquest treball a totes les persones que han estat al meu costat durant tots aquests anys d'estudis: la meva família, els meus companys de feina, companys de la UOC i en especial a la meva parella per suportar-me tot aquest temps.

Donar les gràcies a Pere Tuset, consultor en aquest treball de fi de grau, per la seva implicació en el treball i la seva paciència. A l'empresa SmartCitizen per facilitar-me el material i en especial a Miguel de l'Hangar per donar-me suport amb el protocol SPI i solventar-me el problema que vaig tenir amb la placa.

Summary

The next work presents a practical and economic alternative to free software to create a Web Server that allows us to configure and monitor the parameters of interest in our Wi-Fi connection. In particular, we will use the kit SmartCitizenRTX4100 given by the company called Smart Citizen.

The board Smart Citizen is based on Arduino and has implemented the RTX4100 WiFi module which is ideal for developed solutions with low power consumption. Thanks to him, the application can run without the need for an external host processor because it takes its own firmware that allows developing faster low-power applications.

As a reference, we can find the kit RN131G (Wifly), this will serve as a guide for setting new parameters, such as we have to be able to control various parameters by which you can allow different devices to connect through a browser in AP mode.

To be able to place this design and implementation, RTX provides a complete development environment called Amelie SDK that allows us to test our design because it contains several libraries and examples.

In definitive, we seek to create a Web Server and an AP for the SmartCitizenRTX4100 that allows any mobile device with a WiFi connection can connect just press a button.

Key Words

WiFi, Arduino DUE, Smart Citizen, RTX4100, WebServer, COLApp, AP.

Resum

En el següent treball es presenta una alternativa econòmica i pràctica en software lliure per la implementació d'un servidor web amb el que podrem configurar i monitoritzar els paràmetres que ens interressi en la nostra connexió a la WiFi, en concret farem servir el kit facilitat per l'empresa Smart Citizen que s'anomena *SmartCitizenRTX4100*.

La placa Smart Citizen està basat en Arduino i té implementat el mòdul WiFi RTX4100 el qual és ideal per desenvolupar solucions de baix consum d'energia. Gràcies a ell, les aplicacions es poden executar sense la necessitat d'un processador host extern ja que porta un firmware propi que permet desenvolupar més ràpidament les aplicacions de baixa potència.

Com a referència podem trobar el kit amb el RN131G (WiFly), aquest ens servirà de guia per la configuració dels nous paràmetres, com per exemple haurem de poder controlar diferents paràmetres amb el que podrem permetre a diferents dispositius connectar-se per mitjà d'un navegador en mode AP.

Per poder portar a lloc aquest disseny e implementació, RTX proporciona un entorn de desenvolupament complet anomenat Amelie SDK amb el que podrem fer proves del nostre disseny ja que gràcies a ell, tenim diverses biblioteques i exemples que podem fer servir.

En definitiva, el que busquem és crear un servidor web i un mode AP en la placa SmartCitizenRTX4100 amb el que qualsevol dispositiu mòbil que tingui connexió a WiFi es pugui connectar només premem un botó.

Paraules claus

WiFi, Arduino DUE, Smart Citizen, RTX4100, WebServer, COLApp, AP.

Taula de contingut

Capítol 1 Introducció	1
1.1 Justificació del tema.....	1
1.2 Motivació.....	1
1.3 Objectius generals i específics	2
1.4 Resultats potencials	2
1.5 Organització de la memòria.....	3
Capítol 2 Placa Arduino i RTX4100	4
2.1 Introducció	4
2.2 Placa Arduino Due	4
2.2.1 Programació en Arduino.....	7
2.2.1.1 Estructura del programa.....	9
2.2.2 Lliberies.....	11
2.3 Smart Citizen Kit.....	12
2.3.1 SCK 1.1.....	12
2.3.2 SCK 1.5.....	14
2.3.2.1 Mòdul RTX4100.....	18
2.4. RTX41xx DVK	20
2.4.1 Amelie SDK.....	22
2.5 Entorn de programació	24
2.5.1 Lliberies.....	24
2.5.2 Entorn Arduino IDE.....	26
2.5.3 Entorn Amelie SDK.....	28
2.5.4 Entorn RTX	32
2.6 Access Point (AP)	34
2.7 Servidor Web.....	36
2.7.1. Protocol HTTP	37
2.8 Protocol SLIP.....	39
2.9 WiFi.....	41
2.9.1 Legislació vigent	41
2.9.2 Característiques WiFi	42
2.9.2.1 Categories de xarxa sense fils	42
2.9.2.2 Estàndards WiFi.....	43
2.9.3 Tipus de seguretat	44
Capítol 3 Desenvolupament de la pràctica	45
3.1. Introducció	45
3.2. Creació servidor web amb Arduino	45
3.3. AP Mode	49
3.3.1 SoftAP.....	49
3.3.2 SPI	51
3.3.3 Lliberies	52
3.4. Servidor Web SmartCitizenRTX4100.....	58
3.4.1 SPI.....	59

3.4.2 TempSensor	62
3.4.3 HTML.....	63
3.4.4 Llibreries.....	64
3.5. Integració del codi	70
Capítol 4 Conclusions i línies de futur	71
4.1. Conclusions	71
4.2. Línies de futur.....	72
Capítol 5 Acrònims.....	74
Capítol 6 Bibliografia	76
Capítol 7 Annexos	78

Llista de figures

Il·lustració 1 – Placa Arduino DUE	4
Il·lustració 2 – Mapejat pin Arduino DUE	6
Il·lustració 3 – Editor de text	7
Il·lustració 4 – Led PRW engegat	8
Il·lustració 5 – Board Manager	8
Il·lustració 6 – COM Arduino DUE	9
Il·lustració 7 – Codi engegar un pin	10
Il·lustració 8 – Smart Citizen Kit 1.1	12
Il·lustració 9 – RN 131	13
Il·lustració 10 – Smart Citizen Kit 1.5	14
Il·lustració 11 – Mapejat SCK1.5-davant	17
Il·lustració 12 – Mapejat SCK1.5-darrera	17
Il·lustració 13 – Mòdul RTX4100	18
Il·lustració 14 – Estructura CoLApps	21
Il·lustració 15 – Plataforma Amelie	22
Il·lustració 16 – SmartCitizen_DVK	26
Il·lustració 17 – SmartCitizen_DVK carregat	27
Il·lustració 18 – SmartCitizen_DVK resultat	27
Il·lustració 19 – CreateApp	28
Il·lustració 20 – Arduino Programming Port	29
Il·lustració 21 – RTX EAI Port Server Configuration	29
Il·lustració 22 – Compilació COLApps	30
Il·lustració 23 – Compilació COLApps Fitxer FWS	30
Il·lustració 24 – RTX EAI Port Server	31
Il·lustració 25 – COLA Controller failed	31
Il·lustració 26 – COLA Controller successfully	32
Il·lustració 27 – Firmware RTX	32
Il·lustració 28 – Firmware RTX carregat	33
Il·lustració 29 – PuTTY configuration	33
Il·lustració 30 – PuTTY resposta	34
Il·lustració 31 – SPI	35
Il·lustració 32 – Protocol HTTP	37
Il·lustració 33 – Exemple codi HTML	38
Il·lustració 34 – Exemple codi CSS intern	39
Il·lustració 35 – Format SLIP	40
Il·lustració 36 – Exemple Web Server Arduino	47
Il·lustració 37 – Fritzing	47
Il·lustració 38 – Muntatge amb Fritzing	48
Il·lustració 39 – Ipconfig	56
Il·lustració 40 – IP lliure	56
Il·lustració 41 – AP smartphone	57
Il·lustració 42 – Exemple SPI	61
Il·lustració 43 – Resultat WebServer	64
Il·lustració 44 – Resultat WebServer RTX4100	69

Llista de taules

Taula 1 – Especificacions tècniques Arduino DUE	5
Taula 2 – SPI commands	16
Taula 3 – Estàndards WiFi.....	43

Llista de diagrames

Diagrama 1 – Diagrama de flux Web Server Arduino	46
Diagrama 2 – Web Server Arduino Codi	46
Diagrama 3 – Diagrama de flux Flux AP Mode	49
Diagrama 4 – Diagrama de flux SoftAP	50
Diagrama 5 – Diagrama de flux SPI_AP	51
Diagrama 6 – Diagrama de funcions AppWifi	53
Diagrama 7 – Diagrama de funcions DrvSpiSlave.....	54
Diagrama 8 – Diagrama de funcions SoftAP	55
Diagrama 9 – Diagrama de flux WebServer.....	58
Diagrama 10 – Diagrama de flux SPI_WebServer	59
Diagrama 11 – Diagrama de flux demo_SPI.....	60
Diagrama 12 – Diagrama de flux TempSensor	62
Diagrama 13 – Diagrama de funcions WebServer_Amelie	65
Diagrama 14 – Diagrama de funcions WebServer	66

Llista Annexos

Annex 1a – Diagrama de Gantt desenvolupat	78
Annex 1b – Diagrama de Gantt	78
Annex 2 – Cronograma WBS.....	79
Annex 3 – PINOUT Arduino DUE	80
Annex 4 – Arduino DUE Schemati.....	81
Annex 5 – Codi WebServer Arduino.....	82
Annex 6 – Demo_SPI.....	84
Annex 7 – Codi SoftAP	87
Annex 8 – Codi WebServer RTX4100.....	89

Capítol 1: Introducció

1.1. Justificació del tema

Actualment les telecomunicacions estan jugant un paper molt important en la societat ja que ens facilita la nostra forma de vida ja sigui realitzant les activitats quotidianes o recercant informació.

Cada vegada més estem veient com s'està estenent el desplegament de les xarxes sense fils per proporcionar accés a Internet. Podem veure que en diversos llocs municipals (biblioteques, gimnàs municipal) s'està implementat la tecnologia WiFi per poder donar accés a Internet als ciutadans.

A causa de la creixent demanda, en aquest treball es pretén dissenyar i implementar una solució econòmica i fàcil d'actualitzar, crearem un servidor web que ens permeti la configuració de paràmetres en el processador host per al mòdul WiFi RTX4100.

Com ha precedent podem trobar la tesi *Analysis, Improvement, and Development of New Firmware for the Smart Citizen Kit Ambient Board*¹ de Miguel Colom, el qual ja té implementat una COLApp amb les funcionalitats principals per al funcionament del SCK. Per tant, ampliarem la seva tesi amb el nostre mòdul WiFi.

1.2. Motivació

En un principi volia fer la implementació de la WiFi en una empresa però ho veia molt senzill. Per primer any en la UOC se'ns ha ofert fer Arduino i investigant una mica vaig pensar que podria ser interessant barrejar el tema d'electrònica amb programació ja que l'electrònica m'agrada i en canvi la programació no és el meu punt fort. Per això aquest treball pot resultar-me interessant per millorar en el tema de la programació.

Sobre l'elecció del tema vaig escollir aquest perquè és el que hem sembla més fàcil d'aconseguir sense tenir coneixements previs de la placa i a més, configurarem diferents paràmetres de la WiFi que és el que en un primer dia m'interessava fer.

¹ Colom, M. *Analysis, Improvement, and Development of New Firmware for the Smart Citizen Kit Ambient Board* <http://openaccess.uoc.edu/webapps/o2/bitstream/10609/40042/6/mcolombTFG0115memoria.pdf>

1.3. Objectius generals i específics

En el present projecte hi podem trobar quatre clars objectius i diversos objectius generals, els quals podem organitzar de la forma següent:

Com a objectius específics tenim:

- Configurar un servidor web que permeti interaccionar la placa Arduino amb el mòdul RTX4100.
- Visualitzar diferents paràmetres en la WiFi com la connectivitat i l'estat dels sensor.
- Implementar el mode AP que permeti connectar a diferents equips al mòdul RTX4100.
- Elaborar la documentació de l'aplicació desenvolupada.

Els objectius generals són:

- Comprendre el funcionament i connexions de Arduino.
- Conèixer les diferents cues que es poden aplicar.
- En què consisteix SmartCitizenRTX4100.
- Proporcionar una tecnologia d'última generació que pugui ser eficient en els pròxims anys i la fàcil modificació de la mateixa.
- Conèixer quins son els diferents tipus de serveis es poden oferir per mitjà de les xarxes WiFi.
- Comunicació entre RTX4100 i la connexió sense fils.
- Estudi de les aplicacions necessàries per la connexió.
- Estudi del codi necessari per la seva implementació.
- Implementar el projecte.

1.4. Resultats potencials

La tecnologia WiFi és una de les tecnologies líder en la comunicació sense fil, i el suport per a WiFi s'està incorporant cada vegada més en aparells com portàtils o smartphone.

El que es pretén és la creació i implementació d'una infraestructura que permeti la connexió sense fils amb facilitat per monitoritzar i configurar els diferents paràmetres que ens pugui interessar com pot ser garantir la privacitat de la nostra xarxa per mitjà de verificació de l'usuari, diferents perfils WAN depenent els permisos accés, variació dels temps de connexió, cobertura dels equips, etc. A més, aquesta ha de ser econòmica, tenir un baix consum i ser fàcil de configurar.

1.5. Organització de la memòria

- **Capítol 1: Introducció.** En aquest capítol trobarem la introducció del projecte amb els seus objectius.
- **Capítol 2: Placa Arduino i RTX4100.** Farem la introducció d'Arduino i de l'SmartCitizenRTX4100. Estudiarem l'entorn de programació, veurem quines funcions i llibreries es poden fer servir i els protocols i mètodes que utilitzarem.
- **Capítol 3: Implementació de la pràctica.** Desenvolupament de la proposta, amb el codi, llibreries i muntatge.
- **Capítol 4: Conclusions i línies de futur.** Per finalitzar comentarem els èxits i les problemàtiques del nostre projecte i farem diverses propostes de futur en base del nostre resultat.
- **Capítol 5: Acrònims.**
- **Capítol 6: Bibliografia.**
- **Capítol 7: Annexos.** En aquest apartat podrem veure els annexos del treball com per exemple el diagrama de Gantt, el cronograma, part de codi, etc.

Capítol 2: Placa Arduino i RTX4100

2.1 Introducció

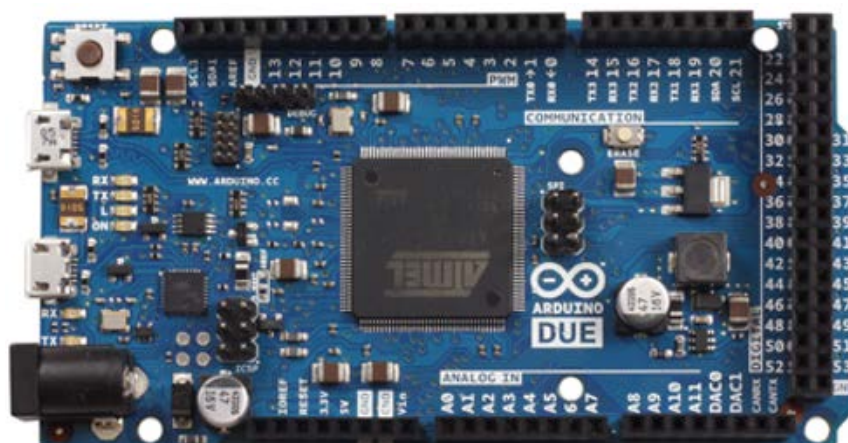
Una vegada temin plantejat el tema del projecte entrarem a veure les especificacions del material que farem servir.

La plataforma de desenvolupament consistirà en una variant d'Arduino DUE i una shield customitzada que integra un RTX4100 i un sensor SHT (temperatura i humitat). Arduino té la característica de ser de codi obert cosa que permet adaptar-lo a les nostres necessitats sense haver de necessitar llicència. Primer de tot farem una ullada a aquesta placa base per veure quines característiques ens ofereix.

També es proporcionarà un firmware customitzat capaç d'utilitzar mitjançant un únic port USB la càrrega de firmware en el propi MCU, el RTX4100 i l'accés al port de sèrie secundari del RTX4100 per debugging i test. Veurem com es compila i es carrega el codi tant com per la placa Arduino com per el mòdul RTX4100.

2.2 Placa Arduino Due

Arduino és una plataforma d'electrònica oberta que posseeix una alta gama de plaques prefabricades que s'ajusten a les diferents necessitats. En el nostre cas farem servir una variant d'Arduino Due facilitat per l'empresa SmartCitizen.



Il·lustració 1: Placa Arduino DUE

Arduino Due està basat en un microcontrolador de 32bits a 84MHz millorant les versions anteriors com UNO i Leonardo que tenen 8bits a 16MHz. El seu microcontrolador és un Atmel AT91SAM3X8E que permet disposar d'entrades i sortides analògiques amb resolució de 12bits. Disposa de dos ports micro-USB, un per la programació i comunicació ("Programming") i l'altre que el permet actuar com client/servidor permetent connectar perifèrics externs USB ("Native"). A més, porta

integrat dos botons: “erase” amb el que podem esborrar la memòria Flash del microcontrolador i el “reset” que permet reiniciar el programa que s’executa. La seva taxa de mostreig va de 15ksps fins a 1000ksps.

Tots els seus pins estan numerats del 0 a 53 i poden ser utilitzats indistintament com entrades o sortides digitals les qual treballen a 3,3V, cada pin pot subministrar una corrent de 3mA a 15mA i disposen d’una resistència pull-down de 100KΩ.

El llenguatge de programació d’Arduino és el “Wiring²” i el seu entorn de desenvolupament està basat en el llenguatge “Processing³” que és un llenguatge de programació i entorn de desenvolupament integrat de codi obert basat en Java, de fàcil utilització i que està enfocat a l’ensenyança i producció de projectes multimèdia i interactius de disseny digital.

Especificacions

A continuació mostrem les especificacions tècniques més rellevants de la placa Arduino DUE:

Especificacions tècniques Arduino DUE	
Microcontrolador	AT91SAM3X8E
Voltatge operatiu	3.3V
Voltatge entrada (recomanat)	7-12V
Voltatge entrada (min./màx.)	6-20V
Pins digitals I/O.....	54 (6 amb PWM)
Pins d’entrada analògics.....	12
Pins de sortida analògics.....	2 DAC
Total de corrent directa per tot I/O.....	130 mA
Corrent directa per el pin 3.3V	800 mA
Corrent directa per el pin de 5V.....	800 mA
Memòria Flash.....	512 KB
SRAM	96 KB (64 + 32 KB)
Velocitat del rellotge.....	84 MHz
Ports micro-USB.....	2
Altura / Amplada.....	101.52 mm / 53.3 mm
Pes	36g

Taula 1: Especificacions tècniques Arduino DUE

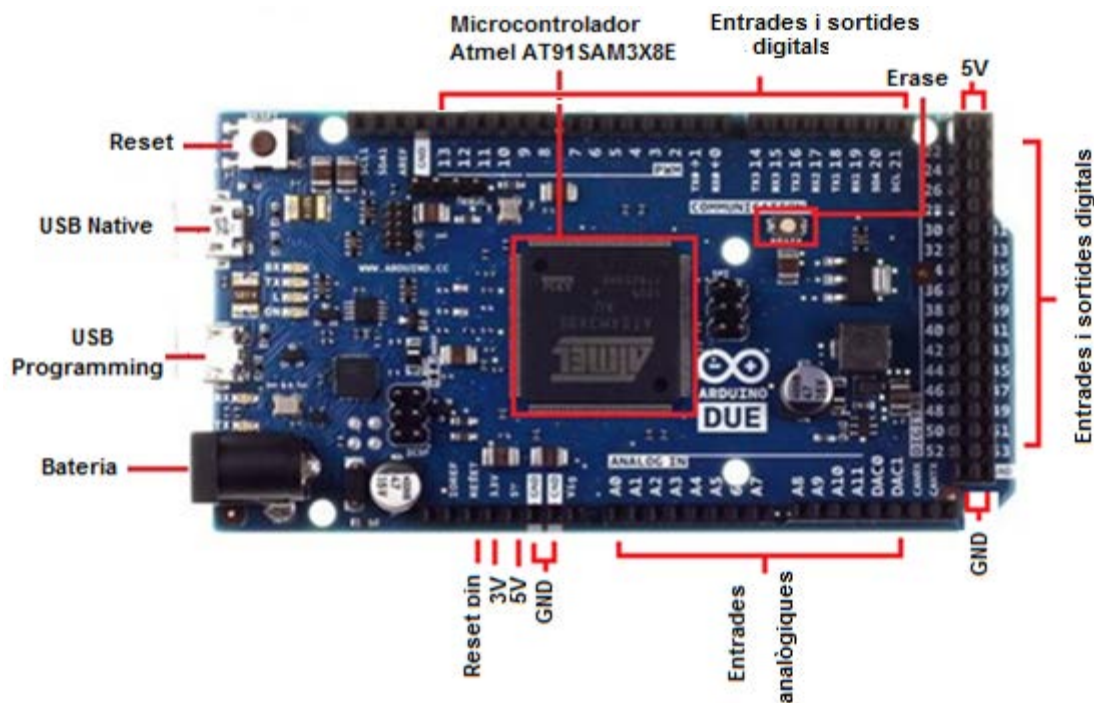
² <http://wiring.org.co>

³ <http://processing.org>

Mapejat de Pin

En l'Annex 3 podem trobar el mapejat de pin d'Arduino DUE i el seu esquema.

A continuació mostrem els principals elements que componen la placa Arduino DUE:



Il·lustració 2: Mapejat pin Arduino DUE

Com es pot veure tenim el botó d'erase i el reset, un port USB Native i un port USB Programming, una bateria per si volguéssim connectar-la a la corrent, diverses entrades i sortides digitals i el microcontrolador Atmel AT91SAM3X8E. Com hem dit abans els pins s'enumeren del 0 al 53 i poden ser utilitzats tant com entrades com sortides digitals. Totes les entrades/sortides treballen a 3,3V però cada pin pot suportar una corrent de 3mA-15mA. També tenen una resistència de *pull-down* que per defecte està desactivada i és de 100KΩ. A més, alguns pins tenen funcions específiques com per exemple:

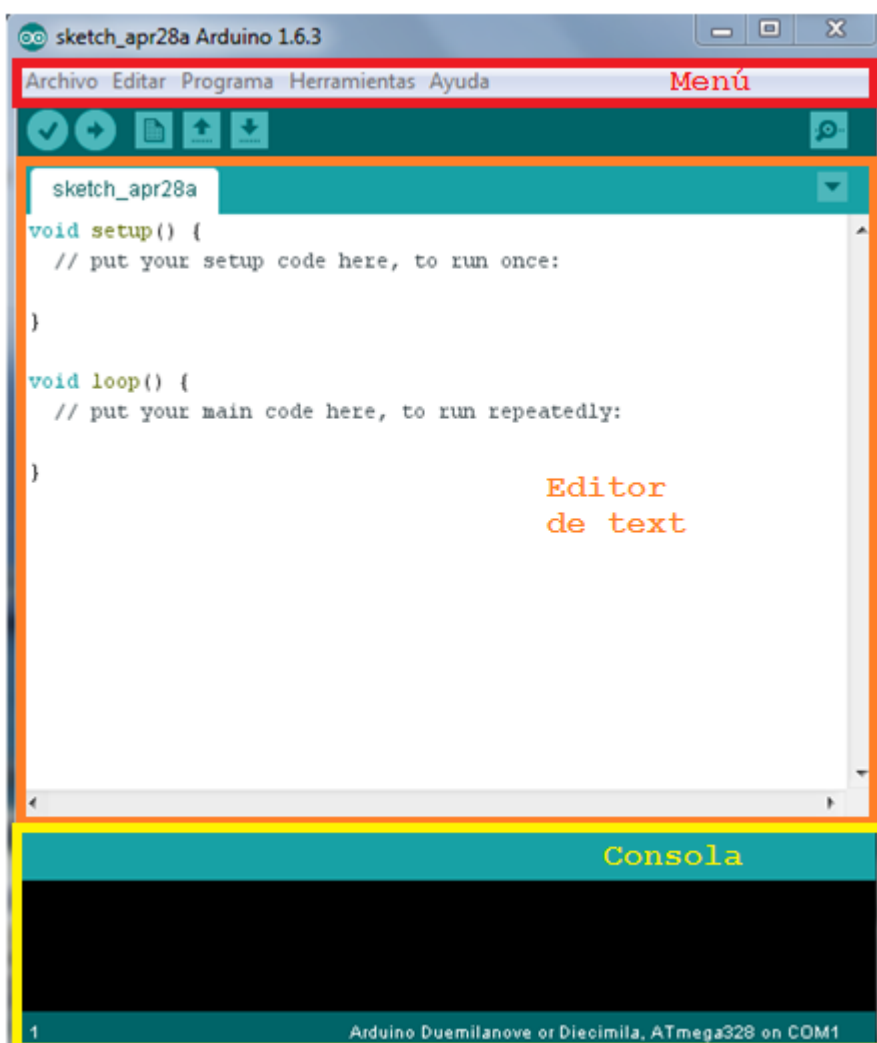
- Pins de 2 a 13: Sortides PWM de 8 bits de resolució.
- Pins de A0 a A11: Entrades analògiques.
- Pins DAC0 y DAC1: Proporcionen una sortida analògica amb una resolució de fins 12 bit (4096 nivells).
- 4 canals de comunicació serial RX0 – TX0.
- Interfase I²C (SDA, SCL).
- AREF: Referència externa per a voltatge d'entrades analògiques.

2.2.1 Programació en Arduino







En el següent apartat veurem un exemple bàsic per comprovar que la nostra placa està ben configurada i funciona, aquest exemple ens servirà per començar a introduir-nos en la programació d'Arduino. Abans però, expliquem el seu entorn de programació.

Primer de tot, hem de descarregar-nos i instal·lar la última versió de l'IDE d'Arduino, el podem trobar fàcilment a <http://arduino.cc/es/Main/Software>.

L'entorn de desenvolupament d'Arduino està basat en un editor de text, una consola i una àrea de missatges:



Il·lustració 3: Editor de text

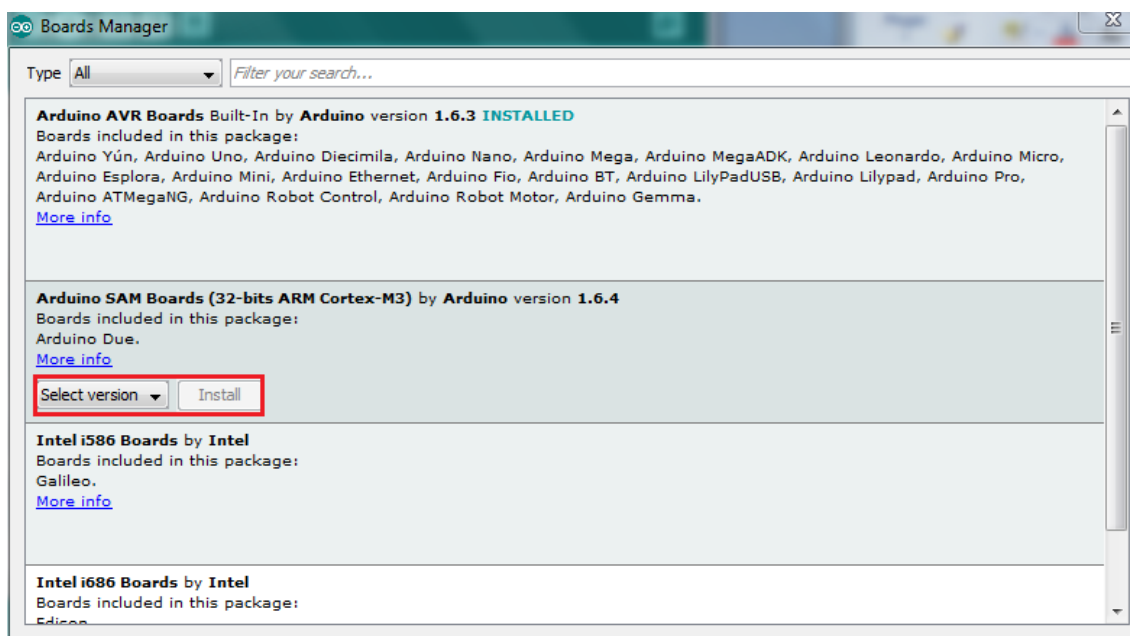
Els botons de sota la barra de Menú serveixen per:  Verificar el codi,  carregar l'sketch,  obrir una fulla en blanc,  cercar codi ja creat,  guardar el codi i  veure el monitor sèrie.

Connectem la placa Arduino DUE (USB Programming) al nostre PC per mitjà d'un cable USB – microUSB. Si li arriba la corrent s'engegarà el led PRW. Al connectar la placa al sistema operatiu Windows ens demanarà la instal·lació dels divers que en el nostre cas al fer servir W7 es fa automàtic.



Il·lustració 4: Led PRW engegat

Executem l'aplicació Arduino i comprovem que amb la instal·lació bàsica el model de la nostra placa no està instal·lat, per això tindrem que anar a la opció “Herramientas/ Placa/ Boards Manager”, aquí ens sortiran diverses plaques que es poden instal·lar, la que ens interessa és la Arduino SAM Boards, seleccionem la última versió i la instal·lem.

