Universitat Oberta de Catalunya

Treball de fi de grau

Disseny i implementació del sistema de gestió web del Smart Citizen Kit

Treball de fi de grau en Tecnologies de Telecomunicació

Jordi Casas Marfil

Supervisat per:

Pere Tuset Peiró

Juny 12, 2015

Llicència

Agraïments

En primer lloc, voldria agrair a l'empresa Smart Citizen, per donar-me la oportunitat de realitzar aquest projecte final de grau, facilitant-me les eines necessàries per la seva realització. Concretament a Alex i Guillem, que han sigut les figures més visibles amb les que he estat en contacte.

També voldria agrair el suport proporcionat pel Miguel Angel de Heras, de la fundació Hangar. Encara que vam parlar poc, les converses van ser molt productives i em van ajudar a solucionar problemes sorgits del bus SPI.

Finalment, també donar les gràcies a en Pere Tusset, per guiar-me al llarg de tot el projecte i per tots els consells i suggeriments que m'ha proporcionat al llarg d'aquest.

Resum

El propòsit d'aquest projecte consisteix en facilitar l'administració del dispositiu SCK[1] (Smart Citizen Kit), el qual proveeix l'empresa Smart Citizen[2]. Aquest dispositiu es basa principalment en la plataforma Arduino[3] i el seu propòsit és enviar els resultats proporcionats pels seus sensors (temperatura, humitat, llum), a un servidor centralitzat per tractar aquesta informació i realitzar estudis ambientals. Actualment SCK està usant un chip anomenat RN131[4] per a la dur a terme la comunicació WiFi, però està estudiant la possibilitat d'usar un nou chip que porti a terme aquesta comunicació; en concret el chip RTX4100[5] de l'empresa RTX.

Els objectius principals d'aquest projecte consisteixen en crear un servidor web i configurar el mode AP pel dispositiu. D'aquesta manera, quan un usuari vulgui modificar la configuració del SCK, només tindrà que prémer un botó (per executar el AP virtual) i connectar-se. Un cop l'usuari s'hagi connectat, podrà configurar el dispositiu mitjançant una interfície web proporcionada pel mateix RTX. El principal avantatge que presentarà aquesta funcionalitat serà la d'eliminar tot requisit, per dur a terme la configuració. En altres paraules, qualsevol dispositiu mòbil amb WiFi, podrà configurar el dispositiu.

Per a dur a terme el canvi de mòdul WiFi (RN131 a RTX4100), es disposarà d'una placa de desenvolupament anomenada Smart Citizen DVK[6] (Development Kit). Al ja existir un estudi[7] previ per aquest mòdul WiFi, el qual va ser realitzat pel Sr. Miguel Colom, el present treball tindrà la finalitat d'ampliar el codi i la funcionalitat existent.

En quant al desenvolupament, s'usarà Amelie[8] (SDK proporcionat per RTX). Aquest està basat en C i servirà per desenvolupar l'aplicació desitjada.

Paraules clau

Smart Citizen

RTX4100

WiFi

Abstract

The purpose of this work is to facilitate the administration of the device SCK[1] (Smart Citizen Kit), which provides the company Smart Citizen.[2] This device is mainly based on the Arduino platform[3] and its purpose is to send the results provided by the sensors (temperature, humidity, light) to a centralized server for posterior analysis. SCK is currently using a chip called RN131[4] to carry out communication WiFi, but Smart Citizen is studying the possibility of using a new chip for the Wi-Fi communication(specifically the RTX4100 chip[5] produced by RTX.)

The main objectives of the project, is to create a web server and configure the soft AP mode. Thus, when a user wants to change AP settings, he need only press a button (for set soft AP mode) and connect to it. Once the user has connected, he can configure the device using a web interface, which will be provided by the same RTX41xx Wi-Fi Module. The main advantage of this function will be to eliminate any requirement to configure the device. In other words, any mobile device with WiFi capability can be used to configure the device.

To carry out the change in WiFi module (RN131 to RTX4100), we will have a development board called Smart Citizen DVK[6] (Development Kit). Because there is preliminary work[7] done by Mr. Miguel Colom, this work will seek to expand its code and its functionality.

In terms of development, we will use the SDK provided by the company RTX, creators of the WiFi module. This SDK called Amelie[8], is based on C and will be used to develop the desired application.

Índex

Llistats de recursos

1 Introducció

1.1 Motivació

El present projecte de fi de grau, sorgeix de la necessitat que té l'empresa Smart Citizen en desenvolupar un sistema que faciliti la configuració del seu sistema. A aquest canvi, se li suma la possibilitat d'elegir un nou mòdul WiFi amb més funcionalitat que l'anterior. Aquest mòdul serà el RTX4100, desenvolupat per l'empresa RTX, el qual millora notablement la capacitat del antic model (RN131) afegint capacitats addicionals com la de ser directament programat. Això vol dir que certes tasques relacionades amb la comunicació, es podran dur a terme en el mateix chip i no serà necessari ocupar cicles de rellotge del sistema base Arduino.

Aquest treball buscarà la millor forma de desenvolupar aquest software, sent ho més transparent possible per l'usuari i permetent que pugui ser re-configurat amb facilitat. El punt fort del treball consistirà en programar les rutines necessàries per tal de que el chip RTX4100 compleixi aquest requisit, sense oblidar que s'haurà de fer un desenvolupament ho més modular possible per a permetre una fàcil modificació en cas que sigui necessària.

1.2 Objectius principals

El present treball busca facilitar l'accés al sistema, per part dels usuaris del SCK. La principal fita consistirà en no dependre de cap plataforma ni de cap sistema per a poder accedir o modificar la configuració. Per a aconseguir aquesta fita, es realitzaran els següents desenvolupaments:

1. Implementació un servidor web dins del mòdul RTX4100, que sigui capaç d'interaccionar amb el SCK (Sensors, EEPROM...). La web que vindrà inclosa en aquest servidor web, facilitarà les següents funcions:

a. Visualització/configuració de paràmetres relacionats amb la connectivitat. Es facilitarà la manipulació dels paràmetres que permetran connectar l'aparell a Internet. És a dir, quan es desactivi el mode de configuració (que inclou el soft AP), i el SCK passi a ser un client, aquest serà capaç de connectar-se a una altre xarxa WiFi, mitjançant els paràmetres introduïts en aquesta pàgina web.

b. Visualització/configuració de paràmetres relacionats amb els sensors, facilitant l'accés a l'estat dels sensors i les seves mesures.

2. Implementar un AP virtual: El dispositiu haurà de ser capaç d'emular un AP. Consegüentment, un altre dispositiu amb WiFi, podrà establir una connexió directa i connectar-se al servidor explicat en el punt anterior. Cal dir, que aquest mode serà temporal (el RTX4100 pot funcionar en mode Client o mode AP). Per tant, si s'entra en mode AP, el SCK no serà capaç d'enviar les seves dades al servidor de Smart Citizen. No obstant, quan es recuperi la connexió al servidor, es podran enviar les dades que prèviament no han pogut ser entregades.

1.2.1 Objectius específics

Encara que en l'apartat anterior s'han comentat els objectius d'aquest treball, en aquest punt s'especificaran en major detall d'aquests objectius:

- Desenvolupament d'un mode soft AP pel RTX4100, que compleixin els següents requisits:
    - Activació del mode, mitjançant un comandament per SPI[9]: En aquest punt, s'ampliarà la API i es crearà un codi que accionarà aquesta funció.

- Creació d'una xarxa oberta amb un SSID únic, consistent d'un denominador comú i un identificar únic.

- Redireccionament de les peticions TCP a la IP del propi mòdul. Aquest punt consistirà en desenvolupar un petit servidor DNS, que traduirà totes les peticions, a la IP del AP. Conseqüentment, qualsevol petició que usi DNS, es redirigirà a la IP del propi mòdul.

- Desenvolupament d'un servidor web que faciliti la visualització/configuració de paràmetres mitjançant l'intercanvi de dades amb el MCU del SCK, i que compleixin els següents requisits:

  - Activació del mode, mitjançant un comandament per SPI: En aquest punt, s'ampliarà novament la API i es crearà un codi que accionarà aquesta funció.

  - Suportar parcialment HTTP/1.1 (Funcionalitat GET/POST): En aquest apartat, es desenvoluparà un petit servidor web. No s'utilitzarà el proveït pel SDK, degut a que presenta unes limitacions no salvables, com es veurà en el següent punt.

  - Realitzar el desenvolupament tenint en compte que el mòdul RTX4100 només proporciona 3kB de RAM per les aplicacions d'usuari. Per resoldre aquest problema, s'usarà la targeta SD prevista pel kit DVK. Aquest fet provocarà tenir que crear una nova funció en la API, que serà cridada via SPI. No s'usarà el servidor Web inclòs(HTTP_API), degut a que aquest no permet controlar les respostes del servidor cap al client a nivell de paquet, característica necessària per a enviar fitxers amb mida superior a la memòria del dispositiu.

Segons aquests requisits, es desenvoluparan les aplicacions: servidor DNS i servidor Web. També s'ampliarà la API actual que utilitza SCK per comunicar-se amb el mòdul RTX4100.

1.3 Beneficis

Existiran dos beneficis claus, que faran que el projecte cobri sentit. Aquests són:

1. Interoperabilitat: Qualsevol usuari amb accés físic al SCK, serà capaç de connectar-se mitjançant el seu dispositiu mòbil (o qualsevol dispositiu que tingui WiFi). Per tant, el major benefici serà la interoperabilitat entre dispositius i la possibilitat de poder realitzar tasques de control, com la lectura de sensors o modificar informació del dispositiu sense complir uns requisits de software específics.

2. CPU pròpia: Gracies al chip RTX4100, es podran desenvolupar aplicacions que ataquin directament contra la CPU d'aquest chip (24 kByte flash, 3 kByte RAM). Això permetrà no usar la CPU d'Arduino i poder reservar espai de memòria i cicles de rellotge per les funcions pròpies de la SCK (per exemple sensors). Per tant, tota la part de configuració WiFi i servidor Web quedarà localitzada en el corresponent mòdul.

1.4 Estructura del document

Al llarg del present document, s'explicaran diferents aparts en els que es recull tota la informació relacionada amb el desenvolupament del projecte. A continuació s'indiquen quins són aquests apartats:

Inicialment es realitzarà una breu introducció del hardware i el software del mòdul RTX4100, amb el que s'ha desenvolupat l'aplicació.

Seguidament es realitzaran unes pinzellades sobre SPI (s'explicaran característiques i configuracions ha tenir en compte per a que els dispositius interactuïn correctament). Un cop fetes es farà una introducció a la placa DVK, que és sobre la que es realitzarà el desenvolupament. En aquest moment es farà un petit resum als protocols que s'han d'implementar (DNS & HTTP).

Arribats a aquest punt i amb tots els conceptes clars, es començarà amb el desenvolupament pròpiament dit. Primer es comentarà quin és l'estat actual del codi i es revisarà la API actual per si s'ha de modificar. Posteriorment es mostraran els aspectes més significatius del codi, incloent el funcionament bàsic de l'aplicació i les funcions més importants.

En quants als annexos, s'incorporarà la documentació rellevant del projecte. Dins d'ella es trobarà:

- Hardware DVK: S'entregaran els diagrames electrònics juntament amb els components usats en la placa de desenvolupament.

- Documentació del codi: Es veurà una descripció de totes les funcions i les relacions entre elles. En aquest apartat s'inclourà tant la documentació de RTX com la d'Arduino.

2 Disseny del software

2.1 Justificació del canvi de mòdul WiFi

Abans de començar a explicar com funciona el nou chip WiFi, és interesant conèixer la raó per la qual l'empresa Smart Citizen, decideix realitzar aquest canvi, amb totes les modificacions que comporta.

Actualment la versió actual del SCK utilitza el mòdul RN131. Aquest mòdul, suporta 802.11 b/g i té un sistema propi de comandes per a poder configurar-lo. Malauradament, no existeix cap SDK obert per a treballar directament sobre el seu firmware, i per tant només pot ser programat mitjançant les citades comandes.

Aquest fet obliga a Smart Citizen a buscar un substitut que s'adapti millor en un sistema integrat. És en aquest moment on sorgeix la idea de substituir el aquest mòdul pel de RTX4100. Això és deu a que en el cas del RTX4100 sí que és possible desenvolupar aplicacions pròpies, mitjançant el corresponent SDK.

A part d'aquest avantatge, el nou mòdul disposa d'un consum energètic menor en suspensió (estat on pesarà el major temps). Aquest consum es pot veure en la Taula 1.

	RX(mA)	TX(mA)	standby(mA)	sleep(uA)
RN131	40	212	15	4
RTX4100	100	300	0.76-2.6 (temps d'espera: 100ms-1s)	2.7

Taula 1: Consums de potència entre els mòduls RN131 i RTX4100

Com a conseqüència, Smart Citizen decideix fer les modificacions adients tant en el hardware com en el software per usar aquest nou mòdul.

2.2 Introducció al mòdul WiFi RTX4100

El mòdul RTX4100 WiFi, consisteix en un mòdul que suporta l'estàndard 802.11b/g/n[10] i té una MCU EFM32[11] de baixa potencia. Aquest mòdul està enfocat en aplicacions que envien dades de forma pausada; de fet, està enfocat a aplicacions de baix consum.

El hardware amb el que es centra aquest treball serà el model RTX4100 (el qual munta el model de MCU EFM32G230F128, corresponent a la família *Gecko*). El model superior és el RTX4140 (el qual munta el model EFM32GG230F1024 que correspon a la família *Giant Gecko*). Les principals diferencies es poden veure en la Taula 2.

Product Families

Family	Core	Speed (MHz)	Flash Memory (kB)	RAM	USB	LCD	Communications
Zero Gecko	ARM Cortex-M0+	24	4, 8, 16, 32	2, 4	No	No	I2C, I2S, SPI
Tiny Gecko	ARM Cortex-M3	32	4, 8, 16, 32	2, 4	No	Yes	I2C, I2S, SPI, USART
Gecko	ARM Cortex-M3	32	16, 32, 64, 128	8, 16	No	Yes	I2C, SPI, UART, USART
Leopard Gecko	ARM Cortex-M3	48	64, 128, 256	32	Yes	Yes	I2C, I2S, SPI, UART, USART
Giant Gecko	ARM Cortex-M3	48	512, 1024	128	Yes	Yes	I2C, I2S, SPI, UART, USART
Wonder Gecko	ARM Cortex-M4	48	64, 128, 256	32	Yes	Yes	I2C, I2S, SPI, UART, USART

Taula 2: Comparació entre microcontroladors EFM32. Font: Silabs[9]

Aquest canvi en la MCU afecta directament al mòdul WiFi. Concretament de la següent manera: mentre que en el RTX4100 només hi ha 24kB de flash i 3kB de RAM per les aplicacions, en el model RTX4140 ofereix 512kB i 64kB de RAM segons es pot veure en la pàgina web del fabricant[5].

2.2.1 Arquitectura del sistema RTX:

El mòdul WiFi RTX4100, conté dos blocs ben diferenciats. Per una part el mòdul WiFi (Qualcomm Atheros AR4100P[12]) i per una altre l'aplicació MCU. Aquesta última conté tot els components de software necessaris per a implementar el dispositiu WiFi (incloent l'aplicació).

El diagrama de la Figura 1, il·lustra les capes d'aplicació que proporciona el MCU. En tot cas, aquesta arquitectura està dividida en dos parts ben diferenciades:

1) Aplicació d'usuari: També coneguda com Co-Located Application (CoLA).

2) Firmware del mòdul: Conegut com a Plataforma del Firmware.

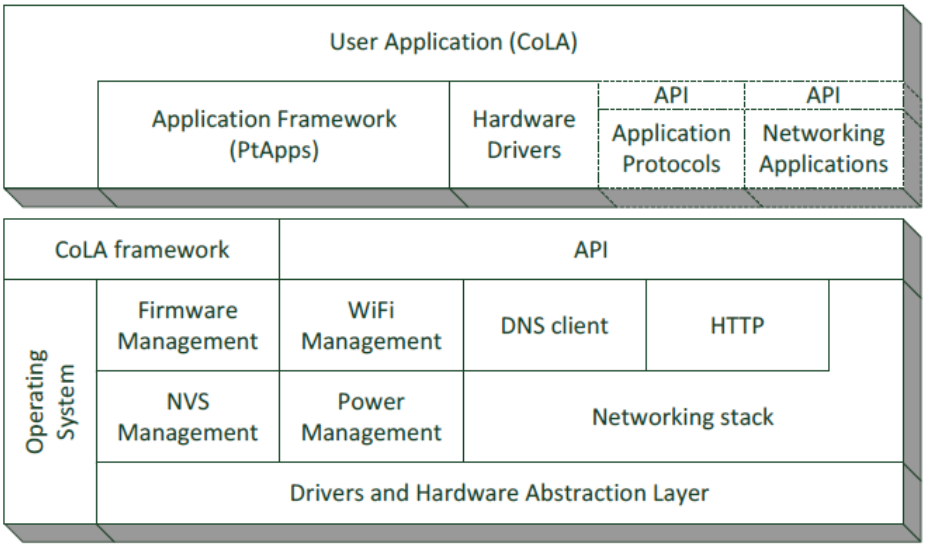


Figura 1: Capes de l'aplicació del RTX4100. Font: Amelie SDK[8]

Per defecte, el mòdul RTX usa una versió de firmware que suporta les aplicacions CoLA. Aquestes aplicacions usaran les API que proporciona el firmware i el framework de CoLA. Les aplicacions desenvolupades, en cap moment sobreescriuran el firmware del mòdul, ja que només es sobreescriurà la part de CoLA).

A continuació es passa a descriure més detalladament els dos blocs que formen l'arquitectura de RTX:

1) Blocs de les aplicacions basades en CoLA:

    a) *Aplicació d'usuari*: Component que implementa la funcionalitat de l'aplicació en el chip WiFi. És la part que es programa amb l'objectiu d'adaptar el mòdul als requisits desitjats.

    b) *Framework de l'aplicació*: És un conjunt de petites aplicacions, les quals solen ser re-usades per les aplicacions d'usuari. Es pot dir que són eines bàsiques (servidor web, servidor DHCP)...

    c) *Drivers de hardware*: És el component que implementa els drivers més usuals. Un bon exemple usat en el present projecte, són els drivers SPI.

    d) *Protocols d'aplicació*: Aquest bloc és opcional i en cas d'usar-se, proporciona protocols específics. Exemples d'aquests protocols poden ser: codificació/de-codificació de missatges XML, anàlisis de missatges... Aquestes aplicacions solen ser accessibles mitjançant una API.

    e) *Aplicacions de xarxa*: Aquest bloc també és opcional i ofereix funcionalitats d'aplicacions basades en xarxa. Alguns exemples són: servidors de FTP, HTTP... De la mateixa manera que els protocols d'aplicació, aquestes aplicacions solen ser usades mitjançant la seva corresponent API.

2) Blocs del firmware del mòdul:

    a) *Co-Located Application (CoLA) Framework*: Aquesta capa implementa un model en el que l'aplicació enllaça directament amb les llibreries dels blocs inferiors. L'aplicació és compilada i enllaçada com un programa separat, posteriorment serà carregat com una tasca en el sistema operatiu del qual es parlarà pròximament.

b) *API*: És una interfície proporcionada per la plataforma de firmware, la qual proporciona totes les funcionalitats necessàries per implementar la part WiFi. Totes les API's segueixen el model de missatge la qual es parlarà en la part del sistema operatiu.

c) *Sistema Operatiu*: RTX disposa d'un sistema operatiu el qual implementa les funcionalitats internes necessàries per les aplicacions basades en CoLA.

d) *Stack de xarxa*: Aquesta capa implementa la capa IP (IPv4 i IPV6) al sistema.

e) *DNS Client*: Servei que proporciona un petit client DNS.

f) *HTTP*: Breu implementació d'un servidor/client HTTP. Aquest bloc és especialment interesant pel present treball. En qualsevol cas, com s'ha dit en la part anterior, tindrà que ser descartat ja que no podrà complir els requisits de Smart Citizen. El requisit que no pot complir, es basa en poder retornar peticions parcialment en base a fer peticions via SPI contra altres parts del SCK.

g) *Administració WiFi*: Aquest component controla aspectes de la connexió tals com mode WiFi, tipus de seguretat y/o gestió d'energia.

h) *Gestió d'energia*: Aquest component és capaç de controlar el rellotge intern de la MCU. El seu propòsit serà minimitzar al màxim l'ús de l'energia, amb l'objectiu de prolongar la vida de la bateria.

i) *Administració de firmware*: Aquest component proporciona la possibilitat d'efectuar una actualització en remot de l'aplicació CoLA.

j) *Administració de NVS*: El RTX4100 implementa una memòria NVS (Non-Volatile Storage) en una part de la flash interna de la MCU, amb l'objectiu de poder salvaguardar informació que ha de ser persistent.

k) *Drivers*: Aquesta capa implementa tant els drivers per poder treballar amb els perifèrics de la MCU, com els drivers per poder treballar amb la interfície física del mòdul WiFi.

2.2.2 El sistema operatiu ROS

El sistema operatiu de RTX, anomenat ROS (RTX Operating System), està enfocat a realitzar implementacions al voltant d'una gran varietat d'aplicacions. La principal característica de ROS consisteix en ser un sistema cooperatiu basat en un sistema de missatges, que permet la comunicació entre tasques que poden no ser relacionades entre si. Un fet interesant és que les tasques poden estar executant-se indefinidament en mode IDLE, i recuperar el control quan un missatge ho indiqui. Aquestes tasques, treballen amb protothreads els quals són processos lleugers i optimitzats per aquests tipus de sistemes.

A continuació és mostren els trets més característics d'aquests sistema:

1) Sistema de missatges ROS: El sistema ROS usa els missatges per a poder comunicar les tasques que s'estan executant. Els missatges enviats, s'identifiquen amb una etiqueta, anomenada primitiva i una sèrie de paràmetres.

   Una tasca, serà capaç d'enviar-se missatges a ella mateixa, funció que rebrà sentit quan un protothread necessiti comunicar-se amb un protothread que està en un estat d'espera.

   Aquests missatges reben especial sentit en l'ús de drivers, on aquests seran capaços d'enviar missatges davant certs esdeveniments (rebre un paquet SPI per exemple).

2) Actuació davant l'entrada/sortida de missatges: La funció de la ColaTask consisteix en mantenir els processos controlats. Per a fer-ho, contempla almenys dos tipus de missatges que seran usats sempre. Aquests corresponen a la inicialització i la terminació de la tasca. Al realitzar la inicialització, normalment també es duu a terme la inicialització del hardware en el framework.