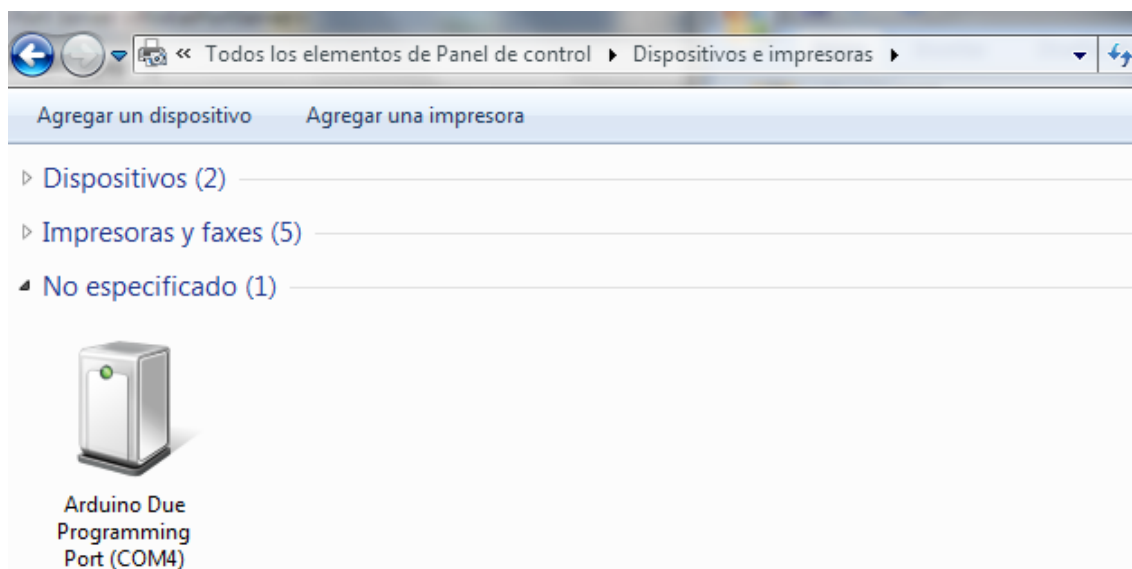


Il·lustració 5: Board Manager

Finalment, ja podem seleccionar el model de la nostra placa (Herramientas / Placa / Arduino Due (Programming port)). També cal comprovar que el port estigui ben

configurat (Herramientas / Puerto Serial), en principi per defecte ens agafarà el COM on estigui connectada la nostra placa però per si a cas ho comprovarem.

Per saber on està connectada anirem a “Panel de control\Todos los elementos de Panel de control\Dispositivos e impresoras”, aquí veurem el nostre dispositiu i el seu COM:



Il·lustració 6: COM Arduino DUE

Arribat a aquest punt ja podríem pujar el programa a la nostra placa. Abans però explicarem com és l'estructura del programa, les llibreries predefinides que podem trobar i les constants que es poden fer servir. Podeu però, veure un exemple de com carregar un programa en l'apartat 2.5.2. Entorn Arduino IDE

2.2.1.1 Estructura del programa

El llenguatge de programació d'Arduino està basat en el llenguatge de programació C/C++. El programa en Arduino es coneix amb el nom de “sketch⁴” i posseeixen tres parts principals: estructura, variables i funcions. A més divideix l'execució en dos parts, setup() i loop():

- setup(): Es carrega quan s'inicia un programa o sketch i només s'executa una vegada. És la part encarregada de contenir les variables i les seves declaracions.
- loop(): S'executa constantment de forma cíclica mentre la placa estigui en funcionament. Desenvolupa el codi principal del programa i s'executa després de setup().

Per tant, setup() conté la configuració i loop() conté el programa.

⁴ Més informació a www.arduino.cc/es/Tutorial/Sketch

Constants

Existeixen constants predefinides per assignar un valor a un pin d'entrada/sortida:

TRUE: Cert, valor 1 o diferent a 0.

HIGHT: Nivell alt.

INPUT: Assignar pin com entrada.

FALSE: Fals, valor 0.

LOW: Nivell baix.

OUTPUT: Assignar pin com sortida.

Funcions específiques

Com a funcions específiques podem trobar les següents:

pinMode(pin, mode): Configura el pin per comportar-se com entrada o sortida digital.

digitalRead(pin): Llegeix el valor d'un pin digital i retorna un valor "HIGH" o "LOW".

digitalWrite(pin, value): Introdueix un "1" (HIGH) o "0" (LOW) en el pin digital especificat.

analogRead(pin): Llegeix el valor analògic d'un pin amb una resolució de 10 bits predeterminada. Només funciona en els pins analògics (A0 – A11). El valor resultant és un enter de 0 a 1023.

analogWrite(pin, valor): Escriu un valor de 0 – 255 (resolució de 8 bits) en el pin especificat. Es pot utilitzar els pins del 0 – 53 com sortida.

analogReference(type): Configura el valor de voltatge utilitzat per l'entrada analògica, per defecte és de 3.3 Volts.

delay(valor en ms): Realitza una pausa en el programa segons el temps especificat.

millis(): Retorna la quantitat de milisegons que porta la placa executant el programa actual, pot contar fins a 50 dies, passat aquest temps comença novament.

Una vegada hem vist per sobre quines constants i funcions podem fer servir, passem a veure un exemple molt bàsic d'sketch amb el que engegarem un pin, esperem 1 segon, apaguem el pin i esperem un altre segon:

```
void setup() {
  Serial.begin(9600);           //Iniciem serial
  pinMode(13, OUTPUT);         //Iniciem "pin 13" com sortida
}
void loop() {
  digitalWrite(13, HIGH);      //Activa pin
  Serial.println("HOLA MON");  //Escriu per el monitor serial
  delay(1000);                 //Espera 1 segon
  digitalWrite(13, LOW);      //Desactiva pin
  delay(1000);                 //Espera 1 segon
}
```

Il·lustració 7: Codi engegar un pin

2.2.2 Lliberies

Com hem dit anteriorment, Arduino fa servir el llenguatge C/C++ per tant, les seves lliberies també estan en el mateix llenguatge. Al ser un programa obert podem crear les nostres pròpies lliberies encara que totes han de seguir uns estàndards definits⁵.

A continuació enumerem algunes de les lliberies ja creades i que estan relacionades amb la nostra proposta:

Standard Libraries

- EEPROM - Reading and writing to "permanent" storage.
- Ethernet - For connecting to the internet using the Arduino Ethernet Shield.
- Firmata - For communicating with applications on the computer using a standard serial protocol.
- SD - For reading and writing SD cards.
- SPI - For communicating with devices using the Serial Peripheral Interface (SPI) Bus.
- WiFi - For connecting to the internet using the Arduino WiFi shield.

Due Only Libraries

- USBHost - Communicate with USB peripherals like mice and keyboards.

USB Libraries

- Keyboard - Send keystrokes to an attached computer.
- Mouse - Control cursor movement on a connected computer.

Contributed Libraries

- Messenger - For processing text-based messages from the computer.
- Simple Message System - Send messages between Arduino and the computer.
- Webduino - Extensible web server library (for use with the Arduino Eth. Shield).
- XBee - For communicating with XBees in API mode.
- SerialControl - Remote control other Arduino over a serial connection.

Sensing:

- Capacitive Sensing - Turn two or more pins into capacitive sensors.
- Debounce - For reading noisy digital inputs (e.g. from buttons).

Timing:

- DateTime - A library for keeping track of the current date and time in software.

⁵ Guia de desenvolupament lliberies Arduino: <http://arduino.cc/en/Hacking/LibraryTutorial>

2.3 Smart Citizen Kit

Smart Citizen és una plataforma que serveix per generar processos participatius entre les persones en la ciutat. L'objectiu de la plataforma és servir com a node productiu per la generació d'indicadors i les eines distribuïdes. El seu projecte es basa en la geolocalització, en Internet i el hardware i software lliure per la captura de les dades.

El Smart Citizen Kit (SCK) consisteix en dos PCBs, la primera dedicada al processament, transmissió i emmagatzemant de dades (Data Board) i la segona als sensors i els seus complements (Sensor Board - Ambient sensor board). Tot el firmware està desenvolupat sobre les llibreries Arduino, extraient les funcionalitats principals a llibreries en C++ natiu per simplificar el codi.

Actualment es fa servir el Smart Citizen Kit 1.1 que a continuació veurem com està format, encara que per el nostre projecte se'ns facilitarà per part de Fab Lab Barcelona i Hangar la versió Smart Citizen Kit 1.5.

2.3.1 SCK 1.1

La versió actual de la Data Board és el SCK 1.1⁶ que està basat en un Atmel ATMEGA32U4 (AVR 8-bits) que és la MCU principal. La principal característica d'aquest MCU és que disposa de suport USB natiu per lo que no requereix un convertidor USB-sèrie en la placa.



Il·lustració 8: Smart Citizen Ki 1.1t

⁶ SCK 1.1 Hardware design files <https://github.com/fablabbcn/Smart-Citizen-Kit/tree/master/hardware/Kickstarter/v1.1b>

També porta un mòdul WiFi que es comunica mitjançant comandes per sèrie amb el AT32U4 per a la transmissió de les dades dels sensors a la plataforma web⁷.

La versió BETA porta:

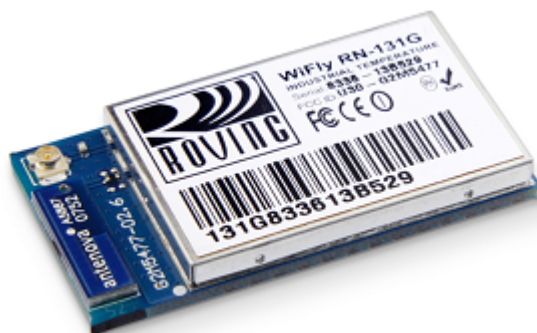
- Base Board v1.1
- Sensor ambiental v1.1

El SCK és una peça de hardware que està compost per dos plaques d'unions planes: una placa filla intercanviable i una taula de processament de dades compatible amb Arduino.

La placa del sensor porta sensors que mesuren la composició de l'aire (CO i NO₂), la temperatura, la intensitat de la llum, els nivells de so, i la humitat. Una vegada que s'ha establert la connexió, pot transferir les dades mesurades per els sensors per mitjà de la WiFi .

RN 131

El mòdul WiFi que porta és un Microchip RN 131⁸ (WiFly) el qual és independent i integra els estàndards 802.11 b/g. És ideal per a les aplicacions sense fils mòbils gràcies a la seva petita forma i al molt baix consum d'energia.



Il·lustració 9: RN 131

El mòdul WiFly incorpora una ràdio de 2,4 GHz, processador, pila TCP/IP, rellotge en temps real, gestió d'energia, i sensor analògic. Només necessita quatre connexions (PWR, TX, RX i GND) per crear una connexió de dades sense fil. A més, el sensor interfície proporciona la temperatura, so, moviment, acceleració, i altres dades analògiques sense requerir maquinari addicional. El mòdul està programat i controlat amb llenguatge ASCII. Una vegada que el mòdul està configurat, pot escanejar per trobar un accés punt, associat, autenticar, i connectar-se a través de qualsevol xarxa WiFi.

⁷ Plataforma web Smart Citizen <http://smarcitizen.me>

⁸ Microchip RN1313 <http://ww1.microchip.com/downloads/en/DeviceDoc/rn-131-ds-v3.2r.pdf>

Actualment, aquest mode del SCK només es pot comunicar mitjançant el sistema de comandes del mòdul per associar-se a una xarxa WiFi i enviar les dades a un host remot mitjançant socket TCP. No existeix cap SDK obert per treballar directament sobre el firmware del SoC (System on Chip) del mòdul, la interacció queda limitada a un sistema de comandes⁹. Podem trobar la implementació d'aquesta arquitectura en el firmware actual del Smart Citizen Kit, amb la versió 0.9.0¹⁰.

2.3.2 SCK 1.5

Actualment s'està treballant en la versió 1.5 de la Data Board (SCK 1.5). El firmware encara està en desenvolupament. Els principals canvis d'aquest disseny és el canvi de MCU i del mòdul WiFi. El nou MCU és un ATMEL ARM Cortex M0+ (SAMD21)¹¹, el mateix controlador utilitzat per l'Arduino Zero (encara en desenvolupament) i amb una arquitectura similar a la del Arduino DUE (Atmel SAM3X8E ARM Cortex-M3).

El firmware s'ha portat a la arquitectura ARM Cortex M3 de l'Arduino DUE i s'està treballant per portar-lo al ARM Cortex M0+ (SAMD21). En quant al firmware del SCK, encara el desenvolupament d'una llibreria que suporti SPI està en procés, només s'han validat mitjançant a nivell de comunicació hardware mitjançant comandes SPI manuals.



Il·lustració 10: Smart Citizen Ki 1.5t

⁹ Basic RN 131 WiFly Commands supported by SCK firmware <https://github.com/fablabbcn/Smart-Citizen-Kit/blob/master/sck-commands.md#basic-sck-setup-commands>

¹⁰ Firmware SCK 1.1 <https://github.com/fablabbcn/Smart-Citizen-Kit/>

¹¹ Atmel Cortex M0+ SAMD21 datasheet http://www.atmel.com/Images/Atmel-42181-SAM-D21_Datasheet.pdf

El nou mòdul WiFi és un RTX4100¹² el qual inclou un MCU Energy Micro Gecko (EFM32 - ARM Cortex M3) de Silicon Labs i un mòdul WiFi Atheros AR4100 WiFi SiP. Més endavant veurem les seves característiques però principalment s'ha escollit per el seu baix consum energètic i la possibilitat de desenvolupar un propi firmware per ell.

El firmware es desenvolupa com un mòdul que s'integra en la arquitectura RTOS propietari que gestiona el RTX4100 que es refereixen com COLApp.

Actualment ja està desenvolupada una COLApp, l'Smart-Citizen-RTX4100¹³, que implementa les funcionalitats principals necessàries per al funcionament del SCK. Aquest desenvolupament forma part de la tesi *Analysis, Improvement, and Development of New Firmware for the Smart Citizen Kit Ambient Board*¹⁴ de Miguel Colom. Les funcionalitats principals que inclouen la implementació són associar-se a una xarxa WiFi i la connexió remota mitjançant un socket TCP. Aquestes accions es realitzen mitjançant l'enviament de comandes per SCI al mòdul des del MCU del SCK.

SPI

Abans d'aprofundir en les comandes implementades actualment, veurem que fa el protocol SPI.

SPI (Serial Peripheral Interface) és un estàndard de comunicacions que es fa servir per la transferència entre circuits integrats. És un procés síncron ja que el flux de bits està regulat per un rellotge i la transmissió i sincronització es realitza per mitjà de quatre senyals:

- **SCLK** (Clock): És el pols que marca la sincronització. Amb cada puls es llegeix o s'envia un bit. També anomenat TAKT.
- **MOSI** (Master Output Slave Input): Sortida de dades del Màster i entrada de dades al Slave. També anomenat SIMO.
- **MISO** (Master Input Slave Output): Sortida de dades del Slave i entrada al Màster. També anomenat SOMI.
- **SS/Select**: Per seleccionar un Slave, o per que el Màster li digui al Slave que s'activi. També anomenat SSTE.

¹² RTX 4100 modules http://www.rtx.dk/RTX41xx_Modules-4024.aspx

¹³ SCK COLApp for RTX41000 <https://github.com/fablabbcn/SmartCitizenRTX4100>

¹⁴ Colom, M. *Analysis, Improvement, and Development of New Firmware for the Smart Citizen Kit Ambient Board* <http://openaccess.uoc.edu/webapps/o2/bitstream/10609/40042/6/mcolombTFG0115memoria.pdf>

En l'enviament de comandes al mòdul des del MCU SCK el programa espera ordres al canal SPI, les executa i torna de nou informació utilitzant també la comunicació SPI. La interfície SPI permet comunicar la RTX4100 amb l'exterior a través de 15 diferents ordres. La comanda ha de ser sempre iniciada per la capa superior mitjançant l'enviament d'un byte que identifica la comanda que ha de ser executat.

Com a avantatges principals podem trobar que té major velocitat de transmissió i que consumeix menys energia I²C o SMBus, no és necessari arbitratge o mecanisme de resposta davant d'errors ni tampoc necessiten el seu propi rellotge ja que fan servir el que envia el servidor. I com a desavantatges principals tenim que no hi ha control de flux per hardware, no hi ha senyal d'assentiment i el servidor podria estar enviant informació sense adonar-se que cap client estigués connectat.

SPI commands reference

La llista resumida de les comandes¹⁵ implementades és la següent:

SPI commands references
1. Get status
2. DNS33 resolve
3. IP config
4. TCP start
5. Associate and connect to the AP
6. Disassociate and disconnect from the AP
7. Setup AP
8. Close TCP connection
9. TCP receive
10. TCP send
11. WiFi chip power on/off
12. WiFi set power save profile
13. WiFi set transmit power
14. WiFi chip suspend
15. WiFi chip resume

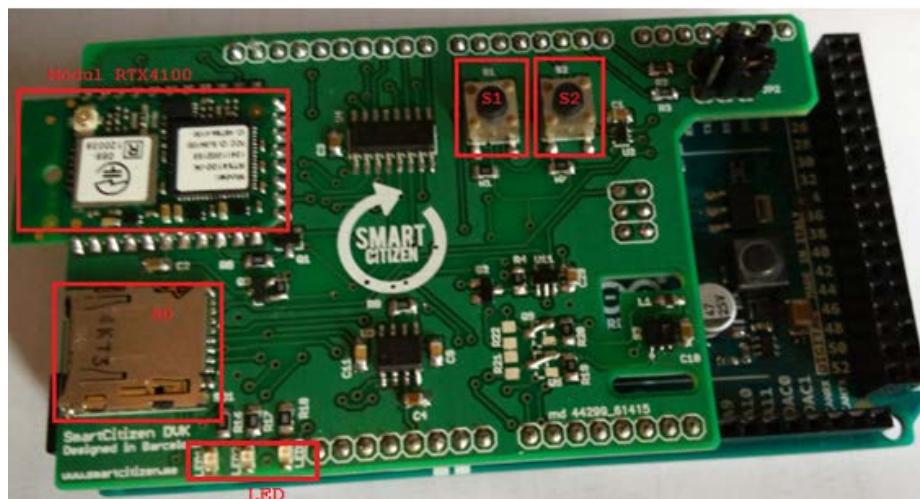
Taula 2: SPI commands

¹⁵ Quick command ref. for the RTX SCK implementation

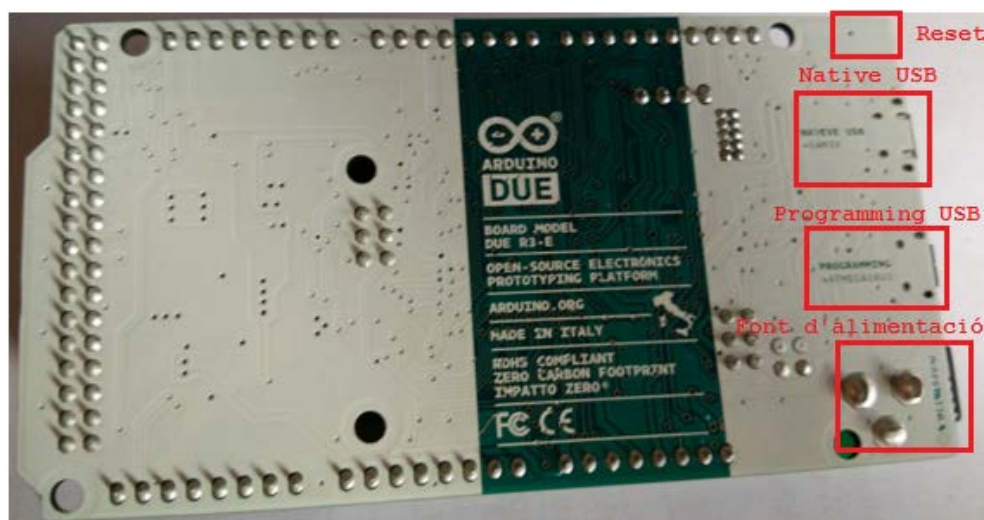
<https://github.com/fablabbcn/SmartCitizenRTX4100#spi-commands-reference>

Mapejat de Pin

A continuació representarem el mapejat de la nostra placa SCK1.5, com hem dit consta de 2PCB, l'SCK 1.5 i Arduino DUE:



Il·lustració 11: Mapejat SCK1.5-davant



Il·lustració 12: Mapejat SCK1.5-darrera

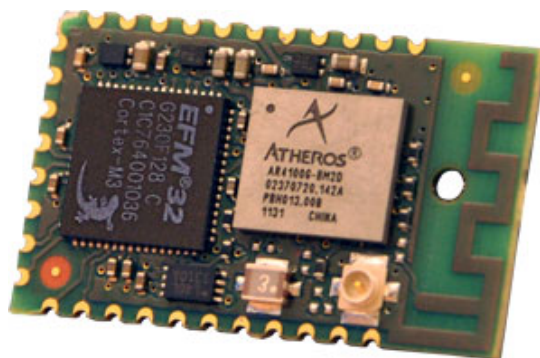
Les parts assenyalades són les més importants

No entrarem en detall en el seu mapejat, només hem assenyalat els components que són fàcils d'identificar i que nosaltres farem servir. Podem identificar fàcilment en la placa formada per Arduino DUE l'USB Native, l'USB Programming i el botó de reset. En la placa de l'SCK trobem el mòdul RTX4100, dos botons que explicarem més endavant (S1 i S2) i els LEDS que ens indicaran l'estat de la placa.

2.3.2.1 Mòdul RTX4100

El Mòdul WiFi RTX4100¹⁶ és una petita placa que suporta l'estàndard 802.11b/g/n WiFi. El seu processador d'aplicacions és de baixa potència i el seu nucli MCU és de 32 bits. També té interna la memòria flash i RAM.

És una solució de baix consum, que s'aconsegueix mitjançant l'ús de la Micro Gecko Energia i el xip Atheros AR4100 WiFi SIP que es pot apagar si no es fa servir (ultra-low power mode). A més, porta dos LDO's per controlar l'alimentació del mòdul WiFi i de la flash.



Il·lustració 13: Mòdul RTX4100

A diferència d'altres mòduls semblants, el RTX4100 permet desenvolupar aplicacions customitzades mitjançant un SDK propi, l'Amelie SDK (veure punt **¡Error! No se encuentra el origen de la referencia. ¡Error! No se encuentra el origen de la referencia.**).

Mòdul WiFly

Els mòduls WiFly suporten diversos mètodes per accedir a xarxes WiFi. En la versió firmware 2.45 suporten l'Access Point (AP). Aquest mode ha de proporcionar diversos avantatges respecte el mode Ad-Hoc:

- Crear una xarxa AP perquè els dispositius Android puguin unir-se.
- S'executarà el servidor DHCP i emetrà adreces IP a set clients.
- Donarà suport a la seguretat en futures versions.
- Suportarà l'enrutament entre els clients.

Els mòduls WiFly poden operar de dos maneres:

- Mode Infraestructura - El mòdul pot unir-se a una xarxa creada per un punt d'accés (AP).
- La manera Soft AP - El mòdul es comporta com un AP amb una funcionalitat limitada.

¹⁶ Més informació a http://www.rtx.dk/Files/Billeder/RTX_T/RTX4100docs/RTX4140_Datasheet_DS2.pdf

El mòdul de firmware porta:

- **Aplicació CoLA:** Aquest component implementa un model de programació on l'aplicació està vinculada dinàmicament amb els serveis prestats per les capes inferiors. L'aplicació es compila i es vincula com un programa separat que en temps d'execució es carrega i s'executa com una tasca sota el sistema operatiu.
- **API:** És la interfície exposada per la plataforma de firmware. S'exposen totes les funcionalitats que necessita l'aplicació per implementar un dispositiu WiFi, com ara un sensor o un actuator.
- **Sistema operatiu:** Sistema operatiu de baix consum RTX que implementa la funcionalitat necessària per organitzar tasques internes, així com l'aplicació de CoLA.
- **Pila de xarxa:** És una capa funcional, l'aplicació de la pila de xarxa IP per a IPv4 i IPv6. La pila de xarxa s'executa en el xip AR4100P WiFi RTX4100.
- **Client DNS:** El client DNS s'utilitza per traduir noms de les IP.
- **HTTP:** Servidor HTTP comú i aplicació HTTP client. La implementació del servidor HTTP inclou la gestió de connexions TCP, l'anàlisi de la petició HTTP, la generació/enviament de missatges de resposta HTTP i l'emmagatzematge de fonts HTTP (pàgines web) i una funció de resposta que s'utilitza per generar el contingut. L'aplicació client HTTP inclou la manipulació de connexió TCP, la generació/enviament de missatges de sol·licitud HTTP i l'anàlisi de HTTP de resposta.
- **Gestió de WiFi:** S'encarrega de tots els aspectes de la connexió WiFi a un punt d'accés WiFi, com per exemple la seguretat de la connexió sense fils, la gestió de l'energia WiFi, etc.
- **Administració d'energia:** S'assegura que qualsevol part externa o interna de la MCU està funcionant només per la quantitat de temps apropiat.
- **Gestió de Firmware :** Aquest component implementa una funcionalitat per realitzar una actualització del firmware remot de l'aplicació Co Situat.
- **Gestió NVS :** Aquest component implementa un emmagatzematge no volàtil (NVS) en una part de la FLASH interna de la MCU .
- **Drivers:** Aquesta és una capa funcional implementar una sèrie de controladors de maquinari per a perifèrics MCU, així com la interfície física a la WiFi.

2.4 RTX41xx DVK

El RTX41xx DVK (Kit de Desenvolupament) ofereix una completa forma d'avaluar el rendiment i les característiques del RTX41xx així com els mitjans per desenvolupar una aplicació personalitzada (CoLA).

El DVK consisteix en la WSAB, un acoblament d'estació per descarregar CoLA i JTAG de depuració i associar-lo al programari de PC per poder controlar i càrrega una aplicació (CoLA) al mòdul RTX41xx.

L'aplicació de terminal precarregada en el mòdul RTX41xx proporciona un mitjà per connectar-se a un altre client o servidor de la xarxa a través d'un punt d'accés (AP) de la xarxa WiFi i enviar dades de manera eficient per aquesta.

El WSAB es pot utilitzar per avaluar el consum de corrent i per proves de rang. El programari de PC és compatible amb una eina basada en GNU i un controlador CoLA per al desenvolupament fàcil d'aplicacions, la seva descàrrega i depuració.

El DVK inclou diverses aplicacions de referència en el codi font per a utilitzar com a font d'inspiració per al seu propi desenvolupament i una cadena d'eines de cost zero.

COLApps

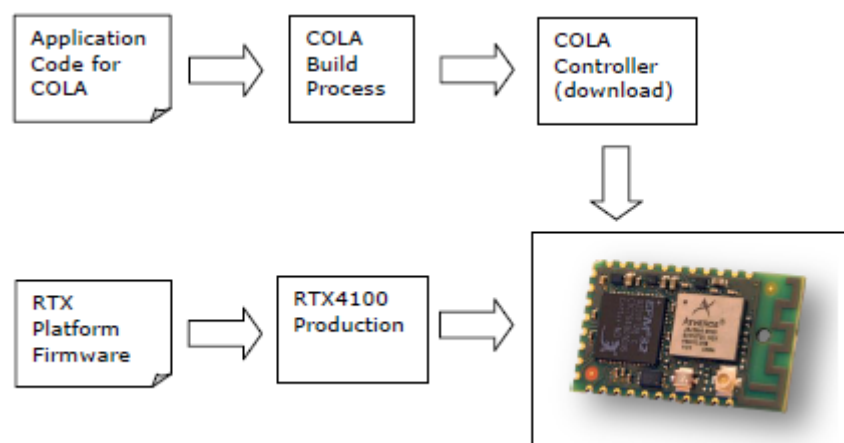
CoLA és l'abreviació de CO-Located Application¹⁷, una aplicació d'usuari que resideix al costat de la plataforma firmware.

Quan CoLA es carrega a la flash, el firmware de la plataforma detecta la seva presència i l'executa com una tasca simple en el seu sistema operatiu. Aquest concepte de CoLA facilita als programadors d'aplicacions crear les seves aplicacions sense deixar de tenir totes les característiques del firmware subjacent per mitjà de les APIs (Application Programmers Interfaces).

El RTX4100 ve precarregat amb la plataforma de firmware RTX, amb suport per a aplicacions CoLA . El desenvolupador de l'aplicació pot construir les seves pròpies aplicacions basat a les API definides pel microprogramari de la plataforma, i descarregar l'aplicació en el mòdul per a la seva execució.

El CoLA precarregat en el sensor de la WSAB subministrat amb el DVK és l'aplicació que s'anomena Terminal.

¹⁷ SCK CoLApp for RTX41000 <https://github.com/fablabbcn/SmartCitizenRTX4100>



Il·lustració 14: Estructura CoLApps

Les diferents parts que formen part d'un CoLApps són :

- **Aplicació d'usuari:** És el component que implementa la funcionalitat de l'aplicació del dispositiu WiFi. Normalment està escrit pel programador de l'aplicació/desenvolupador utilitzant l'API disponible.
- **Protocols d'aplicació:** Són capes funcionals específiques del producte opcional d'execució protocols per a una funcionalitat específica, com per exemple, COAP o MQTT. Els protocols d'aplicació són una part del SDK de RTX, i estan disponible com a codi font o biblioteca binària.
- **Aplicacions de xarxa:** Són components funcionals específics del producte per la implementació d'una varietat d'aplicacions de xarxa, SNMP, HTTP, servidor web, etc. Són una part del SDK de RTX, i estan disponible com a codi font o biblioteca binària.

Software

Totes les targetes vénen amb:

- Firmware Arduino SmartCitizen_DVK.ino¹⁸.
- RTX firmware carregat.
- Aplicació de CoLA i comandes¹⁹ per la comunicació amb el mòdul WiFi.

Per fer servir l'RTX Shield hem de recordar que cal connectar un cable al port "programming" de la placa Arduino DUE (és el que està més a prop del DC power).

¹⁸ Es pot descarregar en http://www.rtx.dk/Download_Center_Login-3980.aspx

¹⁹ Guia d'usuari:

https://github.com/fablabbcn/SmartCitizenDVK/blob/master/Documentation/RTX4100/RTX4100_User_Guide_Module_Evaluation_UG1.pdf

L'RTX té dos polsadors, S1 i S2. Amb aquests dos botons es pot canviar entre 4 modes de funcionament:

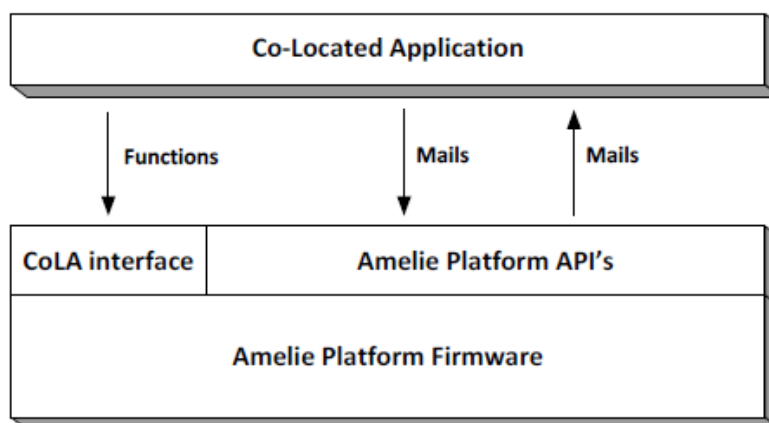
- **Mode 1:** Aquest és el mode per defecte quan es connecta la placa, el LED verd està encès. Envia per mitjà del USB-serial a 9600 bauds, els valors capturats pels sensors de temperatura i humitat i també si es detecta una targeta SD o no (es poden trobar en el main() del SmartCitizen_DVK.ino).
- **Mode 2:** Si premeu el botó S1, estem en mode terminal, el LED verd comença a parpellejar i es pot carregar una aplicació CoLApp, si el LED groc està encès, l'aplicació estarà carregada. Si es pressiona de nou S1 es tornar al mode 1.
- **Mode 3:** Si premeu el botó S2, el LED blau s'engega, ara es pot carregar aplicacions CoLa amb ColaController que es troba dins de la carpeta ColaController de l'Amelie SDK.
- **Mode 4:** Si premeu el botó S2 un cop més, els LED verd i blau s'engeguen, s'entra en mode de càrrega de microprogramari RTX i el LED blau començarà a parpellejar.

2.4.1 Amelie SDK

Construir el codi per el RTX requereix un compilador i enllaçador. Amelie SDK²⁰ es basa en ARM GCC Embedded. Aquest SDK està proporcionat per l'empresa RTX i ens ajudarà a programar el RTX4100.

L'API s'utilitza per a aplicacions d'ubicació conjunta de programes (CoLA) a la part superior de la plataforma Amelie WiFi.

L'API té la següent estructura:



Il·lustració 15 Plataforma Amelie

²⁰ API Specification Amelie Platform: www.rtx.dk/LPW/RTX4100

El desenvolupament d'aplicacions es pot separar a grans trets en tres tasques principals:

- Edició de codi i/o modificació de la font.
- Construir o compilar i enllaçar l'aplicació.
- Carregar l'aplicació en el mòdul.

La descàrrega del SDK de RTX inclou diferents codis d'exemples, els que a nosaltres ens interessa són:

- Blinky: Petit exemple que fa servir un temporitzador i fa que parpellegi un LED.
- Servidor TCP/client.
- Servidor UDP/client.
- Aplicació Terminal.
- 3 implementacions del sensor de temperatura.
- TempSensorSoftApCfg: Demuestra com és possible configurar un mòdul d'arrencada en SoftAP.
- SoftApTcpServer: Implementació AP amb el servidor TCP.
- WebServer: Aplicació de servidor web molt simple.
- Capçalera d'arxius per a les API disponibles.
- RTX EAI Port del servidor : aplicació per a PC utilitzat per a la comunicació amb el maquinari
- CoLA controlador DLL: S'utilitza per seleccionar la imatge activa i l'actualització de firmware.
- CoLA controlador GUI: Exemple d'aplicació utilitzant l'arxiu DLL.

2.5 Entorn de programació

Com hem anomenat anteriorment, el llenguatge de programació que es fa servir és C/C++. En aquest cas fa servir les mateixes constants i funcions que hem vist en l'apartat Arduino.

2.5.1 Llibreries

Existeixen diferents llibreries ja creades que podem fer servir, però en aquest projecte ens interessa la implementació de l'arquitectura en el firmware actual del Smart Citizen Kit versió 0.9.0²¹.

Si ens descarreguem el firmware SmartCitizen_DVK podem veure que està compost dels següents fitxers:

- Constants.h: Fitxer on es troben les constants de SCDVK.cpp.
- SCDVK.cpp: On es defineixen les funcions.
- SCDVK.h: Arxiu header de l'SCDVK.
- SmartCitizen_DVK.ino: Fitxer on s'executen les funcions definides de l'SCDVK.cpp.

Les funcions relacionades amb la comunicació es troben en les següents llibreries:

- Llibreria SCKBase.h²²
- Llibreria SCKServer.h²³

Podem fer servir les llibreries ja creades per Arduino com per exemple:

- SPI²⁴ - For communicating with devices using the Serial Peripheral Interface (SPI) Bus.
- WiFi - For connecting to the internet using the Arduino WiFi shield.

La instal·lació de les llibreries és molt fàcil, només s'ha de copiar a la ruta "C:\Users\lkjlk\Documents\Arduino\libraries" o bé guardar-les a la mateixa ruta del projecte.

²¹ El podeu trobar a <https://github.com/fablabbcn/SmartCitizenDVK/archive/master.zip>

²² Llibreria SCKBase.h https://github.com/fablabbcn/Smart-Citizen-Kit/blob/master/sck_beta_v0_9/SCKBase.h

²³ Llibreria SCKServer.h https://github.com/fablabbcn/Smart-Citizen-Kit/blob/master/sck_beta_v0_9/SCKServer.h

²⁴ Més informació sobre les comandes en <http://www.arduino.cc/en/Reference/SPI>

Protothreads

Els protothreads²⁵ són un conjunt de macros per fer un codi en C monolític que tingui un model d'execució basat en esdeveniments sigui més llegible i més econòmic de mantenir, d'aquesta forma, en comptes de complexes arquitectures de màquines d'estat, es pot dissenyar un codi amb funcions similars als threads.

La principal diferència entre els threads i els protothreads és que aquests no necessiten reservar una quantitat de memòria per el context de cada thread, cada protothread només requereix 2 bytes de memòria RAM.

Les seves principals característiques són:

- Sobrecàrrega de RAM petita: Només 2 bytes per protothread i no hi han piles addicionals
- Altament portàtil: La seva biblioteca és 100% codi C i sense codi assemblador específic.
- Proporciona espera de bloqueig sense multi-threading complet.
- Disponible sota una llicència de codi obert.

Comentem per sobre el seu funcionament:

- **PT_BEGIN:** Marca l'inici d'un protothread.
- **PT_END:** Marca el final d'un protothread.
- **PT_WAIT_UNTIL:** Bloqueja el protothread deixant l'execució a la resta de protothreads.
- **PT_EXIT:** Sortir del protothread.
- **PT_YIELD():** Bloqueja de forma incondicionada el protothread, aquest continuarà la seva execució quan torni a ser cridat i ho farà en el mateix punt on es va quedar bloquejat.
- **PT_SPAWN(pt, pt_estat):** Els protothreads es poden nidar, i mitjançant aquesta macro es crida a un protothread fill, la variable d'estat (2 bytes) s'emmagatzemen a la variable del pare pt_estat. El protothread pare queda bloquejat fins que el protothread fill finalitzi, amb PT_EXIT o PT_END, si el protothread fill es bloqueja, la resta de protothreads d'igual rang jeràrquic o més gran que el del pare es seguiran executant. Si en aquest estat s'invoca al protothread pare, es consultarà la condició de bloqueig del protothread fill.
- **PT_INIT(pt):** Cal inicialitzar els protothreads mitjançant aquesta macro abans de ser cridats.

²⁵ Més informació a <http://en.wikipedia.org/wiki/Protothreads>

Principalment el nostre codi s'ha de basar en COLApp que s'ha explicat en l'apartat 2.4. RTX41xx DVK, per tant, farem servir protothreads per simplificar el codi.

2.5.2 Entorn Arduino IDE

A continuació s'explica com carregar el firmware d'Arduino, en aquest cas fem servir el que ens han facilitat per defecte però seria el mateix procediment per carregar un nou firmware o un sketch. Per tornar a carregar-ho necessitem l'aplicació Arduino IDE²⁶ i el firmware de l'SmartCitizen DVK²⁷.

Primer de tot descomprimim el fitxer *.zip del firmware, aquí dintre trobarem la carpeta Firmware amb els fitxers Constants.h, LICENSE.txt, SCDVK.cpp, SCDVK.h i SmartCitizen_DVK.ino, aquests fitxers els haurem de ficar en una carpeta que s'anomeni "SmartCitizen_DVK".

Obrim l'aplicació Arduino IDE i carreguem el fitxer SmartCitizen_DVK.ino, veurem que se'ns carrega amb les seves llibreries:

```

SmartCitizen_DVK Arduino 1.6.3
Archivo Editar Programa Herramientas Ayuda
SmartCitizen_DVK Constants.h SCDVK.cpp SCDVK.h
#include "SCDVK.h"
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include "Constants.h"

SCDVK DVK;

void setup()
{
  DVK.begin();
}

void loop() // run over and over
{
  DVK.execute();
}

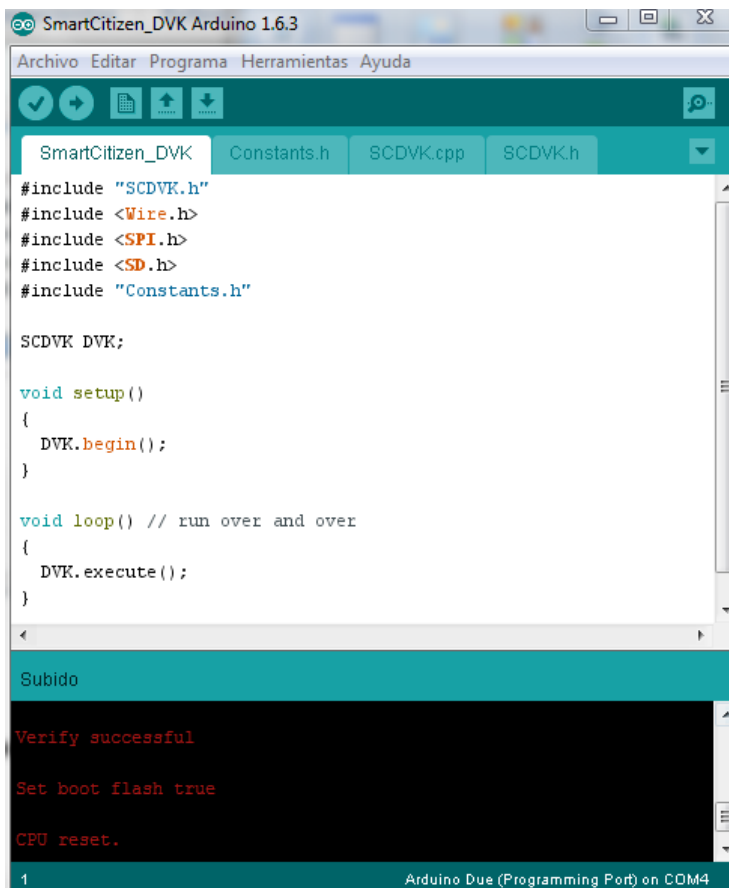
Compilado
Sketch uses 34,056 bytes (6%) of program storage space. Maximum
is 524,288 bytes.
1 Arduino Due (Programming Port) on COM4
    
```

Il·lustració 16: SmartCitizen_DVK

²⁶ El podeu trobar a <http://www.arduino.cc/en/main/software>

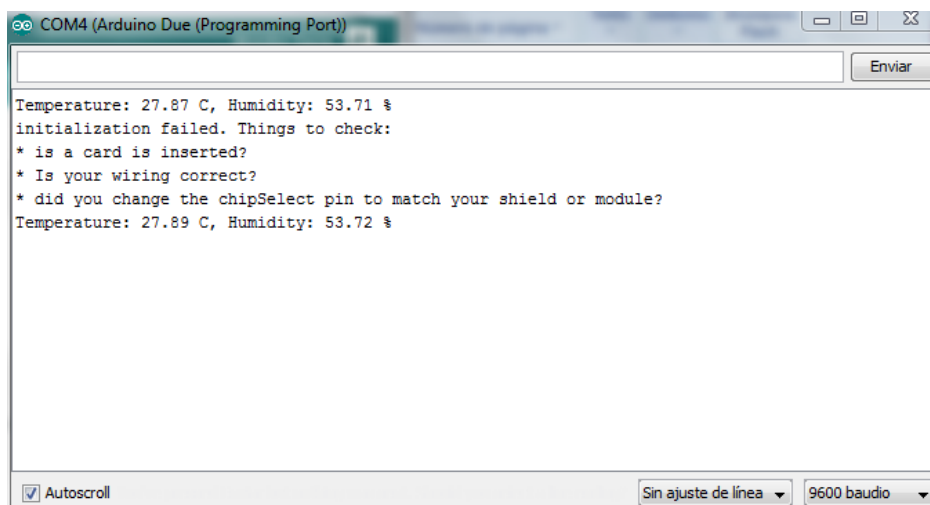
²⁷ El podeu trobar a <https://github.com/fablabbcn/SmartCitizenDVK/archive/master.zip>

Connectem la nostra placa per el port USB Programming i configurem el COM i la placa en l'aplicació tal com hem vist a l'apartat Programació en Arduino. Un cop ho tenim li donem a pujar l'sketch (fletxa cap a la dreta), se'ns engegarà el LED blau de la placa i podrem veure en la part de consola de l'aplicació que s'està carregant. Al cap d'una estona ja tindrem el firmware d'Arduino pujat.



Il·lustració 17: SmartCitizen_DVK carregat

Per comprovar que està ben carregat anem a la opció de monitor sèrie (lupa) i veurem les temperatures del mòdul:



Il·lustració 18: SmartCitizen_DVK resultat

2.5.3 Entorn Amelie SDK

L'entorn Amelie SDK ens ajudarà a programar el mòdul RTX4100. Recordem que l'API s'utilitza per a aplicacions d'ubicació conjunta de programes (CoLA).

Requisits

Primer de tot haurem de descarregar-nos l'Amelie SDK i instal·lar-lo al nostre PC, aquesta instal·lació no té complicacions.

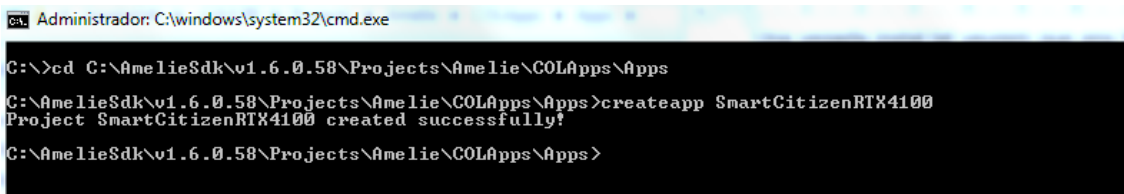
Com ja sabem, el llenguatge de programació és C+ per lo que per manipular els fitxers hauríem de tenir un editor de text²⁸ instal·lat el nostre PC.

Per construir la nostra pròpia CoLa , la versió del SDK necessita un GCC, per això ens hem d'instal·lar també el GCC ARM Embedded 7,7²⁹.

Una vegada tot instal·lat veurem que ens ha creat diversos accessos: Amelie SDK trace (RSX), COLA Controller i RTX EAI Port Server.

Creació del projecte

Primer de tot, hem de crear la carpeta del nostre projecte, si ens fixem en el manual de programació de RTX, la ruta ha de ser: "C:\AmelieSdk\v1.6.0.58\Projects\Amelie\COLApps\Apps". En aquesta ruta trobem un fitxer anomenat "CreateApp.bat", per això entrarem a la consola de comandaments de Windows i cridarem el fitxer createapp de dintre de la ruta junt amb el nom de la nostra carpeta, que en el nostre cas l'anomenarem SmartCitizenRTX4100.



```

Administrator: C:\windows\system32\cmd.exe
C:\>cd C:\AmelieSdk\v1.6.0.58\Projects\Amelie\COLApps\Apps
C:\AmelieSdk\v1.6.0.58\Projects\Amelie\COLApps\Apps>createapp SmartCitizenRTX4100
Project SmartCitizenRTX4100 created successfully!
C:\AmelieSdk\v1.6.0.58\Projects\Amelie\COLApps\Apps>

```

Il·lustració 19: CreateApp

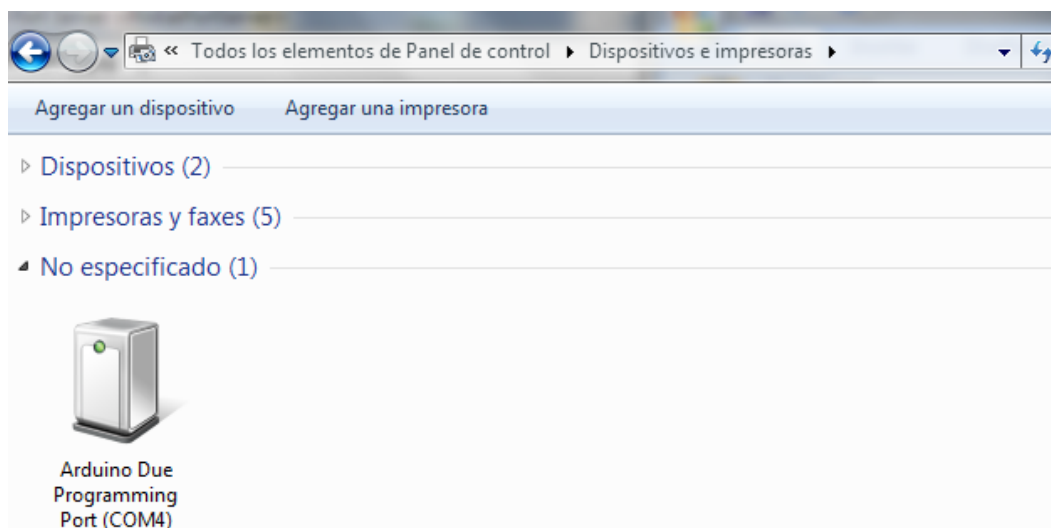
Si anem a la ruta, veurem que ens ha creat la carpeta SmartCitizenRTX4100 i una subcarpeta Build, també tenim un fitxer Main.c, que és on tindrem que ficar el nostre codi, i el fitxer node.ncf que és el que fa de node i uneix els fitxers que es troben dins del mateix projecte. Dintre de la carpeta Build veurem el model del nostre mòdul RTX i diversos fitxers que més endavant ens serviran per compilar el codi que creem en aquest projecte.

²⁸ Exemples: <http://notepad-plus-plus.org> o <http://sublime.com>

²⁹ Descarregar de <https://launchpad.net/gcc-arm-embedded/+milestone/4.7-2013-q3-update>

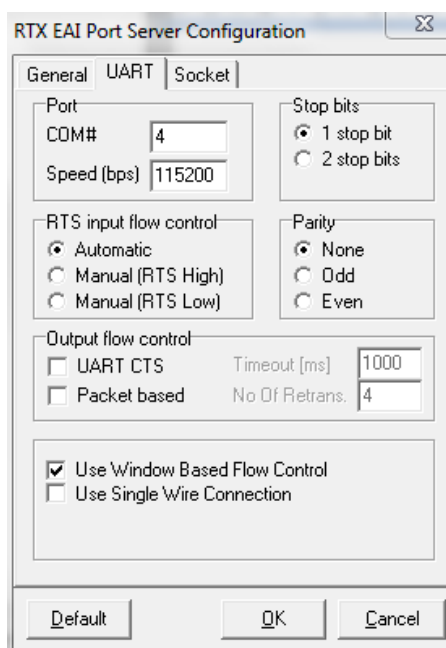
Configuració

Obrim l'aplicació RTX EAI Port Server i seleccionem el COM on estigui connectada la placa, per saber on està connectada anirem a "Panel de control\Todos los elementos de Panel de control\Dispositivos e impresoras", aquí veurem el nostre dispositiu que en el meu cas està en el COM4.



Il·lustració 20: Arduino Programming Port

Una vegada sabem en quin COM es troba, tornem a l'aplicació RTX EAI Port Server. Entrem a l'aplicació i fem botó dret en mig de la pantalla, ens sortiran diverses opcions, la que ara ens interessa és Setup (Ctrl+S), des d'aquí podem configurar el nostre COM, per això anem a la pestanya UART Transport Layer i en el Port COM# fiquem el que estem fent servir. A més, hem de marcar la opció "User Windows Based Flow Control". La configuració ha de quedar com la següent:



Il·lustració 21: RTX EAI Port Server Configuration

Compilació

Una vegada configurat, ja podrem compilar el codi, per això hem d'executar el fitxer ba.bat que es troba a la següent ruta:

C:\AmelieSdk\v1.6.0.58\Projects\Amelie\COLApps\Apps\SmartCitizenRTX4100\Build\RTX4100_WSAB

Per fer-ho es recomana cridar el programa des de la consola de comandaments de Windows, d'aquesta forma podem veure si hi ha cap error en la compilació:

```

C:\AmelieSdk\v1.6.0.58\Projects\Amelie\COLApps\Apps\SmartCitizenRTX4100\Build\RTX4100_WSAB>ba
Cleaning up
Tool check: "C:\Program Files\GNU Tools ARM Embedded\4.7 2013q3\bin\arm-none-eabi-gcc.exe"
4.7.4
Building filelist.mak
Processing: Projects\Amelie\COLApps\Apps\SmartCitizenRTX4100\Build\RTX4100_WSAB
Finished: Projects\Amelie\COLApps\Apps\SmartCitizenRTX4100\Build\RTX4100_WSAB
Time: 0:0
Updating C:\AmelieSdk\v1.6.0.58\Projects\Amelie\COLApps\Config\BuildInfo.inc

AppCommon.c
Main.c
AppSocket.c
AppLed.c
AppWiFi.c
AppSntp.c
C:\AmelieSdk\v1.6.0.58\Projects\Amelie\COLApps\Apps\SmartCitizenRTX4100\Main.c:74:21: warning: 'Pt_AppDnsd' define
C:\AmelieSdk\v1.6.0.58\Projects\Amelie\COLApps\Apps\SmartCitizenRTX4100\Main.c:75:21: warning: 'Pt_AppWebd' define
C:\AmelieSdk\v1.6.0.58\Projects\Amelie\COLApps\Apps\SmartCitizenRTX4100\Main.c:502:8: warning: 'PtWifi_setup_AP' d
onl
    
```

Il·lustració 22: Compilació COLApps

Si tot va bé ens crearà un fitxer FW que només ens faltaria carregar-lo a la placa.

```

Linking SmartCitizenRTX4100.hex
5068 bytes code
924 bytes data
485 bytes const
5553 bytes flash
Creating intel-hex file
Creating binary file
Creating FW update file
Time: 00:27:363
C:\AmelieSdk\v1.6.0.58\Projects\Amelie\COLApps\Apps\SmartCitizenRTX4100\Build\RTX4100_WSAB>
    
```

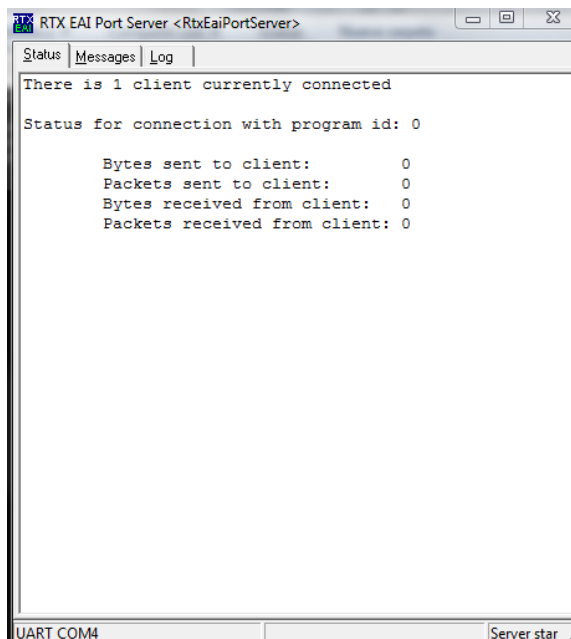
Il·lustració 23: Compilació COLApps Fitxer FWS

En el cas de que hi hagués cap error en la compilació podríem veure els warnings en la mateixa consola i modificar el nostre codi, encara que automàticament a la carpeta RTX4100_WSAB del mateix projecte ens crea un log amb els error (BuildLog.txt).

Carregar del codi

Ja només ens queda carregar el codi a la placa. Connectem la placa per l'USB del port Programming al nostre PC. Des del RTX podem veure si la connexió amb la placa és correcta o no.

En el nostre cas, podem veure que a baix de la finestra ens surt UART COM4 que és al que tenim connectada la placa i Server started, per tant ja podem carregar el fitxer *.fws que hem generat anteriorment. L'RTX EAI també ens indica si hi han clients i transmissions, en la següent captura podem veure que hi ha un client connectat però no hi ha transmissió:



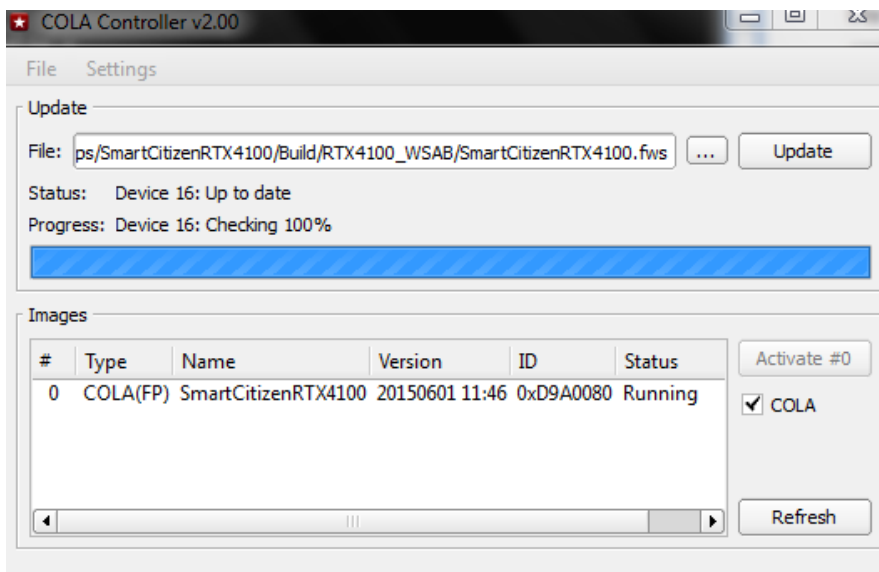
Il·lustració 24 RTX EAI Port Server

Per carregar el firmware hem d'obrir l'aplicació COLA Controller que s'ha instal·lat amb l'Amelie SDK. Per defecte ens dirà que no troba la placa:



Il·lustració 25 COLA Controller failed

Per això hem de prémer el botó S2 de la placa per entrar al mode 4. Ara, si està tot ben configurat, ens detectarà la nostra placa i podrem carregar el firmware creat. Busquem el fitxer *.fws i si tot és correcte li donem al botó Update, d'aquesta manera carregarem el programa per COLA a la placa, veurem que la barra de Progress s'anirà movent i una vegada estigui carregat ens donarà el OK:



Il·lustració 26 COLA Controller successfully

Premem el botó de RESET de la placa per reiniciar-la i ja tindrem el programa carregat.

2.5.4 Entorn RTX

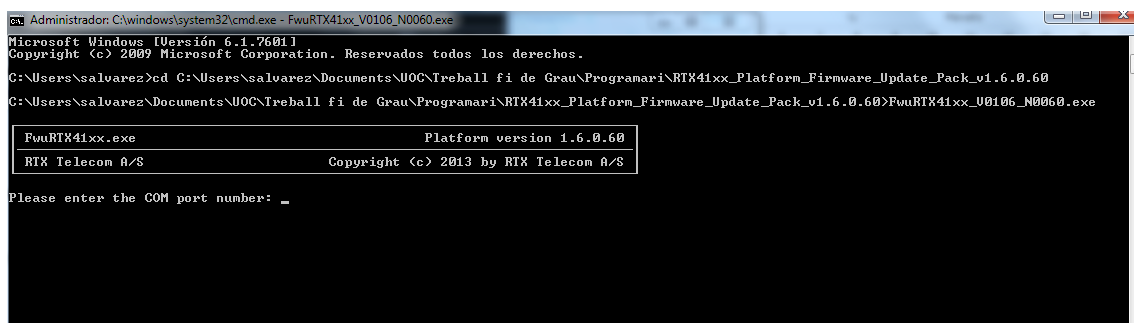
En aquest apartat explicarem com carregar el bootloader RTX, és a dir, el firmware del mòdul RTX4100.

Requisits

Primer de tot hem de descarregar-nos el Firmware per el RTX4100³⁰. Descomprimim la carpeta i veurem diversos fitxers però el fitxer que ens interessa és el FwuRTX41xx_V0106_N0060.exe.