En el cas de que es necessites redistribuir els missatges a altres aplicacions, s'usa la funció PtDispatchMail();

3) ROS Timer System: ROS també disposa de temporitzadors, que són capaços d'enviar missatges a les tasques. Aquestes tasques es podran subscriure a aquests temporitzadors de manera individual.

4) Non Volatile Storage: El sistema també facilita una part de la flash amb funcionalitat de guardar dades. Aquesta informació pot ser manipulada mitjançant les funcions NvsRead() i NvsWrite().

   També es disposa de les funcions ColaNvsOffset() i ColaNvsSize() les quals marquen l'inici i la mida d'aquesta memòria. Mitjançant això, es podrà recuperar la informació desitjada.

   Un exemple que recupera el 8é byte de la NVS i el deposita en la variable r, és el següent:

```
rsuint 8 r;
NvsRead(ColaaNvsOffset+8, sizeof(b), &b);
```

2.2.3 Recursos de Hardware

Existeixen una sèrie de recursos que proporciona el MCU i que és interesant conèixer-los, ja que alguns d'ells seran usats en aquest projecte. Abans d'això cal dir que es podrà atacar directament contra la MCU per controlar els drivers, sempre que aquests puguin ser prèviament desactivats pel FW RTX. Per a controlar-los, s'haurà de fer ús de les llibreries "emlib", les quals estan incloses en el SDK de RTX.

Els recursos facilitats per RTX són els següents:

- *LEUART1*: Utilitzat pel RTX EAP UART. Interfície que va a una velocitat de 9600 bps. Pins PC6, PC7.

- *USART1*: Utilitzat pel RTX EAP UART. Interfície que va a una velocitat de 115200 bps. Pins PA4/PC0, PA3/PC1. Usat per les descarregues de les aplicacions CoLA mitjançant el corresponent controlador.

- *USART0*: Utilitzat per les comunicacions SPI entre la MCU i el chip Atheros (WiFi). Els pins usats son PE9-PE13. En qualsevol cas aquests no podran ser usats, ja que la seva funcionalitat no pot ser desactivada.

- *GPIO ISR*: Les interrupcions GPIO són gestionades pel mòdul FW. Quan una interrupció es detecta, serà enviada a la CoLA mitjançant missatges de comunicació.

- *GPIO's*: Els pins PF5, PA5 i PA6 s'usen internament per la MCU i no poden ser modificats. El PB13, es pot usar per habilitar una font d'alimentació externa. En tot cas, aquesta característica s'haurà d'habilitar des de la CoLA mitjançant el missatge corresponent. Tots els pins restants estaran disponibles.

- *Gestió de rellotge*: El mòdul del FW activa/desactiva els dos rellotges que disposa la MCU per estalviar energia. Aquest mòdul disposa d'un rellotge d'alta freqüència (28MHz) i un rellotge de baixa freqüència (32768kHz).

- *RTC*: El rellotge en temps real és usat pel temporitzador que disposa el sistema ROS.

- *Cortex-M3 SysTick*: És utilitzat pel mòdul del FW per implementar un comptador força sensible, el qual es capaç de treballar sota una resolució de 28MHz.

- *Interfície de depuració*: Els pins PF0 i PF1 poder ser usats per connectar algun depurador. Si no s'usen podran ser usats com a GPIO.

Totes aquestes característiques, es poden veure en la Taula 3. La numeració dels pins la podrem veure en la Figura 2.

N. pin	Nom del pin	Tipus	N. pin	Nom del pin	Tipus
1	GND	GND	16	PB14	I/O
2	PA0	I/O	17	GND	GND
3	PA1	I/O	18	PD5	I/O
4	VCC2	VCC2	19	PC6	I/O
5	GND	GND	20	PC7	I/O
6	VCC1	VCC1	21	PC5/PB11	I/O
7	VIO	0	22	PC4/PD4	I/O
8	PA3/PC1	I/O	23	PC2	I/O
9	PA4/PC0	I/O	24	PC3	I/O
10	PB7	I/O	25	PF0	I/O
11	PB8	I/O	26	PF1	I/O
12	GDN	GND	27	PF2	I/O
13	RESETn	I-PU	28	GND	GND
14	PB12	I/O	29	ANT	I/O
15	PB13	I/O	30	GND	GND

Taula 3: Recursos de hardware del RTX4100

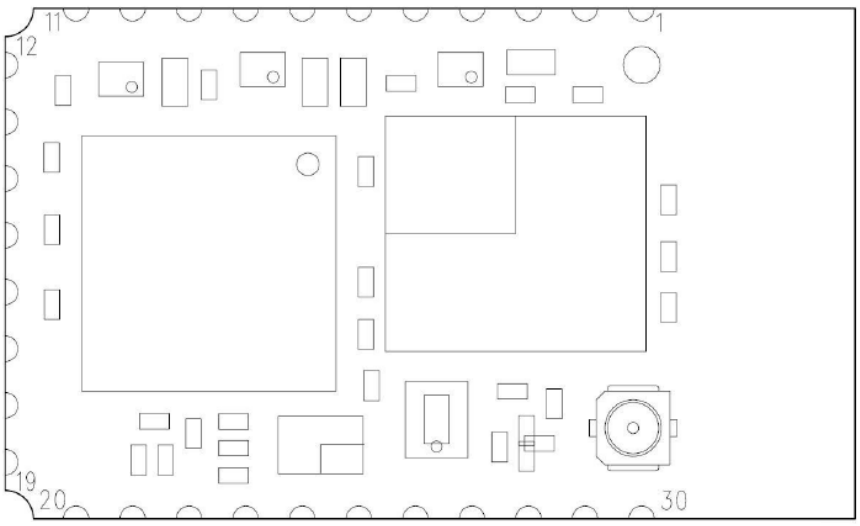


Figura 2: Esquema gràfic dels pins de la placa RTX. Font: Amelie SDK[8]

2.2.4 Exemple d’un desenvolupament

Després de veure tota la base teòrica sobre el RTX i ROS és necessari acabar d’assolir els conceptes bàsics mitjançant un exemple. Dit exemple es tractarà de l’aplicació Blinky, la qual és proporcionada pel SDK de RTX. L'aplicació en qüestió farà parpellejar un LED. (Sembla evident pensar que es realitzarà mitjançat l’ús de temporitzadors).

L'exemple, tindrà en compte 3 possibles estats, els quals estaran caracteritzats pels missatges rebuts:

1) INITTASK: L'aplicació es posa en marxa.
2) TERMINATETASK: ROS rep l'ordre d'apagar el sistema.
3) TIMEOUT: El temporitzador finalitza.

A continuació es descriuran les parts que formaran l’aplicatiu, el qual s’ha de recordar es basarà en el llenguatge de programació C.

I. Càrrega de llibreries externes:

```
//Definicions estàndards de RTX (definicions, estructures...)
#include <Core/RtxCore.h>

//Definicions de ROS (Tipus de missatges, definició de tasques)
#include <Ros/RosCfg.h>

// Definicions de ports I/O (Leds, polsadors...)
#include <PortDef.h>

// Prototips de funcions per les trucades a la API
#include <Cola/Cola.h>

// Definició de ports del EFM
#include <em_gpio.h>
```

II. Variables locals:

```
RosTimerConfigType AppLedTimer = ROSTIMER(COLA_TASK, TIMEOUT,
                                          APP_LED_TIMER);
```

Aquesta instrucció, defineix un temporitzador (RosTimerConfigType). Els paràmetres que rep són: La tasca, la primitiva i el contingut del missatge. En concret s'està definint un missatge amb primitiva TIMEOUT, que tindrà com a paràmetre APP_LED_TIMER.

III. Funció principal:

```
void ColaTask(const RosMailType* Mail)
```

Aquesta és la funció principal que s'executarà al iniciar la CoLA. És l'analogia a la funció main() de C. En el cas tractat, aquesta funció completa serà:

```
void ColaTask(const RosMailType* Mail)
{
  switch (Mail->Primitive)
  {
    case INITTASK:
      GPIO_PinModeSet(LEDPORT, LEDPIN, gpioModePushPull, 1);
      RosTimerStart(APP_LED_TIMER, 1 * RS_T1SEC, &AppLedTimer);
      break;

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

    case TIMEOUT:
      switch (Mail->Timeout.Parameter)
      {
        case APP_LED_TIMER:
          GPIO_PinOutToggle(LEDPORT, LEDPIN);
          RosTimerStart(APP_LED_TIMER, 5 * RS_T100MS, &AppLedTimer);
          break;
      }
      break;

    default:
      break;
  }
}
```

Explicades aquestes parts, ja es pot passar a descriure el que fa l'aplicació:

1. Al iniciar l'aplicació es rebrà un Mail amb la Primitiva INITTASK. Al rebre'l, es donarà corrent al PIN que fa referencia al LED i s'iniciarà el temporitzador.

   Com a paràmetres s'està passant el paràmetre enviat al complir-se el temporitzador (APP_LED_TIMER) i el temps que es triga a fer saltar el temporitzador (1*1SEC = 1s). També es passarà l'objecte del temporitzador que s'ha configurat al principi (en variables locals).

2. Quan hagi passat un segon, es rebrà un nou missatge. Aquest tindrà la primitiva TIMEOUT i el paràmetre APP_LED_TIMER (tal com s'ha definit en el temporitzador del INITTASK). En aquest punt, es commutarà el valor del PIN del LED (per encendre o apagar-lo) i s'establirà de nou un temporitzador (en aquesta ocasió de 5 segons). En aquest moment s'entrarà en un bucle i cada 5 segons s'executarà el mateix. Això serà fins que s'apagui el sistema (lògicament si es fa de forma controlada). En aquest cas, es passarà a la primitiva TERMINATETASK.

3. La última primitiva, TERMINATETASK, crida la funció ***RosTaskTerminated(ColaIf->ColaTaskId);*** Aquesta funció, indicarà al sistema que la tasca ha finalitzat.

Encara que en aquest cas no es necessari modificar-lo, cal mencionar el fitxer "node.cnf". Aquest fitxer és l'encarregat de compilar els elements necessaris pel correcte funcionament del programa. En aquest cas, únicament s'usa un arxiu, però en projectes majors és més que possible que el codi estigui repartit en variïs arxius.

Aquest fitxer s'ha de personalitzar a les necessitats del projecte, però almenys ha de disposar de les següents parts:

- NODE_NAME = Nom de la CoLA.
- NODE = Ruta on figuren els arxius de l'aplicació.
- C-FILES: Aplicacions que s'han de compilar/enllaçar per a que l'aplicació funcioni correctament.

En el cas de l'exemple anterior, aquest fitxer contindrà:

```
NODE_NAME=Blinky
NODE=Projects\Amelie\COLApps\Apps\Blinky
C-FILES=
        Main.c
```

2.2.5 Depuració del codi desenvolupat

En qualsevol desenvolupament és important tenir eines de cara a depurar el codi i controlar possibles errors. Per a depurar el mòdul RTX4100 existeixen 3 eines que ens ho permetran fer:

1. Utilitzant funcions *print*: Aquesta és la funció més primitiva i que pot ser usada en qualsevol llenguatge de programació. Consisteix en imprimir missatges determinats en punts estratègics del codi, amb l'objectiu d'entendre el que està succeint (podem arribar a mostrar contingut de certes variables).

   En qualsevol cas, no existeix una funció *print*, però mitjançant algun exemple, es pot arribar a realitzar. Per a mostrar-ho, és interesant l'exemple proporcionat pel SDK de RTX, el qual es basa a enviar/rebre missatges pel port sèrie, en concret, el port LEUART.

D'aquest exemple es pot arribar a extreure la funció següent, que s'encarregarà d'enviar missatges via sèrie:

```
#include <Drivers/DrvLeuart.h>
#define PRINT(x) UartPrint(x)
static void UartPrint(char *pStr)
{
  while (*pStr != '\0')
  {
    if (*pStr == '\n')
    {
      DrvLeuartTx('\r');
    }
    DrvLeuartTx(*pStr++);
  }
}
```

2. Utilitzant l'aplicació RSX: RTX disposa d'una aplicació anomenada RSX per poder monitoritzar els missatges que entrega ROS. A part dels missatges, el desenvolupador també podrà enviar els seus propis missatges usant la funció RtxEaiPrintf() (a l'estil *printf*).

   L'aplicació també permet definir filtres per a mostrar missatges determinats. Un exemple d'aquests filtres es pot veure en la Figura 3.

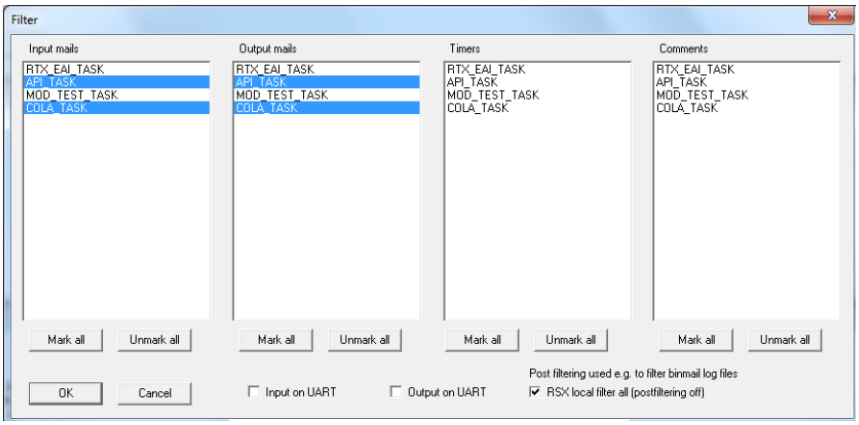


Figura 3: Filtre de l'aplicació RSX

3. Utilitzant RTX2040 Unity II: Aquesta eina, consisteix en una interfície que podrà depurar varies MCU mitjançant la interfície SDI+ (Serial Debug Interface Plus). Encara que aquesta eina és molt potent, no se li dedicarà massa atenció degut a que es un producte pel que s'ha de demanar llicencia. El programa a part de depurar, pot carregar el codi directament sobre el mòdul RTX4100. En quant a la depuració, afegeix opcions més avançades, com la possibilitat de realitzar breakpoints (aturar el codi en un moment determinat), veure les variables en estats determinats, etc.

Després d'haver realitzat aquest petit resum en quant a la depuració, el sistema que s'usarà en el projecte és el de l'aplicació RSX. El motiu és que es podran veure els missatges que enviï ROS i es podran arribar a enviar els missatges que es necessitin mitjançant la funció RtxEaiPrintf(), la qual no haurà de ser definida com era el cas de la funció *print*. A part, la funció RtxEaiPrintf() necessita menys codi que la funció *print*. En concret, la funció *print* suposa un increment de 105B de codi en l'aplicació Blinky, mentre que la funció RtxEaiPrintf només en suposa un increment de 57B.

2.3 Característiques de SPI

SPI (Serial Peripheral Interface), és un estàndard de comunicacions que està format per un bus de dades, en el que es transmeten paquets de 8 bits en full-dúplex. Aquest bus està pensat per comunicacions on la distancia dels elements és curta (sol comunicar microcontroladors amb actuadors per exemple).

Existeixen dos punts característics dels sistema SPI i que cal destacar, de cara a una correcta configuració dels dispositius:

- Diferencies entre MSB i LSB: SPI està constituït per enviar paraules binaries de 8 bits. El concepte MSB(Most Significant Bit)/LSB(Least Significant Bit), està relacionat amb l'elecció de la posició en la que es troben els bits que tenen valor. Si aquests bits es troben a l'esquerra, seran els de major pes (MSB) mentre que si es troben en la dreta, seran de menor pes (LSB).

- Configuració dels rellotges (Polaritat i fase):
  - La polaritat del rellotge indica la base del rellotge (l'estat en repòs).
  - La fase indica el moment en el qual s'ha de llegir/col·locar la informació. El valor 0, indica que és llegeix el valor MISO en el flanc de pujada i MOSI en el flanc de baixada. El valor 1 indica que es llegeix el valor MOSI en el flanc de pujada i MISO en el flanc de baixada.

Un exemple gràfic, d'aquests conceptes de rellotge es poden trobar en la Figura 4.

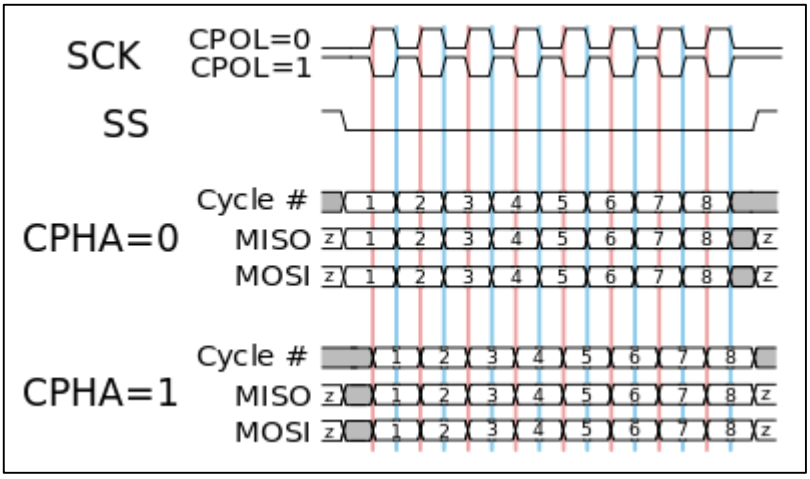


Figura 4: Diagrama de transmissió mitjançant el bus SPI. Font: Wikipedia[9]

Degut a que cada perifèric té les seves peculiaritats, és important veure com es comportarà tant Arduino DUE com el mòdul WiFi Rtx4100 respecte aquest bus:

a) Arduino DUE: Segons les característiques que es poden veure en la seva web[13], aquest model disposa d'una CPU amb una freqüència de 84MHz. En Arduino, la velocitat del SPI ve marcada per la funció setClockDivider[1] la qual té un valor de 21 en aquest processador; això significa que la velocitat del SPI, serà 84MHz/21 = 4MHz. Aquesta freqüència no és cap sorpresa, ja que per defecte Arduino usa una relació divisor/CPU per tenir un valor de 4MHz.

[1] http://arduino.cc/en/Reference/SPISetClockDivider

Convé recordar, que no es necessari agafar la velocitat més ràpida que pugui suportar un processador (s'ha de recordar que ha de ser compatible amb la velocitat dels altres elements i que a major velocitat, la relació d'errors augmentarà).

En quan a configuracions de rellotge, si s'accedeix a la corresponent llibreria d'Arduino[2] dins del seu SDK, es pot veure que implementa 4 modes basats en el segon bit per la fase i tercer per la polaritat, dins la variable SPCR. Amb aquestes podrem confeccionar las Taula 4.

Mode	Valor	Valor(binari)	Fase/Polaritat
SPI_MODE_0	0x00	....00..	0/0
SPI_MODE_1	0x04	....01..	0/1
SPI_MODE_2	0x08	....10..	1/0
SPI_MODE_3	0x0C	....11..	1/1

Taula 4: Configuracions del bus SPI en Arduino

b) RTX4100 (Energy Micro EFM32): Revisant el codi proporcionat pel SDK de RTX, es pot veure un fitxer anomenat DrvSpi.h[3]. En ell es pot veure la configuració del rellotge (polaritat i fase). Com es veu indica que CLKPOL = 0 y CLKPHA = 0. També es pot veure que el firmware està implementat per a treballar amb trames de 8bits.

Per una altre banda i segons la documentació del EFM32G230[14],en l'apartat 3.9.2, per l'apartat SPI s'usa un cristall que permet variar la seva velocitat de 4MHz a 32MHz. Malauradament com s'ha explicat en l'apartat de components del RTX4100, l'actual firmware que controla la gestió del rellotge no permet la variació de freqüència, permetent únicament l'activació o desactivació.

Si s'aplica aquesta freqüència i segons la formula proporcionada per EFM32G230[15] els valors assignables tindran un rang de 8MBps a 244.11bps.

[2] libraries/SPI/SPI.h
[3] \\Projects\Amelie\Components\Drivers

c) Configuració SPI per la comunicació RTX4100-Arduino DUE: En aquest punt es decidirà la millor forma de configurar el sistema Arduino i el sistema RTX4100 per a una correcta comunicació:

- Donat que no s'especifica directament, per elegir l'ordre de bytes (MSB/LSB), s'haurà de provar quin dels dos mètodes usen els dispositius. Donat que realitzar aquest canvi en Arduino, és una tasca fàcil, serà aquest sistema el que s'adaptarà al de RTX4100. Després de fer les pertinents proves (enviar un byte i mostrar-lo), s'arriba a la conclusió que RTX funciona en mode LSB.

- Amelie utilitza la següent configuració pel rellotge: CLKPOL=0 y un CLKPHA=0. Segons els drivers d'Arduino, això es basarà en usar SPI_MODE0.

- Configuració mestre/esclau: donat que Arduino, tindrà que comunicar-se amb altres elements a part del mòdul RTX (SDCard per exemple), s'usarà aquest com a mestre. Els elements restants quedaran per tant com a esclaus.

- Velocitat: la velocitat vindrà definida per la mida mitjana que necessitem enviar pel port. Pel desenvolupament actual, es preveu que la mixa màxima ho marquin algunes imatges que es tinguin que proporcionar d'Arduino al mòdul WiFi. Cal recordar que Arduino per defecte està configurat a 4MHz. Això proporciona (4·10^6mbps = 488.28kB) de velocitat bruta, velocitat més que acceptable, mentre que el rellotge de RTX4100 és capaç de suportar fins 32MHz (4MHz x 8) fet que asseguraria un correcte mostreig.

  No obstant, degut a que no es necessària tanta velocitat, s'usaran dades més conservadores. En concret Arduino emetrà a 1Mbps mentre que RTX4100 es configurarà per un mostreig de 8Mbps, proporcionant una velocitat efectiva de ~1000b/s via bus SPI.

2.4 Introducció al kit de desenvolupament

Un cop explicades les funcionalitats bàsiques de SPI i de RTX4100, és necessari fer una ràpida menció al kit de desenvolupament que s'usarà en el projecte. Aquest kit, estarà constituït de dos parts ben diferenciades: El *Smart Citizen Kit DVK* i l'*Arduino DUE*.

- Smart Citizen Kit DVK: Aquest kit és comportarà com un shield[16] per Arduino. Aquests elements s'usen per ampliar les funcionalitats que proporciona l'Arduino elegit. La imatge física d'aquest kit, és pot veure en la Figura 5 mentre que l'interconnexionat es veurà en la Figura 6. Aquest kit de desenvolupament, consta de diversos elements, però cal fer especial menció al chip RTX4100 i a la targeta SDCARD, ja que són els dos elements s'usaran principalment en el treball.

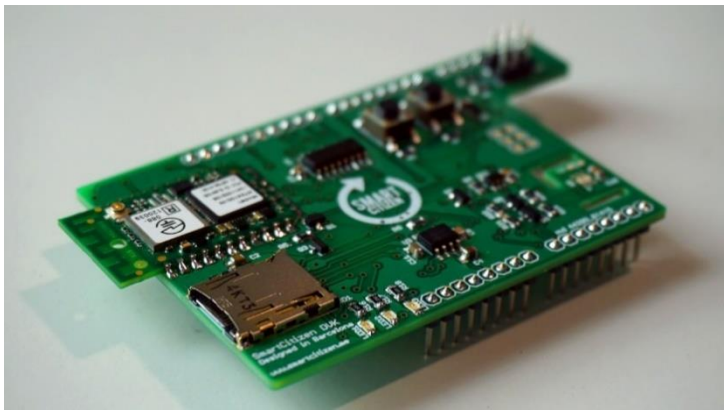


Figura 5: Imatge del DVK. Fotografia: fablabbcn[6]

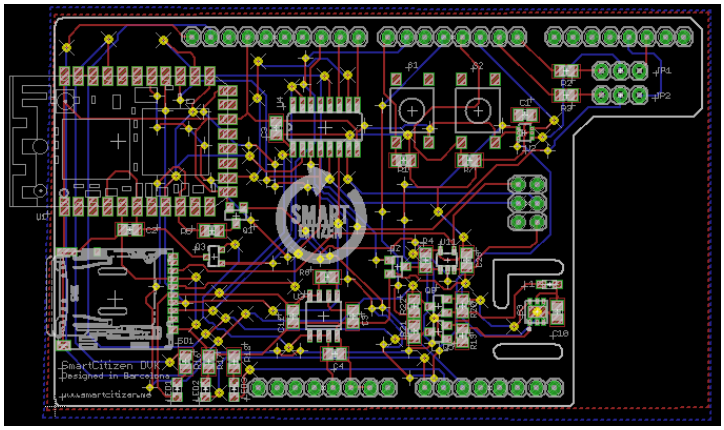


Figura 6: Interconnexionat del DVK

- Arduino DUE: Per part de Smart Citizen s'ha proporcionat una placa Arduino DUE[17] amb la qual poder fer el corresponent desenvolupament. Cal recordar que el desenvolupament es realitzarà sobre el Kit DVK, sent l'elecció d'Arduino trivial. De fet l'opció desitjada per Smart Citizen, serà l'Arduino Zero[18], la qual actualment, encara no ha sortit al mercat. L'únic requisit desitjat, que la velocitat de rellotge de pla placa, pugui arribar a proporcionar una velocitat de 1MHz en la comunicació via SPI (fet que compleixen totes les plaques d'Arduino, doncs com s'ha explicat la velocitat per defecte és de 4MHz).

Una comparació de les dues plaques, tant comparativament com gràficament les podem observar en la Taula 5 i la Figura 7 respectivament. En aquesta comparació la qual no està a escala, és pot veure com la placa Zero és força més petita que la DUE. En qualsevol cas, ambdues plaques disposen del port SPI requerit i podran proporcionar una velocitat major al MHz requerit.

Model	Zero	Due
Microcontrolador	ATSAMD21G18	AT91SAM3X8E
Voltatge operatiu	3.3V	3.3V
Pins Digitals I/O	14 12(PWM i UART)	54 12(PWM)
Pins analògics d'entrada	6 (12-bit ADC)	12
Pins analògics de sortida	1 (10-bit DAC)	2 (DAC)
Tensió per pin I/O	7mA	800 mA
Memòria Flash	256 KB	512 KB
SRAM	32 KB	96 KB
EEPROM	16KB (amb emulació)	Disposa de la Flash
Velocitat de rellotge	48 MHz	84 MHz

Taula 5: Comparativa entre Arduino Zero i Arduino Due Fotografia: Arduino[17] & [18]

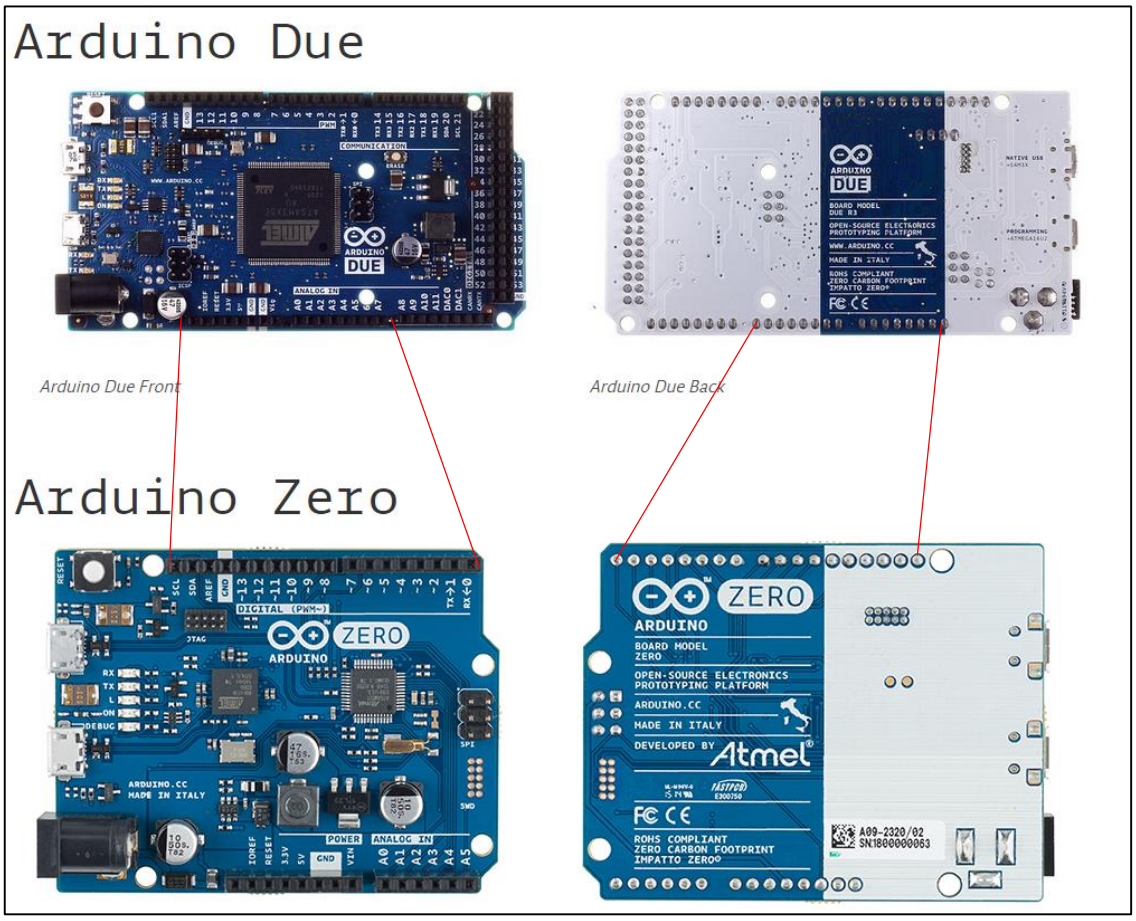


Figura 7: Comparativa entre Arduino Zero i Arduino Due

Encara que en l'annex -***Diagrama hardware DVK***-, es poden veure les connexions realitzades entre el DVK i l'Arduino DUE, en la figura 8 es veu com es comparteix el SPI entre el mòdul RTX4100 i la targeta SDCARD. També es definirà l'element EN74AHC1G125[19] que servirà per activar/desactivar l'entrada de dades en la línia MISO segons l'estat del RTX_CS_3V i finalment es veuran diversos inversors CD74HC4050M[20] que serviran per normalitzar la senyal, per l'esperada pel dispositius del DKV/Arduino (3.3V/5V).

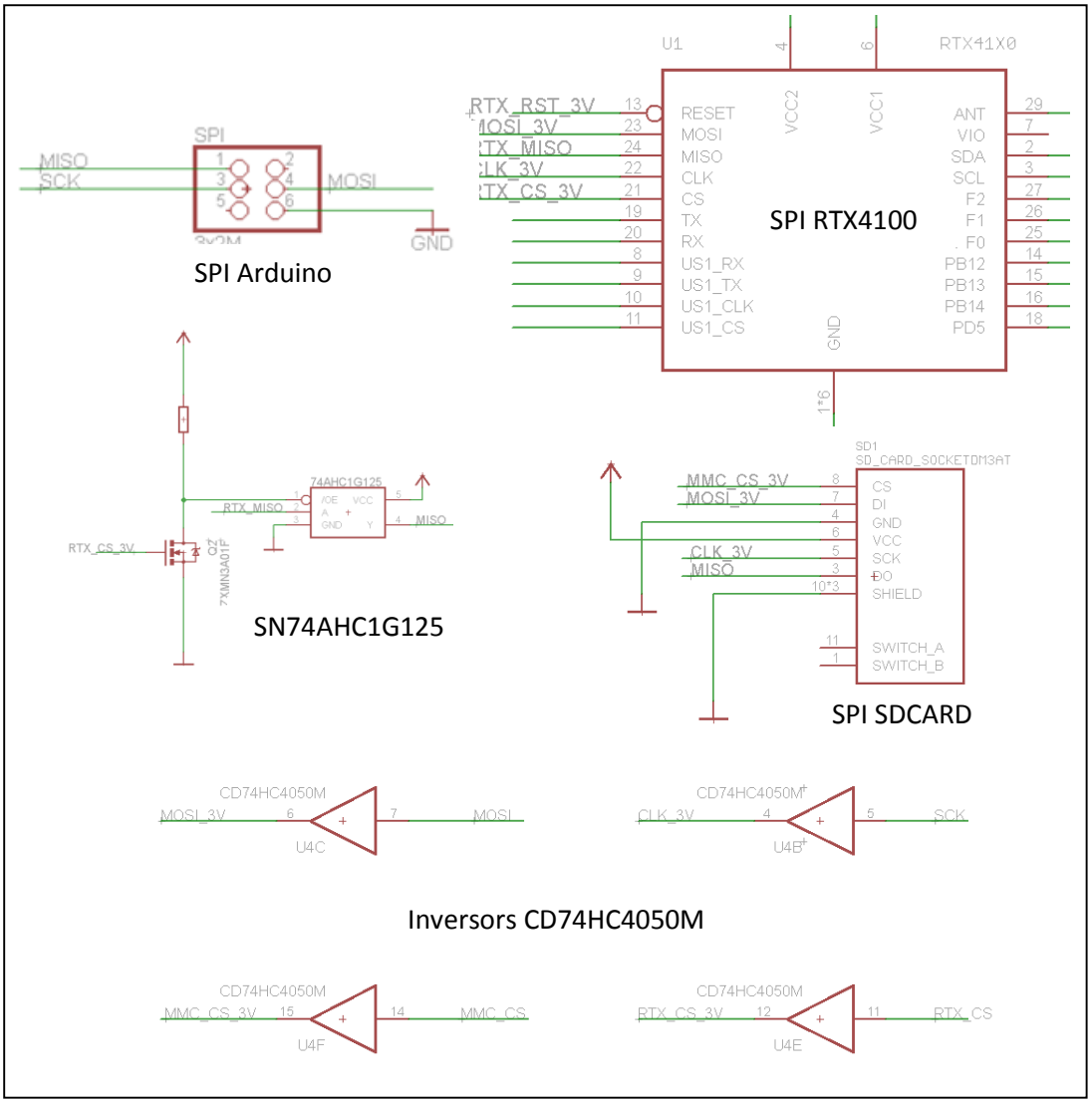


Figura 8: Diagrama de connexions SPI (Arduino & RTX)

2.5 Conceptes preliminars de protocols IP

2.5.1 Conceptes bàsics de DNS

Per a poder fer el correcte desenvolupament del servidor de noms, s'haurà de recórrer a la seva especificació[21]. En ella es pot veure informació útil com per exemple que les peticions de DNS s'atenen de manera general pel port 53 UDP. També s'observa que una consulta a un servidor de DNS consisteix en una única petició que consta d'una capçalera i la petició que es forma.

Un exemple pràctic d'aquesta teoria, es pot veure en la Figura 9, la qual tracta d'una consulta de DNS contra la direcció "www.amazon.com" i capturada a través del programa Wireshark[22].

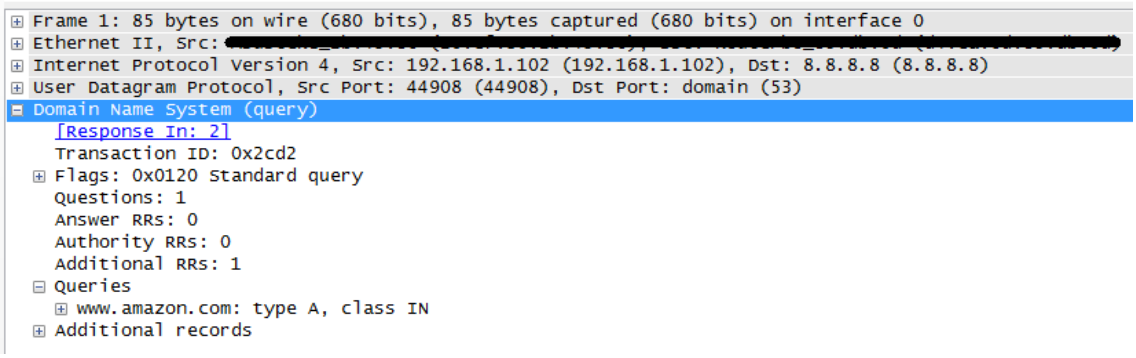


Figura 9: Paquet de DNS (petició de pàgina)

La mateixa resposta a aquesta consulta, es pot veure en la Figura 10.

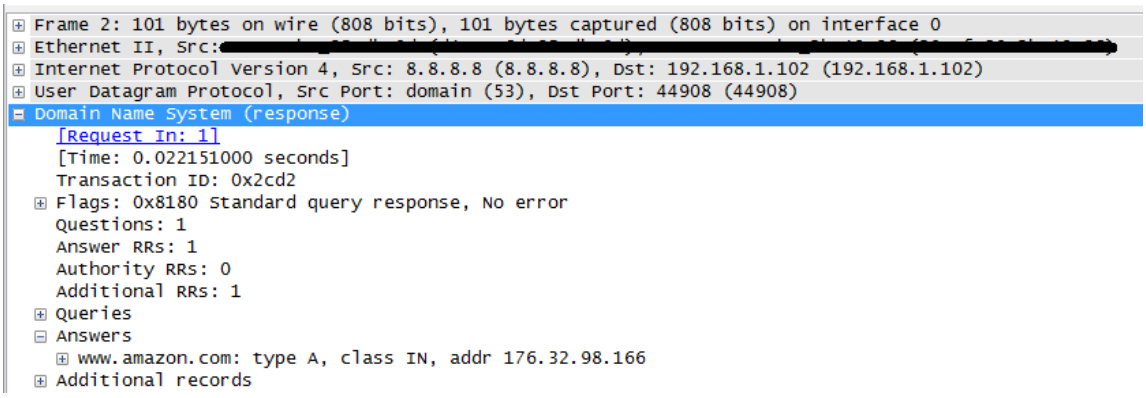


Figura 10: Paquet de DNS (resposta de pàgina)

Degut a que el propòsit del treball no és realitzar un servidor DNS, tan sols es realitzarà un breu repàs dels camps necessaris per dur a terme una resposta correcta.

- En una resposta, el camp “Transaction ID” haurà de coincidir amb el de la petició. Es pot veure que aquest camp val 0x2cd2.

- El camp flags, serà estàtic(sempre tornarà el valor 0x8180). Aquest valor vol dir que: és una resposta (primer byte += 8), que s’ha sol·licitat recursiu com fan la majoria d’aplicacions web (segon byte += 2), que aquesta recursivitat és permesa pel servidor (tercer byte += 8) i res més (quart byte = 0).

- El camp “queries”, serà una replica de la pregunta que es realitza i com a tal, tant en la petició com en la resposta tindrà el mateix valor, com es pot veure en les Figures 11 i 12 respectivament.

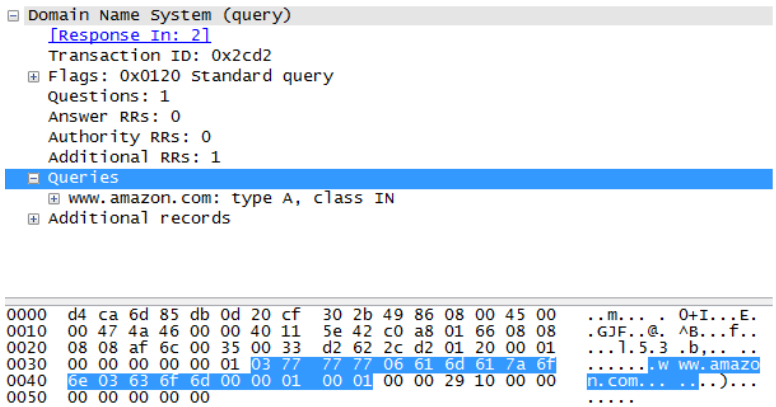


Figura 11: Paquet de DNS (contingut petició de pàgina)

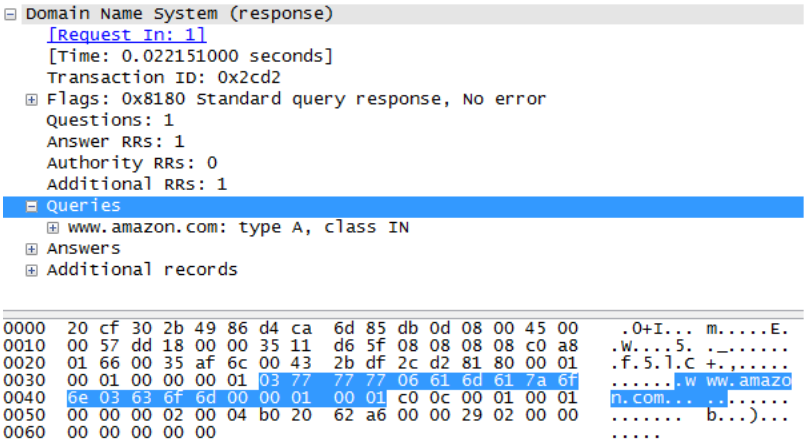


Figura 12: Paquet de DNS (contingut resposta de pàgina)

- Finalment, en quant a la resposta, el valor també serà estàtic. Aquest valor es centrarà únicament a dir que resolem la petició del client, que és tipus A(Address), que és de classe IN(Internet), que té un temps de vida de 900 segons, que la IP té una longitud de 4bytes (per 32 bits) i que la IP és la IP del nostre AP.

2.5.2 Conceptes bàsics de HTTP

Donat que no es vol desenvolupar un servidor que compleixi amb tots els estàndards HTTP, no es tindran en compte tots els requisits que marca l'estàndard HTTP1.1[23].

Degut a que el servidor ha e suportar els mètodes GET/POST, es passa a explicar quin és el funcionament d'una petició amb aquests mètodes:

a) GET: La petició GET només té en compte la URI que s'està enviant. En la URI, es diferenciarà la URL de la resta de les variables mitjançant el símbol "?". Les variables constaran de dos parts. El nom de variable i el seu valor, que anirà precedir del símbol "=". Per concatenar diferents variables s'usarà el símbol "&". Això es pot veure més clarament en l'exemple de la Figura 13, el qual és una petició de la cerca de la paraula "prova" a la web de google.

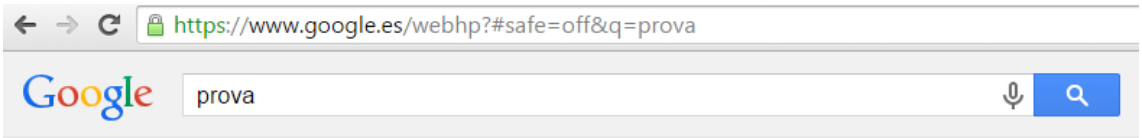


Figura 13: Exemple petició web (GET)

La URL és: "/webhp" i les variables que té són: #save, q. D'aquestes variables la que ens interessarà és la última, ja que té el valor "prova", que és la paraula que hem buscat en el cercador.

b) POST: La petició POST és una mica més complexa que la GET, i consisteix en afegir les variables al final de la petició web. Per fer aquesta prova, es capturaran les traces d'una web que no usa HTTPS en la autenticació. Això es podrà veure en la Figura 14.

Figura 14: Exemple petició web (POST)

En aquest nou exemple, es pot veure com es passen diversos valors, però no van en la URI (només apareix /index.php). Aquests valors són: acció, email, password i remember.

Per implementar aquestes funcions, s'haurà de rebre la petició enviada pel client, i segons sigui POST o GET, analitzar uns camps o uns altres. En el cas del GET, únicament s'analitzarà la URI, mentre que en el cas del POST, s'haurà d'analitzar tot el missatge enviat, ja que cal recordar que les variables del POST estan en el final del missatge.

Un cop s'hagi processat la petició, s'analitzarà per veure quin contingut la forma (si és un comandament que s'ha d'executar en el SCK o si és un contingut estàtic que s'ha de cercar en el SCK).

Una de les funcions que també es valora, és la possibilitat d'usar el SCK per a recuperar arxius pesats. Amb aquesta opció, es podran carregar fins i tot imatges en la mateixa web. Aquesta funcionalitat presentarà un nou problema, i és que s'ha de recordar que el RTX4100 només té una memòria de 3kB de RAM. Això vol dir, que no es podrà retornar una petició major de 3kB.

Per solucionar aquest problema, es partirà l'arxiu a enviar en diverses parts, i seran enviades una rere l'altre (com peticions diferents). En tot cas, no es farà ús dels chunks que implementa HTTP1.1, ja que en tot moment s'estan parlant de fitxers de pocs k's d'espai. El que si que es farà, és servir la petició per parts (les parts seran tant gram com admeti el buffer).

Tot això es veurà una mica més clar en el Diagrama 1. Una petició entrant seguirà els següents passos:

1) Primer s'analitzarà i es prendrà informació útil com el mètode emprat, les comandes i la URL a la que fa referencia.