Carregar el codi

Per executar-ho carreguem la consola de comandaments de Windows i executem el fitxer:

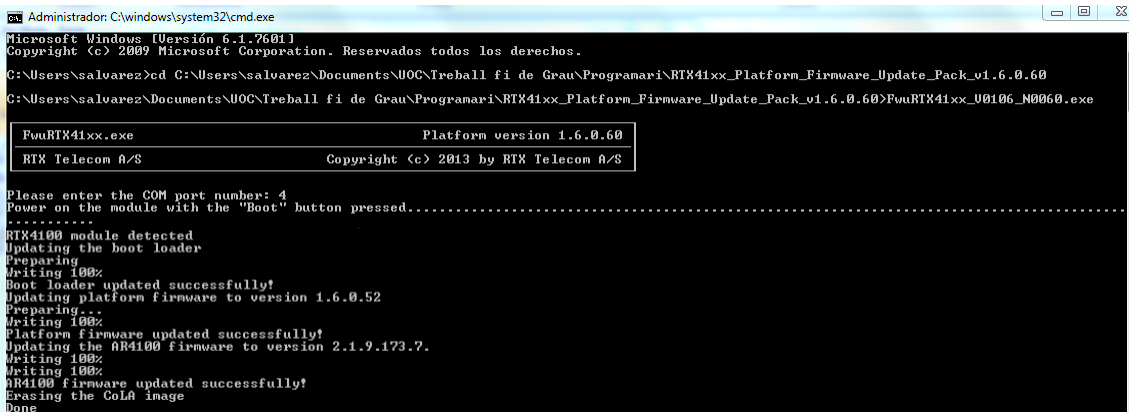


Il·lustració 27 Firmware RTX

³⁰ Firmware RTX4100

http://www.rtx.dk/Files/Billeder/RTX_T/RTX4100docs/RTX41xx_Platform_Firmware_Update_Pack_v1.6.0.60.zip

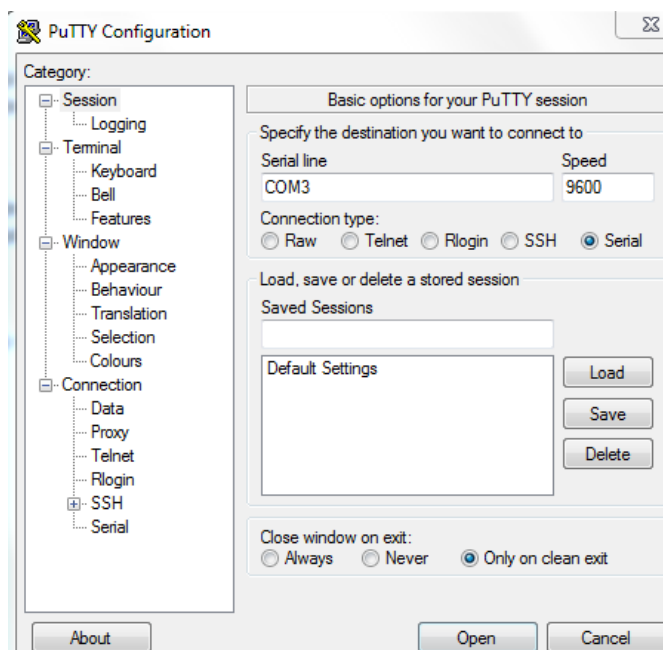
Ens demana el COM al que tenim connectat la placa, connectem la placa per el port USB Programming i li indiquem en quin COM està. Cliquem el botó S2 dos cops i entrarem en el mode 4, per això, el LED es ficarà de color blau, després de poca estona ja tindrem el bootloader fet.



Il·lustració 28 Firmware RTX carregat

Comprovació

Per comprovar si està ben carregat podem fer servir per exemple l'aplicació PuTTY³¹ que podem descarregar gratuïtament de la seva web <http://www.putty.org/>. Aquesta aplicació el que ens permet és fer un telnet i poder veure que està enviant o rebem. La configuració és molt simple, només hem d'indicar quin COM fem servir i la connexió ha de ser Serial:



Il·lustració 29 PuTTY configuration

³¹ Més informació a <http://www.putty.org/>

Una vegada li donem a Open se'ns obrirà el terminal i podrem veure que estem enviant i rebent dades:

```

COM3 - PuTTY
* Is your wiring correct?
* did you change the chipSelect pin to match your shield or module?
Temperature: 27.88 C, Humidity: 39.56 %
initialization failed. Things to check:
* is a card is inserted?
* Is your wiring correct?
* did you change the chipSelect pin to match your shield or module?
Temperature: 27.85 C, Humidity: 39.62 %
initialization failed. Things to check:
* is a card is inserted?
* Is your wiring correct?
* did you change the chipSelect pin to match your shield or module?
Temperature: 27.84 C, Humidity: 39.65 %
initialization failed. Things to check:
* is a card is inserted?
* Is your wiring correct?
* did you change the chipSelect pin to match your shield or module?
Temperature: 27.82 C, Humidity: 39.70 %
initialization failed. Things to check:
* is a card is inserted?
* Is your wiring correct?
* did you change the chipSelect pin to match your shield or module?
Temperature: 27.79 C, Humidity: 39.72 %
    
```

Il·lustració 30: PuTTY resposta

Per tant, ho tenim tot correcte, ja tornem a tenir el firmware original del RTX4100 i podem continuar fent proves amb la placa.

2.6 Access Point (AP)

Un punt d'accés (AP) és un dispositiu que interconnecta dispositius de comunicació sense fils per formar una xarxa sense fils. S'encarrega de ser una porta d'entrada a aquesta xarxa en un lloc específic i per una cobertura de radio determinada per qualsevol dispositiu que sol·liciti l'accés, sempre i quan, estigui configurat i tingui els permisos necessaris.

La placa que ens han facilitat, com hem comentant abans, porta un mòdul WiFly que fa aquesta funció.

Nosaltres haurem de desenvolupar un mode Soft AP per al RTX4100 basat en les notes d'aplicació del fabricant³². Aquest mòdul s'ha de desenvolupar com una extensió de la COLApp ja existent i ha de suportar les següents funcions:

1. Activació del mode mitjançant una comanda per SPI des del SCK.
2. Creació d'una xarxa oberta amb un SSID únic per kit consistent d'un denominador comú i un identificador únic.
3. Servidor mínim DHCP per assignar IP's als clients.

³² RTX Documentation - [RTX4100 Application Note SoftAP_AN8.pdf](#)

SPI

El bus SPI (Serial Peripheral Interface) és un estàndard de comunicacions que es fa servir principalment per la transferència d'informació en equips electrònics. Serveix per controlar quasi qualsevol dispositiu electrònic digital que accepti un flux de bits regulat per un rellotge (comunicació síncrona).

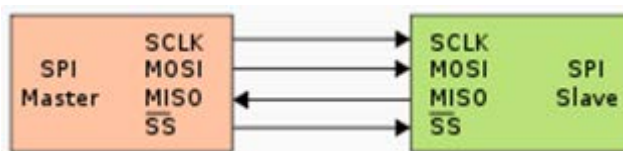
Inclou una línia de rellotge d'entrada, sortida i un pin de chip Select que connecta o desconnecta la operació del dispositiu amb el que es desitgi comunicar-se. D'aquesta forma permet multiplexar les línies de rellotge.

La seva avantatge principal és que minimitza el número de conductors, pins i la mida del circuit integrat per lo que redueix el cost de fabricar, muntar i provar la electrònica.

El hardware consisteix en senyals de rellotge, data in, data out i chip select per a cada circuit integrat que ha de ser controlat. En quasi qualsevol dispositiu digital pot ser controlat amb una combinació de senyals. Existeixen diversos tipus, uns llegeixen la dada quan el rellotge puja/ baixa i altres amb la pujada/baixada del franc del rellotge.

Per tant, SPI és un protocol síncrona i la sincronització i transmissió de dades es realitza per mitjà de quatre senyals:

- **SCLK (Clock):** És el pols que marca la sincronització. Amb cada pols d'aquest rellotge, es llegeix o s'envia un bit. També anomenat TAKT.
- **MOSI (Master Output Slave Input):** Sortida de dades del Master i entrada de dades al Slave. També anomenat SIMO.
- **MISO (Master Input Slave Output):** Sortida de dades del Slave i entrada al Master. També anomenat SOMI.
- **SS/Select:** Per seleccionar un Slave, o per que el Master li digui al Slave que s'activi. També anomenat SSTE.



Il·lustració 31 SPI

La cadena de bits és enviada de manera síncrona amb els polsos del rellotge, és a dir amb cada pols, el Master envia un bit. Per que comenci la transmissió, el Master baixa la senyal SSTE ó SS/Select a zero, amb això, el Slave s'activa i comença la transmissió, amb un pols de rellotge al mateix temps que el primer bit és llegit.

2.7 Servidor Web

Un servidor web és un programa que atén i respon les diferents peticions dels usuaris, proporcionant els recursos sol·licitats fent ús del protocol HTTP o HTTPS. El servidor web bàsic és el que acostuma a realitzar les següents accions en un bucle:

1. Espera peticions en el port TCP indicat.
2. Rep una petició.
3. Busca el recurs.
4. Envia el recurs utilitzant la mateixa connexió per la que va rebre la petició.
5. Retorna al segon punt.

El funcionament del servidor web que implementarem nosaltres té la següent forma:

1. Establiment d'una connexió Arduino amb el PC mitjançant el protocol SLIP.
2. Recepció de les peticions del client.
3. Tractament de les peticions del client.
4. Resposta a les peticions del client ja sigui fent ús d'un actuador o retornant el valor d'un sensor.

El nostre servidor web ha de permetre la visualització i configuració de paràmetres mitjançant l'intercanvi de dades amb el MCU de l'SCK i ha de suportar les següents funcionalitats:

1. Activació del servidor web amb una ordre per SPI des del SCK.
2. Suportar parcialment HTTP/1.1 limitant-se a les funcionalitats bàsiques de GET i POST.
3. Servir una única pàgina d'html que podrà contenir diferents formularis amb CSS incrustat.
4. Permetre l'enviament d'informació mitjançant formularis HTML de tornada al mòdul.
5. Complementar aquests requeriments analitzant la possibilitat de servir continguts estàtics més pesats com imatges i el suport de peticions XHR (Ajax).
6. Per tal de poder validar la interfície web pot desenvolupar-se una pàgina de proves que permet la visualització i configuració dels paràmetres.
 - a. Paràmetres de xarxa (SSID, Network security, Key).
 - b. Paràmetres de servidor (Temps entre updates i nombre de updates conjuntes).
 - c. Mostrar les dades de l'última lectura dels sensors.

2.7.1 Protocol HTTP

HTTP és un protocol de transferència d'hipertext (HyperText Transfer Protocol) que treballa a nivell d'aplicació utilitzat per a la transferència d'informació. HTTP defineix la sintaxi i semàntica que és utilitzada per a la comunicació entre els diversos elements de l'arquitectura del programari. Aquest protocol és usat per arquitectures client-servidor amb un esquema petició-resposta.

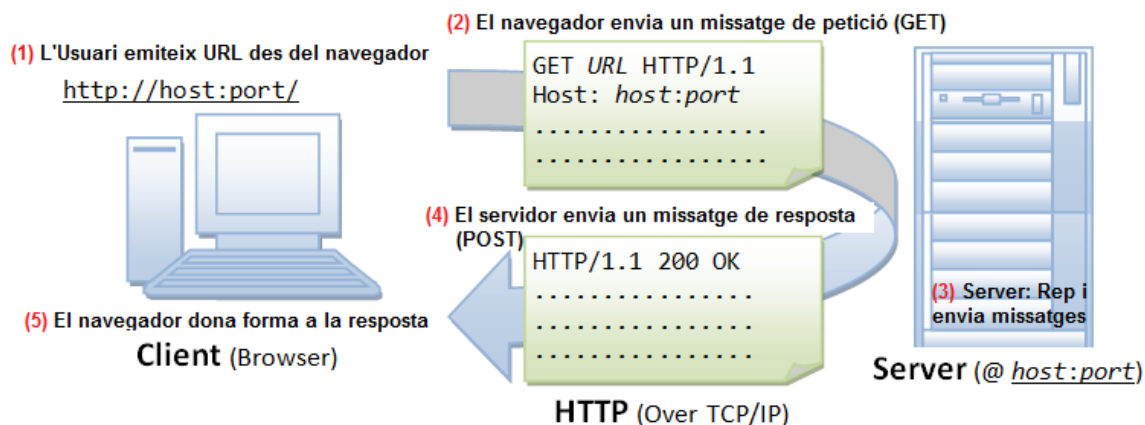
HTTP és un protocol sense estat, el que significa que no guarda cap informació de les seves connexions anteriors. Per donar una solució a aquest problema es van crear les galetes, les quals són informació que el servidor guarda en els clients.

Segueix el següent esquema: Un client realitza una petició que estarà formada per un mètode, una URI i una versió del protocol en la primera línia, a continuació s'envien les capçaleres de la petició i finalment si és necessari, una possible dada. El servidor contesta amb una línia d'estat que inclou la línia del protocol i un codi d'èxit o error, en les següents línies envia les capçaleres de la petició i finalment i si fos necessari algun tipus de dades.

Mètodes HTTP

Existeixen diferents mètodes però a nosaltres per realitzar el nostre projecte només ens interessen dos d'ells:

- **GET:** Demana una representació del recurs especificat. Per seguretat no hauria de ser usat per aplicacions que causin efectes ja que transmet informació a través de la URI afegint paràmetres a la URL.
- **POST:** S'utilitza per realitzar peticions en les que el servidor destí accepta el contingut de la petició com un nou subordinat del recurs demanat. També canvia l'estat del servidor.



Il·lustració 32 Protocol HTTP

HTML

L'estàndard HTML³³ (Hypertext Markup Language) fa referència al llenguatge per l'elaboració de pàgines web. Està definit per la W3C³⁴ (World Wide Web Consortium) que és la organització encarregada de l'estandardització de quasi totes les tecnologies lligades en l'entorn web i per tant, fa referència a l'elaboració de pàgines webs. Defineix una estructura bàsica i un codi HTML per la definició del contingut web.

Una de les seves avantatges és que fa servir les referències, això vol dir que si incrusten un element extern a la pàgina aquest no s'incrusta directament en el codi si no que fa referència a la ubicació de dit element mitjançant text. D'aquesta forma, la pàgina web només conté text i la tasca d'unir i visualitzar la pàgina final recau en el navegador web.

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Ejemplo1</title>
  </head>
  <body>
    <p>ejemplo1</p>
  </body>
```

Il·lustració 33: Exemple codi HTML

CSS

Cascading Style Sheets (CSS)³⁵ és un mecanisme simple que descriu com es mostrarà un document. Per tant, és un llenguatge que serveix per definir i crear la presentació d'un document estructurat escrit en HTML i XML separant el contingut de la presentació. Aquesta forma de descripció d'estils ofereix el control total sobre l'estil i el format dels documents, qualsevol canvi en l'estil marcat d'un element es veurà afectat a totes les pàgines vinculades a aquest CSS.

CSS funciona a base de regles i tenen dos parts: un selector i la declaració, per exemple `h1 {color: red;}`, `h1` és el selector i `{color: red;}` és la declaració.

Existeixen tres formes per donar un estil a un document: Utilitzant una fulla d'estil externa vinculada al document, aplicar l'estil directament sobre els elements i incrustar el codi a l'HTML que en aquest cas és el que ens interessa:

³³ Podeu trobar una guia bàsica a <https://www.uv.es/jac/guia/>

³⁴ Més informació a <http://www.w3c.es/>

³⁵ Més informació a http://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada

- Utilitzant l'element `<style>` en el interior del document al que se li vol donar estil. D'aquesta forma els estils seran reconeguts abans de que la pàgina es carregui per complet.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN">
<html>
  <head>
    <title>Fulla d'estil intern</title>
    <style type="text/css">
      body {
        padding-left: 11em;
        font-family: Georgia, "Times New Roman", serif;
        background-color: #d8da3d;
      }

      h1 { font-family: Helvetica, Geneva, Arial, sans-serif;
      }

    </style>
  </head>
  <body>
    //El que fem aquí serà amb estil body
    <h1>Aquí s'aplicarà l'estil de lletra per a h1</h1>
  </body>
</html>
```

Il·lustració 34: Exemple codi CSS intern

2.8 Protocol SLIP

El protocol SLIP (Serial Line Internet Protocol) és un estàndard de transmissió de datagrames IP per a línies sèrie. És un protocol amb una senzilla implementació i per això no proporciona ni direccionament, ni identificació de paquet, ni detecció/correcció d'errors ni mecanismes de compressió.

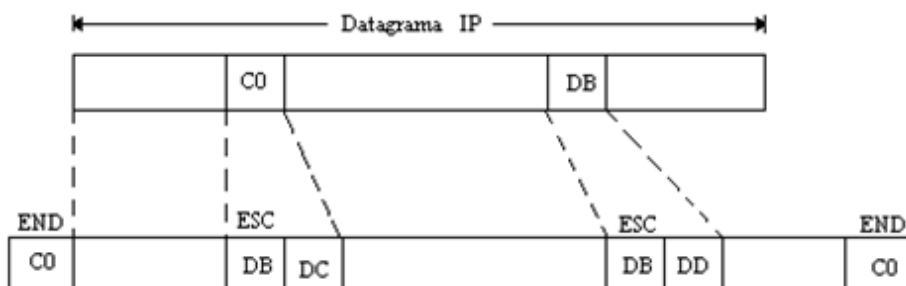
SLIP es pot fer ús en línies amb velocitats compreses entre 1200bps i 19.2kbps. Les configuracions més comuns són: host-host, host-router i router-router. Permet que es faci servir el Protocol d'Internet (IP) sobre un enllaç sèrie, això vol dir, que es pot fer servir el protocol TCP/IP.

Format SLIP

SLIP modifica cada datagrama IP afegint un caràcter especial C0 o "SLIP END" que permet distingir entre diferents datagrames. Per prevenir soroll de línia s'acostuma enviar un al principi també; de manera que es doni per acabada qualsevol tipus de connexió errònia anterior.

Si el caràcter C0 es presenta en el contingut del datagrama, s'utilitza la seqüència de doble byte DB, DC. El caràcter DB és el caràcter d'escapament de SLIP

Si en el contingut es presenta el caràcter d'escapament; es reemplaça per la seqüència DB, DD.



Il·lustració 35: Format SLIP

Tipus SLIP

Existeixen dos tipus de SLIP:

- **SLIP dinàmic:** El servidor proveïdor d'Internet identifica al ordinador proporcionant una direcció IP que és assignada dinàmicament de entre un conjunt de direccions i només dura el temps de la connexió.
- **SLIP estàtic:** El servidor proveïdor d'Internet assigna una direcció fixa a l'ordinador per al seu ús en totes les sessions.

A causa del seu senzill disseny, el protocol presenta diverses deficiències encara que també els seus avantatges:

És fàcil d'implementar, addiciona pocs bytes d'overhead i per això hi ha moltes aplicacions que el fan servir encara que no és un protocol estàndard. Només reconeix la IP però s'ha de conèixer la IP de cada extrem. No efectua detecció ni correcció d'errors per lo que és poc fiable.

2.9 WiFi

Definirem que és una xarxa sense fils i a rel de la definició anirem entrant en les seves característiques com estàndards, protocols i tipus de seguretat. Abans però, mirarem si s'ha de legislació vigent

2.9.1 Legislació vigent

En Espanya, la Comissió del Mercat de Telecomunicacions, Organisme Públic regulador independent dels mercats nacionals de comunicacions electròniques i de serveis audiovisuals, fou creada per el Real Decret-Llei 6/1996, de 7 de juny, de Liberalització de les Telecomunicacions, a través de la qual s'ampliaren i perfilaren les funcions que van ser inicialment atribuïdes a la Comissió del Mercat de les Telecomunicacions i es va definir una nova composició del Consell.

Ordre IET/787/2013, de 25 d'abril, per la que s'aprova el quadre nacional d'atribució de freqüències (CNAF 2013). L'Article 5 del Reglament de desenvolupament de la Llei 32/2003, de 3 de novembre, General de Telecomunicacions, en relatiu a l'ús del domini públic radioelèctric, aprovat per el Real Decret 863/2008, de 23 de maig, estableix que mitjançant Orde del Ministeri de Industria, Turisme i Comerç, s'aprova el Quadre Nacional d'Atribució de Freqüències per als diferents tipus de serveis de radiocomunicació, definits en el Reglament de Radiocomunicacions de la Unió Internacional de Telecomunicacions. Defineixen l'atribució de bandes de freqüència als seus respectius serveis amb les característiques tècniques que siguin necessàries.

Circular 1/2010, de 15 de juny de 2010, de la Comissió del Mercat de les Telecomunicacions, per la que es regulen les condicions d'explotació de xarxes i la prestació de serveis de comunicacions electròniques per la Administració Pública.

Real Decret 1720/2007, de 21 de desembre, per el que s'aprova el Reglament de desenvolupament de la Llei Orgànica 15/1999, de 13 de desembre, de protecció de dades de caràcter personal («BOE» 19 gener 2008). En la que s'estableix la protecció de les persones físiques en lo que respecta al tractament de dades personals i circulació d'aquestes dades.

Real Decret 1066/2001, de 28 de septiembre, per la que s'aprova el Reglament que estableix condicions de protecció del domini públic radioelèctric, restriccions a les emissions radioelèctriques i mesures de protecció sanitària enfront a emissions radioelèctriques. En el que s'especifiquen les restriccions bàsiques de freqüència i els nivells de referència que s'han de complir per garantir-les com els nivell d'exposició.

2.9.2 Característiques WiFi

Abans de definir les característiques de la WiFi mirem de definir que és una xarxa sense fils.

Una xarxa sense fils és una xarxa en la que dos o més terminals es poden comunicar sense la necessitat d'una connexió per cable. Un usuari pot seguir estant connectat mentre es desplaça dintre d'una determinada àrea geogràfica.

Es basen en un enllaç que utilitza ones electromagnètiques, segons la seva freqüència de transmissió, l'abast i la velocitat es poden fer servir tecnologies diferents.

WiFi és un mecanisme de connexió de dispositius electrònics de forma sense fils. És una marca de WiFi Alliance³⁶ que adopta, prova i certifica que els equips compleixen els estàndards 802.11 relacionats a xarxes sense fils d'àrea local.

2.9.2.1 Categories de xarxa sense fils

Existeixen quatre tipus de categories generals de xarxa sense fils:

- **Xarxes sense fils d'àrea personal (WPAN: Wireless Personal Area Network):**
Cobreixen l'abast d'algunes desenes de metres. Generalment es fa servir per connectar dispositius a un ordinador sense connexió per cable.
La tecnologia principal WPAN és Bluetooth (IEEE 802.15.1) que ofereix una velocitat màxima d'1 Mbps amb un abast màxim d'uns trenta metres.
La tecnologia Zigbee (IEEE 802.15.4) funciona en la banda de freqüència de 2,4 GHz i pot arribar a una velocitat de transferència de fins 250 Kbps amb un abast màxim d'uns 100 metres.
- **Xarxes sense fils de xarxa local (WLAN: Wireless Local Area Network):**
Cobreixen uns pocs centenars de metres, es basen en l'estàndard IEEE 802.11 (WiFi) i les seves variants. Ofereixen una velocitat màxima de 54 Mbps en una distància de varis cents de metres.
HiperLAN 2 (High Performance Radio LAN 2.0), permet als usuaris arribar a una velocitat màxima de 54 Mbps en una xarxa aproximadament de cent metres, i transmet dintre del rang de freqüències de 5150 y 5300 MHz.
- **Xarxes sense fils de xarxa metropolitana (WNAM: Wireless Metropolitan Area Network):** Es basen en l'estàndard IEEE 802.16. Els bucles locals sense fils ofereixen una velocitat total efectiva d'1 a 10 Mbps, amb un abast de 4 a 10 kilòmetres.

³⁶ WiFi Alliance: www.wi-fi.org

- **Xarxes sense fils de xarxa extensa (WWAN: Wireless Wide Area Network):**
Tenen l'abast més ampli de totes las xarxes sense fils. La família d'estàndards IEEE 802.20 o UMTS són els més representatius. Les tecnologies principals son:
 - GSM (Global System for Mobile Communication).
 - GPRS (General Packet Radio Service).
 - UMTS (Universal Mobile Telecommunication System).

2.9.2.2 Estàndards WiFi

Com hem dit abans, les xarxes WiFi permeten la connectivitat d'equips i dispositius mitjançant ones de radi.

Al llarg del temps s'han definit diversos estàndards amb l'objectiu de millorar la connectivitat i el seu rendiment. Els estàndards més utilitzats actualment són els següents:

Estàndard	Nom	Descripció
802.11a	WiFi5	Rang de freqüència: 5 GHz. Velocitat màxima de connexió: 54 Mbps.
802.11b	WiFi	Rang de freqüència: 2,4 GHz Velocitat màxima de connexió: 11 Mbps. Abast de fins 300 m en un espai obert.
802.11d	Internacionalització	Permet l'ús internacional de les xarxes 802.11 locals i operar en diferents rangs de freqüència segons el dispositiu.
802.11e	Millora de la qualitat del servei	Defineix els requisits de diferents paquets en quant a l'ampla de banda i al retard de transmissió
802.11f	Itinerància	Fa servir el protocol IAPP que permet a un usuari itinerant canviar-se clarament d'un punt d'accés a un altre mentre està en moviment
802.11g		Rang de freqüència: 2,4 GHz Velocitat màxima de connexió: 54 Mbps.
802.11n		Rang de freqüència: 2,4 GHz – 5 GHz Velocitat màxima de connexió: 600 Mb.

Taula 3: Estàndards WiFi

2.9.3 Tipus de seguretat

La seguretat és el punt feble de les xarxes sense fils, ja que el senyal es propaga per l'aire en totes les direccions i pot ser captada. Això fa que les xarxes sense fils siguin vulnerables a ser interceptades.

Existeixen diverses alternatives per garantir la seguretat d'aquestes xarxes. Les més comunes són la utilització de protocols de xifrat de dades per als estàndards WiFi com el WEP, el WPA o el WPA2 que s'encarreguen de codificar la informació transmesa per protegir la seva confidencialitat, proporcionats pels propis dispositius sense fil:

- **WEP (Wired Equivalency Privacy):** Basat en el mètode de criptografia RC4 que utilitza criptografia de 64 bits o 128 bits. Les dues utilitzen un vector de inicialització de 24 bits. No obstant això, la clau secreta té una extensió de 40 bits o de 104 bits.
- **WPA (WiFi Protected Access):** Posseeix el protocol TKIP (Temporal Key Integrity Protocol) amb un vector de 48 bits i una criptografia de 128 bits. Amb la utilització del TKIP la clau és alterada en cada paquet i sincronitzada entre el client i l'Access Point, també fa ús d'autenticació de l'usuari per un servidor central.
- **WPA2 (WiFi Protected Access 2):** Millora de WPA que utilitza l'algoritme d'encryptació AES (Advanced Encryption Standard).
- **WPA PSK (WiFi Protected Access Pre-Share Keyda):** Hi ha una clau compartida per tots els integrants de la xarxa prèviament a la comunicació. La fortalesa de la seguretat resideix en el nivell de complexitat d'aquesta clau.
- **MAC (Mesurava Access Control):** Cada placa de xarxa té el seu propi i únic número de direcció MAC. D'aquesta manera, és possible limitar l'accés a una xarxa només a les plaques els números MAC estiguin especificats en una llista d'accés.
- **NAC (Network Access Control):** Permeten controlar qui es pot connectar i també des de quins dispositius es pot realitzar aquest accés.
- **VPN (Virtual Private Network):** Permet una extensió de la xarxa local sobre una xarxa pública. Pot controlar qui i quina màquina es connecta i el nivell d'accés. Per assegurar la integritat de la informació utilitza funcions de hash: Message Digest (MD2 i MD5) i Secure Hash Algorithm (SHA). Per assegurar la confidencialitat fa ús d'algorismes de xifrat com DES (Data Encryption Standard), Triple DES (3DES) i EAS (Advanced Encryption Standard).
- **Servidor AAA (Authentication Authorization Accounting):** En el cas d'organitzacions es poden utilitzar servidors d'autenticació centralitzats, com Remote Authentication Dial-In User Service (RADIUS), que manté una llista d'usuaris i claus d'accés, que serà sol·licitada a l'accedir a la xarxa.

Capítol 3: Desenvolupament de la pràctica

3.1 Introducció

Arribats a aquest punt ja hem complert els objectius generals del projecte. Hem vist quines son les característiques de l'SCK i d'Arduino, quins components i protocols podem fer servir i com es fan les crides.

En aquest capítol ens centrarem en els objectius específics. S'implementarà el mode AP que permetrà connectar els diferents equips que disposin de WiFi amb el nostre mòdul RTX4100 i configurarem un servidor web que permeti interaccionar la placa amb el mòdul RTX4100, en aquesta web visualitzarem diferents paràmetres com ara la connectivitat i la temperatura.

Els diferents codis fets servir s'entregaran com a fitxers adjunts a la finalització d'aquest projecte.

3.2 Creació servidor web amb Arduino

Per veure de forma senzilla com funciona un servidor web el que farem serà crear-ne un amb Arduino. Primer de tot hem de connectar la nostra placa al nostre equip i assignar-li una IP lliure, a continuació escriurem el codi HTML de la nostra web en l'sketch. Quan un dispositiu es vulgui connectar ens enviarà una petició HTTP que la placa reconeixerà i enviarà el codi HTML per mostrar-nos la web creada.

Per aquesta pràctica necessitarem una placa Arduino, una resistència i un LED. Si no disposem del material ho podem fer de forma virtual fent servir el programa que ens facilita l'empresa Fritzing³⁷.

De forma resumida el que farà el nostre programa serà escoltar si hi ha peticions HTTP, si hi ha la llegirà que serà engegar o apagar el LED i enviarà la resposta (engegar o apagar el LED).

³⁷ <http://fritzing.org>

A continuació mostrem el diagrama de flux:

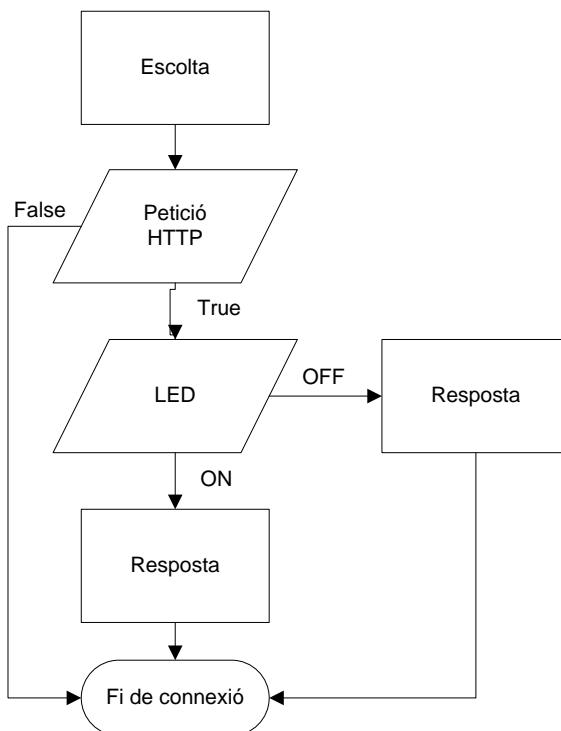


Diagrama 1: Diagrama de flux Web Server Arduino

Després de veure com seria el diagrama general, entrem a comentar el seu codi:

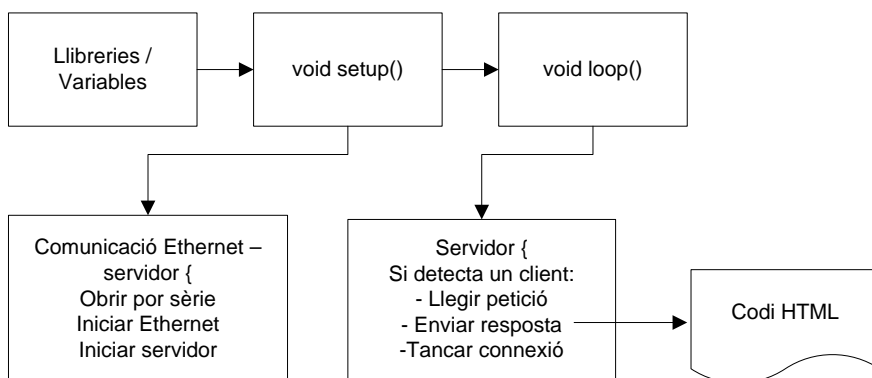


Diagrama 2: Web Server Arduino Codi

Per poder fer servir diverses funcions hem d'afegir al començament del nostre sketch les llibreries SPI i Ethernet. També hem d'assignar una IP a la placa i obrir el port 80 per poder rebre les peticions HTTP. A continuació definim les variables i realitzem la comunicació entre el servidor i l'Ethernet. Finalment creem el servidor web, el que farà serà que quan detecti que el client envii una petició HTTP aquest el llegirà caràcter per caràcter i d'aquesta forma podrem fer una recerca i veure l'estat del LED. Quan detecti una línia en blanc serà que ha acabat i enviarà la resposta.

El resultat serà semblant al següent:

Web Server Arduino



LED=OFF

Il·lustració 36: Exemple Web Server Arduino

Un cop inserit el codi de la web HTML, muntem el circuit i només quedarà carregar el programa a la nostra placa Arduino (veure codi sencer en Annex 5: Codi WebServer Arduino).

Fritzing

Per muntar el circuit farem servir Fritzing. És una iniciativa de hardware lliure que ens ofereix una eina de software i diversos documents i exemples per fer circuits amb Arduino.

Per instal·lar-lo només hem d'anar a la web <http://fritzing.org> a l'apartat Download i escollir el nostre sistema operatiu. Un cop descarregat l'instal·lem sense cap complicació, és descomprimeix i s'executa el fitxer fritzing.exe.

L'aplicació té 4 seccions: Menú, col·lecció d'elements, opcions d'element i zona de treball:

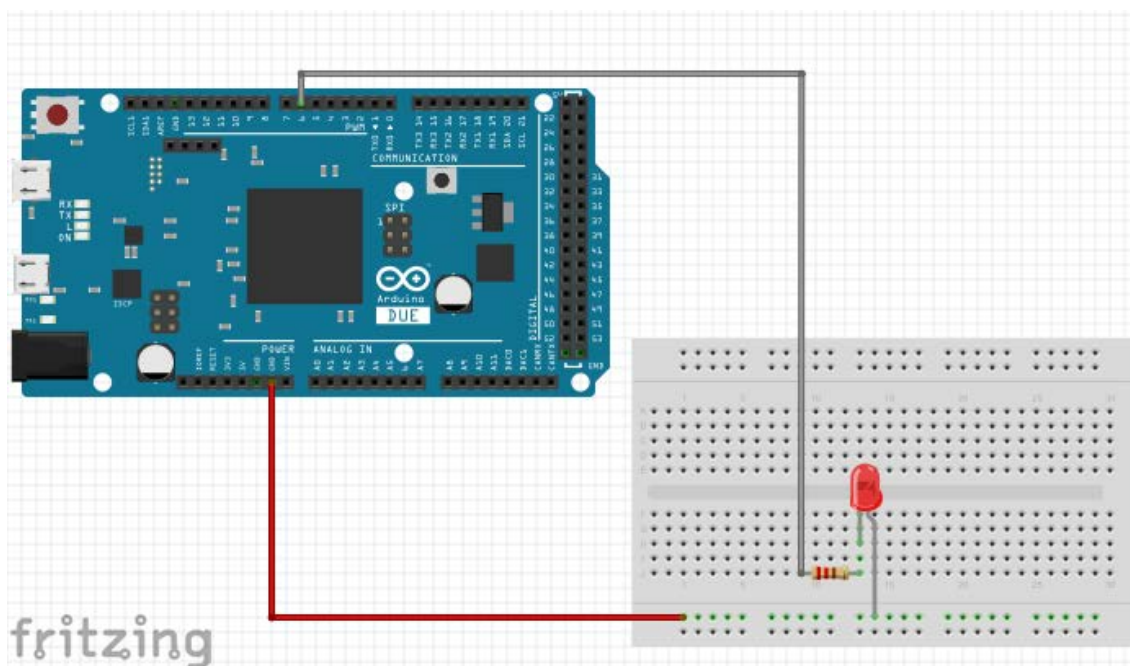


Il·lustració 37: Fritzing

Per muntar el nostre circuit anirem a menú i seleccionarem la vista del projecte Protoboard. En el menú de col·lecció d'elements veiem tots els elements que podem fer servir, en el nostre cas buscarem la placa Arduino DUE i com a components un LED i una resistència, una vegada localitzat només haurem d'arrossegar-ho fins a la zona de treball i connectar-ho. En aquest cas no fa falta modificar cap valor en les opcions d'elements. Hem de recordar que en l'sketch hem dit que el LED estarà connectat al PIN 6 i que la resistència s'ha de connectar en sèrie amb el LED, per això al fer la connexió del LED a la placa s'ha de connectar al PIN 6.

Una vegada tinguem el disseny carreguem el codi a la vista <>Code. Un cop el tenim li donem a la opció Update per pujar el programa i fer funcionar el circuit. Aquesta aplicació també disposa de l'apartat Serial Mode que és una consola amb el que podrem monitoritzar el seu funcionament.

El resultat ha de ser semblant al següent:



Il·lustració 38: Muntatge amb Fritzing

3.3 AP Mode

En aquest apartat ja ens centrem en el que se'ns demana en el projecte. Crearem un Access Point que faci servir COLApp i suporti de forma resumida les següents funcions:

1. Activació del mode mitjançant una comanda per SPI des del SCK.
2. Creació d'una xarxa oberta amb un SSID únic.
3. Servidor DHCP per assignar IP's als clients.

Abans però, hem de tenir configurat el nostre ordinador per poder crear, compilar i executar el programa (veure apartat 2.5.3 Entorn Amelie SDK).

Una vegada explicat com funcionarà la compilació i càrrega del programa passem a veure quines són les parts principals del nostre AP:

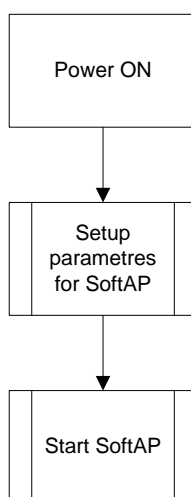


Diagrama 3: Diagrama de flux AP Mode

Com es pot veure el codi consta de diverses parts. De forma resumida podem dir que el mòdul softAP serà el que farà d'AP, permetrà als diferents dispositius que tinguin WiFi connectar-se al mòdul. A continuació mirem els diagrames una mica més a fons cada mòdul per saber exactament que fa cadascun.

3.3.1 SoftAP

El nostre programa ha de fer servir l'estàndard de comunicacions SPI per poder donar servei AP, per això caldrà afegir un mode AP. Si ens fixem en el manual de "RTX41xx: User Guide UG3 Application Development" veurem que en el punt 5.6 "The SoftAPTcpServer Application" se'ns indica que l'aplicació SoftAP Tcp Server implementa les funcionalitats del SoftAP, mentres que al mateix temps fa la integració del TCP Server. Nosaltres aprofitarem aquest exemple per guiar-nos i fer la implementació de l'SPI.

El diagrama de flux ha de ser el següent:

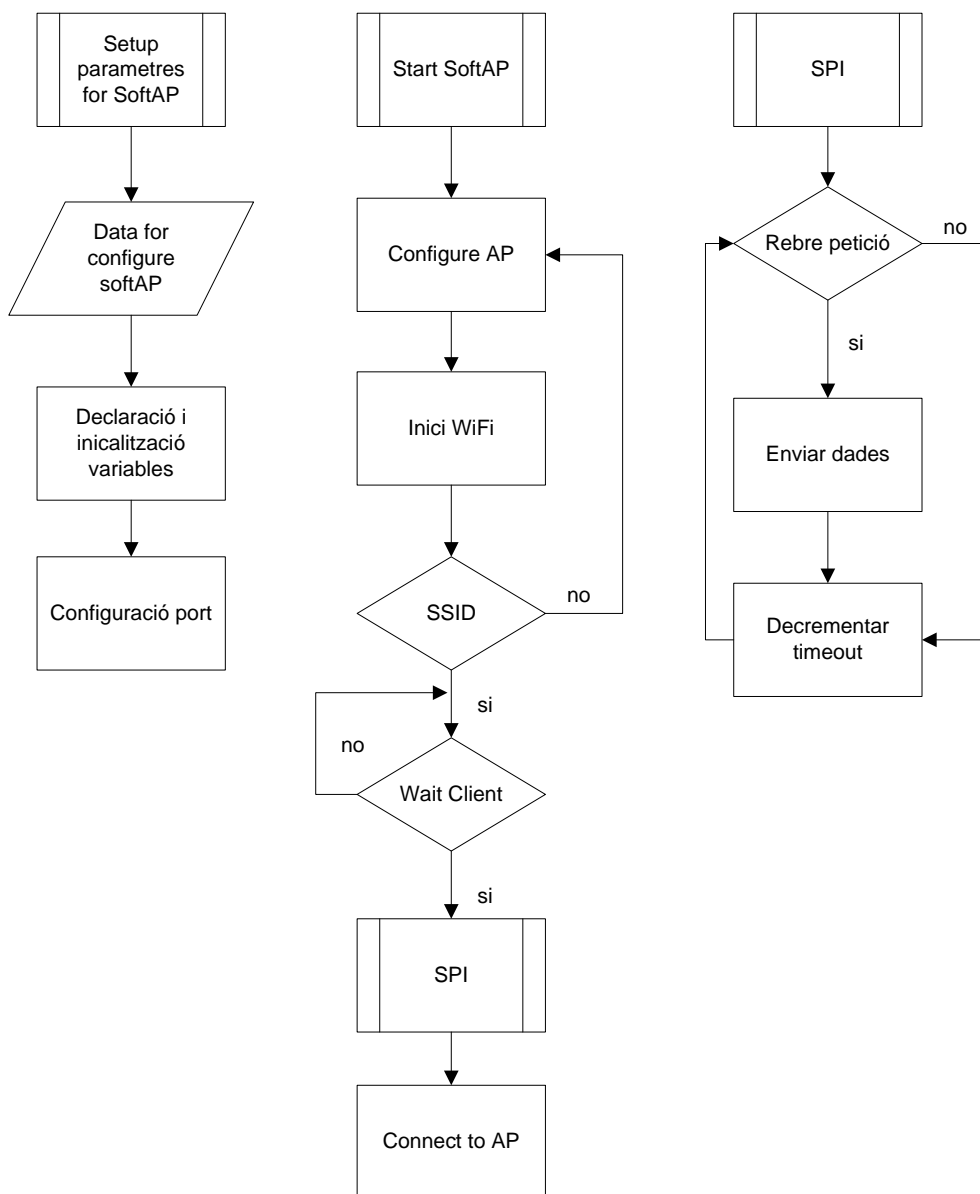


Diagrama 4: Diagrama de flux SotfAP

El diagrama de flux del softAP és una mica més complex ja que porta més mòduls. Es configuraran els paràmetres de l'AP i s'iniciarà, activem la WiFi i comprovarem si tenim activat l'SSID que serà el que ens permetrà connectar el nostre dispositiu, si no ho està s'iniciarà. Esperarem la petició del client, aquesta sol·licitud es farà a través de l'SPI, i donarem accés a l'AP. Una vegada finalitzada la connexió tancarem el socket.

Veiem que abans d'iniciar el softAP hem de configurar els paràmetres que tindrà, aquí és on trobarem totes les variables, definicions i inicialitzacions. Una vegada configurat iniciarem el mode softAP.

3.3.2 SPI

Per mitjà del mòdul SPI farem l'enviament de les dades. El nostre AP ha de connectar-se per mitjà d'una comanda SPI. Veiem el seu diagrama de flux:

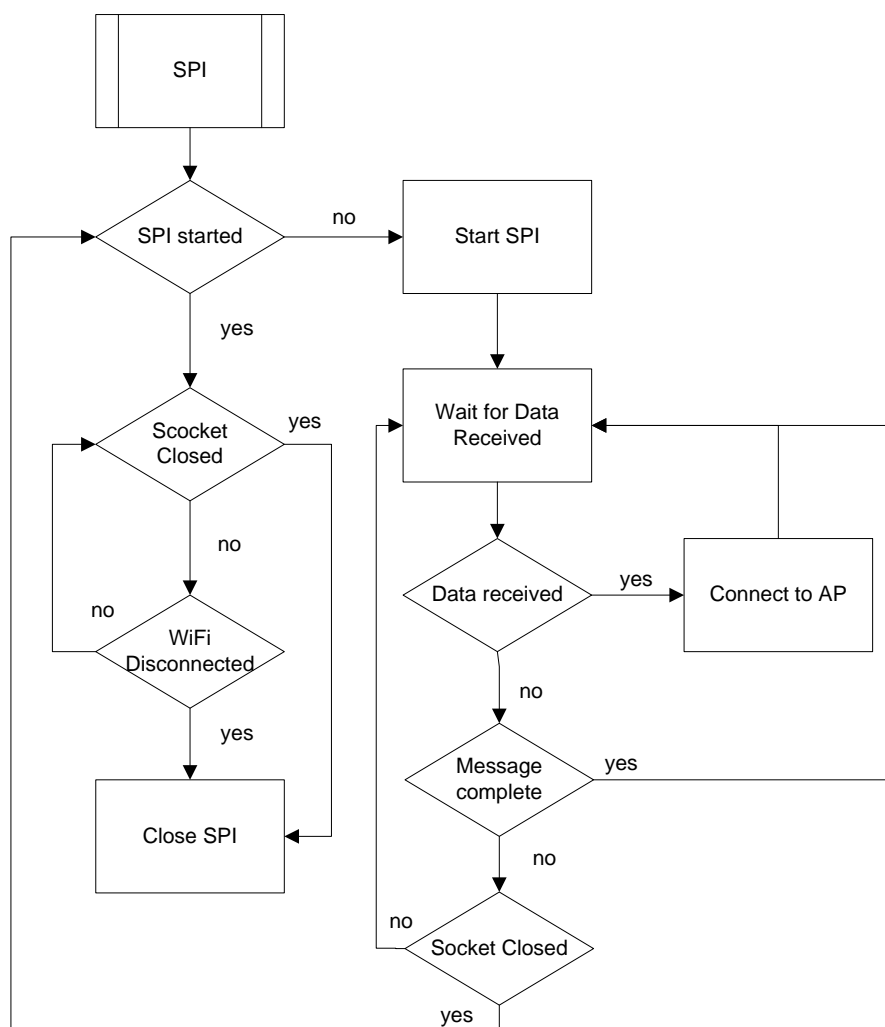


Diagrama 5: Diagrama de flux SPI_AP

Tal com hem dit el mòdul farà la connexió de l'AP per mitjà del protocol SPI. Iniciarem el mòdul i esperarem a rebre la petició de connexió, una vegada rebuda la redirigirem a l'AP i esperarem si hi ha cap petició més. Una vegada finalitzada la connexió del client enviem la petició de desconnexió. En quant es confirmi es tancarà la connexió per mitjà de la comanda de l'SPI.

3.3.3 Lliberies

Abans d'entrar en detall en el nostre codi, expliquem les diferents lliberies a les que el mode AP farà crides:

AppWifi.c

Dintre d'aquest codi el que trobem és la configuració de la estructura que s'anomenarà `AppWifiDataType`, aquesta estructura consta de diversos elements com la informació de l'access point, l'índex de la IP utilitzada o de la última connectada, la configuració dels paràmetres del mòdul WiFi, la configuració de la IP tant v4 com v6, etc. També trobarem diversos flags de modificació com WiFi connectada, AP iniciat, connexió iniciada, DHCP autoritzat i iniciar el mode SoftAP.

Si mirem per sobre la implementació ens trobem amb el primer Protothread anomenat `PtAppWifiStatus`, aquí dintre trobem la variable `AppWifiData`, creada anteriorment, si està connectat farà les comprovacions del `usestaticip` i del `dhcpenabled`, si està connectat enviarà la configuració de la IP (`sendApIpv4ConfigReq`) i finalment enviarà el `RSS_SUCCES`, o en cas contrari, es desconnectarà.

A continuació trobem `AppWifiInit` on tenim la configuració SoftAP per defecte, els paràmetres de la WiFi i l'estat del protothread.

Després hi ha diverses sentències amb les configuracions del Wifi com per exemple el `AppWifiGetSsid` que ens retornarà el nom de l'SSID, dintre d'aquest podem veure que `AppWifiSetApInfo` ens retornarà diversos paràmetres com l'índex, `SsidLength`, `Ssid`, `SecurityType`, `Key`.

A partir de les línies 612 ens trobem la part interessant per el softAP. Aquí trobem que `AppWifiApGetSoftApSsid` ens retorna el SSID i la configuració del softAP (`SecurityType`, `keylength`, `key`, `channel`, `country code`, `beacon`), a part trobarem la configuració de la IP.

Finalment trobem diverses funcions per treure (GET) o modificar (SET) característiques del softAP com el canal, el country o el password.

A continuació mostrem el diagrama de funcions de l'`AppWifi` i les lliberies que crida:

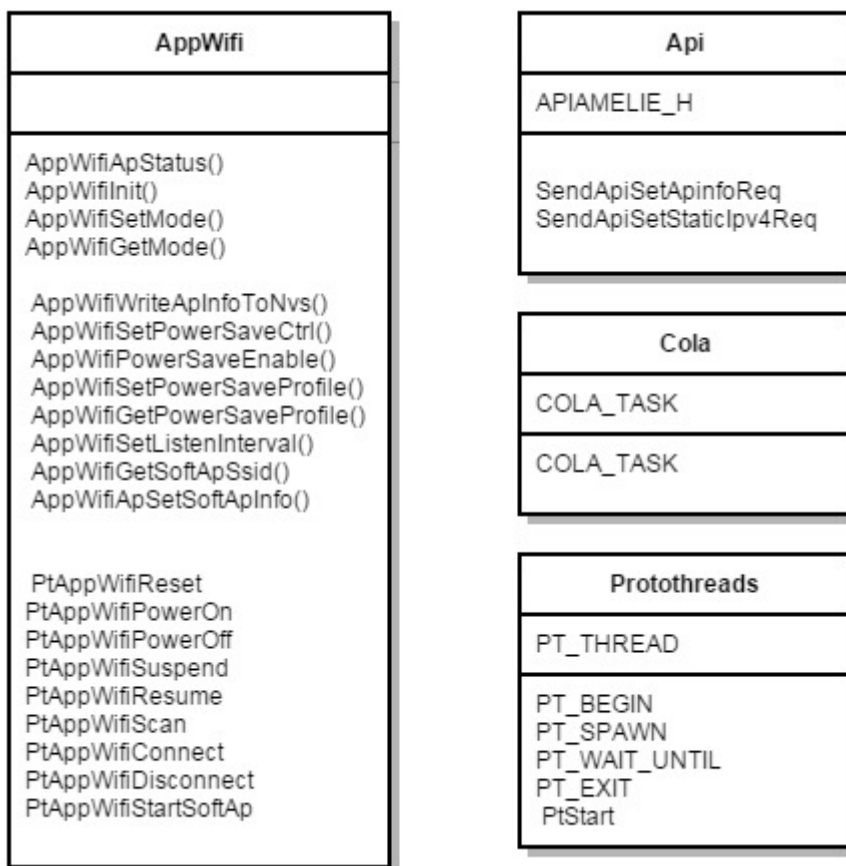


Diagrama 6: Diagrama de funcions AppWifi

Com podem veure dintre de l'AppWifi trobem diverses funcions i mètodes però també tenim referències a tres llibreries que es troben dintre de l'Amelie SDK. Si mirem dintre del codi podem veure a que fan referència:

- **API:** Dintre del ApiAmelie.h estem cridant, en concret, a dos funcions MPS per l'enviament del mails:

Funció	Explicació
SendApiEnableStepUpPortReq	API_ENABLE_STEP_UP_PORT_REQ = 0x6F21
SendApiSetStaticIpv4Req	API_SET_STATIC_IPV4_REQ = 0x6F34

- **COLA:** Fem servir COLA que recordem que facilita als programadors d'aplicacions crear les seves aplicacions sense deixar de tenir totes les característiques del firmware subjacent per mitjà de les APIs
- **PROTOTHREAD:** Aquí trobem els diferents protothreads que hem estudiat anteriorment.

DrvSpiSlave

La connexió amb les dades s'ha de fer per el protocol SPI, la seva estructura és la següent:



Diagrama 7: Diagrama de funcions DrvSpiSlave

Funció	Explicació
DrvSpiSlaveRxGetSize	Calcula i retorna el número de bytes del buffer de RX
SpiSlaveRxIntHandler	Manejador d'interrupcions MOSI
SpiSlaveTxIntHandler	Manejador d'interrupcions MISO
SpiSlaveTxIntHandler	Manejador d'interrupcions MISO
PtDrvSpiSlaveInit	Protothread: Inicialització SPI
PtDrvSpiSlaveClose	Protothread: Tanca SPI
DrvSpiSlaveInit	Inicialització
DrvSpiSlaveRx	Llegeix dades del buffer RX
DrvSpiSlaveRxFlush	Buida el buffer RX
DrvSpiSlaveTxStart	Inicialització de la transmissió del buffer

Codi de SoftAP

Un cop tenim clar el funcionament expliquem com serà el nostre SoftAP.

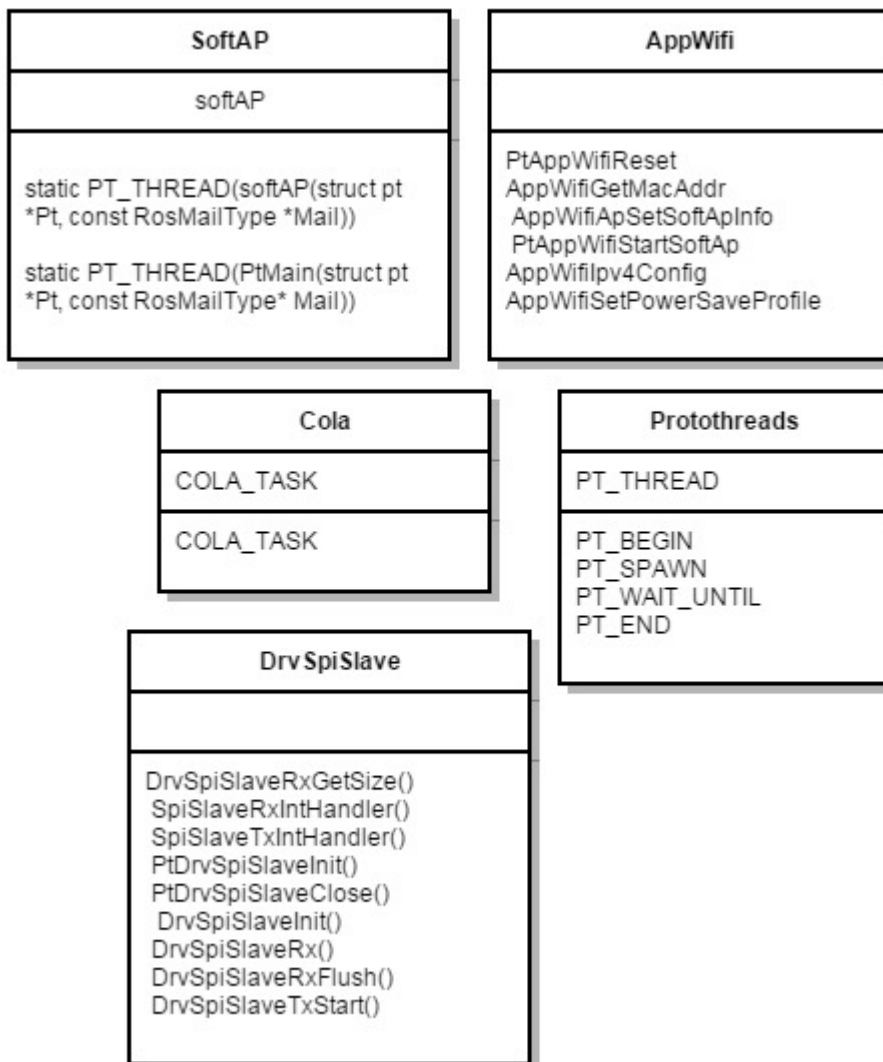


Diagrama 8: Diagrama de funcions SoftAP

Hem de tenir en compte que primer de tot hem de fer les definicions pertinents, hem de configurar el tipus de seguretat, la password, el canal, la IP, la màscara, etc.

Per assignar-li una IP lliure primer buscarem quina tenim configurada en el nostre Pc, és molt senzill, només hem d'obrir una finestra de comandament de Windows i ficar ipconfig, aquesta comanda ens retornarà la IP els valors de la nostra connexió:

```

C:\Users\lkjlk>ipconfig

Configuración IP de Windows

Adaptador de LAN inalámbrica Conexión de red inalámbrica:

    Sufijo DNS específico para la conexión. . . : Home
    Vínculo: dirección IPv6 local. . . . . : fe80::f4ad:acbc:78b5:e9d3%11
    Dirección IPv4. . . . . : 192.168.1.134
    Máscara de subred . . . . . : 255.255.255.0
    Puerta de enlace predeterminada . . . . . : 192.168.1.1

```

Il·lustració 39: Ipconfig

Com podem veure la nostra IP és 192.168.1.134, ara el que farem serà buscar una lliure, per això farem un ping a una adreça i si no rebem resposta és que està lliure:

```

C:\Users\lkjlk>ping 192.168.1.135

Haciendo ping a 192.168.1.135 con 32 bytes de datos:
Respuesta desde 192.168.1.134: Host de destino inaccesible.
Respuesta desde 192.168.1.134: Host de destino inaccesible.
Respuesta desde 192.168.1.134: Host de destino inaccesible.
Respuesta desde 192.168.1.134: Host de destino inaccesible.

Estadísticas de ping para 192.168.1.135:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (<0% perdidos>),

C:\Users\lkjlk>

```

Il·lustració 40: IP lliure

Per tant, podem veure la .135 està lliure i es pot assignar.

En el nostre cas no necessitem que l'AP tingui seguretat, per tant haurem d'iniciar la seguretat com AWST_NONE, el channel que farem servir serà el 2437 i 10 minuts d'inactivitat.

La primera part del codi anirà després del codi del TCP connection, aquí hem de iniciar el mode softAP creant un protothread i fent la crida al fill. Creem el SSID que serà la suma de la MASK i la MAC, configurem els valors del softAP que hem comentat abans i iniciem softAP. En aquest moment ja esperem la petició del client. Un cop rebuda connectem amb l'AP.

Veiem més a prop com es fan les crides:

Dintre del softAP iniciarem el protothread com fill i l'iniciarem, també cridarem al PtAppWifiReset que es troba dintre de la llibreria AppWiFi, aquesta acció el que fa es reiniciar el mòdul de la WiFi. A continuació creem l'SSID que com hem dit és la suma

de la MASK i de la MAC, funció que torna a cridar a la llibreria AppWiFi per poder construir-ho. Iniciem el mòdul fent la crida a l'AppWiFi que el que fa és fer les comprovacions de la WiFi, llegir la MAC i fer la configuració, tot això es fa per COLApp. Un cop iniciat s'espera la connexió del client.

A part, dintre de l'apartat Start TCP connection haurem d'afegir el nostre mòdul:

```
else if (strcmp(argv[0], "softap") == 0) {  
PT_SPAWN(Pt, &childPt, softAP(&childPt, Mail));  
}
```

I per últim dintre del SPI_Communication hem de ficar un nou case amb el SoftAP:

```
case 16: { // Wifi mode SoftAP  
PT_SPAWN(Pt, &childPt, softAP(&childPt, Mail));  
break;  
}
```

Una vegada compilat el codi i pujat a la nostra RTX4100 podrem veure que s'ha configurat un AP. Si busquem des de qualsevol dispositiu que disposi de WiFi podrem veure que ens detecta una nova i ens podem connectar a ella sense problemes ja que no disposa de cap tipus de seguretat:



Il·lustració 41: AP smartphone

Si volem comprovar que el funcionament de les peticions és correcte, per mitjà de l'aplicació Wireshark³⁸ podem veure el tràfic.

³⁸ <https://www.wireshark.org/>

3.4 Servidor Web SmartCitizenRTX4100

La part del servidor web ha d'estar integrada en l'AP que hem creat abans. El nostre servidor web ha de suportar de mode resumit les següents funcions:

1. Activació del servidor web.
2. Suportar parcialment HTTP/1.1 limitant-se a les funcionalitats bàsiques de GET i POST.
3. Servir una única pàgina d'HTML.
4. Permetre la visualització i configuració dels paràmetres.
 - a. Paràmetres de xarxa (SSID).
 - b. Mostrar les dades de l'última lectura dels sensors.