2) Posteriorment, s'analitzaran les variables, per veure si existeix alguna comanda que es tingui que executar. Tota comanda a processar haurà de començar per la variable ($web_optioncmd). En cas que existeixi, s'analitzarà per veure si s'ha d'executar localment (si comença per la cadena definida per $web_optioncmd_RTX) o via el bus SPI. En aquest últim cas, s'usarà la API corresponent i s'enviarà a Arduino per a executar la comanda. En ambdós casos, el sistema retornarà un resultat i aquest serà mostrat al client web. D'aquesta manera es podran realitzar des de consultes (sensors, estats actual del sistema) a accions que modifiquin l'estat del sistema (canviar la configuració del punt d'accés, o dades de connexió al servidor de Smart Citizen).

En cas que no existeixi cap comanda a executar, s'analitzarà el fitxer: Si no hi ha cap fitxer definit (path=/), es carregarà l'arxiu definit per la variable $HTTP_DEFAULT_WEBPAGE. Seguidament s'analitzarà si l'arxiu a carregar conté una cadena determinada per la variable ($web_sdmask). Això indicarà si l'arxiu s'ha de buscar en la SD o no. En cas que no estigui en la SD, es tornarà un error de pàgina no existent (404). Si l'arxiu aparentment es troba en la SD, es reclamarà l'arxiu i en cas que existeixi serà enviat al client. El motiu de no passar tots els arxius contra la SD, és la de no sobrecarregar el sistema (cal recordar que es redirigeixen totes les peticions contra el servidor web).

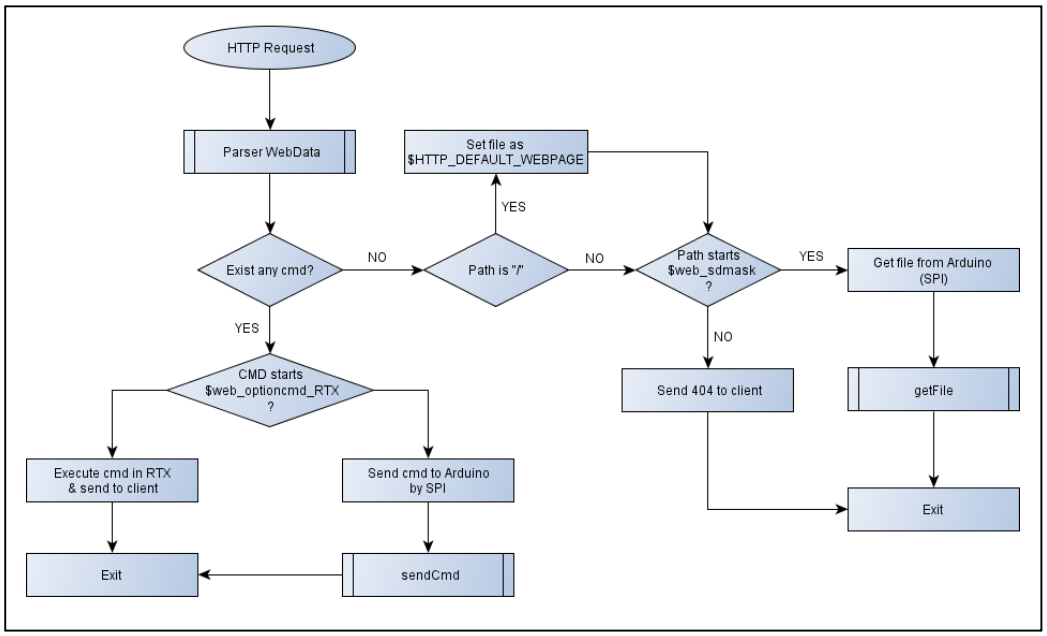


Diagrama 1: Aplicació HTTP

Els codis d'error que contemplaran aquesta aplicació són:

- 200 OK : Petició entregada correctament.
- 404 File not found: L'arxiu no existeix.
- 501 Not Implemented: Mètode desconegut (només es suporta GET/POST)
- 500 Internal Server Error: El SPI no retorna cap resultat. (Només afectarà a les peticions que s'hagin d'enviar per SPI.

Degut a que es vol confeccionar un servidor ho més senzill possible, és necessari comentar les plantilles especifiques emprades. Aquestes plantilles que es poden veure en l'annex –***Documentació del codi***- són:

1) Plantilla Error404: Quan un fitxer no existeixi, s'enviarà l'error 404 i es redirigirà a la pàgina principal en 3 segons.

2) Plantilla Error501: Quan s'usi un mètode no suportat, s'enviarà l'error 501 i es redirigirà a la pàgina principal en 3 segons.

3) Plantilla CMD: Quan s'enviï una comanda, es retornarà el resultat de la mateixa i s'oferirà la possibilitat de tornar a la pàgina principal mitjançant un enllaç. En cas de no polsar-lo, al cap de 10 segons es redirigirà a la pàgina principal.

4) Plantilla arxiu: Sempre que es sol·liciti un fitxer, en cas d'existir, s'enviarà una capçalera que tindrà la opció "Cache-Control: public, max-age=3600". Aquesta opció intentarà cachejar el contingut en el client web, per evitar tornar-lo a descarregar. No obstant, al no tenir una data fiable que enviar, la possibilitat de cachejar dependrà de la implementació del client web.

2.6 Desenvolupament de l'aplicatiu

2.6.1 Entorn de desenvolupament usat

Per a desenvolupar el projecte, es tindran en compte les següents eines, les quals es comenten a continuació. Degut a possibles canvis en futures versions, es fa menció a les versions usades en el desenvolupament:

- Arduino[24]. Versió 1.6.3: Programa emprat per programar la placa d'Arduino DUE (part complementaria de les API's necessàries).

- Netbeans IDE[25]. Versió 8.0.2: IDE de desenvolupament emprat per a programar el codi del mòdul WiFi RTX4100.

- Amelie SDK[8]. Versió 1.6.0.63. Exemples i eines proporcionades per RTX per a programar el mòdul WiFi. El desenvolupament va començar en la versió 1.6.0.58, però el 7 d'Abril del 2015 van treure una nova versió. Els canvis[4] d'aquesta versió, incloïen una optimització en el protocol TCP (emprat pel servidor web), motiu pel qual es decideix actualitzar.

- Bootloader RTX[8]. Versió 1.6.0.60. El desenvolupament va començar usant la versió 1.6.0.52, però tal com es comenta en el punt anterior, el 6 d'Abril RTX treu una versió que optimitza el protocol TCP.

[4] http://www.rtx.dk/Files/Billeder/RTX_T/RTX4100docs/Release_note_Amelie_SDK_ver_1_6_0_63.pdf

2.6.2 Estat actual del codi

Com s'ha dit en diverses ocasions, el present treball busca ampliar les funcionalitats d'un desenvolupament prèviament realitzat. De cara a realitzar aquest desenvolupament, es realitzarà de la manera més modular possible, de cara a futures modificacions. En tot cas, és evident que per a ampliar el codi, prèviament s'haurà de revisar i fer els canvis imprescindibles per a la correcta execució del nostre desenvolupament.

La part a analitzar, serà principalment la comunicació SPI (que és la que s'usarà en el present treball. Sobre aquesta comunicació i partint del codi existent[26], es poden destacar els següents punts:

- ✓ Per activar aquestes funcions s'ha de definir "SPI_COMMUNICATION".

- ✓ En el codi actual, el bus SPI s'inicialitza mitjançant el protothread PtDrvSpiInit() i la funció DrvSpiInit(). Aquests mètodes són accessibles mitjançant la llibreria "Drivers/DrvSpi.h", la qual fa referencia al mode mestre. També es pot veure, com s'està inicialitzant a una velocitat de 9600bps (~1,17KB).

- ✓ Les comandes que actualment implementa la API, estan dins d'un bucle (posterior a la inicialització del SPI). Això vol dir que cada cop que entri una comanda via SPI s'executarà en cas que sigui necessari. Cal tenir en compte, que hi ha comandes que esperen informació addicional, com és el cas de la consulta DNS (0x02), la qual necessita el nom de host per a traduir-lo.

2.6.3 Modificacions en la API

Actualment existeixen 15 comandes que permeten interactuar amb el mòdul WiFi RTX4100. Aquestes comandes són enviades mitjançant el bus SPI el qual disposarà d'un byte per aquest propòsit, per tant hi ha un màxim de 256 comandes a configurar (0→255). Lògicament encara que no és el cas, es podrien estendre aquest número de comandes (per exemple, el caràcter 255 podria indicar que s'ha de llegir un nou byte, ampliant aquest número de comandes conseqüentment).

A continuació es mostren les comandes actuals. Es farà especial èmfasis en les comandes que és modificaran (explicant el seu funcionament). També s'ampliaran aquestes comandes amb 3 noves en sentit (Mestre → Esclau), i 2 noves en sentit contrari.

Arduino → RTX4100

- 0x01 (1): Obtenció de l'estat del sistema. Retornarà 1 byte del qual:
  - XXXX XXXX: Associat a AP. (0:NO 1:SI)
  - XXXX XXXX: Connexió establerta. (0:NO 1:SI)
  - XXXX XXXX: Informació TCP pendent. (0:NO 1:SI)
  - XXXX XXXX: Chip suspès. (0:NO 1:SI)
  - XXXX XXXX: Estat soft AP. (0:NO 1:SI)
  - XXXX XXXX: Servidor Web. (0:NO 1:SI)

- 0x02 (2): Resolució de nom (DNS)

- 0x03 (3): Configuració de IP

- 0x04 (4): Establir nova connexió TCP

- 0x05 (5): Associat i connectar-se a un AP

- 0x06 (6): Des-associar-se i desconnectar-se a un AP

- 0x07 (7): Configuració de AP

- 0x08 (8): Tancar connexió TCP
- 0x09 (9): Rebre la informació TCP pendent i marcar-la com llegida
- 0x0A (10): Enviar informació TCP
- 0x0B (11): Encendre/apagar el chip WiFi
- 0x0C (12): Establir nou perfil d'energia WiFi
- 0x0D (13): Elecció de la potencia de transmissió WiFi
- 0x0E (14): Suspensió del chip WiFi
- 0x0F (15): Des-suspensió del chip WiFi
- 0x10 (16): Activació mode soft AP: Aquest mode activarà el punt d'accés virtual i redirigirà totes les peticions a si mateix. Retornarà 0 si la comanda ha estat executada correctament o 1 si no ho ha estat.
- 0x11 (17): Activar Servidor Web: Encendrà un servidor Web que serà capaç d'enviar comandaments bàsics a l'Arduino. En aquest punt serà el RTX4100 qui marcarà la connexió SPI mitjançant interrupcions. (Quan un usuari vulgui enviar un comandament, avisarà a l'Arduino de que té dades per enviar i aquest les llegirà). Retornarà 0 si la comanda ha estat executada correctament o 1 si no ho ha estat.
- 0x12 (18): Desconnexió Servidor Web & soft AP: Aquesta comanda només podrà ser cridada després de l'anterior. El seu objectiu serà finalitzar el Servidor Web i desconnectar el soft AP.

RTX4100→Arduino

- 0xA0 (160): Executar comandament: Aquesta comanda, anirà seguida d'un byte, que indicarà el número de bytes a reservar per la comanda. Posteriorment a aquest byte s'enviarà la comanda i s'esperarà un byte que indicarà la mida de la resposta. Un cop enviat, s'esperarà la resposta.

Un possible exemple per l'ordre "get value sensor 1" seria:

→0xA0
→0x12
→"get value sensor 1"
← 0x1
← 0x23

- 0xA1 (161): Petició de fitxer: Aquesta comanda servirà per a sol·licitar un fitxer. Degut a que és una mica complexa es partirà en les següents parts:

    1) Seguida a la comanda s'enviarà el nombre de bytes a reservar per la ruta del fitxer, seguidament s'enviarà la ruta del fitxer.

    2) Un cop l'Arduino rebi aquesta informació, passarà a comprovar si el fitxer existeix i en cas afirmatiu consultar la seva longitud. La resposta que farà Arduino, constarà del número de bytes a reservar per poder guardar la mida del fitxer. Si aquest fitxer no existís, aquesta mida seria 0. Posteriorment a la mida, s'enviarà la longitud del fitxer.

    3) En aquest punt, RTX enviarà el seu buffer de SPI. Un cop Arduino el rebi i comparant-lo amb el seu, s'elegirà el buffer idoni (és a dir el més petit). El motiu de l'elecció del buffer, serà per a que els dos dispositius coneguin quantes transaccions s'haurà de fer per a enviar el fitxer. Aquest valor del buffer, independentment del seu valor, ocuparà 2 bytes (sent el valor màxim 65535b).

    4) En aquest punt, RTX (sabent el numero de transaccions necessàries), es començarà a descarregar el fitxer. Cada cop que Arduino enviï un tros del fitxer, s'aturarà i esperarà a que RTX confirmi que ha rebut el tros (mitjançant interrupcions). En aquest moment, si encara queden parts del fitxer per enviar, es seguirà el mateix procediment. Cal recordar que RTX té una memòria molt limitada, per tant, cada cop que es rebi un fragment, aquest s'enviarà via TCP al client que ha sol·licitat el recurs.

5) Quan s'hagin enviat totes les parts dels fitxers, ambdues parts donaran per completada la comanda i quedaran expectants de noves comandes.

Tot això es pot veure millor amb el següent exemple: En el cas que existeixi un fitxer anomenat “\FILE” que ocupa 300bytes i que el buffer de RTX és de 128 mentre que el d'Arduino és de 500, per a descarregar-lo segons el procediment anterior s’enviarien les següents transaccions:

→0xA1 0x05 “\FILE”	(Punt 1)
←0x02 0x012C (300)	(Punt 2)
→0x80 (128) ←0x80 (128)	(Punt 3)
←[128data] ←[128data] ←[44data]	(Punt 4)

2.6.4 Modificacions rellevants del codi

El desenvolupament realitzat en aquest treball, intentarà afectar en la menor mida possible al desenvolupament ja realitzat. Per a fer-ho, s'usarà un disseny modular que permetrà la seva modificació de forma senzilla. En tot cas hi haurà una sèrie de modificacions que es tindran que fer sobre el codi ja realitzat.

- SPI: Aparentment la part de SPI ja està desenvolupada. En tot cas, s’haurà de revisar, ja que en l’anterior treball no estaven definits quin eren els requisits.

  1) El primer que s’haurà de fer, serà canviar els rols mestre/esclau, ja que en el codi actual, RTX s’usa de mestre, opció que no serà valida pel nou codi segons els requisits que s’han pogut veure en els apartats anteriors.

  2) També serà necessari elegir una nova velocitat, ja que el codi actual usa una velocitat de 9600bps (~1,17KB/s). Això és deu en que en l’actual codi, només és transmetien cadenes curtes les quals donaven sentit a aquest valor. Si ara volem enviar un fitxer de 200KB pel servidor web, amb la velocitat actual es trigarà prop de 3 minuts, temps que inacceptable per qualsevol usuari.

3) Serà necessari modificar els drivers SPI[5] usats pel RTX4100. Això es deu a que la placa proporcionada per Smart Citizen (DVK), no compleix els estàndards en quant a SPI, el qual considera els valors vàlids sempre que CS sigui nul.

En la Figura 15, es veu el disseny de la DVK en el qual es veu un buffer triestat, que deixarà passar la senyal d'entrada (RTX_MISO) en cas que la senyal OE sigui nul·la. Aquest disseny va en contra de les especificacions genèriques ja que el transistor NMOS actua de inversor. si (RTX_CS_3V = 0 → OE = VCC → MISO = 0).

Degut a aquest punt, s'hauran de modificar els drivers corresponents i indicar-li al RTX que tindrà que enviar dades quan RTX_CS_3V sigui VCC. Degut a que serà necessari fer aquest canvi, s'aprofitarà i s'afegirà un altre, el qual avisarà mitjançant un mail (RTX) de quan s'hagin rebut una sèrie de bytes nous. (Aquesta modificació es pot veure en l'annex -***Diagrama hardware DVK***).

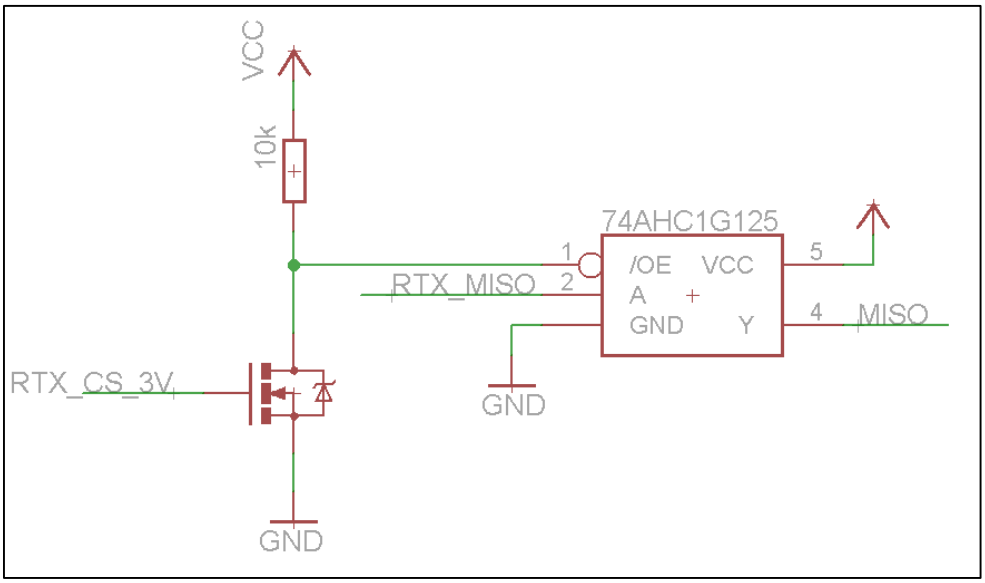


Figura 15: Circuit del bus SPI (RTX)

[5] \Projects\Amelie\Components\Drivers\DrvSpiSlave.c

4) API: Encara que prèviament ja s'han definit les API's, en aquest punt s'entrarà en més detall de quines tasques realitzaran:

   I. Mode soft AP: Al executar aquesta comanda, RTX farà una interrupció (apagarà el LED) per a que Arduino tingui en compte que s'està executant la comanda. Posteriorment executarà pròpiament la comanda, i un cop finalitzada, tornarà a fer una interrupció (encendrà el LED). D'aquesta manera, Arduino sabrà en tot moment quin és l'estat de la tasca. La comanda en si, realitzarà les següents tasques:

      - Reinicialitzar el mòdul WiFi: Aquesta operació col·locarà el mòdul WiFi en un estat operatiu.

      - Configuració del ESSID: El ESSID que prendrà el AP, correspon a una cadena fixa (SCK_AP) seguida dels últims 3 blocs hexadecimals que formen la MAC.

      - Iniciació WiFi: Seguidament es realitzaran configurarà informació relativa al enllaç (canal, potencia, encriptació) i posteriorment s'intentarà iniciar el mode Soft AP.

      - Configuració IP: Un cop iniciat, es proporcionarà una IP. La configuració es traurà de la definició de les variables STATIC_IP i SUBNET_MASK respectivament.

      - Configuració DNS: Finalment s'iniciarà un petit servidor DNS (el qual redirigirà totes les peticions contra la nostre IP).

   II. Mode servidor Web: Aquesta comanda té una peculiaritat especial. Al executar-se, RTX farà una interrupció a Arduino (apagarà el LED). Seguidament iniciarà el servidor web. Quan ho hagi fet, farà una interrupció (encendrà el LED). D'aquesta manera Arduino sabrà que s'ha executat correctament i actualitzarà els seus estats.

En aquest punt no es sortirà de la comanda, com pesava amb la resta de comandes, i és que RTX esperarà a rebre un missatge sol·licitant la desconnexió del servidor web (la següent comanda). Això es deu, a que al entrar en aquest estat, s'intercanvien els rols, i és RTX qui inicia les comunicacions via SPI (encara que segueix sent l'esclau). És important tenir en compte, que aquest missatge només aplicarà quan el servidor web no estigui cursant una petició. Això es deu a que seria possible que davant la petició d'un arxiu, aquest contingues la comanda, fet que ens produiria problemes.

III. Desconnexió servidor Web: Aquesta comanda només podrà ser executada després de l'estat d'inici del mode web. La seva funció serà desconnectar el servidor web, tal com s'ha dit en la comanda anterior. La comanda produirà que RTX enviï un missatge, sol·licitant la desconnexió del servidor web, fet que anul·laria el WAIT que hi ha en la comanda anterior produint la desconnexió del servidor web i el reinici de la connexió WiFi.

5) Arduino: En aquest punt, es realitzaran una sèrie de modificacions sobre el codi proveït per SC, les quals són:

I. Modificació dels estats: Per aquesta implementació, seran necessaris 5 estats, els són:

1. normal_mode: Els dos LEDS (blau,verd), estaran apagats. Es cridarà al main() y es creuaran els serials (això serà útil per debugejar). En el nostre exemple, el main serà trivial. Aquesta serà l'opció per defecte del sistema.

2. softap_mode: El LED verd està encès i el LED blau apagat. Al entrar en aquest mode, s'escoltaran les interrupcions que provinents de RTX. Posteriorment s'enviarà la comanda 0x10 (iniciació softap). S'esperarà a que s'apagui el LED i s'encengui (interrupcions provocades per RTX), les quals informaran que s'ha iniciat correctament. Un cop realitzades, es desconnectaran les interrupcions. A partir d'aquest moment, només es creuaran els serials (fet, que ens permetrà debugar el RTX).

3. webserver_mode: Els dos LEDS (blau, verd) s'encendran. S'activarà la interrupció i s'enviarà la comanda d'inici de servidor web. S'esperaran les corresponents interrupcions per confirmar que el servei s'inicia. En aquest punt s'inicia la targeta SD. A partir d'aquest moment, només es creuaran els serials (fet, que ens permetrà debugar el RTX) i es revisarà si hi han dades en el SPI (mitjançant les interrupcions).

4. Web_end_mode: En aquest punt, s'enviarà l'ordre per desactivar els servidors HTTP/DNS i el soft AP al RTX. Per a fer-ho, s'enviarà la corresponent comanda (0x12) i es desactivaran les interrupcions. Seguidament es tornarà al mode 0.

5. Test_mode: Aquest mode servirà per provar el servidor web (RTX també ha d'estar en aquest mode, ja que no s'enviarà l'activació), sense tenir que activar cap boto. En aquest mode, únicament s'activarà la targeta i s'activaran les interrupcions. Com en el webserver_mode, s'entrarà en un bucle, el qual creuarà els serials i revisarà si hi han dades en el SPI (mitjançant les interrupcions).

II. Modificació de la funció dels botons: El botó 1 serà capaç de passar del estat 0 al 1 i del 1 al 2. Sobre el botó 2, si l'estat actual és 2, permetrà passar a l'estat 0. Això voldrà dir que el botó 1, permetrà activar els estats soft AP & Server Web, mentre que el boto 2, permetrà finalitzar aquests estats (enviant la comanda corresponent a RTX).

2.6.5 Aspectes a destacar del codi desenvolupat

- Diagrama d'exemple d'una comunicació SPI entre Arduino i RTX: En apartats anteriors s'ha vist la necessitat de fer ús del bus SPI. En aquest punt s'acabarà d'aprofundir en aquest tema mitjançant un diagrama de temps.

Cal recordar els rols que s'han definit i és que el mòdul RTX4100 al ser l'esclau, no podrà iniciar mai una comunicació directament. Per tal de solucionar aquest problema, s'usarà la senyal IRQ-RTX per indicar a Arduino que el RTX té dades per enviar. Com es pot veure en la figura 16, quan RTX modifiqui el valor de IRQ-RTX, en uns instants Arduino serà capaç de reconèixer aquest senyal i procedir a la emissió/lectura de dades.

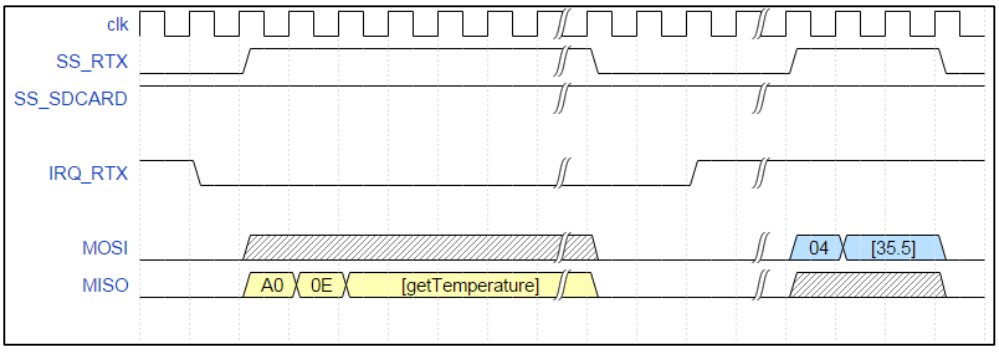


Figura 16: Cronograma d'exemple, d'una comunicació via SPI bus

En l'exemple, RTX envia a Arduino la comanda A0 (executar comanda), seguit de la cadena 0E (14 caràcters). Aquí hi ha un salt de temps, per evitar mostrar els 14 cicles que ocupa aquesta cadena. Posteriorment hi ha un nou salt de temps, que consisteix en el temps que triga Arduino en executar la comanda internament. Finalment, i sempre després de que IRQ_RTX valgui 1 (senyal que confirma que RTX està preparat per rebre la resposta), Arduino respon els valors 0x04 (indicant que a continuació enviarà 4 caràcters) i la temperatura actual (35.5).

- Interrupcions entre Arduino/RTX4100: Donat que no existeix cap línia dedicada que pugui ser usada com a interrupció entre Arduino i el RTX, s'usarà la senyal del LED (taronja). Per a aquest propòsit, es tindrà que portar la senyal corresponent, a un pin lliure de la placa Arduino. En aquest cas, es decideix usar el pin 22 tal com es pot veure en la Figura 17. El senyal del LED, tal com s'ha mostrat en la Taula 3 correspon a la entrada numero 27 del RTX la qual rep el nom PF2.

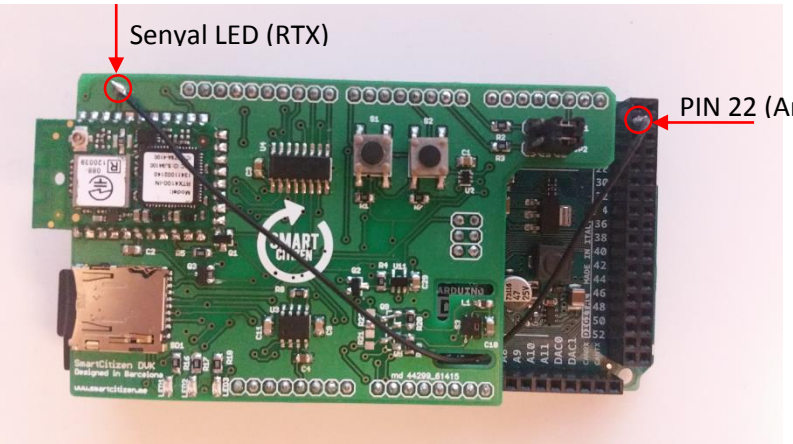


Figura 17: Placa RTX (afegida línia de dades pel bus SPI)

Aquest senyal tindrà com a propòsit aconseguir que el RTX (estant en mode esclau), pugui avisar al mestre que té informació per enviar (mitjançant l'ús d'aquesta interrupció). El mestre al detectar un canvi de senyal en el pin, executarà un enviament/lectura de dades del port SPI.

- Diagrama de decisions: A continuació es mostren els diagrames de decisions dels apartats més crítics del desenvolupament i que es considera que han de tenir especial presencia en aquesta memòria. Aquests apartats són:

    1) Diagrama 2 (API que representa la connexió/desconnexió del servidor web).
    2) Diagrama 3 (API usat per a enviar una comanda).
    3) Diagrama 4 (API usat per l'enviament d'un fitxer).

El rectangle groc simbolitza el mòdul RTX, mentre que el verd simbolitza l'Arduino. Les fletxes puntejades simbolitzen la comunicació entre els dispositius.

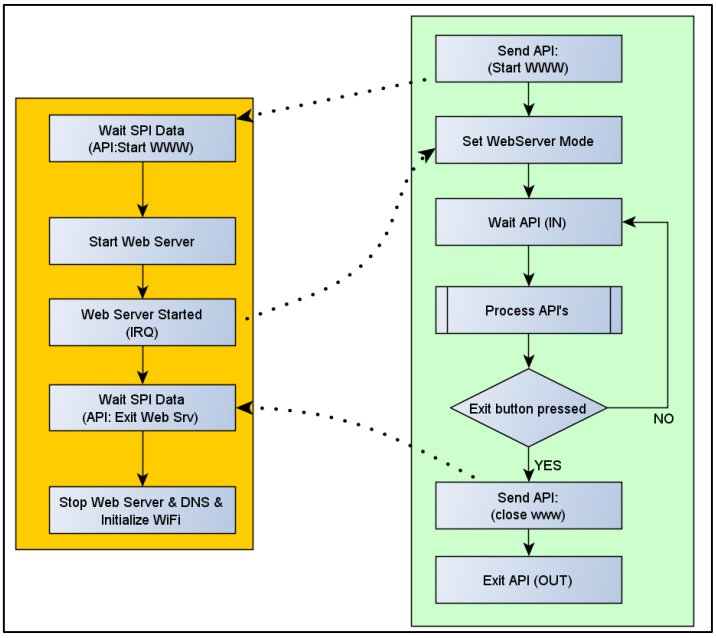


Diagrama 2: API (SPI: connexió/desconnexió servidor web)

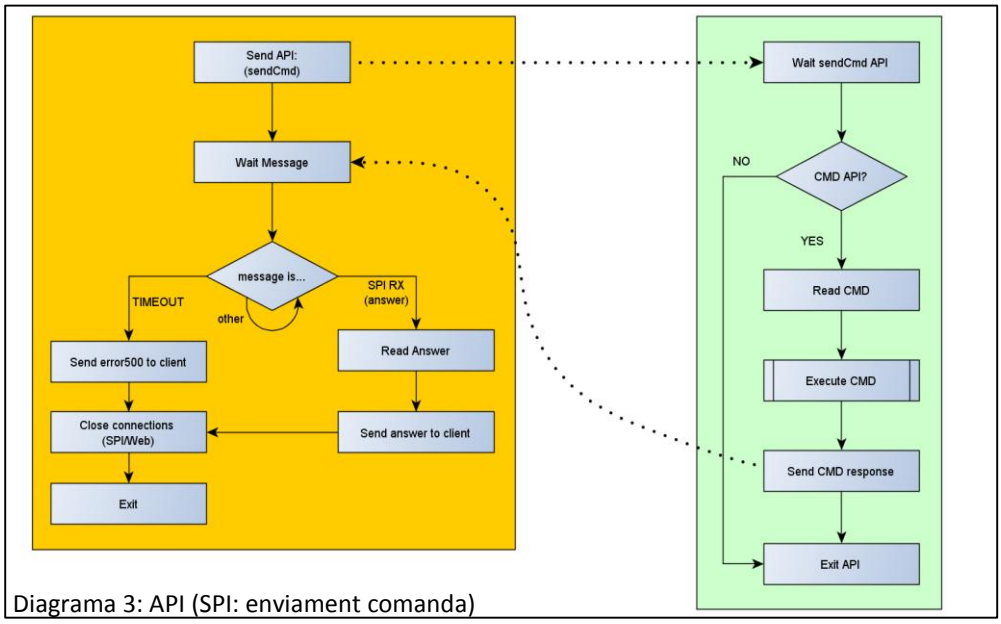


Diagrama 3: API (SPI: enviament comanda)

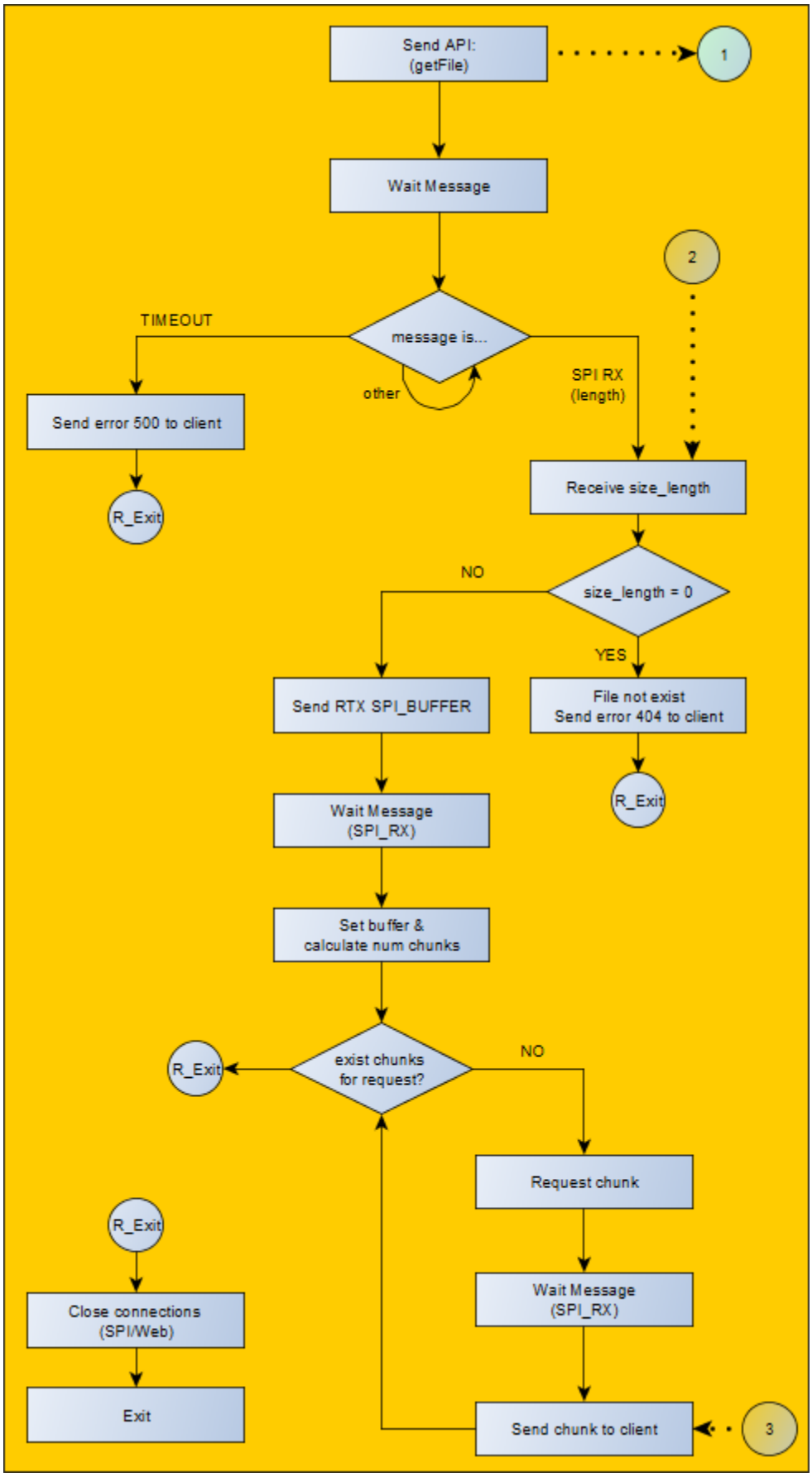


Diagrama 4: API (enviament arxiu (RTX))

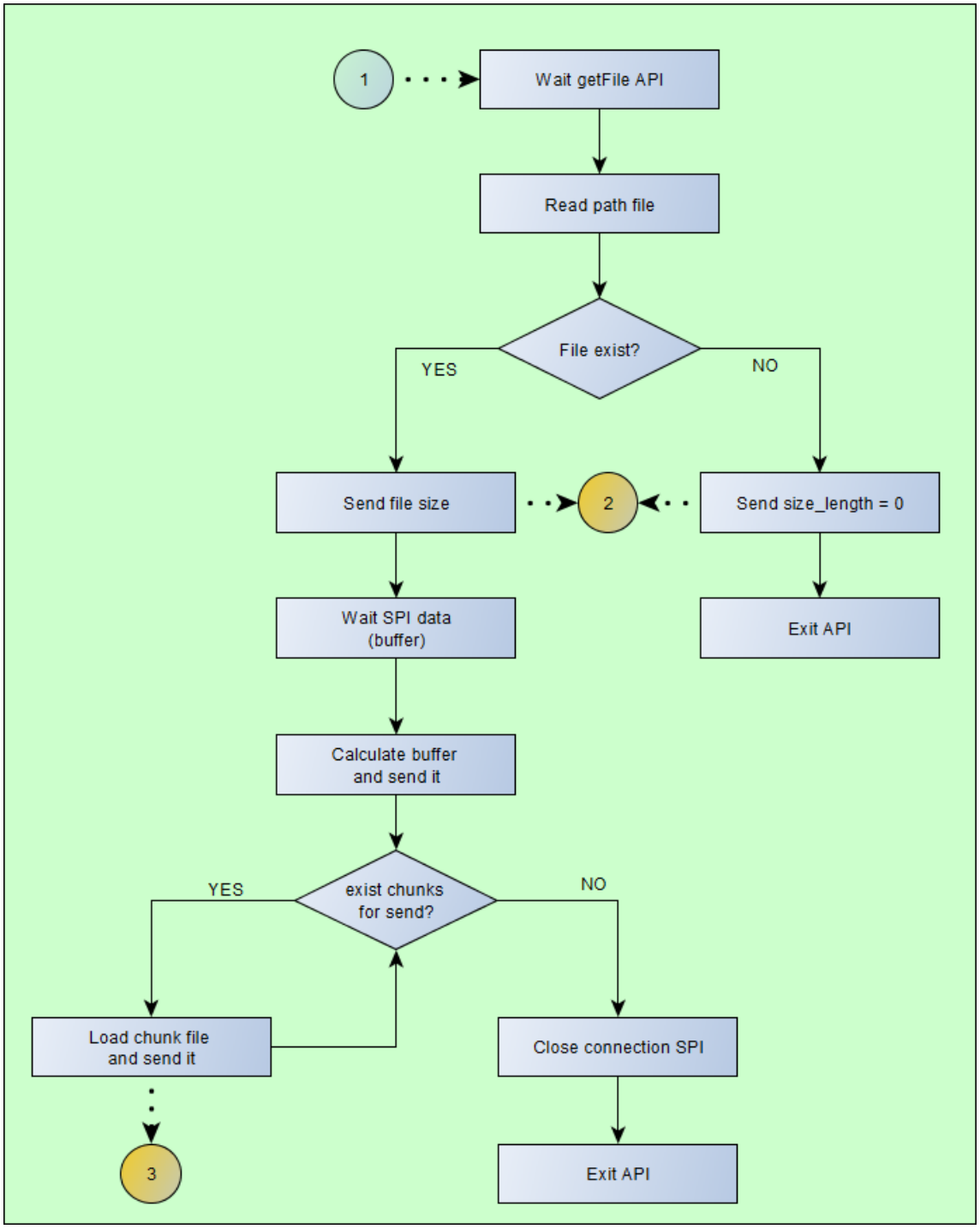


Diagrama 5: API (enviament arxiu (Arduino))

- Missatges definits en el sistema (Mails ROS):

  - SPI_SLAVE_RX_TRIGGER: Aquest missatge ens indicarà que s'ha arribat a enviar una quantitat de bytes prèviament decidida (DrvSpiSlaveSetTrigger). Això evitarà consultar contínuament si s'han rebuts els caràcters necessaris.

  - APP_WEBSERVER_BUSY: Cada cop que s'envia al servidor web una petició, s'enviarà un missatge "busy", per indicar a tots els demes serveis que s'està processant una comanda. Això és especialment útil quan es vol desactivar el mode web.

  - APP_WEBSERVER_IDLE: De la mateixa manera que succeeix en el punt anterior, quan es finalitza la petició i ja no es rebran més informació SPI, es procedirà a monitoritzar de nou els missatges, esperant que entri la comanda de finalització de servidor web.

  - APP_WEBSERVER_EXIT: Un cop que la API de sortida es rebi, s'enviarà un missatge, indicant que s'han de finalitzar certs serveis(Soft AP i servidor web).

- Limitacions del desenvolupament:

  1) SDCARD: Les llibreries d'Arduino usades, tenen una sèrie de limitacions, tant com és pot llegir en la documentació[6] corresponent. D'elles es destaquen:

     I. Ús recomanable de FAT16[27], amb les limitacions que això comporta (en el nostre projecte aquestes limitacions no afectaran).
     II. Els noms dels arxius hauran de complir la nomenclatura 8.3[28]. Fruit d'aquestes limitacions, l'arxiu només podrà contenir 8 lletres pel nom, seguit d'un punt i 3 lletres per l'extensió (sent el punt i l'extensió opcionals). A part, tant els noms dels fitxers com el dels directoris, haurà de ser en majúscules.

[6] http://www.arduino.cc/en/Reference/SDCardNotes

2) Connexions HTTP simultànies: Degut a la limitació de memòria del chip RTX4100 (3kB), només es permetrà una connexió simultània. Degut a que els navegadors web[7] solen obrir múltiples connexions per accelerar la carrega d'una web, és un fet que s'haurà de tenir en compte de cara a realitzar el corresponent disseny web (evitant usar un ús excessiu de recursos).

2.6.6 Funcions i llibreries desenvolupades

Encara que en l'annex –Documentació del codi- es pot accedir a tota la documentació del codi, a continuació es comentaran les noves llibreries que s'han usat i s'indicaran les funcions modificades o creades amb més rellevància en el projecte.

Per part del RTX4100, s'ha treballat amb 8 fitxers (RosPrimitiv.h, DrvSpiSlave.c, Main.c, DrvSpiSlave.h, rtx_AppDns.c, rtx_AppDns.h, rtx_AppWeb.c i rtx_AppWeb.h). Tots els que començant per "rtx_" s'han desenvolupat en aquest projecte. El motiu de generar-los ha sigut el d'estructurar el desenvolupament per a que pugui ser ampliat en un futur.

En quant a les llibreries, s'han usat 3 noves (a part de les nostres), de cara a complir els nostres requisits. Aquestes han sigut: RtxEai, em_gpio i DrvSpiSlave. La seva funció és:

- RtxEai: Servirà per a debugar el dispositiu. Gracies a ella s'enviaran missatges per conèixer l'estat actual del dispositiu i el que està succeint.

- em_gpio: Aquesta s'ha usat en el servidor web i en el Main. La seva funció és de interactuar amb els PINs del mòdul. En el cas que ens ocupa, s'ha usat per a encendre/apagar el LED (fet que provocava les interrupcions en Arduino).

[7] http://www.browserscope.org/?category=network

- DrvSpiSlave: Aquesta llibreria, també s'ha usat tant en el servidor web com en el Main, i s'ha carregat en detriment de la DrvSpi la qual proporcionava les funcions de SPI mestre. Serà una funció important, doncs gran part del desenvolupament depèn de SPI.

A continuació, es mencionaran les funcions principals defininides en els corresponents fitxers:

- **RosPrimitiv.h**

  En aquest fitxer, s'han afegit 4 missatges (els quals s'han comentat anteriorment). Aquests són: SPI_SLAVE_RX_TRIGGER, APP_WEBSERVER_BUSY, APP_WEBSERVER_IDLE, APP_WEBSERVER_EXIT.

- **DrvSpiSlave.c**
  - void SpiSlaveRxIntHandler(rsuint8 vector);
    - S'ha modificat la funció amb l'objectiu de que al rebre 'l' bytes definits per la funció DrvSpiSlaveSetTrigger(rsuint16 l), s'envii el missatge (SPI_SLAVE_RX_DATA).
  - void DrvSpiSlaveSetTrigger(rsuint16 l);
    - Indicarà a SPI el número de bytes que farà saltar el trigger.
  - rsuint16 DrvSpiSlaveGetBufferSize();
    - Recuperarà el valor del buffer SPI definit en el Driver.

- **Main.c:**

  - *static PT_THREAD(**PtWifi_softAP**(struct pt *Pt, const RosMailType *Mail));*
    - Aquesta funció configurarà el AP (AppWifiApSetSoftApInfo) i posteriorment l'iniciarà en mode soft AP (PtAppWifiStartSoftAp) . A part, activa el servei de DNS.

Diagrama 6: Connexions funció PtWifi_softAP (RTX)

  - *static PT_THREAD(**force_wifimode**(struct pt *Pt, const RosMailType *Mail));*
    - Aquesta funció activarà el mode soft AP i el servidor web, sense esperar cap API. S'usarà per a realitzar proves de funcionament.