Per tant, el que tindrem que fer serà crear un WebServer que ens permeti connectar-nos a una web que tingui el format HTML i aquest ha de suportar HTTP i CCS i ha de rebre la temperatura que facilita el sensor. Per això, tindrem que permetre realitzar peticions i donar respostes.

El digrama de flux del nostre WebServer simplificat serà:

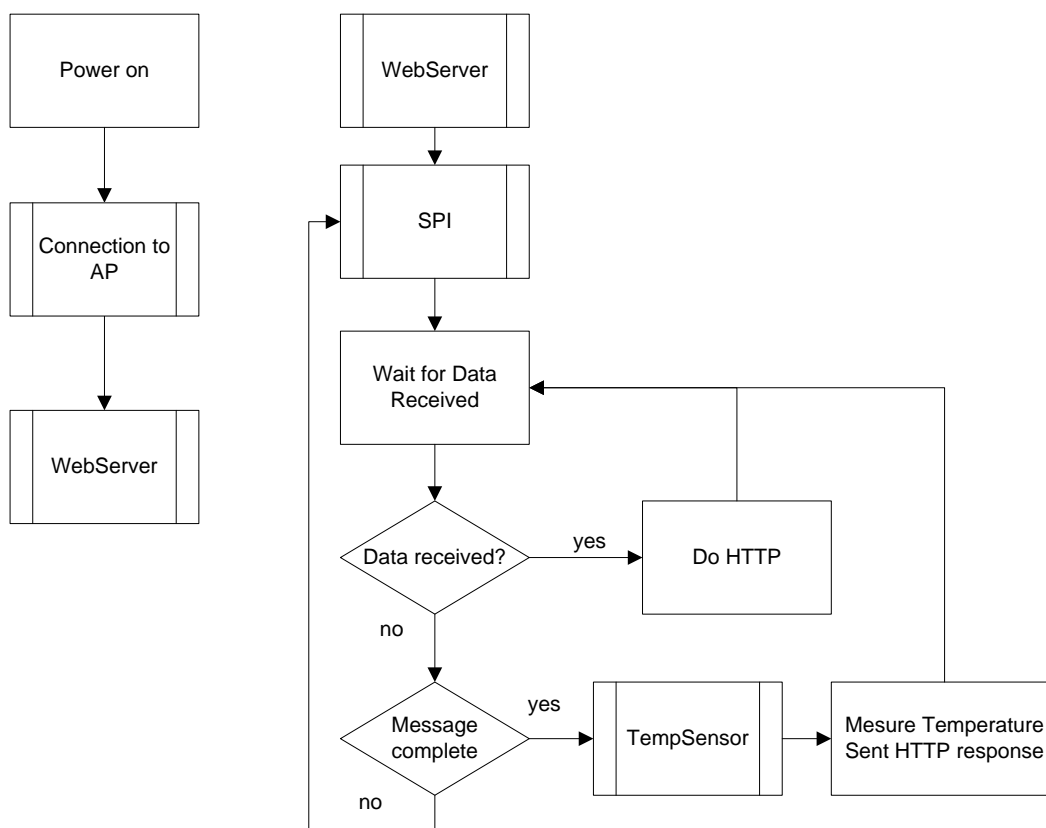


Diagrama 9: Diagrama de flux WebServer

Anteriorment ja hem vist com està format l'AP el que ens falta veure és com estarà format el TempSensor que serà qui ens faciliti la temperatura del mòdul per mitjà de SPI i per tant, com funciona l'SPI.

3.4.1 SPI

Per mitjà del mòdul SPI farem l'enviament de les dades. El nostre AP ha de connectar-se per mitjà d'una comanda SPI. Veiem el seu diagrama de flux:

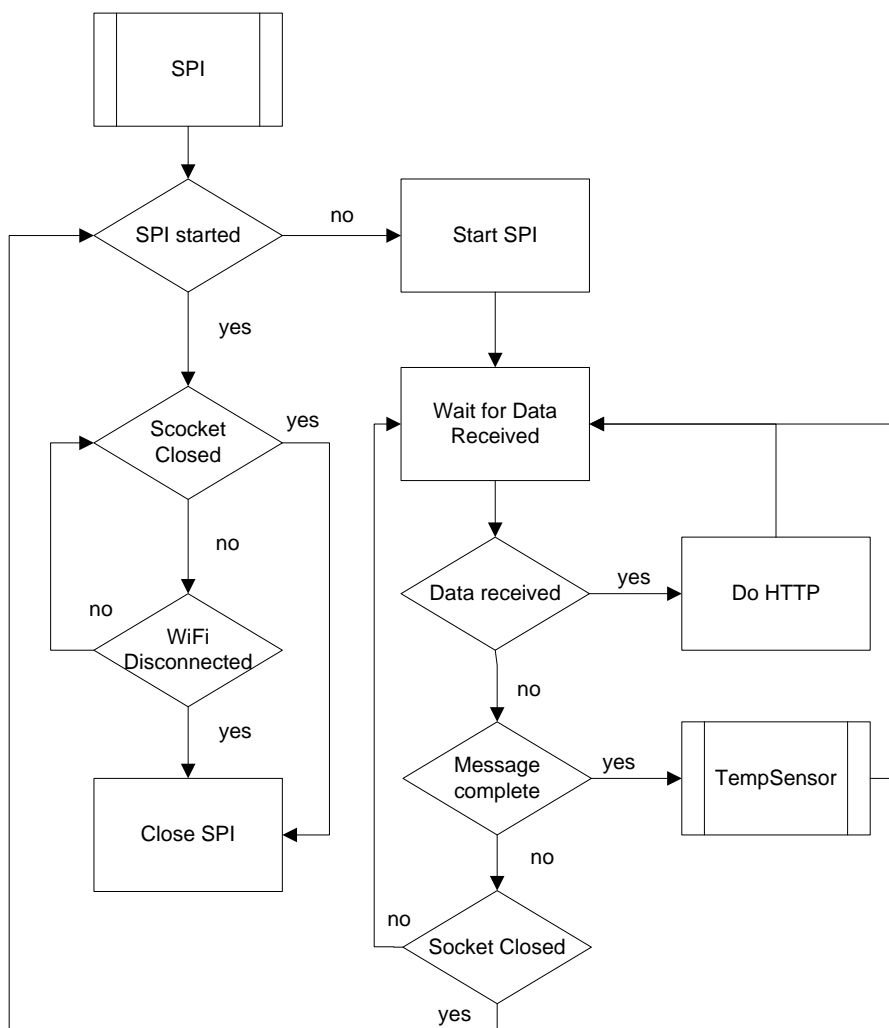


Diagrama 10: Diagrama de flux SPI_WebServer

Tal com hem dit el mòdul farà la connexió de l'enviament de les dades per mitjà del protocol SPI.

Segons el que hem vist, el que ens faltaria serà que la placa ens proporcioni els valors indicats. Per això hem d'introduir les variables corresponents, abans però, haurem d'indicar quines mesures volem fer. S'ha d'implementar les funcionalitats de SoftAP i TempSensor al mateix temps. El propòsit de l'aplicació és poder associar el mòdul

RTX4100 a l'AP i enviar-li les dades del TempSensor que es demanin. Aquí podem veure com és la configuració de baixa potència del mòdul i com informar a través de la WiFi del control de la temperatura.

Seguidament veurem un exemple per tenir clar com funciona el SPI.

Exemple SPI

Primer de tot hem d'anar a la ruta "C:\AmelieSdk\v1.6.0.58\Projects\Amelie\Components\Drivers" i modificar el fitxer DrvSpiSlave.c que trobarem, més endavant ens fixarem quines son les funcions que hi ha dintre, de moment, només hem d'afegir una línia entre el 'Activate Auto Chip select' i el 'route Usart RX and TX pins' (línies 273-376), amb aquest canvi el que aconseguim és que es processin les dades quan CS=HIGH, més endavant veurem com està format aquest codi.

```
// Enable Chip Select Invert
USART->CTRL |= USART_CTRL_CSINV;
```

Perquè la comunicació SPI funcioni, el mòdul RTX ha d'estar configurat com SLAVE i Arduino com MASTER, per això haurem de modificar el firmware de la placa perquè sigui capaç d'enviar i rebre les dades. A continuació veurem el diagrama de flux de la part SPI que afegim (veure codi en l'Annex 6: Demo_SPI – Codi Firmware):

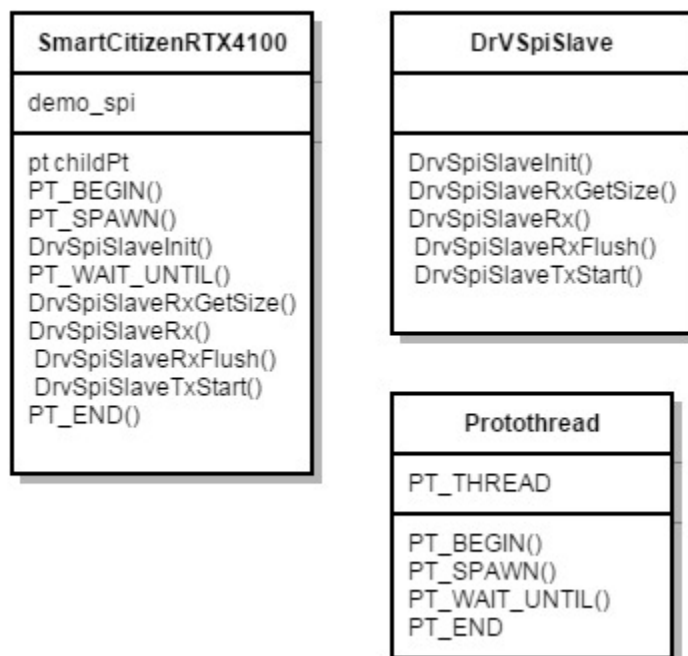


Diagrama 11: Diagrama de flux demo_SPI

Com podem veure, el nostre mètode demo_spi fa crides a dos llibreries que ja hem pogut veure quan creaven el nostre AP. El que fa es iniciar el protothread i fixar la

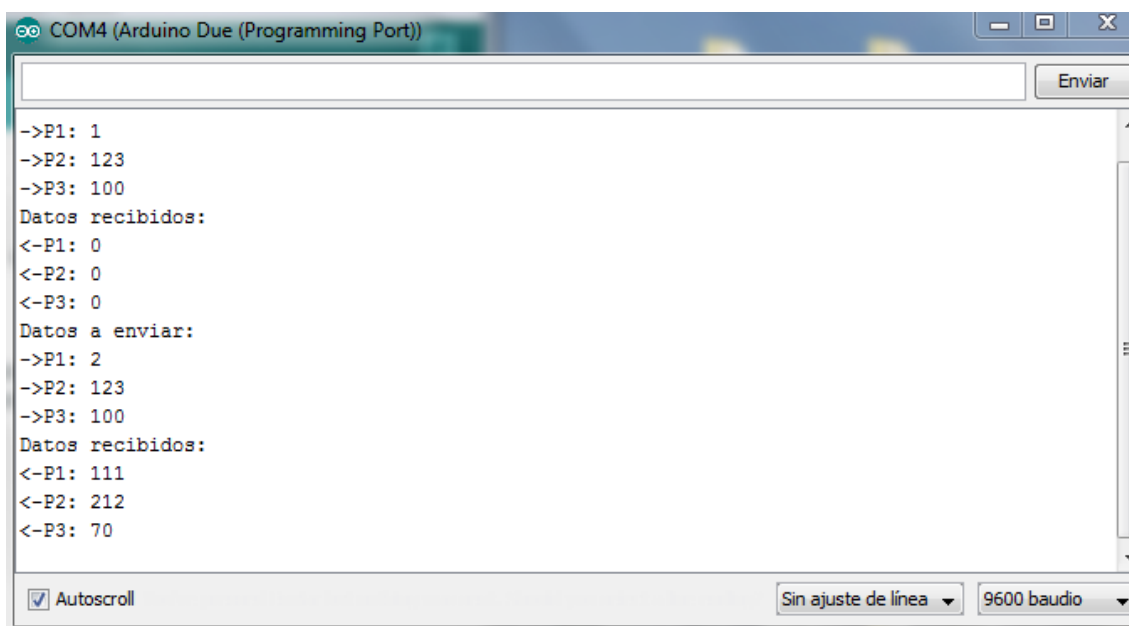
velocitat de transmissió, esperem la recepció de les dades i les gestionem. Finalment les enviem per SPI i finalitzem el protothread.

Aquest firmware l'haurem de pujar a la placa per mitjà del COLA tal com s'explica a l'apartat 2.5.3 Entorn Amelie SDK.

Per veure si la comunicació SPI funciona només ens queda crear un codi en Arduino amb el port SPI configurat (veure codi sencer en l'Annex 6: Demo_SPI – Codi Arduino).

Aquest codi s'ha de pujar per mitjà de l'aplicació Arduino tal com es comenta a l'apartat 2.5.2 Entorn Arduino IDE.

Si recordem l'aplicació Arduino té un monitor on podem veure que el resultat serà el següent:



The screenshot shows the Arduino Serial Monitor window for COM4 (Arduino Due (Programming Port)). The window contains the following text:

```
->P1: 1
->P2: 123
->P3: 100
Datos recibidos:
<-P1: 0
<-P2: 0
<-P3: 0
Datos a enviar:
->P1: 2
->P2: 123
->P3: 100
Datos recibidos:
<-P1: 111
<-P2: 212
<-P3: 70
```

At the bottom of the window, there are settings: Autoscroll, Sin ajuste de línea, and 9600 baudio.

Il·lustració 42: Exemple SPI

Un cop hem vist com fa l'enviament de dades podem passar a veure el següent mòdul.

3.4.2 TempSensor

Segons el que hem vist, el que ens faltaria serà que la placa ens proporcioni els valors indicats. Per això hem d'introduir les variables corresponents, abans però, haurem d'indicar quines mesures volem fer. Aquesta informació serà enviada per mitjà del protocol SPI.

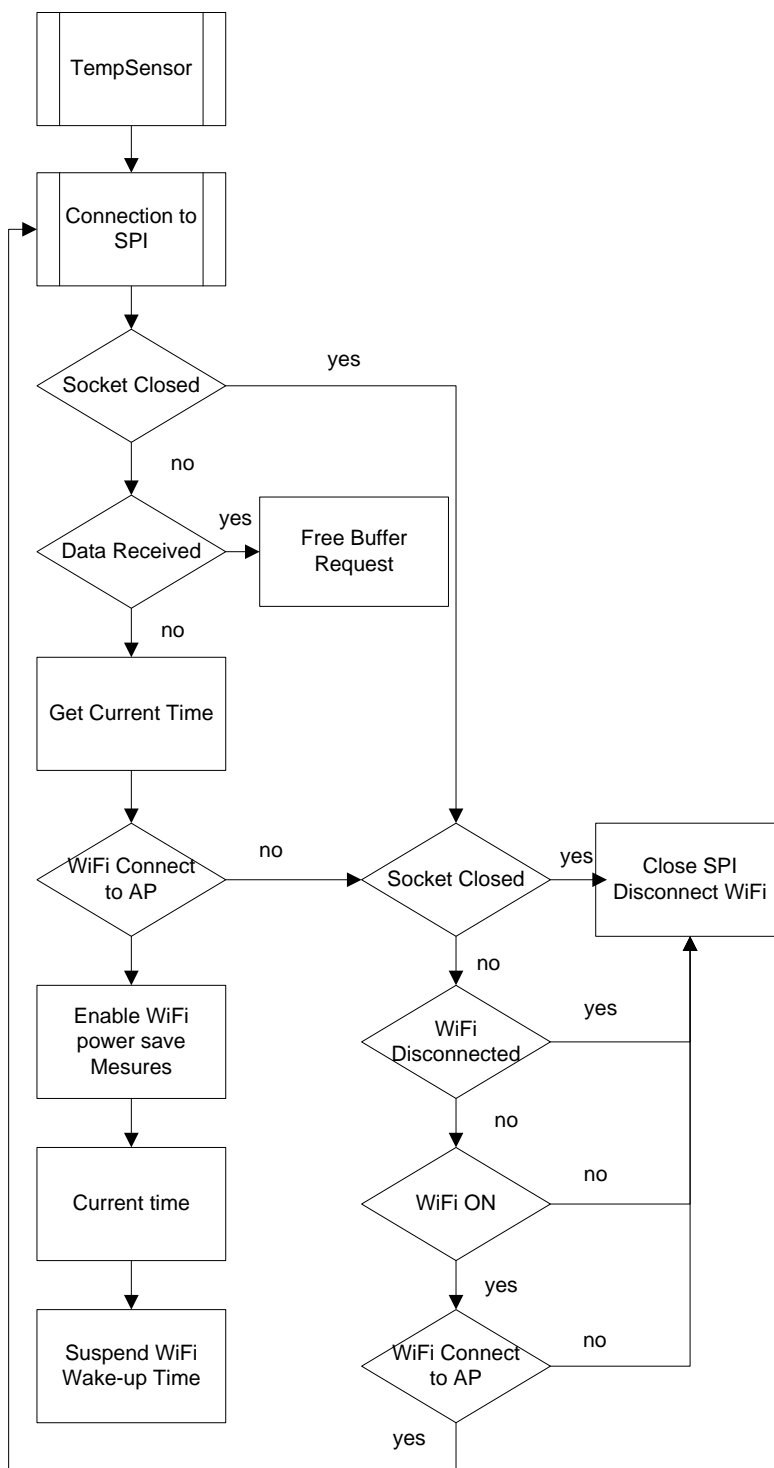


Diagrama 12: Diagrama de flux TempSensor

Com podem veure en el diagrama farà servir el protocol SPI per transferir les dades. En mode resumit el seu funcionament és el següent: engegarem el mòdul i es connectarà a la última AP a la que estava connectat, iniciarà la comunicació SPI i enviarà les mesures registrades. Una vegada transferides les dades tancarem la connexió SPI. Més endavant veurem com és la seva estructura i com fa la connexió amb el WebServer.

3.4.3 HTML

El codi HTML ha d'estar dintre del mateix WebServer, més endavant veurem la seva ubicació. La idea és crear la estructura de la web en HTML5 i els estils en CSS3. Se'ns demana que el codi estigui inserit en el codi del WebServer, per tant, el codi de la web serà semblant al següent:

```

/** Part HTML */
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>WEB SERVER RTX4100</title>
<style>
/** Part CSS */ /*Definició dels estils*/
*{
font-family: sans-serif;
font-size: 12px;
color: #798e94;
}
body{
width: 400px;
margin: auto;
background-color: #E2ECEE;
}
.Parametros{
border: 1px solid #CED5D7;
border-radius: 6px;
padding: 45px 45px 20px;
margin-top: 50px;
background-color: white;
box-shadow: 0px 5px 10px #B5C1C5, 0 0 10px #EEF5F7 inset;
}
.Parametros label{
display: block;
font-weight: bold;
}
.Parametros div{
margin-bottom: 15px;
}
</style> /*Fi definició d'estils*/
//<link rel='stylesheet' href='estilos.css'>
</head>
<body>
<br>
<h1 align='center'> Web Server RTX4100</h1>
<form class='Parametros'> //Crida a l'estil de Parametros

```

```
<div><label>IP:</label><output type='text' value=''></div>
<div><label>SSID:</label><output type='text' value=''></div>
<div><label>Temperatura:</label><output type='text' value=''></div>
<div><input type='submit' value='Reset'></div>
</form>
</body>
</html>
```

Aquest codi és una primera versió, el modificarem depenent dels valors que vulguem obtenir o rebre i tenint en compte la mida que se'ns permeti carregar.

El resultat del codi anterior és el següent:



Il·lustració 43: Resultat WebServer

Ja tenim el disseny HTML del WebServer. Ara mirarem quines parts formen el codi del WebServer,

3.4.4 Llibreries

A continuació estudiem les funcions i crides que necessitem per crear el WebServer.

Estructura del codi de WebServer

Per crear el WebServer, com hem dit abans, ens fixarem en l'exemple que tenim d'Amelie, per tant, estudiem el seu codi:

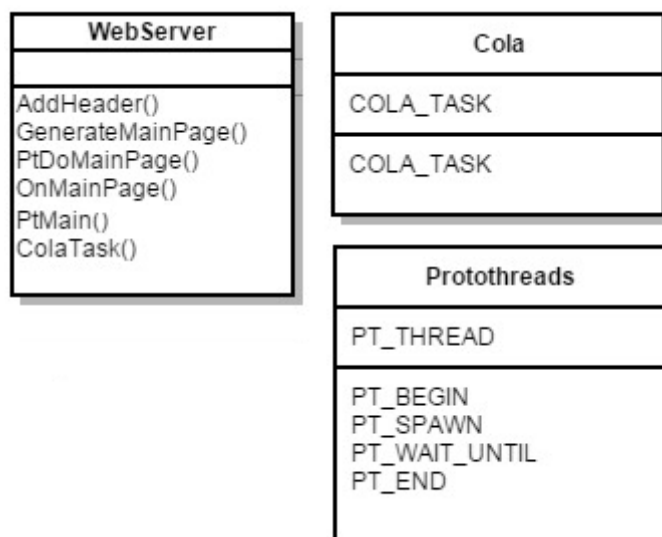


Diagrama 13: Diagrama de funcions WebServer_Amelie

A continuació explicarem les seves funcions:

MAIN	Funció	Explicació
WebServer	AddHeader	Genera la capçalera d'HTTP
WebServer	GenerateMainPage	Genera la web HTML, aquí és on hem de ficar el nostre codi
WebServer	PtDoMainPage	Protothread: Traurem el valor de la temperatura
WebServer	OnMainPage	Enviament HTTP (RSS)
WebServer	PtMain	Protothread que crida el Main
COLA	COLA_TASK	COLA
Protothread	PT_BEGIN	Inici d'un protothread
Protothread	PT_SPAWN	Crida a un protothread fill
Protothread	PT_WAIT_UNTIL	Protothread en espera
Protothread	PT_END	Fi d'un protothread

Aquest era el codi que hi havia d'exemple, com podem veure en diagrama de flux, nosaltres el que volem és implementar el WebServer amb l'AP, per tant, tindrem que

implementar els dos mòduls. Recordem que cadascun té crides a diferents llibreries, veiem com serà el digrama de funcions:

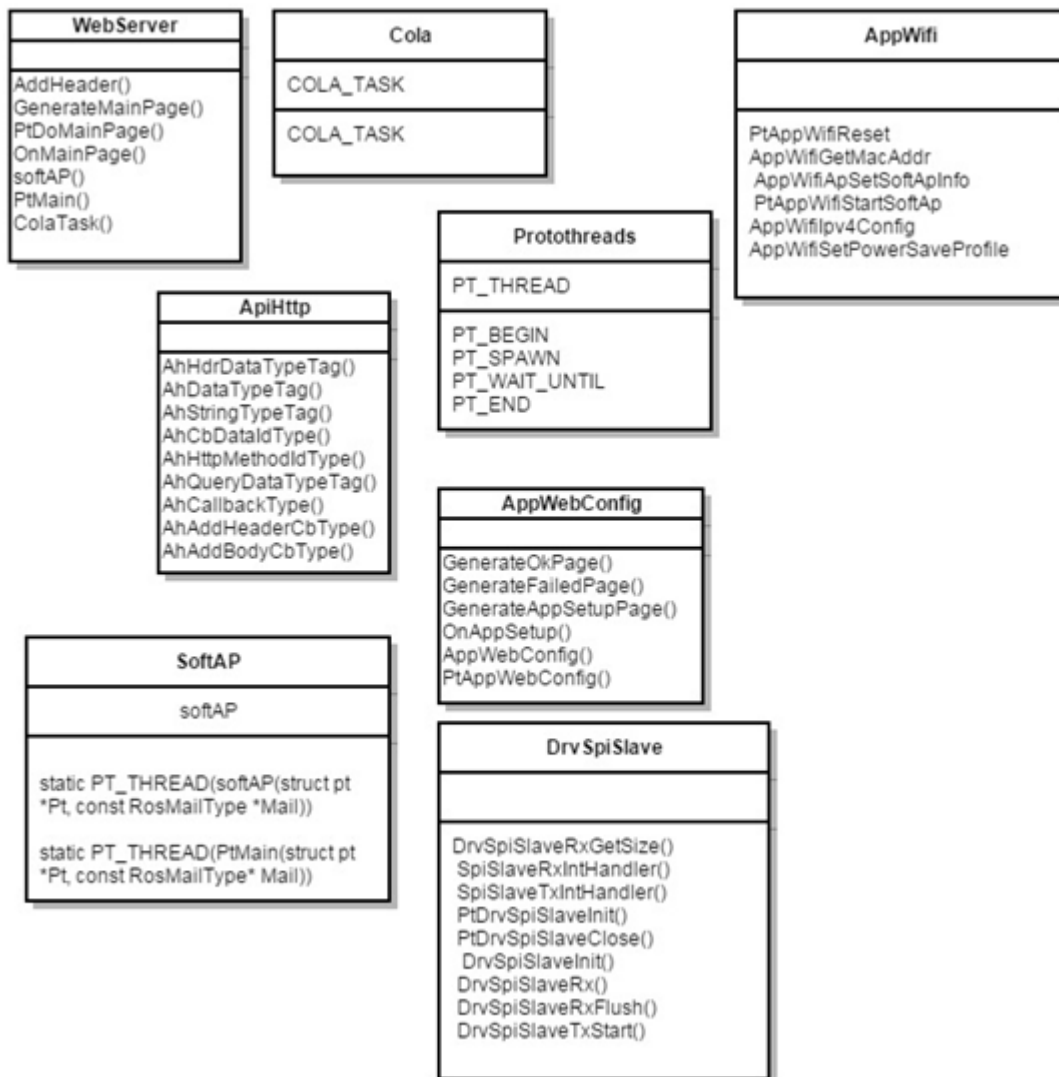


Diagrama 14: Diagrama de funcions WebServer

El que fem és cridar al protothread de SoftAP des del mateix WebServer.c. El protothread pare serà el PtMain, des d'aquí s'executaran tots els fills (inicia el servidor HTTP i afegeix la web creada a GenerateMainPage). Primer ens interessa carregar el SoftAP perquè s'iniciï abans per tenir la connexió SoftAP i una vegada estigui la connexió feta llancem el bucle de la temperatura i carreguem la web gràcies a la llibreria ApiHttp³⁹ que ens permet comunicar el servidor web amb el navegador. Per tant, primer es genera la web amb el GenerateMainPage i després es crida des de OnMainPage. Hem de tenir en compte que el GenerateMainPage té un buffer de 32Kbytes, per això, el nostre codi no pot ocupar més ja que no el carregarà sencer.

³⁹ Es troba a "C:\AmelieSdk\v1.6.0.58\Include\IOT\App\Http"

Dintre del mètode PtDoMainPage trobem com treure les mesures, hem de dir que només hem aconseguit treure la temperatura del mòdul, l'SSID que ens dona no és correcte i la data encara que ens doni una que no és la del dia és normal ja que segons la web oficial la data s'ha de configurar el dia en la que engeguem la placa per ficar-la a l'hora.

Mirem més a fons que fa cadascuna de les llibreries que s'implementen:

ApiHttp

Com hem dit, és l'encarregat de fer la connexió del servidor web i el client. Mirarem per sobre les funcions que ens interessen:

Funció	Explicació
AhHdrDataTypeTag	Manté la capçalera HTTP
AhDataTypeTag	Emmagatzema les dades del cos del missatge
AhStringTypeTag	Devolució de la trucada
AhCbDataIdType	Identifica el tipus de dades de la devolució de trucada
AhHttpMethodIdType	Identifica el mètode que suporta
AhQueryDataTypeTag	Manté el punter i retorna el valor de dades de la consulta
AhCallbackType	Manté el punter i retorna el valor de dades de la consulta
AhResourceCbType	Devolució de la trucada, 404: desconegut, 501: RSS_NOT_SUPPORTED, 500: RSS_FAILED
AhAddHeaderCbType	Genera/afegeix camps de capçalera al missatge HTTP que s'envia
AhAddBodyCbType	Annexa dades del cos del missatge HTTP enviat

Com podem veure ens genera la connexió amb l'HTTP, en aquesta llibreria trobarem totes les funcions necessàries per poder fer la crida, generació i enviament de la nostra web. Si ens fixem en la llibreria, a partir de la línia 427 podem veure els prototipus de funció per a l'emalatge i l'enviament de correu de funcions (MPS).

AppWebConfig

Si ens fixem en l'exemple de WebServer podem veure que fa una crida a la llibreria AppWebConfig⁴⁰, aquesta llibreria configura els paràmetres de l'HTML i inicia tant el softAP com el WebServer. A continuació explicarem el seu funcionament:

Funció	Explicació
GenerateOkPage	Si es genera sense errors l'HTML
GenerateFailedPage	Si es genera amb errors l'HTML
GenerateAppSetupPage	Configuració dels paràmetres de l'HTML
OnAppSetup	On es carrega el mètode
AppWebConfig	Modifiquen diferents paràmetres del SoftAP, Ssid, key, channel...
PtAppWebConfig	Prothread: Inicia el SoftAP, DHCP i webserver

Per tant, aquesta llibreria va de la mà amb l'ApiHttp.

Codi de WebServerRTX4100

El codi que hem vist anteriorment s'ha de configurar junt amb el softAP, el codi softAP és el mateix que hem creat per l'AP, l'únic que per fer la crida al softAP necessitem un altre Prothread que nosaltres hem anomenat PtMain. Creem el prothread fill, configurem els paràmetres del temps d'escolta que es troba com sempre a l'AppWiFi i iniciem el prothread del softAP. S'iniciarà el servidor HTTP gràcies a la crida a la llibreria ApiHttp per el port 80 i afegim la web creada per mitjà de la crida novament a la llibreria ApiHttp que el que fa és facilitar la cadena de ruta que ha d'estar disponible sempre que el servidor estigui en execució, per tant, genera la sol·licitud del navegador web.