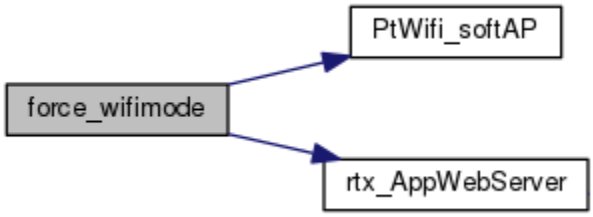


Diagrama 7: Connexions funció force_wifimode (RTX)

  - *static PT_THREAD(**PtMain**(struct pt *Pt, const RosMailType *Mail));*
    Aquesta funció prèviament existent, i es modificarà amb els següents objectius:
    - Modificació de la part SPI: En el codi actual està en mode esclau.
    - Davant l'entrada SPI de la API 16, activa la funció soft AP explicada anteriorment.
    - Davant l'entrada SPI de la API 17, activa el servei de servidor web i es queda a l'espera de la senyal de la SPI de finalització.

- void **ColaTask**(const RosMailType *Mail);
    - Es controla el missatge SPI_SLAVE_RX_DATA. En cas d'estar en mode web i rebre'l, es mirarà si correspon a la API de finalització de servidor web i softAP, per a finalitzar-los en cas necessari.

Un cop carregat el firmware, la ColaTask farà les inicialitzacions pertinents i carregarà el PtMain com a protothread inicial. Aquest protothread en el mode actual, vigilarà l'entrada SPI provinent d'Arduino i actuarà en funció de la seva API. Cal dir que aquesta funció ja existeix, i per tant només es realitzen les modificacions indicades prèviament.

- **Rtx_AppDns.c:**

- *static void **reverse_rsuint32**(rsuint32* data);*
    - Aquesta funció servirà per invertir l'ordre de dades en variables de 32 bits, entenent com a unitat mínima el caràcter (8bits).
    - Exemple: Una entrada de: 0x10,0x20,0x30,0x40, tornaria: 0x40,0x30,0x20,0x10.

- *static PT_THREAD(**PtOnDNSreq**(struct pt *Pt, const RosMailType* Mail));*
    - Aquesta funció analitzarà cada petició de DNS entrant i prepararà la resposta en base a ella.
    - S'usarà la funció auxiliar reverse_rsuint32, per poder utilitzar la IP local com a resposta.

Diagrama 8: Connexions funció PtOnDNSreq (RTX)

- *PtEntryType** ***rtx_AppDnsServer****(RsListEntryType *PtList);*
  - Aquesta funció servirà per a associar una connexió contra una funció determinada. En aquest cas, associarà la funció PtOnDNSreq als paquets UDP amb port 53.

Diagrama 9: Connexions funció rtx_AppDnsServer (RTX)

- **Rtx_AppWeb.c:**

  - *void* ***find_var_cmd****();*
    - Aquesta funció servirà per buscar les variables necessàries. De fet, posicionarà el punter(*ptr_cmd*) en la posició idònia. En cas necessari, també fixarà el caràcter nul al final de la variable. Això serà útil de cara a treballar amb el punter amb les llibreries string.

  - *void* ***parser_webdata****(rsuint8* ptr_data);*
    - Aquesta funció s'encarregarà d'examinar els camps d'una petició web. Es posicionarà el punter(*ptr_url*) en la part URL de la petició i s'examinarà quin és el mètode de la petició. En cas de ser GET, s'executarà la funció anterior, amb l'objectiu de que el punter(*ptr_cmd*) apunti on toca.

Diagrama 10: Connexions funció parser_webdata (RTX)

  - *static PT_THREAD(****Pt_SPI_getfile****(struct pt *Pt, const RosMailType *Mail, AppSocketDataType *pInst, rsuint8 *file));*
    - Aquesta funció intentarà descarregar l'arxiu indicat (*file*), a traves de SPI. Donat que probablement es necessitarà entregar l'arxiu en diverses peticions, es passarà també la instancia del client per a entregar-li els paquets. En cas que el fitxer no existeixi, s'enviarà un 404 al client.

- *static PT_THREAD(**Pt_SPI_sendcmd**(struct pt *Pt, const RosMailType *Mail, AppSocketDataType *pInst, rsuint8 *cmd));*
  - Aquesta funció intentarà executar una comanda via SPI (a Arduino). La comanda en qüestió serà la ubicada en el punter(*cmd*). El resultat serà enviat directament al client.
  - Com a nota addicional, serà possible enviar/rebre qualsevol cadena de caràcters via SPI.

- *static PT_THREAD(**PT_process_rtx_command**(struct pt *Pt, const RosMailType* Mail, AppSocketDataType *pInst, rschar *command));*
  - De forma similar a la funció anterior, aquesta funció executarà una comanda i el resultat serà enviat al client. La diferencia radica en que la comanda s'executarà en el mateix RTX.
  - Aquesta funció permetrà llegir/modificar dades del RTX tal com la NVS o consultar dades com la temperatura del chip WiFi.

- *static PT_THREAD(**PtOnHTTPreq**(struct pt *Pt, const RosMailType* Mail));*
  - Aquesta funció s'executarà en cada paquet rebut pel port 80 TCP. La seva funció serà llegir una petició HTTP i preparar la corresponent resposta.
  - La funcionalitat detallada d'aquest mètode, es pot veure en el Diagrama 1.

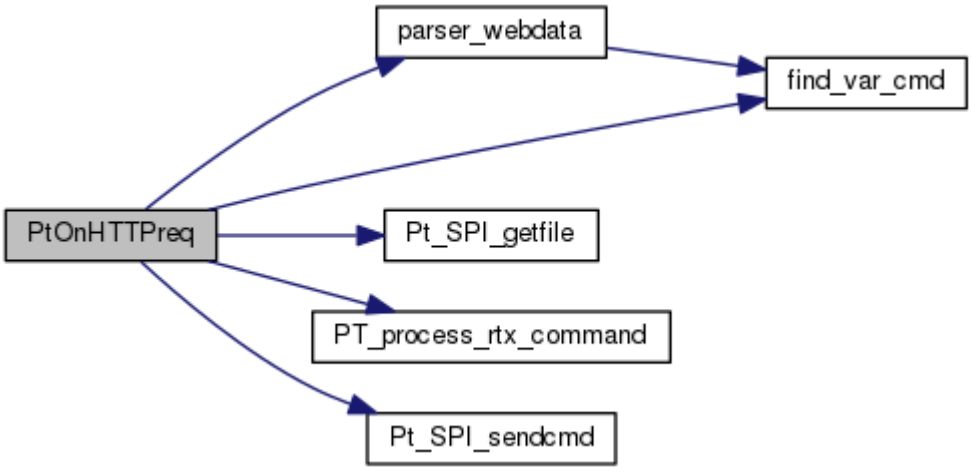


Diagrama 11: Connexions funció PtOnHTTPreq (RTX)

- *PtEntryType** ***rtx_AppWebServer****(RsListEntryType *PtList);*
  - Aquesta funció servirà per a associar una connexió contra una funció determinada. En aquest cas, associarà la funció PtOnHTTPreq als paquets TCP que vagin contra el port 80.

Diagrama 12: Connexions funció rtx_AppWebServer (RTX)

En quant a Arduino, no s'ha usat cap llibreria nova a la proporcionada per SCK. En quant a mètodes si que s'han desenvolupat i modificat els existents (situats en el fitxer *SCDVK.cpp*). La definició de les funcions és:

- void **ISR1**();
  - Canvia el mode d'operació a webserver_mode si el previ és softap_mode, o el passa softap_mode en cas contrari.

- void **ISR2**();
  - Canvia el mode d'operació a webserver_exit si el previ és webserver_mode.

- void **ISR_RTX_SPI**();
  - Funció que indicarà una interrupció (correspon al LED. En certes ocasions indicarà que RTX ha d'enviar/rebre dades).

- byte **dec_to_hex**(byte * dst, long dec_num);
    - Funció que convertirà una variable long, en un array de variables byte. El resultat de la funció, seran els bytes necessaris per representar aquest número.
    - Aquesta funció s'usarà per exemple, per enviar la mida d'un fitxer en bytes.

- void **connread_rtx_spi**(int l);
    - Llegirà un conjunt de *l* bytes via SPI i els col·locarà en una variable per a la posterior lectura. (Per rebre els bytes, en realitat estarà enviant bytes 0x00, per a rebre la corresponent informació).

- void **connsend_rtx_spi**(byte * data, int len);
    - Funció que enviarà via SPI, el contingut del punter *data* (la longitud de la cadena serà la indicada per *len*).
- void **SCDVK::begin**();

Aquesta funció ja existeix. De fet és la executada pel setup() d'Arduino. No obstant, les següents modificacions seran necessàries:

- El MUX estarà en mode de firmware del RTX (per poder depurar).
- El pin SS(SPI) de SDCARD, estarà en ON.
- S'habilitarà el PIN en mode entrada de RTX_IRQ_SPI. (interrupció).
- S'inicialitzarà el port USB debug a 9600bps (per depurar Arduino).
- S'intentarà inicialitzar la SD (comportarà la inicialització de SPI).

- **voidSCDVK::webserver_mode**();
  - Si el servidor WiFi no esta iniciat, envia el codi d'inici de servidor web via SPI (API:17) i activa les interrupcions.
  - En cas d'estar iniciat, es comprova si RTX ha enviat una petició d'enviament. En cas afirmatiu es llegeix.
  - Un cop realitzat, s'entregaran els missatges de depuració pendents.

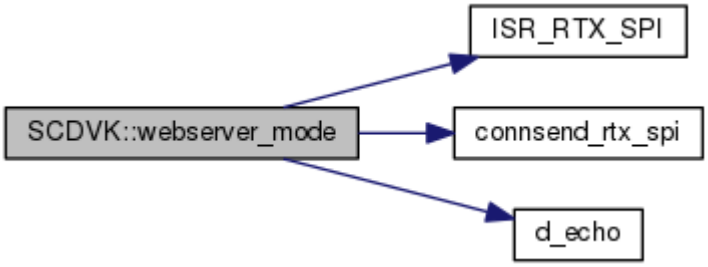


Diagrama 13: Connexions funció webserver_mode (Arduino)

- **void SCDVK::webserver_exit**();
  - Si el servidor web està activat, envia el codi de finalització via SPI (API:18) i desactiva les interrupcions de RTC.
  - Al completar-se l'ordre anterior, l'estat habitual serà el de "normal_mode".

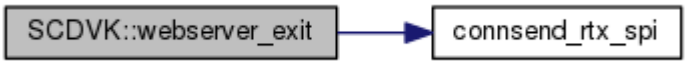


Diagrama 14: Connexions funció webserver_exit (Arduino)

- void **SCDVK::softap_mode**();
  - Si el soft AP no està activat, s'enviarà via SPI la API corresponent per activar-lo (API:16).
  - Al completar-se l'ordre anterior, es pararan les interrupcions de RTX.

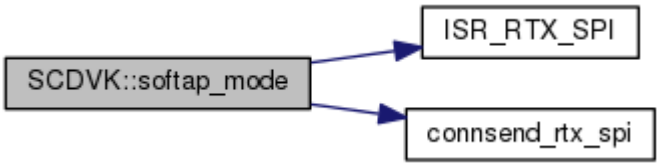


Diagrama 15: Connexions funció softap_mode (Arduino)

- void **SCDVK::test_serverweb**();
  - Funció que força el mode serverweb en el sistema (s'ha d'usar juntament al mode force_wifimode en el RTX). No s'enviarà cap API d'activació usant d'aquest mode. (API's: 16/17).

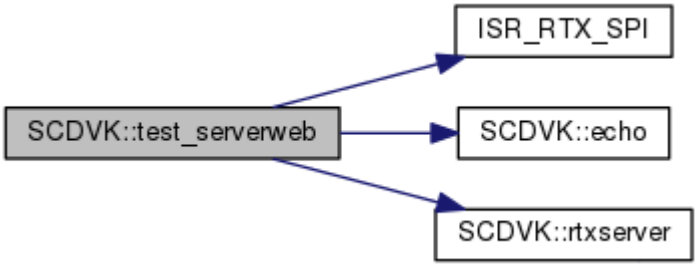


Diagrama 16: Connexions funció test_serverweb (Arduino)

- void **SCDVK::execute**();
  - Aquesta funció s'ha modificat afegint uns quants estats al sistema.
  - Activa un mode o un altre, en base a la variable (operation_mode). Per tant, tindrà que accedir a totes les funcions de mode.

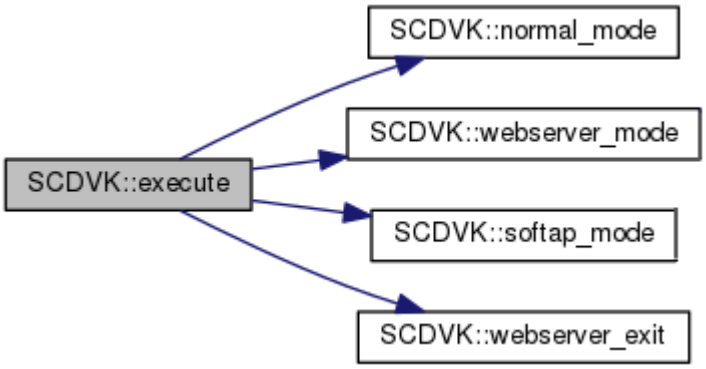


Diagrama 17: Connexions funció execute (Arduino)

- void **SCDVK::rtxserver**();
  - És la funció més important del sistema Arduino i el seu funcionament es pot veure en els diagrames [3] i [5]. Servirà per realitzar les accions que indiqui la API. De moment només es suporten 2 API's:

    I. API 160: Execució comanda de RTX a Arduino. Executarà una comanda en Arduino, la qual provindrà de RTX(SPI). Després es retornarà el resultat. Per a executar-lo en Arduino, farà ús del mètode exec_cmd(..). Posteriorment el contingut de la variable *data* d'aquet mètode, s'enviarà com a resultat al RTX.

    II. API 161: Obtenció d'arxiu. S'intentarà accedir al fitxer indicat per aquest mètode i s'entregarà mitjançant chunks al RTX. En cas de que l'arxiu no existeixi, en l'entrega de la mida de la longitud, s'indicarà (0). És una funció fonamental del sistema, i degut al sincronisme que ha de portar juntament amb RTX (mitjançant les interrupcions), és pot dir que ha sigut la més complexa. Usarà les llibreries SDCard de cara a obtenir el fitxer.

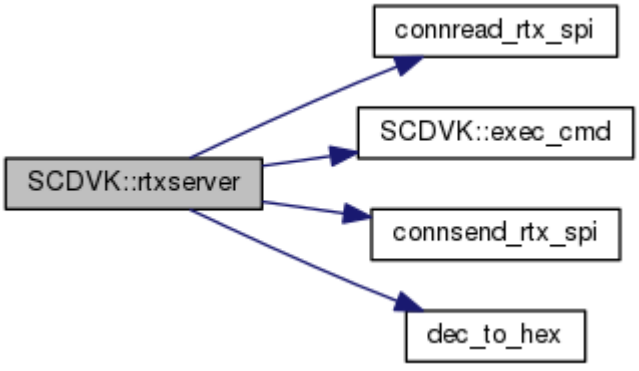


Diagrama 18: Connexions funció rtxserver (Arduino)

- void **d_echo**();
  - Serveix per enviar tots els missatges pendents entre els ports sèrie. Exemples d'ús són: depurar el RTX o pujar una CoLApp.

- void **SCDVK::exec_cmd**(char *cmd, char *data);
  - Serveix per executar un comandament indicat via SPI pel RTX. El resultat, es guardarà en la variable data.

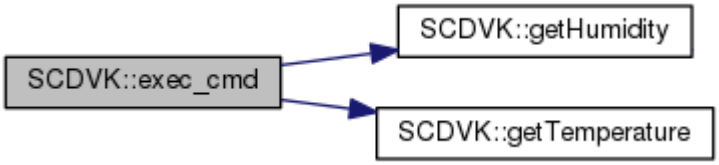


Diagrama 19: Connexions funció exec_cmd (Arduino)

2.6.7 Exemples de funcionament

- Activació mode soft AP & Servidor web: Un cop iniciat el sistema, s'haurà de col·locar en mode soft AP i posteriorment activar el servidor web. En un estat inicial, l'únic LED encès serà el taronja.

Per a fer-ho, caldrà polsar el primer boto en el Arduino DUE. Un cop fet això, Arduino enviarà la comanda a RTX per activar el soft AP. En aquest moment, el punt d'accés ja serà accessible per qualsevol dispositiu, i s'encendrà el LED verd. El debug d'aquests missatges es poden veure en la Figura 18 (RTX i Arduino respectivament). D'altre banda en la Figura 19, es pot veure com apareix el punt d'accés en un escaneig WiFi.

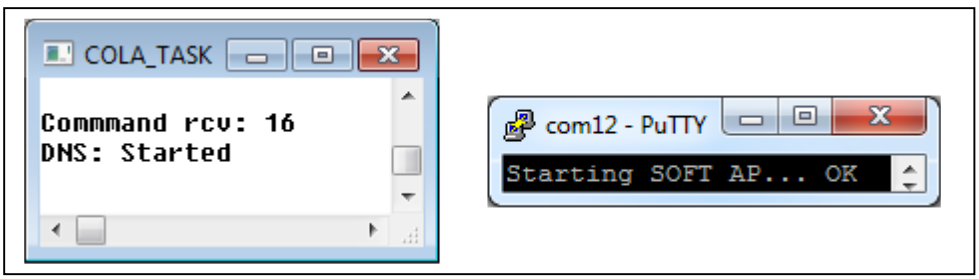


Figura 18: Debug de l'activació del soft AP (RTX&Arduino)

Figura 19: Escaneig del AP virtual

Arribats a aquest punt, s'haurà d'iniciar el servidor web. Per a fer-ho, s'haurà de prémer de nou el primer botó. En aquest mode, també s'encendrà el LED blau, per tant tots els LEDS estaran encesos. El debug d'aquests missatges es poden veure en la figura 20 (RTX i Arduino respectivament).

Figura 20: Debug de l'activació del servidor WEB

- Desconnexió servidor web & soft AP: Un cop activat el servidor web, existeix la possibilitat de tornar el sistema a un estat previ. Per a fer-ho, s'haurà de polsar el segon botó. Cal dir que per prémer aquest boto, el servidor web ha d'estar prèviament iniciat. Degut a que es passarà a l'estat inicial, un cop fet, únicament romandrà encès el LED taronja. El debug d'aquests missatges es poden veure en la figura 21 (RTX i Arduino respectivament).

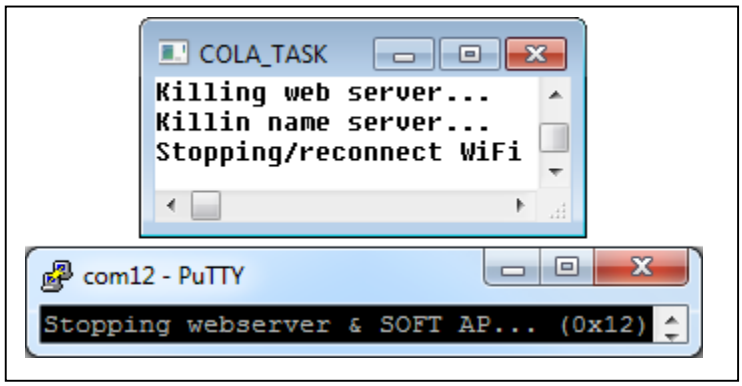


Figura 21: Debug de la desactivació del servidor WEB i el Soft AP

- Exemple de carrega de pàgina mitjançant la SD: Al iniciar el servidor web, serà possible carregar arxius que existeixin en la targeta SD, mitjançant la comunicació del bus SPI. Donat que normalment els arxius seran superiors a la memòria i al buffer SPI del mòdul WiFi, cada cop que es rebi un fragment d'un arxiu (chunk) s'enviarà al client.

  Un exemple d'això, es pot veure descarregant la pàgina inicial. Aquesta pagina, descarregarà dos fitxers tal com es pot veure en la Figura 22 (debug RTX). Es descarregarà la pròpia pàgina (/WEB/INDEX.HTM) i una petita imatge(/WEB/LOGO.PNG) que s'usa de prova de concepte. La Figura 23 mostra el resultat de carrega d'una pàgina web en el navegador.

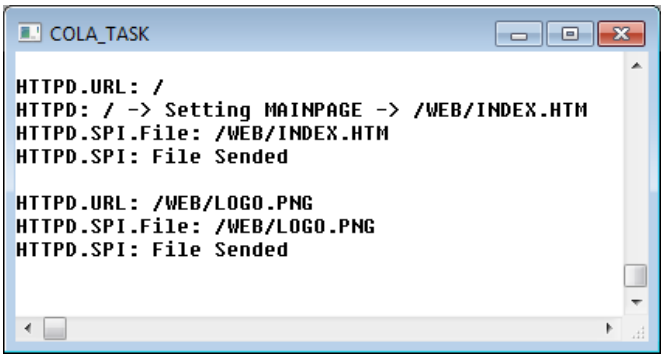


Figura 22: Petició pàgina web d'exemple (debug RTX)

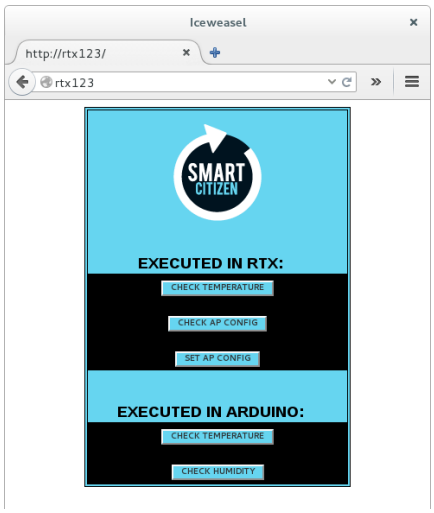


Figura 23: Pàgina web d'exemple (pàgina amb logo)

Com es veu gracies a la Figura 24 (debug d'Arduino), s'entrega l'arxiu fragmentat en diverses parts, que seran enviades al client. En el primer cas, es fragmenta l'arxiu en 4 parts de (1806=511+511+511+273)bytes. En el segon, es fragmentarà l'arxiu en 6 parts (2923=511-511-511-511-511-368)bytes.

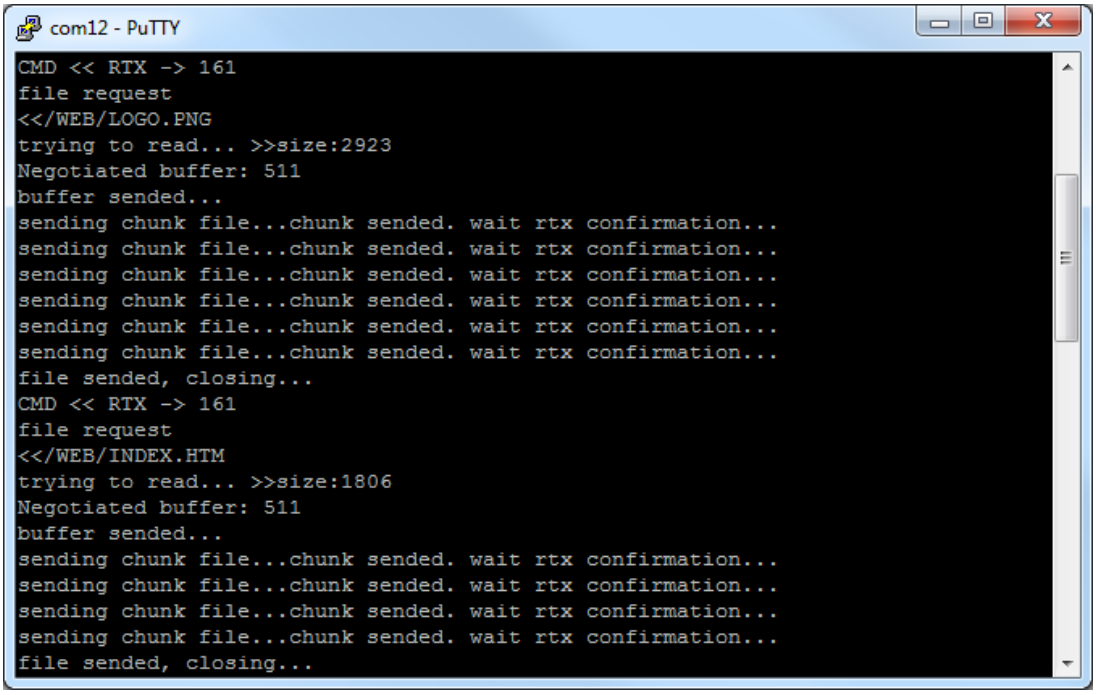


Figura 24: Petició pàgina web fragmentada (debug Arduino)

- Exemple d'execució de comanda RTX: L'execució de comandes en el mateix RTX permetrà prescindir de la comunicació SPI. Això es deu a que la comunicació no sortirà del mateix RTX. Un exemple per veure la temperatura que marca el chip, es pot veure en la Figura 25, mentre que el debug en RTX es pot veure en la Figura 26.

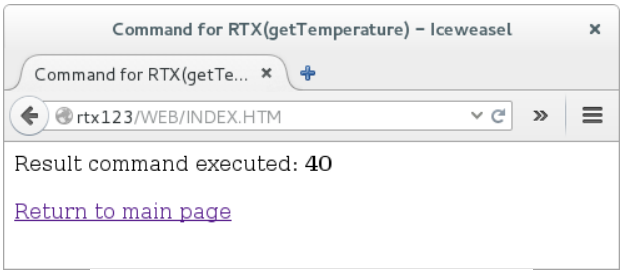


Figura 25: Execució comanda en RTX

En el log es veu la comanda que s'envia al RTX (getTemperature) juntament amb la ruta. En aquest cas la ruta s'ometrà, ja que al existir una comanda, té prioritat sobre l'arxiu.

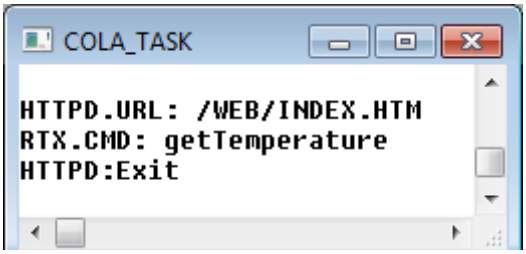


Figura 26: Execució comanda en RTX(Debug RTX)

- Exemple d'execució de comanda Arduino: Aquest cas és una variant del punt anterior. No obstant en aquest cas, s'usarà una comunicació SPI, ja que la comanda s'executarà en l'altre extrem (Arduino). Un cop executada, s'enviarà la resposta mitjançant el mateix bus, i es mostrarà en la web, tal com es pot veure en la Figura 27. En aquest cas, el log de RTX, representat per la Figura 28 indicarà que la comanda s'executa mitjançant SPI. En quant el log de Arduino es pot veure en la Figura 29.

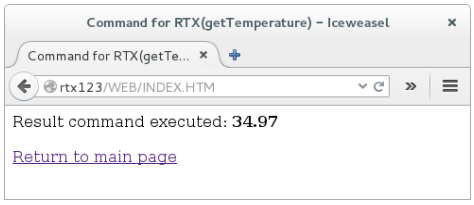


Figura 27: Execució comanda en Arduino

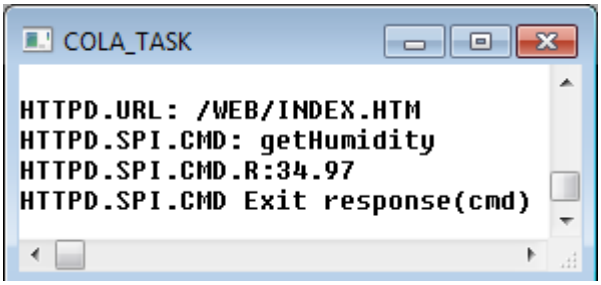


Figura 28: Execució comanda en Arduino (Debug RTX)

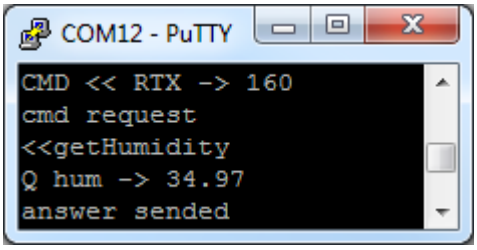


Figura 29: Execució comanda en Arduino (Debug Arduino)

2.6.8 Exemple de funcionament sota un entorn real

En aquest últim punt, s'usarà un smarthpone per a provar el desenvolupament en un entorn real. En la presentació, es pot veure un petit vídeo que mostrarà l'exemple en temps real.

Aquesta prova es realitzarà mitjançant un mòbil Samsung Galaxy S5, i la aplicació "Internet" la qual ve pre-instalada.

- Activació del soft AP & Servidor web: El primer pas, serà activar el mode soft AP i el servidor web. Aquest pas ja s'ha explicat en els punts anteriors, així que s'ometrà.

- Connexió al AP (mòbil): Com es veu en la Figura 30, el primer pas lògic serà connectar-nos al punt d'accés. Com a informació addicional, i com es pot veure en el log del RTX (Figura 31), un cop ho fem, el mòbil intentarà establir algunes connexions HTTP (funcionament habitual dels smartphones). En tot cas, aquestes seran negatives (Error 404) ja que en aquest mode, només es pot accedir a la nostre pàgina web.

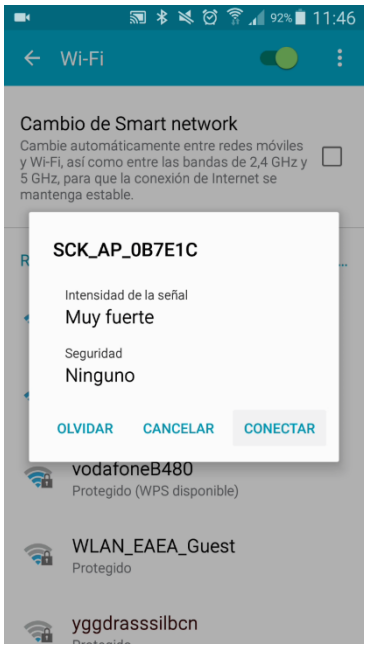


Figura 30: Connexió al AP

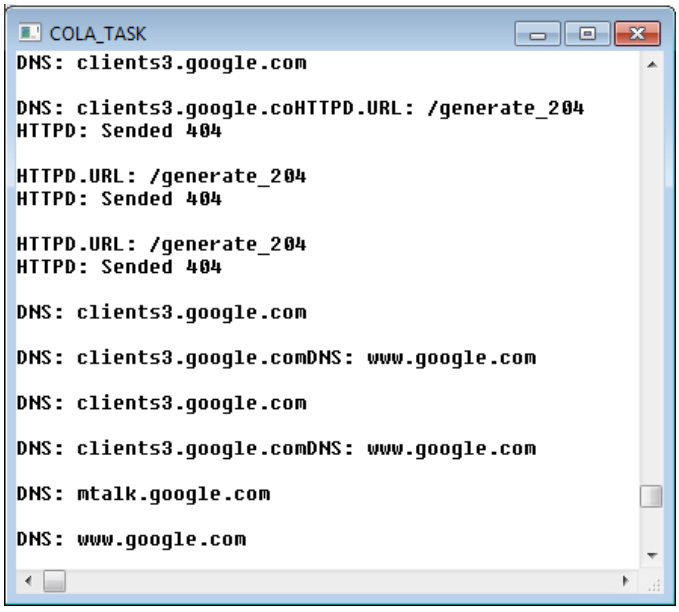


Figura 31: Debug connexió AP

- Accés a una pàgina web: Per provar el servidor web, podem accedir a qualsevol pàgina sempre que sigui HTTP. Per a fer la prova, usarem el domini "test", que lògicament no existeix.

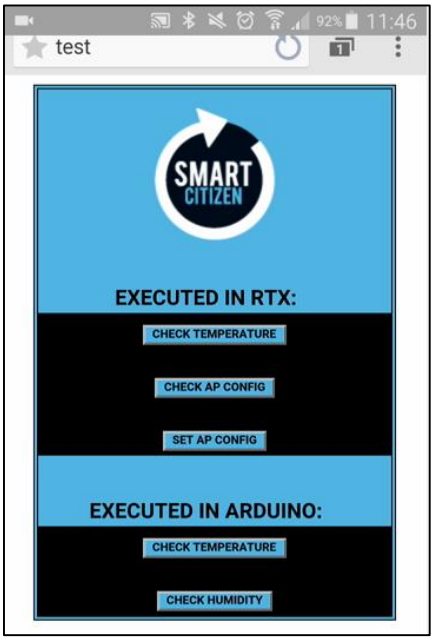


Figura 32: Pàgina principal

Un cop que s'accedeixi, es veurà la pàgina principal (Figura 32). En quant el debug, es pot veure que el navegador ha intentat traduir el nom, i quines són les pàgines descarregades per la pàgina web. També es veu l'intent fallat de l'arxiu "favicon.ico". Aquest fitxer correspon a la icona favicon[8], la qual es usada per identificar la pàgina. En el cas que ens ocupa, no s'ha pujat cap fitxer, per tant es mostrarà el 404. Tot això es veurà en el debug corresponent del RTX, el qual es pot veure en la Figura 33.

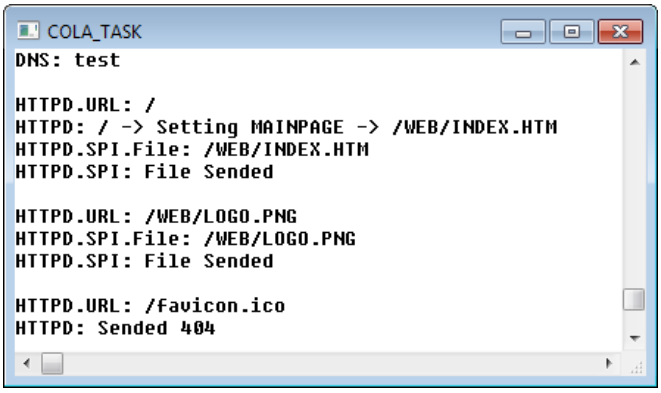


Figura 33: Pàgina principal (debug)

[8] http://es.wikipedia.org/wiki/Favicon

- Funcionalitats dels botons: En aquest punt es mostraran totes les funcionalitats possibles.

    1) Check temperature(rtx): Executarà una comanda en RTX per comprovar quina és la temperatura del mòdul WiFi.

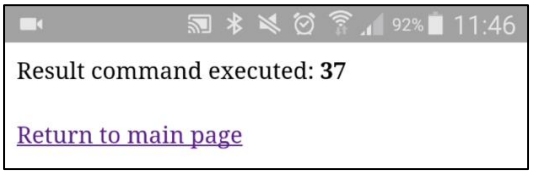


Figura 34: Execució comanda en RTX

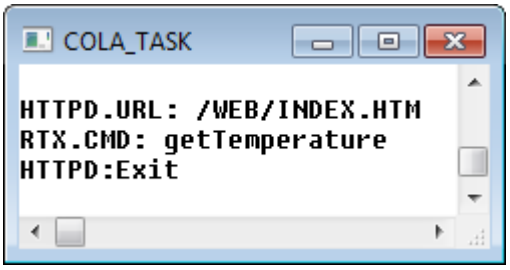


Figura 35: Execució comanda en RTX (debug RTX)

    2) Check apconfig(rtx): Executarà una comanda que permetrà veure la IP i el ESSID del punt d'accés (no la del soft AP, sinó la configuració en mode normal).

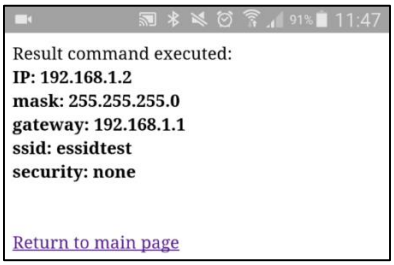


Figura 36: Consulta configuració WiFi

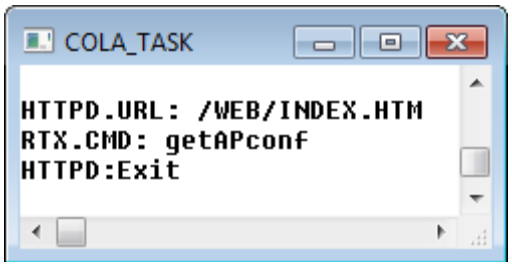


Figura 37: Consulta configuració WiFi(debug RTX)

3) Set apconfig (rtx): Aquest botó executarà una pàgina (Figura 38) que permetrà canviar la configuració de xarxa (IP, mascara, gateway) i la configuració AP (ESSID, xifratge, contrasenya). El seu debug es pot veure en la figura 39.

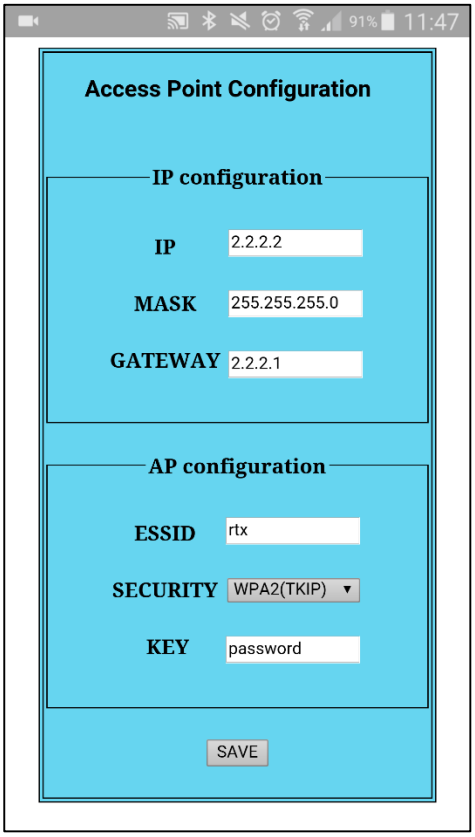


Figura 38: Modificació configuració WiFi

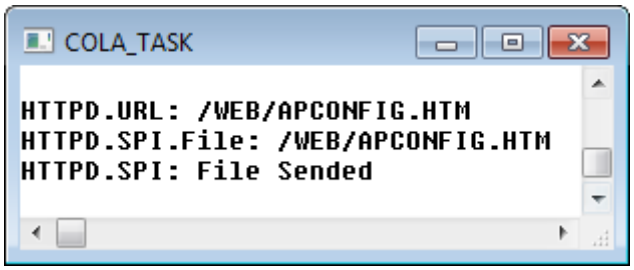


Figura 39: Modificació configuració WiFi(debug RTX)

Al prémer el botó de salvar, s'enviarà la comanda que farà aquesta modificació (Figura 40). Cal dir, que en el debug (Figura 41), la comanda no s'aprecia correctament (s'afegeixen caràcters estranys).

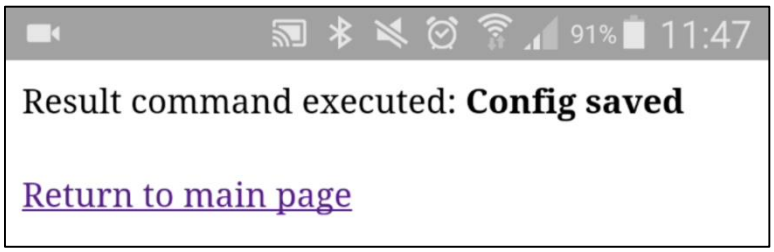


Figura 40: Guardar modificació configuració WiFi

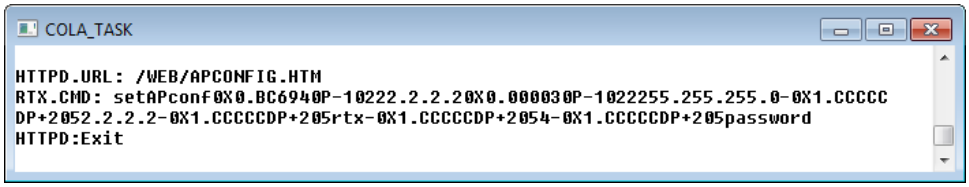


Figura 41: Guardar modificació configuració WiFi(debug RTX)

4) Check apconfig (rtx): De nou es mostrarà aquesta opció per veure si el set apconfig ha funcionat correctament. Com es veu en la Figura 42, el canvi ha sigut correcte.

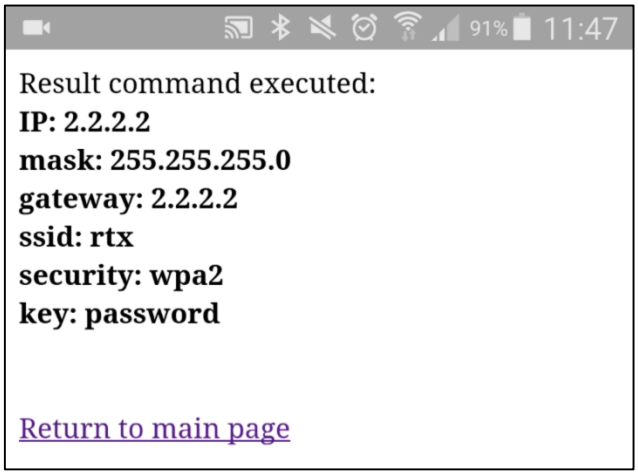


Figura 42: Consulta configuració WiFi

5) Check Temperature: Aquesta comanda enviarà un comandament via Arduino per veure la temperatura ambient (Figura 43). Aquest tipus de comanda serà la més usada pel sistema. El debug tant de RTX com d'Arduino es veuen en les Figures 44 i 45 respectivament.

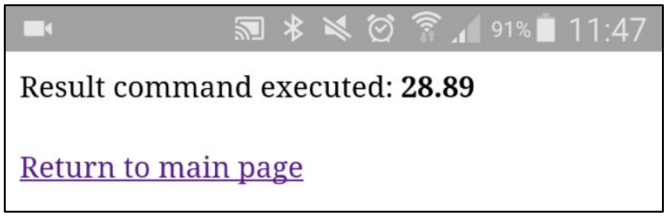


Figura 43: Consulta temperatura en Arduino

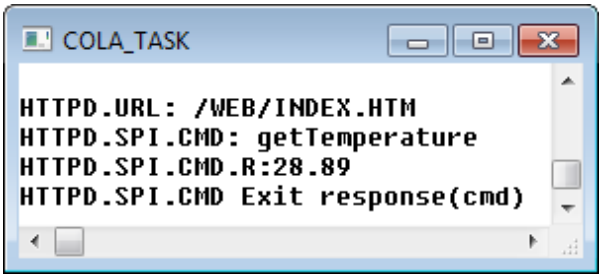


Figura 44: Consulta temperatura en Arduino (debug RTX)

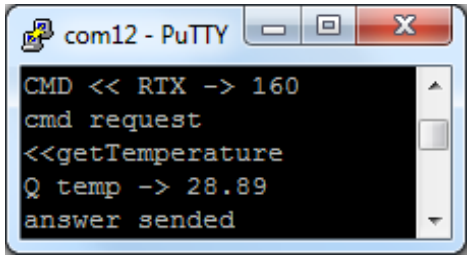


Figura 45: Consulta temperatura en Arduino (debug Arduino)

6) Check Humidity: Com l'anterior, aquesta comanda s'executarà en Arduino i es pot apreciar en la Figura 46. En aquest cas, mostra la humitat ambiental i els debugs tant de RTX com d'Arduino es veuen en les Figures 47 i 48 respectivament.

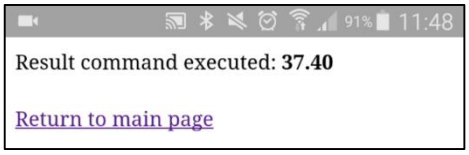


Figura 46: Consulta humitat en Arduino

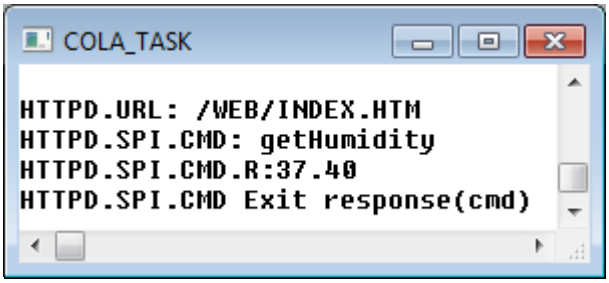


Figura 47: Consulta humitat en Arduino (debug RTX)

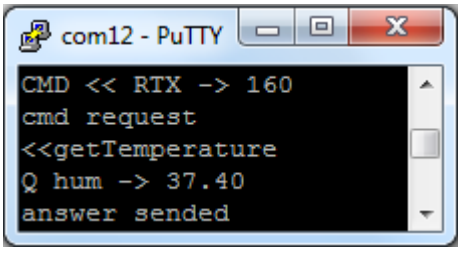


Figura 48: Consulta humitat en Arduino (debug Arduino)

3 Conclusions i treball futur

L'objectiu principal d'aquest projecte, ha sigut ampliar les funcionalitats d'un firmware pel mòdul WiFi RTX4100. En concret, s'han afegit les funcionalitats de soft AP i de servidor Web.