Un cop tot configurat tant el softAP com el WebServer el resultat del nostre codi es pot veure en l'Annex 8: Codi WebServer RTX4100.

⁴⁰ Es pot trobar a C:\AmelieSdk\v1.6.0.58\Projects\Amelie\Components\PtApps

El resultat de la web serà la següent:



Il·lustració 44: Resultat WebServerRTX4100

Com es pot veure tindrem dos seccions, la de configuració del menú i la d'informació. En aquests moments no s'ha creat la part de configuració ja que no ha donat temps de fer proves amb els formularis per tal de modificar-ho però en el codi final es pot veure un esborrany de la seva aproximació.

En la part d'informació es pot veure la data, la qual s'ha de configurar sempre a l'iniciar la placa, la temperatura del mòdul, la humitat i l'SSID. El problema és que l'SSID que ens facilita no és el correcte i la humitat deuria de ser en %. El botó Refresh serà per refrescar la web i podrem veure les dades actualitzades.

Si finalment es fiquen els formularis per la modificació es podrien crear pestanyes per no haver de crear diverses pàgines.

3.5 Integració del codi

Una vegada tenim tant l'AP com el WebServer funcionant és hora de ficar-ho dins del codi del Sr.Colom.

La part de l'AP ja l'hem aplicat en el WebServer, per tant agafarem el codi del WebServer i l'ajustarem al codi general. Tindrem en compte les llibreries amplades i variables definides.

Comentem una mica per sobre com ha sigut la integració del codi. Ens hem trobat que dintre del Main del Sr. Colom ja hi havia un mòdul softAP, el que hem fet ha sigut actualitzar-lo amb la nostra informació. També estava el PtMain que era més complex per lo que li hem canviat el nom al nostre i ha passat a dir-se PtMainSoftAP. Dintre del seu PtMain s'ha fet la trucada al PtMainSoftAP.

Capítol 4: Conclusions i línies de futur

4.1. Conclusions

Podem veure que gràcies al software lliure tenim un ventall molt gran per poder fer múltiples funcions. En el nostre cas gràcies a l'empresa SmartCitizen hem pogut veure una adaptació d'Arduino, en concret hem pogut estudiar l'SCK 1.5. Aquesta placa està formada per dos PCB's que son Arduino DUE que es dedica al processament, transmissió i emmagatzemant de dades (Data Board) i la segona és l'SCK1.5 que està dedicada als sensors i als seus complements (Sensor Board - Ambient sensor board).

En aquest projecte hem pogut veure com està formada la placa Arduino i com és el seu entorn de programació. El firmware de l'SCK es desenvolupa com un mòdul i l'RTX4100 és l'encarregat de gestionar-lo i està desenvolupat com una COLApp. Per poder programar el mòdul RTX4100 l'empresa RTX ha desenvolupat l'aplicació Amelie SDK serveix per compilar i enllaçar. Per això s'ha estudiat la seva constitució, hem vist i fet proves amb els exemples proporcionats i hem pogut pujar el nostre codi a l'SCK. Abans però hem estudiat les llibreries i les funcions que hi ha implicades en la seva comunicació per finalment poder crear el nostre AP i WebServer.

Durant el procés de disseny i desenvolupament del projecte van aparèixer diversos problemes i obstacles que van requerir una major atenció i esforç per poder solventar-los. Principalment van ser conseqüència de la limitació de coneixements i la dificultat d'algunes de les idees plantejades. La part de COLApp ha sigut difícil d'entendre però finalment puc dir que ser com funciona, en canvi l'SPI no m'ha funcionat del tot. Hi havia molta informació que entendre i separar la que ens interessava. També va haver-hi un problema en la recepció del material cosa que va fer que tinguéssim poc temps per poder fer proves i es va haver de modificar la planificació, en el meu cas hagués necessitat un mes o una mica més per poder complir amb tots els objectius proposats.

Cal destacar que finalment s'han complert quasi tots els objectius, s'ha pogut crear un mòdul AP amb el qual ens podem connectar per mitjà de qualsevol dispositiu que tingui incorporat WiFi, també s'ha implementat la comunicació per SPI per poder fer tant la petició de connexió de l'AP com per enviar les dades necessàries a l'HTML, s'ha creat un WebServer molt simple amb HTML i CSS incorporats en el mateix codi encara que només s'ha aconseguit que ens faciliti la temperatura del mòdul.

El que ha faltat fer ha sigut poder treure per web tant l'SSID com la temperatura del sensor. Tant la humitat com la data les dona malament, la humitat no he sigut capaç de que surti en tant per cent i la data s'ha de configurar manualment cada vegada que s'inicia el codi, aquest problema ja està detectat en la web d'Arduino.

Dintre del WebServer es volia fer formularis perquè es poguessin modificar alguns camps, però per falta de temps no s'ha pogut mirar i es ficarà com a proposta. També en la part de l'AP es volia implementar, com a cosa extra, un entorn de comunicació segura entre el servidor i el dispositiu, s'ha pogut mirar la part teòrica però no ha donat temps de mirar la implementació.

Per finalitzar he de dir que aquest treball no ha estat el que m'esperava, la meva idea era poder treballar més amb la part d'Arduino per tocar alguna cosa de programació junt a electrònica però gràcies que l'empresa SmartCitizen ens ha facilitat el kit hem tingut que fer un projecte, com és normal en aquests casos, amb una sèrie de requisits per lo que finalment no ens hem centrat en Arduino. Igualment s'ha de dir que m'ha agradat poder fer proves amb un kit encara en fase de desenvolupament, és cert que li he dedicat moltes hores i semblava que no aconseguia res, però ara veient ja la finalització del projecte puc dir que estic contenta d'haver pogut arribat fins aquí.

4.2. Línies de futur

Trobem diverses ampliacions o línies de futur que podem dividir en diferents apartats:

AP

En el treball s'han explicat els requisits de la WiFi i els diferents tipus de seguretat però per falta de temps no s'ha pogut mirar. Es podrien afegir les funcions de seguretat a l'AP per controlar l'accés a ell i els serveis que aquest pot oferir.

Una altre proposta de millora és que al connectar-se a l'AP s'obri directament el navegador amb el WebServer sense tenir la necessitat d'haver d'obrir el navegador per comprovar la temperatura que ens proporciona, o si finalment fem seguretat, el navegador ens podria demanar la identificació de l'usuari.

WebServer

Com ha proposta futura es podria millorar el servidor web afegint per exemple que suporti components JavaScript. També s'haurien de poder fer modificacions en les opcions del mòdul i estaria bé que guardés un log de connexió amb les dades.

En comptes de crear el codi en el mateix mòdul, podem fer que la informació la agafi de la SD i la tregui per el navegador, d'aquesta forma no estariem pendents de no superar el buffer i podria contindre més informació com per exemple imatges o vídeos.

Llibreria SCK

Aquest apartat no era obligatori fer-ho però si que seria força interessant desenvolupar una llibreria per el firmware de l'SCK que implementi les comandes SPI per la gestió de les funcionalitat de l'AP i el WebServer. D'aquesta manera alliberariem codi del programa principal i només deuríem fer la crida a les funcions necessàries.

Capítol 5: Acrònims

- **3DES:** Triple DES
- **AAA:** Authentication Authorization Accounting
- **APIs:** Application Programmers Interfaces
- **APN:** Acces Point Name
- **BOE:** Bolletí Oficial de l'Estat
- **CMT:** Comissió del Mercat de les Telecomunicacions
- **DHCP:** Dynamic Host Configuration Protocol
- **DVK:** Development Kit
- **EAS:** Advanced Encryption Standard
- **ESS:** Extended service set
- **GHz:** Gigahertz
- **GPRS:** General Packet Radio Service
- **GSM:** Global System for Mobile Communication
- **HiperLAN 2:** High Performance Radio LAN 2.0
- **HTTP:** Hypertext Transfer Protocol
- **HTML:** HyperText Markup Language
- **IEEE:** Institut d'Enginyeria Elèctrica i Electrònica
- **IP:** Internet Protocol
- **LAN:** Local Area Network
- **MAC:** Mesurava Access Control
- **MCU:** Micro Controller Unit
- **Mbps:** Megabits per segon
- **PPP:** Point to Point Protocol
- **RTX:** Real Time Extended
- **SCK:** Smart Citizen Kit
- **SDK:** Software Development Kit
- **SLIP:** Serial Line Internet Protocol
- **SPI:** Serial Peripheral Interface
- **SSID:** Service Set Identifier
- **SoC:** System on Chip
- **TCP:** Transmission Control Protocol
- **UMTS:** Universal Mobile Telecommunication System.
- **VPN:** Virtual Private Network
- **W3C:** World Wide Web Consortium
- **WEP:** Wired Equivalency Privacy

- **WAN:** Wide Area Networ
- **WiFi:** Wireless Fidelity
- **wIPS:** Wireless intrusion protection system
- **WiMAX:** Worldwide Interoperability for Microwave Access
- **WNAM:** Wireless Metropolitan Area Network
- **WPA:** WiFi Protected Access
- **WPA2:** WiFi Protected Access 2
- **WPA PSK:** WiFi Protected Access Pre-Share Keyda
- **WPAN:** Wireless Personal Area Network
- **WWAN:** Wireless Wide Area Network

Capítol 6: Bibliografia

➤ Contingut web

- Arduino. "Arduino Due" [en línia]
<http://arduino.cc/en/Main/arduinoBoardDue>
- Fritzing: "Fritzing electronics made easy" [en línia] <http://fritzing.org>
- Github. "SmartCitizenRTX4100" [en línia]
<https://github.com/mcolom/SmartCitizenRTX4100>
- Introducción a los servidores web. "Carles Mateu i Piñol" [en línia]
http://www.cibernetia.com/manuales/instalacion_servidor_web/
- Javier Longares: "Protothreads: Threads en C sin sistema operativo ni memoria RAM" [en línia] <http://www.javierlongares.com/arte-en-8-bits/protothreads-threads-en-c-sin-sistema-operativo>
- Wikipedia: "HTML" [en línia] <http://es.wikipedia.org/wiki/HTML>
- Wikipedia: "IEEE 802.11" [en línia] http://es.wikipedia.org/wiki/IEEE_802.11
- Wikipedia: "Punto de acceso inalámbrico" [en línia]
http://es.wikipedia.org/wiki/Punto_de_acceso_inal%C3%A1mbrico
- Wikipedia: "Serial Line Internet Protocol" [en línia]
http://es.wikipedia.org/wiki/Serial_Line_Internet_Protocol
- Wikipedia: "WiFi" [en línia] <http://es.wikipedia.org/wiki/WiFi>

➤ Legislació, normatives

- Circular 1/2010, de 15 de juny de 2010, de la Comissió del Mercat de les Telecomunicacions, per la que es regulen les condicions d'exploració de xarxes y la prestació de serveis de comunicacions electròniques per la Administració Pública [en línia]
http://www.boe.es/aeboe/consultas/bases_datos/doc.php?id=BOE-A-2010-12831
- Llei Orgànica 15/1999, de 13 de desembre, de protecció de dades de caràcter personal [en línia]
http://noticias.juridicas.com/base_datos/Admin/lo15-1999.html
- Ordre IET/787/2013, de 25 d'abril, per la que s'aprova el quadre nacional d'atribució de freqüències (CNAF 2013) [en línia]
<http://www.boe.es/boe/dias/2013/05/09/pdfs/BOE-A-2013-4845.pdf>
- OMS: Nota descriptiva 304 [en línia]
<http://www.who.int/mediacentre/factsheets/fs304/en/index.html>

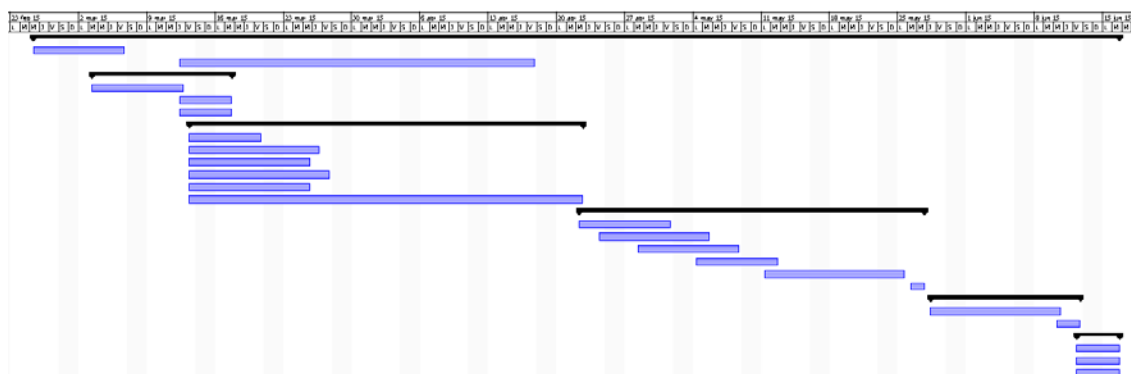
- Real Decreto 1066/2001, de 28 de setembre, *per la que s'aprova el Reglament que estableix condicions de protecció del domini públic radioelèctric, restriccions a les emissions radioelèctriques i mesures de protecció sanitària enfront a emissions radioelèctriques* [en línia] <http://www.boe.es/boe/dias/2001/09/29/pdfs/A36217-36227.pdf>
- Real Decret 1720/2007, de 21 de desembre, per el que s'aprova el Reglament de desenvolupament de la Llei Orgànica 15/1999, de 13 de desembre, de protecció de dades de caràcter personal («BOE» 19 gener 2008) [en línia] <https://www.boe.es/buscar/act.php?id=BOE-A-2008-979>

Capítol 7: Annexos

➤ Annex 1: Diagrama de Gannt

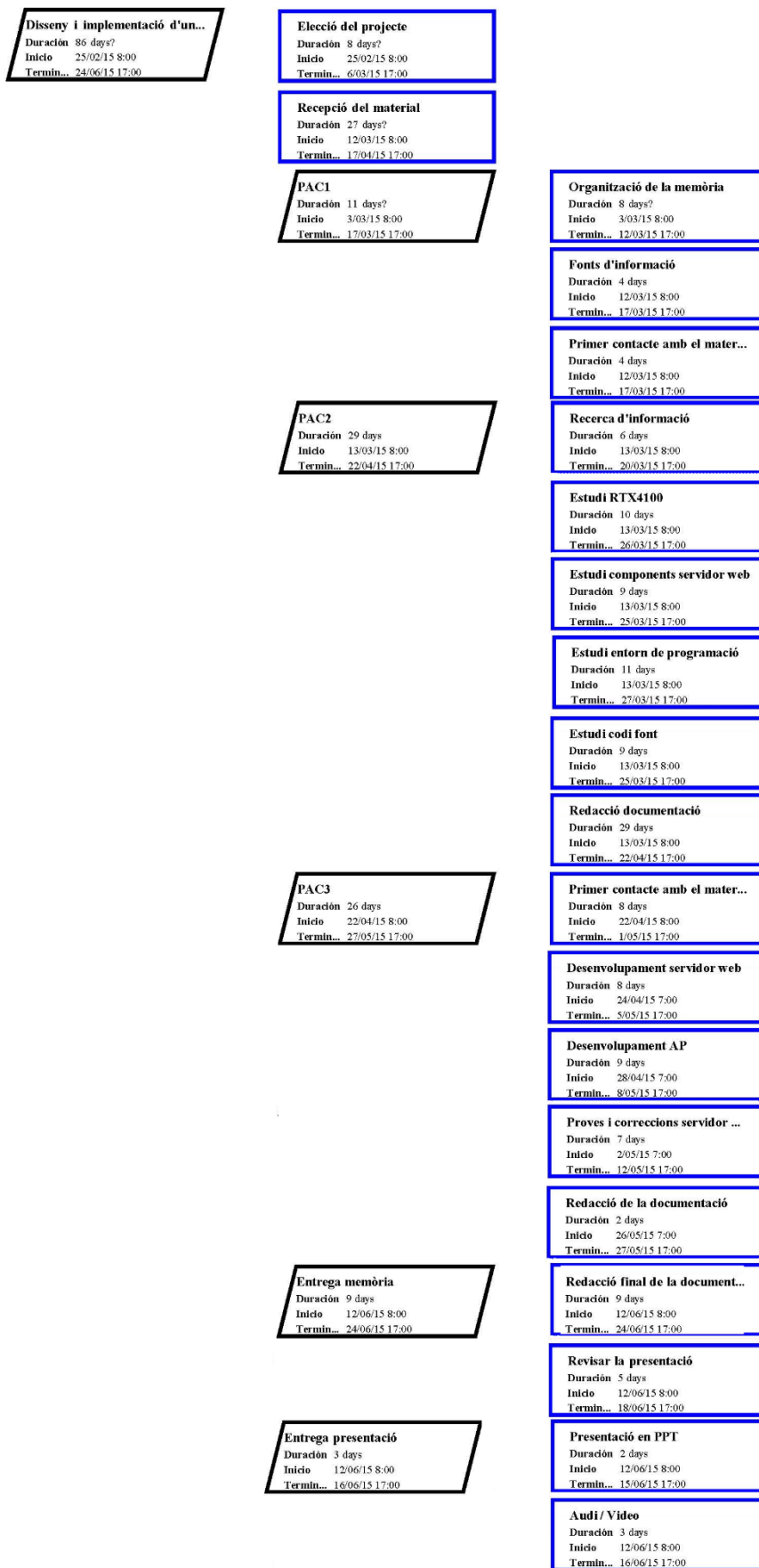
	📌	Nombre	Duració	Inicio	Terminado
1	📌	Disseñy i implementació d'un servidor web amb el mòdul RTX4100	80 days?	25/02/15 8:00	16/06/15 17:00
2		Elecció del projecte	8 days?	25/02/15 8:00	6/03/15 17:00
3	📌	Recepció del material	27 days?	12/03/15 8:00	17/04/15 17:00
4	📌	PAC1	11 days?	3/03/15 8:00	17/03/15 17:00
5	📌	Organització de la memòria	8 days?	3/03/15 8:00	12/03/15 17:00
6	📌	Fonts d'informació	4 days	12/03/15 8:00	17/03/15 17:00
7	📌	Primer contacte amb el material	4 days	12/03/15 8:00	17/03/15 17:00
8	📌	PAC2	29 days	13/03/15 8:00	22/04/15 17:00
9	📌	Recerca d'informació	6 days	13/03/15 8:00	20/03/15 17:00
10	📌	Estudi RTX4100	10 days	13/03/15 8:00	26/03/15 17:00
11	📌	Estudi components servidor web	9 days	13/03/15 8:00	25/03/15 17:00
12	📌	Estudi entorn de programació	11 days	13/03/15 8:00	27/03/15 17:00
13	📌	Estudi codi font	9 days	13/03/15 8:00	25/03/15 17:00
14	📌	Redacció documentació	29 days	13/03/15 8:00	22/04/15 17:00
15	📌	PAC3	26 days	22/04/15 8:00	27/05/15 17:00
16	📌	Primer contacte amb el material	8 days	22/04/15 8:00	1/05/15 17:00
17	📌	Desenvolupament servidor web	8 days	24/04/15 7:00	5/05/15 17:00
18	📌	Desenvolupament AP	9 days	28/04/15 7:00	8/05/15 17:00
19	📌	Proves i correccions servidor web	7 days	2/05/15 7:00	12/05/15 17:00
20	📌	Proves i correcció AP	11 days	9/05/15 7:00	25/05/15 17:00
21	📌	Redacció de la documentació	2 days	26/05/15 7:00	27/05/15 17:00
22	📌	Entrega memòria	12 days	28/05/15 8:00	12/06/15 17:00
23	📌	Redacció final de la documentació	10 days	28/05/15 8:00	10/06/15 17:00
24	📌	Revisar la presentació	3 days	10/06/15 8:00	12/06/15 17:00
25	📌	Entrega presentació	3 days	12/06/15 8:00	16/06/15 17:00
26	📌	Presentació en PPT	3 days	12/06/15 8:00	16/06/15 17:00
27	📌	Audi / Video	3 days	12/06/15 8:00	16/06/15 17:00
28	📌	Edició de la presentació	3 days	12/06/15 8:00	16/06/15 17:00

Annex 1a: Diagrama de Gannt desenvolupat



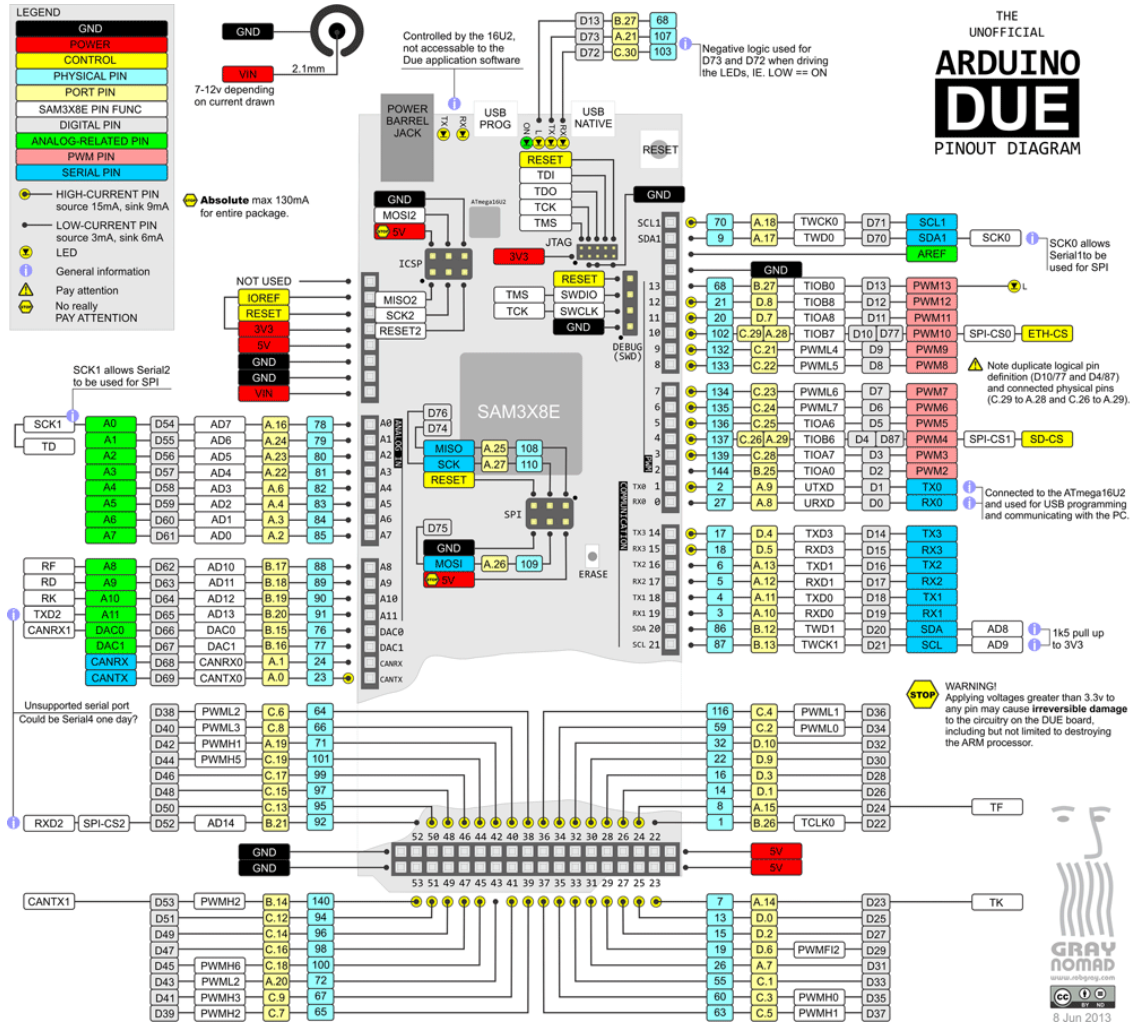
Annex 1b: Diagrama de Gannt

➤ Annex 2: Cronograma WBS



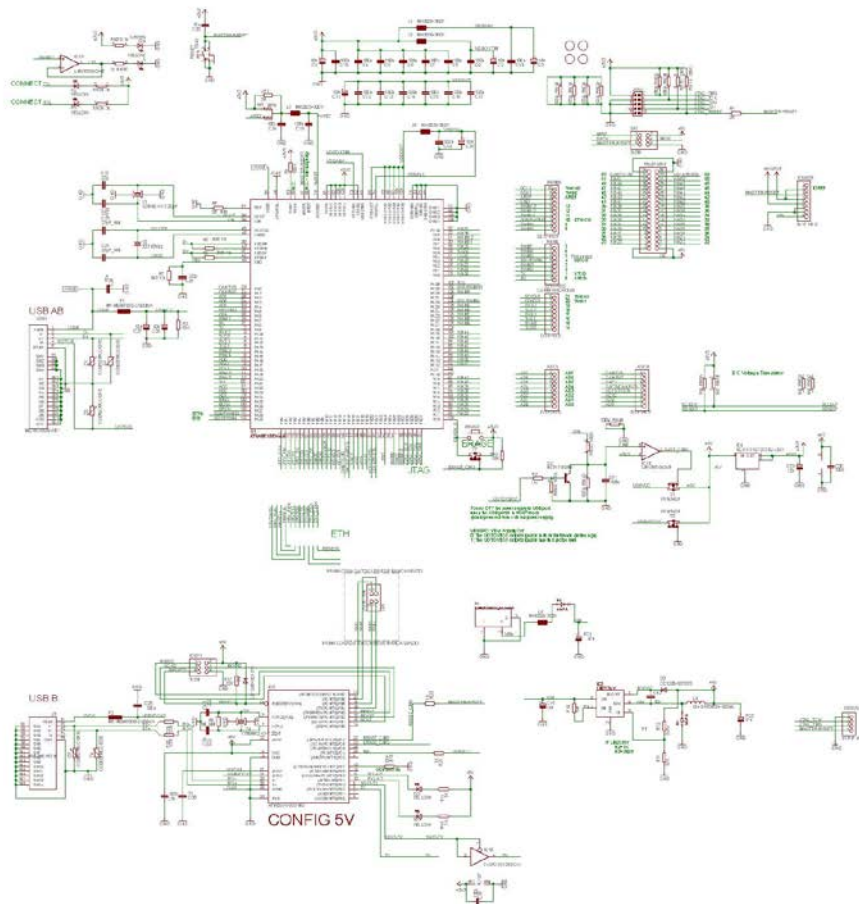
Annex 2: Cronograma WBS

➤ Annex 3: PINOUT Arduino DUE



Annex 3: PINOUT Arduino DUE

➤ **Annex 4: Arduino DUE Schematic**



Annex 4: Arduino DUE Schematic

➤ Annex 5: Codi WebServer Arduino

```

/*****
** Web Server Arduino**
*****/

//Llibreries
#include <SPI.h>
#include <Ethernet.h>

//Assignar IP a la placa
byte mac[] = {0xDE, 0xFD, 0xAE, 0xAF, 0xFE, 0xED };
IPAddress ip(192, 168, 1, 136);

//Rebre peticions HTTP
EthernetServer server(80);
EthernetClient client;

//Variables
int led=6; //Iniciem led en 6
int posicio; //Variable per guardar la posició
String estado="OFF"; //Iniciem el led en OFF
String cadena;

/**Comunicació Ethernet i servidor***/
void setup() {
  Serial.begin(9600); //Obre port serie amb velocitat 9600 baudios
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());
  pinMode(led,OUTPUT);
}

/**Servidor***/
/* El que farà serà que quan detecti que el client enviï una petició H
TTP aquest el llegirà caràcter per caràcter i d'aquesta forma podrem f
er una recerca i veure l'estat del LED. Quan detecti una línia en blan
c serà que ha acabat i enviarà la resposta */
void loop() {
  EthernetClient client = server.available();
  //Si detecta un client petició HTTP
  if (client) {
    Serial.println("new client");
    boolean currentLineIsBlank = true; //Petició HTTP acaba amb línia
en blanc
    while (client.connected()) { //Mentres client connected
      if (client.available()) { //Visualitzem petició HTTP
        char c = client.read(); //Llegim petició HTTP
        Serial.write(c); //Visualitzem petició HTTP
        cadena.concat(c);

        //Recerca en la petició
        posicio=cadena.indexOf("LED="); //Guardem posició LED
        //Si posició=ON, estado=ON
        if(cadena.substring(posicio)== "LED=ON"){
          digitalWrite(led,HIGH);
          estado="ON";
        }
      }
      //Si posició=OFF, estado=OFF
      if(cadena.substring(posicio)== "LED=OFF" ){

```

```

digitalWrite(led,LOW);
estado="OFF";
}
//Si rep línia en blanc = petició HTTP acabada
if (c == '\n' && currentLineIsBlank) {
  //Enviem resposta HTTP
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/html");
  client.println();

  /*** Codi HTML ***/
  /* La nostra web estarà formada per dos botons que
  engegaran/apagaran el LED que connectarem a la placa. Per poder enviar
  els paràmetres per mitjà del codi hem de fer servir URL.*/
  client.println("<!DOCTYPE html PUBLIC>");
  client.println("<html>");
  client.println("<head><meta content='text/html; charset=utf-
8' http-equiv='Content-Type' />");
  client.println("<title></title>");
  client.println("</head>");
  client.println("<body>");
  client.println("<br>");
  client.println("<h1 align='center'> Web Server
Arduino</h1>");
  client.println("<div style='text-align:center;'>");
  client.println("<button onClick=location.href='./?LED=ON\
'");
  client.println("ON");
  client.println("</button>");
  client.println("<button
onClick=location.href='./?LED=OFF\'");
  client.println("OFF");
  client.println("</button>");
  client.println("<br /><br />");
  client.println("<b>LED = ");
  client.print(estado);
  client.println("</b><br />");
  client.println("</b></body>");
  client.println("</html>");

  break;
}
if (c == '\n') {
  currentLineIsBlank = true;
}
else if (c != '\r') {
  currentLineIsBlank = false;
}
}
}
//Control de temps
delay(1);
//Tanquem connexió
client.stop();
Serial.println("client disconnected");
}
}

```

➤ Annex 6: Demo_SPI

Annex 6: Demo_SPI – Codi Firmware

```

/** demo_SPI */
static PT_THREAD(demo_spi(struct pt *Pt, const RosMailType *Mail)) {
    static struct pt childPt;
    PT_BEGIN(Pt);

    //PARTE SLAVE
    //PT_YIELD_UNTIL(Pt, IS_RECEIVED(KEY_MESSAGE));

    const rsuint32 baud_rate = 32000000;

    printf("Arrancando PtDrvSpiSlaveInit\n");
    PT_SPAWN(Pt, &childPt, PtDrvSpiSlaveInit(&childPt, Mail, baud_rate));

    //printf("Arrancando DrvSpiSlaveInit\n");
    //DrvSpiSlaveInit(baud_rate);

    while (1) {
        //PT_YIELD_UNTIL(Pt, IS_RECEIVED(SPI_SLAVE_RX_DATA));
        //PT_YIELD_UNTIL(Pt, IS_RECEIVED(KEY_MESSAGE));
        RosTimerStart(APP_PACKET_DELAY_TIMER, (3000 * RS_T1MS), &PacketDe
layTimer);PT_YIELD_UNTIL(Pt,IS_RECEIVED(APP_PACKET_DELAY_TIMEOUT));
        printf ("Intentando enviar datos...\n");

        rsuint8 recibidospi[DrvSpiSlaveRxGetSize()];
        static rsuint8 enviadospi[5];

        DrvSpiSlaveRx(&recibidospi[0], sizeof(recibidospi));

        enviadospi[0] = 111;
        enviadospi[1] = 212;
        enviadospi[2] = 70;
        //enviadospi[1] = recibidospi[0];
        //enviadospi[2] = recibidospi[1];

        DrvSpiSlaveRxFlush();

        printf ("recibido: %i %i %i\n",
recibidospi[0],recibidospi[1],recibidospi[2]);

        DrvSpiSlaveTxStart(&enviadospi[0], 3);
        printf("ENVIADO\n");
    }
    PT_END(Pt);
}

```

Annex 6: Demo_SPI – Codi Arduino

```

#include <SPI.h>
/** ARDUINO ports definitions - GPIOs and ADCs **/
#define RESET 8 //LDR
#define MUX 2 //LDR
#define PROG_MODE A2
#define BLUE A0
#define GREEN A1
#define MMC_CS 10
#define RTX_CS 9
#define POWER 6
#define SELECT_MODE_1 7
#define SELECT_MODE_2 3

#define SHT21_ADDRESS 0x40 // Direction of the sht21

unsigned long t = 0;
byte seq_byte = 0;

void setup() {
  pinMode(RESET, OUTPUT);
  digitalWrite(RESET, LOW); //RESET ON
  delay(100);
  digitalWrite(RESET, HIGH); //RESET OFF
  pinMode(MUX, OUTPUT);
  digitalWrite(MUX, LOW); //NORMAL MODE
  pinMode(PROG_MODE, OUTPUT);
  digitalWrite(PROG_MODE, LOW); //PROG_MODE OFF
  pinMode(GREEN, OUTPUT);
  digitalWrite(GREEN, HIGH); //GREEN ON
  pinMode(MMC_CS, OUTPUT);
  pinMode(52, OUTPUT);
  digitalWrite(MMC_CS, HIGH); //SPI MMC NO SELECT
  pinMode(RTX_CS, OUTPUT);
  digitalWrite(RTX_CS, HIGH); //SPI RTX NO SELECT
  pinMode(POWER, OUTPUT);
  digitalWrite(POWER, HIGH); //ENABLE POWER RTX
  SPI.begin();
  Serial.begin(9600);
  Serial1.begin(9600);
}

void loop() {
  if (millis() > (t + 3000)) {
    //Test SPI (Arduino=Master)

    digitalWrite(RTX_CS, HIGH); //SPI.begin no ho baixa
    SPI.beginTransaction(SPISettings(1000000, LSBFIRST, SPI_MODE0));

    if (seq_byte == 254) {
      seq_byte == 0;
    } else {
      seq_byte ++;
    }
    Serial.println("Datos a enviar:");
    Serial.print("->P1: "); Serial.println(seq_byte);
    Serial.print("->P2: "); Serial.println(123);
    Serial.print("->P3: "); Serial.println(100);
  }
}

```

```
byte p1 = SPI.transfer(seq_byte);
byte p2 = SPI.transfer(123);
byte p3 = SPI.transfer(100);

Serial.println("Datos recibidos:");
Serial.print("<-P1: "); Serial.println(p1);
Serial.print("<-P2: "); Serial.println(p2);
Serial.print("<-P3: "); Serial.println(p3);
SPI.endTransaction();

digitalWrite(RTX_CS, LOW);
t = millis();
}
}
```

➤ Annex 7: Codi SoftAP

```

/**CODI softAP***/ /** Protothread del SoftAP ***/
static PT_THREAD(softAP(struct pt *Pt, const RosMailType *Mail)) {
    static struct pt childPt;
    PT_BEGIN(Pt);

    PT_SPAWN(Pt, &childPt, PtAppWifiReset(&childPt, Mail));
    //PT_SPAWN(Pt, &childPt, PtAppPrologueNoConnect(&childPt, Mail));
    //PT_SPAWN(Pt, &childPt, PtAppWifiPowerOn(&childPt, Mail));

    //SSID = MASK + MAC //Creació del SSID
    const ApiWifiMacAddrType *pMacAddr = AppWifiGetMacAddr();

    rsuint8 essid[32];
    snprintf((char*)essid, sizeof(essid), "%s%02X%02X%02X",
            SOFT_AP_ESSID, (*pMacAddr)[3], (*pMacAddr)[4], (*pMacAddr)[5]);
    AppWifiApSetSoftApInfo((rsuint8*)essid, SOFT_AP_SECURITY_TYPE,
            FALSE, 0, NULL, SOFT_AP_CHANNEL, SOFT_AP_INACT,
            (rsuint8*)SOFT_AP_COUNTRY_CODE, SOFT_AP_BEACON);

    AppData.SSID=essid;

    //Start SoftAP
    PT_SPAWN(Pt, &childPt, PtAppWifiStartSoftAp(&childPt, Mail));

    //static IP
    ApiIPv4AddressType address, subnetMask, gateway;
    inet_aton(STATIC_IP, &address);
    inet_aton(SUBNET_MASK, &subnetMask);
    inet_aton(GATEWAY, &gateway);
    AppWifiIPv4Config(TRUE, address, subnetMask, gateway, 0);

    //SendApiWifiApSetDhcpPoolReq(COLA_TASK, SOFT_AP_DHCP_POOL_START,
    SOFT_AP_DHCP_POOL_END, SOFT_AP_LEASE_TIME);

    //wait a client
    PT_WAIT_UNTIL(Pt, IS_RECEIVED(API_WIFI_CONNECT_IND));

    PT_END(Pt);
}
/**FI CODI softAP***/
/** SPI ***/
static PT_THREAD(Ptspi(struct pt *Pt, const RosMailType *Mail)) {
    static struct pt childPt;
    PT_BEGIN(Pt);

    const rsuint32 baud_rate = 3200000;

    printf("Arrancando PtDrvSpiSlaveInit\n");
    PT_SPAWN(Pt, &childPt, PtDrvSpiSlaveInit(&childPt, Mail, baud_rate));

```

```
printf("Arrancando DrvSpiSlaveInit\n");
DrvSpiSlaveInit(baud_rate);

while (1) {
    PT_WAIT_UNTIL(Pt, IS_RECEIVED(SPI_SLAVE_RX_DATA));
    RosTimerStart(APP_PACKET_DELAY_TIMER, (3000 * RS_T1MS),
&PacketDelayTimer);PT_YIELD_UNTIL(Pt, IS_RECEIVED(APP_PACKET_DELAY_TIMEOUT));

    printf("prueba de envios..\n");

    rsuint8 recibidospi[DrvSpiSlaveRxGetSize()];
    DrvSpiSlaveRx(&recibidospi[0], sizeof(recibidospi));
    AppData.Temperature = recibidospi[0];
    AppData.Humidity = recibidospi[1];
    static rsuint8 enviadospi[5];

    enviadospi[0]=AppData.Temperature;
    enviadospi[1]=AppData.Humidity;

    enviadospi[0] = recibidospi[0];
    enviadospi[1] = recibidospi[1];
    // enviadospi[2] = recibidospi[2];

    DrvSpiSlaveRxFlush();

    DrvSpiSlaveTxStart(&enviadospi[0], 3);
    printf("ENVIADO\n");

}

PT_END(Pt);
}
/**/ FI SPI /**/
```


➤ **Annex 8: Codi WebServer RTX4100**

```

/*****
*****
* Program/file: Main.c
*
* Copyright (C) by RTX A/S, Denmark.
* These computer program listings and specifications, are the property
of
* RTX A/S, Denmark and shall not be reproduced or copied or used in
* whole or in part without written permission from RTX A/S, Denmark.
*
* DESCRIPTION: Co-Located Application (COLA).
*
*****/

/*****
*****
*
*                               PVCS info
*
*****/

$Author:   mmj  $
$Date:    11 Nov 2013 10:16:30  $
$Revision: 1.16  $
$Modtime: 11 Nov 2013 10:16:06  $
$Archive: J:/sw/Projects/Amelie/COLApps/Apps/WebServer/vcs/Main.c_v
$

*/

/*****
*****
*
*                               Include files
*
*****/

#include <Core/RtxCore.h>
#include <Ros/RosCfg.h>
#include <PortDef.h>
#include <Api/Api.h>
#include <Cola/Cola.h>
#include <Protothreads/Protothreads.h>
#include <SwClock/SwClock.h>
#include <NetUtils/NetUtils.h>

#include <PtApps/AppCommon.h>
#include <PtApps/AppLed.h>
#include <PtApps/AppSocket.h>
#include <PtApps/AppWifi.h>
#include <PtApps/AppSntp.h>
#include <PtApps/AppWebconfig.h>
#include <Drivers/DrvButtons.h>
#include <Drivers/DrvIntTemp.h>
#include <Drivers/DrvApds990x.h>
#include <Drivers/DrvHIH613x.h>
#include <Drivers/DrvLps331ap.h>
#include <Drivers/DrvI2cIntf.h>
#include <Drivers/DrvLsm303dlhc.h>
#include <Drivers/DrvNtcTemp.h>

```

```

#include <Drivers/DrvAdc.h>
#include <Drivers/DrvMvs0409.h>
#include <Drivers/DrvSpiSlave.h>
#include <Drivers/DrvSpi.h>

#include <3Party/HttpParser/http_parser.h>

/*****
*
*                               Macro definitions
*
*****/
#define STATIC_IP    "192.168.1.135"
#define SUBNET_MASK  "255.255.255.0"
#define GATEWAY      "192.168.1.1"
#define HTTP_PORT    80
#define RX_IDLE_TIMEOUT (30*RS_T1SEC)
#define DATE_STR_LEN 30

// Data for configure softAP
#define SOFT_AP_ESSID      "SCK_AP_" //partial essid
#define SOFT_AP_SECURITY_TYPE  AWST_NONE
#define SOFT_AP_KEY       "rtx4lxxpass"
#define SOFT_AP_COUNTRY_CODE "FF" //ES
#define SOFT_AP_CHANNEL   2437//Channel 6 center frequency
#define SOFT_AP_INACT     10 // minutes
#define SOFT_AP_BEACON   100 // ms
#define SOFT_AP_IP        "192.168.1.135"
//#define SOFT_AP_SUBNET_MASK "255.255.255.0"
//#define SOFT_AP_GATEWAY    "192.168.1.1"

#define SOFT_AP_DHCP_POOL_START "192.168.1.200"
#define SOFT_AP_DHCP_POOL_END   "192.168.1.255"
#define SOFT_AP_SUBNET          "255.255.255.1"
#define SOFT_AP_LEASE_TIME      86400
/*****
*
*                               Enumerations/Type definitions/Structs
*
*****/
typedef struct
{
    struct pt ChildPt;
    PtEntryType* PtEntryPtr;
    rschar DateStr[DATE_STR_LEN];
    rsint16 Temperature;
    rsint16 Humidity;
    rsint32 Pressure;
    rsint16 Lux;
    rsint32 SSID;
    rsbool MultiSensor;
    rsbool HttpServerStarted;
    rsuint32 HttpInstance;
} AppDataType;

/*****
*
*                               Global variables/const
*
*****/

```

```

/*****
*****
*                               Local variables/const
*****/
static const char* const Day[] = { "Sun", "Mon", "Tue", "Wed", "Thu",
"Fri", "Sat" };
static const char* const Month[] = { "Jan", "Feb", "Mar", "Apr",
"May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};

static RsListEntryType PtList;
static AppDataType AppData;

// Timers
static const RostimerConfigType PacketDelayTimer =
    ROSTIMER(COLA_TASK, APP_PACKET_DELAY_TIMEOUT,
    APP_PACKET_DELAY_TIMER);

static const RostimerConfigType DnsRspTimer =
    ROSTIMER(COLA_TASK, APP_DNS_RSP_TIMEOUT,
    APP_DNS_RSP_TIMER);

// TCP flags
static char TCP_is_connected; // True when the TCP connection has been
established
static char TCP_received; // True when data has been received at the
TCP connection

static int socketHandle; // The socket ID of the TCP connection
static rsuint8 *TCP_receive_buffer_ptr; // TCP receive buffer
static int TCP_Rx_bufferLength; // Number of bytes received at the TCP
buffer

// Energy control
static rsuint8 is_suspended;
/*****
*****
*                               Local Function prototypes
*****/

/*****
*****
*                               Implementation
*****/
/** Genera la capçalera d'HTTP */
static rsuint32 AddHeader(rsuint8 *BufferPtr, rsuint32 BufferLength,
rsuint8* DataPtr, rsuint32 Instance)
{
    rsuint8 *p = BufferPtr;
    rsuint32 l = 0;

    l += HttpAddHeader(p+l, BufferLength-l, "Content-Type", "text/html;
charset=utf-8");
    l += HttpAddHeader(p+l, BufferLength-l, "Server", "RTX41xx demo WEB
server");
    l += HttpAddHeader(p+l, BufferLength-l, "Date", AppData.DateStr);

    return l;
}

```

```

/** Genera la web HTML, aquí és on hem de ficar el nostre codi */
static rsuint32 GenerateMainPage(rsuint8 *BufferPtr, rsuint32
BufferLength, rsuint32 Offset, rsuint8 *DataPtr, rsuint32 Instance)
{
    rsuint16 n = 0;
    rsuint16 l = BufferLength;
    rschar *p = (rschar*)BufferPtr;
    const char* const start =
        /**CODI HTML**/
        "<!DOCTYPE html>"
        "<html>"
        "<head>"
        "    <meta charset=\"utf-8\" />"
        "    <title>WEB SERVER RTX4100</title>"
        "<style>" //Codi CSS
        "body{"
        "    width: 500px;"
        "    margin: auto;"
        "    background-color: #E2ECEE;"
        "    color: FF0000;"
        "}"

        ".Contenedor {"
        "background-color: #FFFFFF;"
        "color: #000000;"
        "border: 6px groove #0E93AD;"
        "font-family: arial;"
        "text-align: justify;"
        "}"

        /*//Codi CSS per les pestanyes
        "#pestanas {"
        "float: top;"
        "font-size: 3ex;"
        "font-weight: bold;"
        "}"

        "#pestanas ul{"
        "margin-left: -40px;"
        "}"

        "#pestanas li{"
        "list-style-type: none;"
        "float: left;"
        "text-align: center;"
        "margin: 0px 2px -2px -0px;"
        "background: darkgrey;"
        "border-top-left-radius: 5px;"
        "border-top-right-radius: 5px;"
        "border: 2px solid bisque;"
        "border-bottom: dimgray;"
        "padding: 0px 20px 0px 20px;"
        "}"

        "#pestanas a:link{"
        "text-decoration: none;"
        "color: bisque;"
        "}"

        "#contenidopestanas{"

```

```

"clear: both;"
"background: dimgray;"
"padding: 20px 0px 20px 20px;"
"border-radius: 5px;"
"border-top-left-radius: 0px;"
"border: 2px solid bisque;"
"width: 1025px;"
}" */
"</style>"
"</head>"

"<body>"
"<h1>RTX4100 WEB Server</h1>"
"<form class='Contenedor'>"
  "<h2>Configuration menu</h2>"
  "<nav>"
    "<ul>"
      "<li><a href=\"setup\">Set IP config</a></li>" //Falta
afegir noves pags.
    "</ul>"
  "</nav>"

  "<form class='Contenedor'>"
    "<h3>System information</h3>"
    "<div><td><b>Date:</b></td><td>%s</td></tr></div>"
    "<div><td><b>Temperature:</b></td><td>%d.%d
C</td></tr></div>"
    "<div><td><b>Humidity</b></td><td>%i.%i
%s</td></tr></div>";

const char* const multiSensor =
  //"<tr><td><b>Pressure:</b></td><td>%i.%i hPa</td></tr>"
  "<tr><td><b>Light:</b></td><td>%i LUX</td></tr>";

const char* const end =
  "<div><td><b>SSID:</b></td><td>%d</td></tr></div>"
  //"<div><td><b>IP:</b></td><td>%d</td></tr></div>"
  "<br><div><input type='submit' value='Refresh'></div>"
  /*//Formularis, no s'ha provat
"<form action=\"setup\" method=\"POST\">"
  "<input type=\"hidden\" name=\"subject\" value=\"Formulari\">"
  "<table>"
    "<tr><td>IP</td><td><input type=\"text\"
name=\"nombre\"></td></tr>"
    "</td></tr>"
    "<tr>"
      "<td colspan=\"2\" align=\"center\">"
        "<input type='submit' name='Enviar' value='Enviar'>"
        "<input type=\"reset\"></td>"
      "</tr>"
    "</table>*/"
  "</form>"
"</body>"
"</html>";
/****FI CODI HTML****/

n = snprintf(p, l, start, AppData.DateStr, AppData.Temperature/10,
AppData.Temperature%10,AppData.Humidity%10, AppData.SSID);
if (AppData.MultiSensor)
{
  n += snprintf(p ? p+n : NULL, l, multiSensor,

```

```

        (int)AppData.Humidity/10, (int)AppData.Humidity%100,
"\%"\"RH",
        (int)AppData.Pressure/100,
(int)AppData.Pressure%100,
        (int)AppData.Lux);
    }
    n += sprintf(p ? p+n : NULL, l, end);
    return n;
}

/** Protothread: Traurem el valor de les mesures */
static PT_THREAD(PtDoMainPage(struct pt *Pt, const RosMailType *Mail)
{
    static struct pt SensorUpdatePt;

    PT_BEGIN(Pt);

    // 1. Get time //La fecha se tiene que configurar el dia que se
encienda la placa para ponerla a la hora
.http://playground.arduino.cc/Code/DateTime
    PT_WAIT_UNTIL(Pt, IS_RECEIVED(API_GET_TIME_CFM));
    if (((ApiGetTimeCfmType *)Mail)->Status == RSS_SUCCESS)
    {
        SwcTimeType absTime;
        SwcRelative2AbsoluteTime(&absTime, ((ApiGetTimeCfmType *)Mail)-
>Time);
        sprintf(AppData.DateStr, DATE_STR_LEN, "%s, %d %s %d
%02d:%02d:%02d GMT",
                Day[absTime.WeekDay], absTime.Day, Month[absTime.Month-
1], absTime.Year + 1900, absTime.Hour, absTime.Minute,
absTime.Second);
    }

    // 2. Read all sensors and update the sensor data stored in
AppExosite
    if(AppData.MultiSensor)
    {
        rsint16 data;
        rsuint16 udata;
        PT_SPAWN(Pt, &SensorUpdatePt, PtDrvAdc_Measure(&SensorUpdatePt,
Mail, &DrvNtcAdcConfig, &udata));
        AppData.Temperature = DrvNtcTempConvertValue(ntcConvTable1,
sizeofNtcConvTable1, udata);
        PT_SPAWN(Pt, &SensorUpdatePt, PtDrvAps990xGetLux(&SensorUpdatePt,
Mail, &AppData.Lux));
        PT_SPAWN(Pt, &SensorUpdatePt, PtDrvHIH613xMeasure(&SensorUpdatePt,
Mail, &AppData.Humidity));
        PT_SPAWN(Pt, &SensorUpdatePt,
PtDrvLps331apMeasure(&SensorUpdatePt, Mail, &AppData.Pressure,
&data));
    }
    else
    {
        // Measure internal temperature inside the EFM32 chip.
        PT_SPAWN(Pt, &SensorUpdatePt, PtDrvIntTempMeasure(&SensorUpdatePt,
Mail, &AppData.Temperature));
    }

    // 3. Send/generate the WEB page
    SendApiHttpServerSendResponseReq(COLA_TASK, AppData.HttpInstance,
200, "OK",

```

```

                                GenerateMainPage(NULL, 0, 0, NULL,
0), // Calc the size of the main page
                                NULL,
// Static body not used
                                AddHeader,
// Call back used to add HTTP headers
                                GenerateMainPage);
// Callback used to generate the main page
    PT_WAIT_UNTIL(Pt, IS_RECEIVED(API_HTTP_SERVER_SEND_RESPONSE_CFM) &&
                ((ApiHttpServerSendResponseCfmType*)Mail)-
>Instance == AppData.HttpInstance);

    // 4. Clear PtEntryPtr to signal that we are ready to handle a new
request
    AppData.PtEntryPtr = NULL;

    PT_END(Pt);
}

/**/ Enviament HTTP (RSS) /**/
// This function is called when HTTP GET request with path = "/" is
received
static RsStatusType OnMainPage(AhHttpMethodIdType HttpMethod, rschar
*PathPtr, rschar *QueryPtr, rsuint32 Instance)
{
    if (HttpMethod != AHM_GET)
    {
        return RSS_NOT_SUPPORTED;
    }

    // Start a protothread that will read the sensors and generate the
HTML page
    if (AppData.PtEntryPtr == NULL)
    {
        // Start a protothread that will wait for get time cfm, do sensor
reading
        // and send HTTP response message with the WEB page generated.
        AppData.PtEntryPtr = PtStart(&PtList, PtDoMainPage, NULL, NULL);

        // Request current time (CFM is handled by PtDoMainPage())
        SendApiGetTimeReq(COLA_TASK);
    }
    return RSS_SUCCESS;
}

/**/CODI softAP***/ /**/ Protothread del SoftAP /**/
static PT_THREAD(softAP(struct pt *Pt, const RosMailType *Mail)) {
    static struct pt childPt; //Protothread fill
    PT_BEGIN(Pt);

    PT_SPAWN(Pt, &childPt, PtAppWifiReset(&childPt, Mail));
    //PT_SPAWN(Pt, &childPt, PtAppPrologueNoConnect(&childPt, Mail));
    //PT_SPAWN(Pt, &childPt, PtAppWifiPowerOn(&childPt, Mail));

    //SSID = MASK + MAC //Creació del SSID
    const ApiWifiMacAddrType *pMacAddr = AppWifiGetMacAddr();

    rsuint8 essid[32];
    snprintf((char*)essid, sizeof(essid), "%s%02X%02X%02X",
            SOFT_AP_ESSID, (*pMacAddr)[3], (*pMacAddr)[4],
(*pMacAddr)[5]);

```

```

AppWifiApSetSoftApInfo((rsuint8*)ssid, SOFT_AP_SECURITY_TYPE,
    FALSE, 0, NULL, SOFT_AP_CHANNEL, SOFT_AP_INACT,
    (rsuint8*)SOFT_AP_COUNTRY_CODE, SOFT_AP_BEACON);

AppData.SSID=ssid;

//Start SoftAP
PT_SPAWN(Pt, &childPt, PtAppWifiStartSoftAp(&childPt, Mail));

//static IP
ApiIPv4AddressType address, subnetMask, gateway;
inet_aton(STATIC_IP, &address);
inet_aton(SUBNET_MASK, &subnetMask);
inet_aton(GATEWAY, &gateway);
AppWifiIpv4Config(TRUE, address, subnetMask, gateway, 0);

//SendApiWifiApSetDhcpPoolReq(COLA_TASK, SOFT_AP_DHCP_POOL_START,
SOFT_AP_DHCP_POOL_END, SOFT_AP_LEASE_TIME);

//wait a client
PT_WAIT_UNTIL(Pt, IS_RECEIVED(API_WIFI_CONNECT_IND));

PT_END(Pt);
}
/****FI CODI softAP****/

/**/ Protothread: Crida al SoftAP ***/
static PT_THREAD(PtMain(struct pt *Pt, const RosMailType* Mail))
{
//Protothread fill
static struct pt childPt;
PT_BEGIN(Pt);

// Set power save parameters
AppWifiSetPowerSaveProfile(POWER_SAVE_HIGH_IDLE); // 200ms idle
interval
AppWifiSetListenInterval(100); // 100ms

//Iniciar el protohothread del softAP com fill
PT_SPAWN(Pt, &childPt, softAP(&childPt, Mail));

// Start HTTP server
SendApiHttpServerInitReq(COLA_TASK, 80, NULL);
PT_WAIT_UNTIL(Pt, IS_RECEIVED(API_HTTP_SERVER_INIT_CFM) &&
((ApiHttpServerInitCfmType*)Mail)->Port == 80);
if (((ApiHttpServerInitCfmType*)Mail)->Status == RSS_SUCCESS)
{
// Server started
AppData.HttpServerStarted = TRUE;
AppData.HttpInstance = ((ApiHttpServerInitCfmType*)Mail)-
>Instance;

// Add main page
SendApiHttpServerAddResourceReq(COLA_TASK, AppData.HttpInstance,
"/", OnMainPage);
}
else
{
// FIX ME: Failed to start HTTP server
}
}

```



```

// Request current time using Sntp
AppSntpStartTimeSync(&PtList, NULL);
PT_WAIT_UNTIL(Pt, IS_RECEIVED(APP_EVENT_Sntp_SYNC_DONE) ||
IS_RECEIVED(APP_EVENT_Sntp_SYNC_FAILED));

while (1)
{
// Yield to allow other tasks to run
PT_YIELD(Pt);

if (IS_RECEIVED(KEY_MESSAGE))
{
if (Mail->Pl.Pl == KEY_WPS)
{
// Stop the HTTP server
if (AppData.HttpServerStarted)
{
SendApiHttpTerminateReq(COLA_TASK, 80);
PT_WAIT_UNTIL(Pt, IS_RECEIVED(API_HTTP_TERMINATE_CFM) &&
((ApiHttpTerminateCfmType*)Mail)->Instance == AppData.HttpInstance);
AppData.HttpServerStarted = FALSE;
}

// Do WPS!
AppLedSetLedState(LED_STATE_WPS_ONGOING);
PT_SPAWN(Pt, &AppData.ChildPt,
PtAppWifiDoWps(&AppData.ChildPt, Mail));
AppLedSetLedState(LED_STATE_IDLE);

// WPS done -> restart to reconnect
PT_RESTART(Pt);
}
}
}

PT_END(Pt);
}

/** SPI **/
static PT_THREAD(Ptspi(struct pt *Pt, const RosMailType *Mail)) {
static struct pt childPt;
PT_BEGIN(Pt);

const rsuint32 baud_rate = 32000000;

printf("Arrancando PtDrvSpiSlaveInit\n");
PT_SPAWN(Pt, &childPt, PtDrvSpiSlaveInit(&childPt, Mail,
baud_rate));

printf("Arrancando DrvSpiSlaveInit\n");
DrvSpiSlaveInit(baud_rate);

while (1) {
PT_WAIT_UNTIL(Pt, IS_RECEIVED(SPI_SLAVE_RX_DATA));
RosTimerStart(APP_PACKET_DELAY_TIMER, (3000 * RS_T1MS),
&PacketDelayTimer);PT_YIELD_UNTIL(Pt,
IS_RECEIVED(APP_PACKET_DELAY_TIMEOUT));

printf("prueba de envios..\n");

rsuint8 recibidospi[DrvSpiSlaveRxGetSize()];

```

```

        DrvSpiSlaveRx(&recibidospi[0], sizeof(recibidospi));
        AppData.Temperature = recibidospi[0];
        AppData.Humidity = recibidospi[1];
        static rsuint8 enviadospi[5];

        enviadospi[0]=AppData.Temperature;
        enviadospi[1]=AppData.Humidity;

        enviadospi[0] = recibidospi[0];
        enviadospi[1] = recibidospi[1];
        // enviadospi[2] = recibidospi[2];

        DrvSpiSlaveRxFlush();

        DrvSpiSlaveTxStart(&enviadospi[0], 3);
        printf("ENVIADO\n");

    }

    PT_END(Pt);
}
/**/ FI SPI ***/

void ColaTask(const RosMailType* Mail)
{
    // Pre-dispatch mail handling
    switch (Mail->Primitive)
    {
        case INITTASK:
            // Init the Buttons driver
            DrvButtonsInit();

            // Init sensor drivers
            DrvIntTemp_Init();
            if (DrvLps33lap_Init())
            {
                AppData.MultiSensor = TRUE;
                DrvApds990x_Init();
                DrvHIH613x_Init();
                DrvAdc_Init(&DrvNtcAdcConfig);
            }

            // Init the Protothreads lib
            PtInit(&PtList);

            // Init the LED application
            AppLedInit(&PtList);

            // Init the WiFi management application
            AppWifiInit(&PtList);

            // Start the Main protothread
            PtStart(&PtList, PtMain, NULL, NULL);
            break;

        case TERMINATETASK:
            RosTaskTerminated(ColaIf->ColaTaskId);
            break;

        case API_GPIO_INTERRUPT_IND:

```

```
        // Dispatch API_GPIO_INTERRUPT_IND to the button driver and
return
        DrvButtonsOnMail(Mail);
        return;
    }

    // Dispatch mail to all protothreads started
    PtDispatchMail(&PtList, Mail);
}

// End of file.
```