Degut a la poca memòria que ofereix el mòdul, s'han tingut que superar diversos obstacles. No obstant, com s'ha vist en el projecte s'ha decidit usar la targeta SD per a guardar el codi web, fet que ha estalviat la memòria corresponent. El fet de que la SD sigui accessible pel RTX ofereix un món de possibilitats. Ara serà possible carregar imatges o scripts (característica que seria impensable sense la SD).

Un altre objectiu interesant, ha sigut la possibilitat de controlar i d'interactuar en base a les variables de les peticions HTTP. En aquesta part s'han aconseguit els punts:

- Reprogramació de la NVS, per poder salvar configuracions del AP, prèviament introduïdes per formularis web.

- Capacitat d'executar comandes tant en Arduino com en RTX. (Veure temperatura, consultar configuracions...)

Arribats a aquest punt, encara que s'han assolits tots els objectius d'aquest projecte, encara queda feina pendent de cara a obtenir la versió final del SCK. De fet, encara queden tasques per a desenvolupar. Malauradament aquestes, quedaven fora l'abast d'aquest projecte. En tot cas, alguns objectius poden ser:

1) Confeccionar el codi (Arduino), que suporti tots els requisits de Smart Citizen.

2) Desenvolupar una web que compleixi tots els requeriments de Smart Citizen (en el treball únicament s'ha presentat una proba de concepte).

Amb aquests punts s'estarà més a prop de treure una versió final del model SCK.

Glossari

AP: Access Point (Punt d'accés).

API: Application Programming Interface (Interfície de programació d'aplicacions).

BPS: Bauds per segon

C: Llenguatge de programació C

CLK: Clock (Rellotge).

CoLA: Co-Located Application

CPHA: Clock PHAse (Fase de rellotge).

CPOL: Clock POLarity (Polaritat del rellotge).

CPU: Central Processing Unit (Unitat central de processament).

CS: Chip Select (Seleció de chip).

DHCP: Dynamic Host Configuration Protocol (Protocol de configuració dinàmics de host).

DNS: Domain Name System (Sistema de noms de domini).

EEPROM: Electrically Erasable Programmable Read-Only Memory (ROM programable i esborrable elèctricament).

FTP: File Transfer Protocol (Protocol de transferència de fitxers).

GPIO: General Purpose Input/Output (Entrada/Sortida de propòsit general).

HTTP: HyperText Transfer Protocol (Protocol de transferència de hipertext).

I/O: Input/Output (Entrada/Sortida).

IP: Internet Protocol (Protocol d'Internet).

IRQ: Interrupt Request (Petició d'Interrupció).

ISR: Interrupt Service Routine (Rutina d'interrupció de servei).

LED: Light-Emitting Diode (Díode emissor de llum).

LSB: Least Significant Bit (Bit menys significatiu).

MAC: Media Access Control (Control d'accés al medi).

MCU: Micro Control Unit (Microcontrolador).

MISO: Master Input Slave Output (Entrada mestre/sortida esclau).

MOSI: Master Output Slave Input (Sortida mestre/entrada esclau).

MSB: Most Significant Bit (Bit més significatiu).

NVS: Non-Volatile Storage (Emmagatzematge no volàtil)

RAM: Random Access Memory (Memòria d'accés aleatori).

ROS: RTX Operating System (Sistema operatiu de RTX).

RTC: Real Time Clock (Rellotge en temps real).

SDK: Software Development Kit (Kit de desenvolupament).

SPCR: SPi Control Register (Registre de control del SPI).

SPI: Serial Peripheral Interface (Interfície de perifèrics sèrie).

SRAM: Static RAM (RAM estàtica).

SS: Slave Select (Selecció de esclau).

SSID: Service Set IDentifier (Identificador de conjunt de serveis).

TCP: Transmission Control Protocol (Protocol de control de transmissió).

UART: Universal Asynchronous Receiver-Transmitter (Transmissor-Receptor Asíncron Universal).

UDP: User Datagram Protocol (Protocol de datagrama de usuari).

URI: Uniform Resource Identifier (Identificador de recursos uniforme).

USART: Universal Synchronous Receiver-Transmitter (Transmissor-Receptor Síncron Universal).

WiFi : WIreless FIdelity (Fidelitat sense fils).

XML: eXtensible Markup Language (Llenguatge de marques extensible).

Referències

[1] **Smart Citizen**(2015). SmartCitizen Docs. Smart Citizen. [en línia]. [Data de consulta: 7 de març de 2015].
<http://docs.smartcitizen.me/#/start/hardware>

[2] **Smart Citizen**(2015). Smart Citizen. [en línia]. [Data de consulta: 7 de març de 2015].
<https://smartcitizen.me>

[3] **Arduino**(2015). Arduino. [en línia]. [Data de consulta: 7 de març de 2015].
<http://arduino.cc>

[4] **Microchip Technology Inc**(2014). RN131 - Wireless Modules. [en línia]. [Data de consulta: 20 de març de 2015].
<www.microchip.com/rn131>

[5] **RTX A/S**(2013). Low Power Wi-Fi RTX4100. [en línia]. [Data de consulta: 7 de març de 2015].
<http://www.rtx.dk/RTX41xx_Wi-Fi_modules-3921.aspx>

[6] **fablabbcn**(2015). "Smart Citizen Kit Develoment Kit (DVK) for RTX4100 and Arduino Due". [documentació en línia]. [Data de consulta: 8 de març de 2015]
<https://github.com/fablabbcn/SmartCitizenDVK>

[7] **Colom, Miguel** (2015). Analysis, improvement, and development of new firmware for the smart citizen kit ambient board. [treball de fi de grau en línia]. UOC. [Data de consulta: 7 de març de 2015].
<http://hdl.handle.net/10609/40042>

[8] **RTX A/S**(2013). Low Power Wi-Fi RTX4100 RELATED MATERIAL. [en línia]. [Data de consulta: 8 de març de 2015].
<http://www.rtx.dk/Default.aspx?ID=3980>

[9] **Wikipedia**(2015). "Serial Peripheral Interface Bus". [documentació en línia]. [Data de consulta: 20 de març de 2015].
<http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus>

[10] **IEEE**(2012). "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications". [documentació en línia]. [Data de consulta: 15 de juny de 2015].
<http://standards.ieee.org/getieee802/download/802.11-2012.pdf>

[11] **Silicon Labs**(2015). 32-bit MCU [en línia]. [Data de consulta: 23 de març de 2015].
<http://www.silabs.com/products/mcu/32-bit/Pages/32-bit-microcontrollers.aspx>

[12]**Qualcomm**(2013). "AR4100P". [documentació en línia]. [Data de consulta: 15 de juny de 2015].
<http://www.qca.qualcomm.com/wp-content/uploads/2013/11/AR4100P.pdf>

[13]**Arduino**(2015). Arduino - Compare. [en línia]. [Data de consulta: 20 de març de 2015].
<http://arduino.cc/en/Products.Compare>

[14]**Silicon Labs**(2014). "EFM32G230 DATASHEET". [documentació en línia]. [Data de consulta: 21 de març de 2015].
<http://www.silabs.com/Support%20Documents/TechnicalDocs/EFM32G230.pdf>

[15]**Silicon Labs**(2013). "USART/UART - Asynchronous mode". [documentació en línia]. [Data de consulta: 22 de març de 2015].
<http://www.silabs.com/Support%20Documents/TechnicalDocs/AN0045.pdf>

[16]**Arduino**(2015). ArduinoShields. [en línia]. [Data de consulta: 21 de març de 2015].
<http://www.arduino.cc/en/Main/ArduinoShields>

[17]**Arduino**(2015). ArduinoBoardDue. [en línia]. [Data de consulta: 20 de març de 2015].
<http://www.arduino.cc/en/Main/ArduinoBoardDue>

[18]**Arduino**(2015). ArduinoBoardZero. [en línia]. [Data de consulta: 20 de març de 2015].
<http://www.arduino.cc/en/Main/ArduinoBoardZero>

[19]**Texas Instruments**(2014). "SN74AHC1G125 Single Bus Buffer Gate With 3-State Output". [documentació en línia]. [Data de consulta: 23 de març de 2015]
<http://www.ti.com/lit/ds/symlink/sn74ahc1g125.pdf>

[20]**Texas Instruments**(2005). "High-Speed CMOS Logic Hex Buffers, Inverting and Non-Inverting". [documentació en línia]. [Data de consulta: 23 de març de 2015]
<http://www.mouser.com/ds/2/405/schs205i-101882.pdf>

[21]**Mockapetris, Paul**(1983). "DOMAIN NAMES - IMPLEMENTATION and SPECIFICATION". [documentació en línia]. [Data de consulta: 25 de març de 2015]
<https://tools.ietf.org/html/rfc883>

[22]**Wireshark Foundation**(2015). Wireshark. [en línia]. [Data de consulta: 25 de març de 2015]
<https://www.wireshark.org>

[23]**Fielding Roy, Gettys Jim, Mogul Jeff, i altres**(1999). "Hypertext Transfer Protocol -- HTTP/1.1". [documentació en línia]. [Data de consulta: 25 de març de 2015]
<http://tools.ietf.org/html/rfc2616>

[24]**Arduino**(2015). Arduino Software. [en línia]. [Data de consulta: 19 d'abril de 2015].
<http://www.arduino.cc/en/Main/Software>

[25]**Netbeans Community**(2015). Netbeans. [en línia]. [Data de consulta: 23 de març de 2015].
<https://netbeans.org>

[26]**Colom, Miguel**(2015). "SmartCitizenRTX4100". [codi font en línia]. [Data de consulta: 20 de abril de 2015].
< https://github.com/mcolom/SmartCitizenRTX4100/blob/master/Main.c>

[27]**Wikipedia**(2015). "File Allocation Table". [en línia]. [Data de consulta: 4 de abril de 2015].
<http://en.wikipedia.org/wiki/File_Allocation_Table.<http://en.wikipedia.org/wiki/File_Allocation_Table>

[28]**Wikipedia**(2015). "8.3 filename". [en línia]. [Data de consulta: 4 de abril de 2015].
<http://en.wikipedia.org/wiki/8.3_filename>

Annex

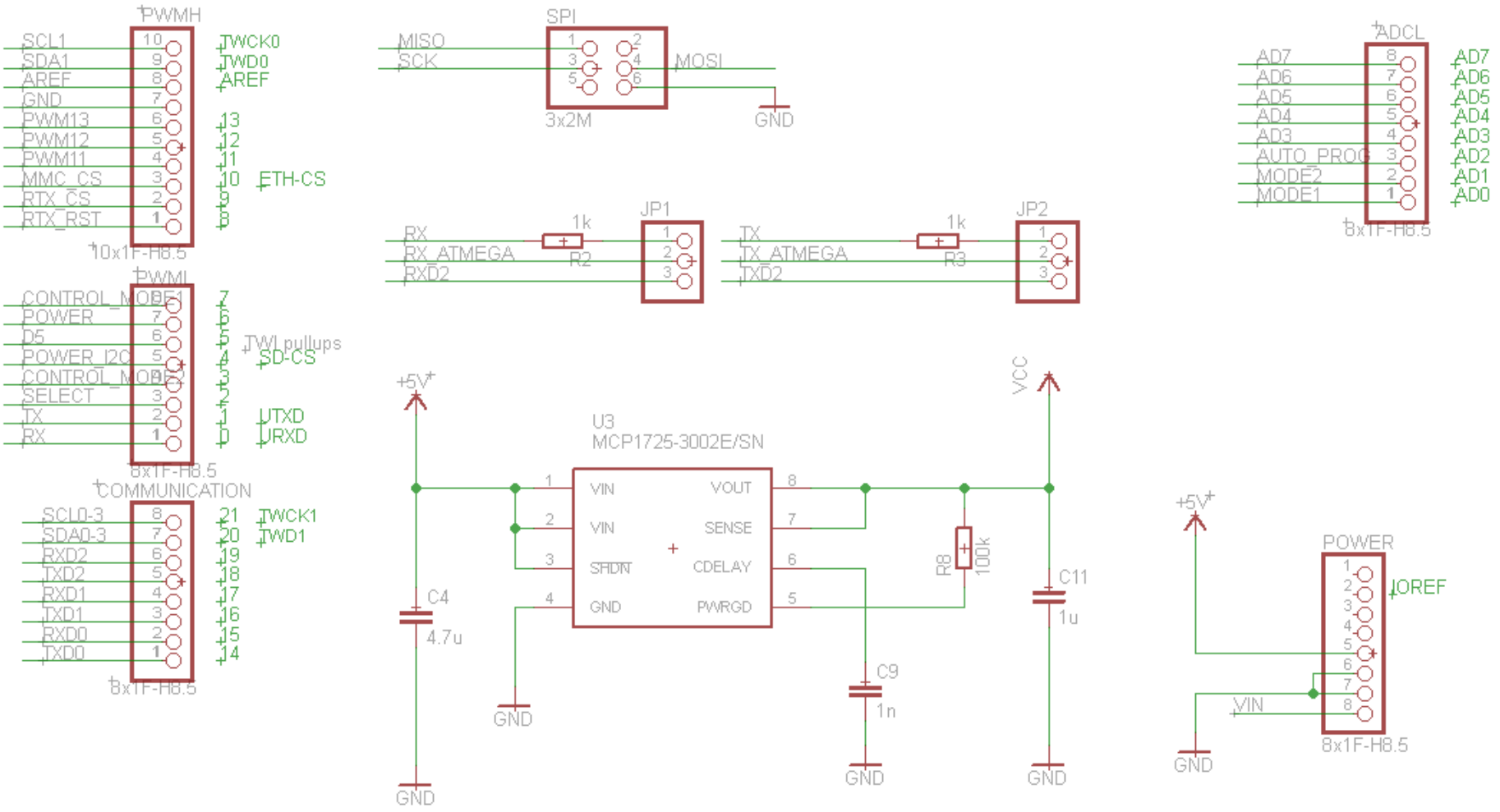
- **Diagrama hardware DVK**-

Proporcionat per Smart Citizen

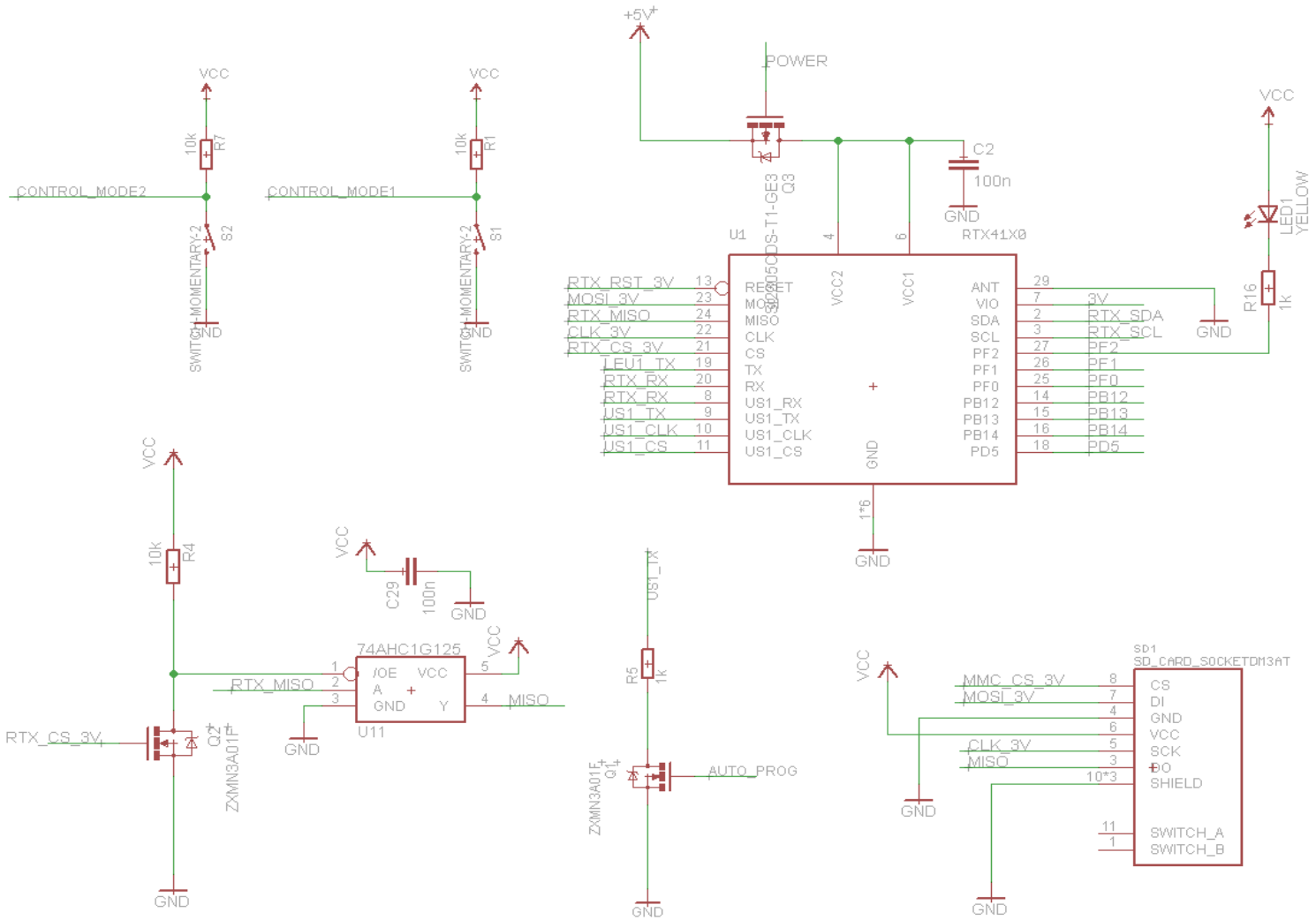
Referencia DVK

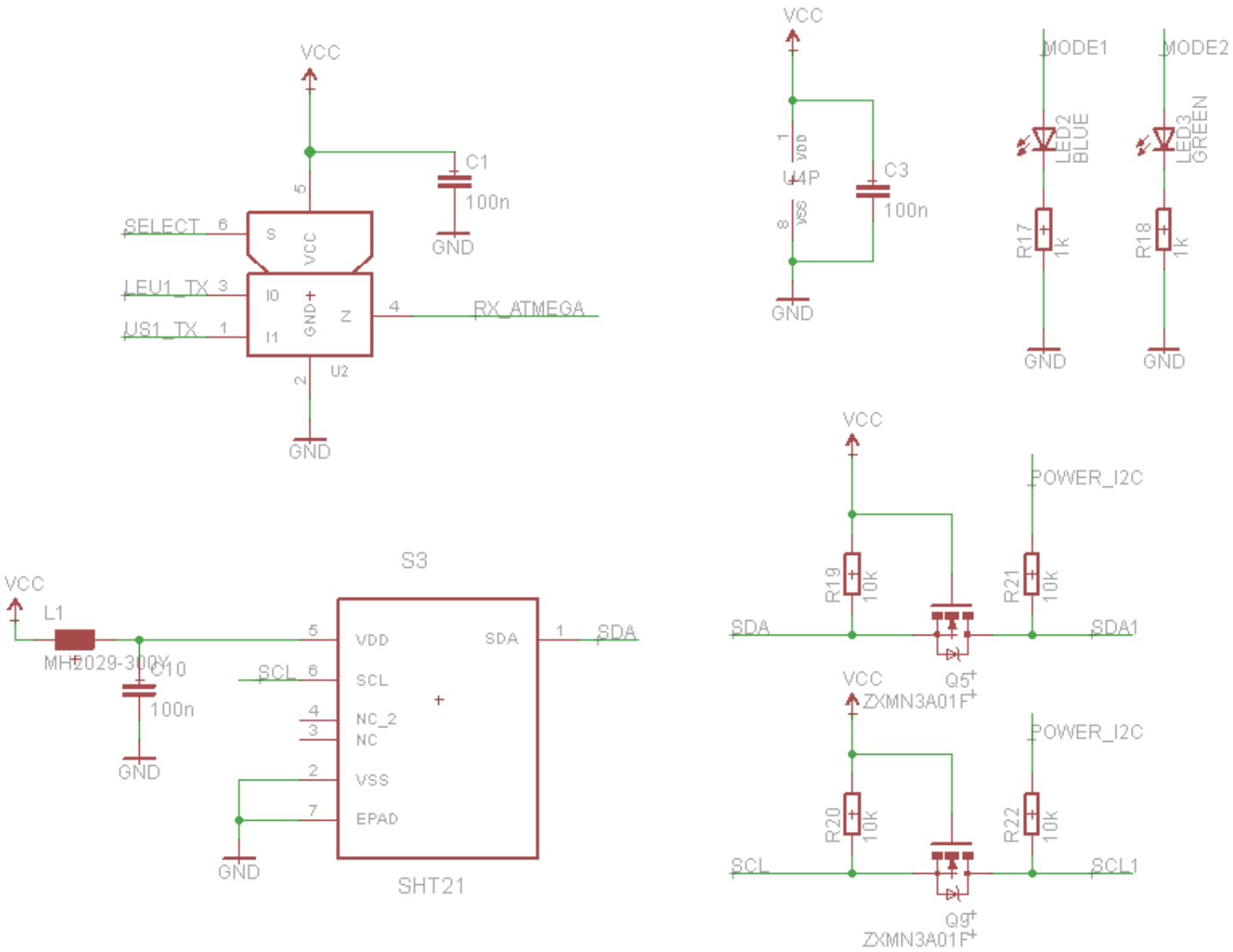
Manufacturer	Manufacturer Part	Name	Designator	Qty	Description
VISHAY	**VJ0805Y104MXXAC**	Capacitors	C1, C2, C3, C10, C29	5	100n
VISHAY	**VJ0805V475MXQTW1BC**		C4	1	4.7u
VISHAY	**VJ0805Y102JXQPW1BC**		C9	1	1n
VISHAY	**VJ0805Y105KXQTW1BC**		C11	1	1u
VISHAY	**PHT0805Y1002BGT**	Resistors	R1, R4, R7, R19, R20, R21, R22	7	10k
VISHAY	**PHT0805Y1001BGT**		R2, R3, R5, R16, R17, R18	6	1k
YAGEO	**RT0805FRE07100KL**		R8	1	100k
Bourns	**MH2029-300Y**	Magnetic Bead	L1	1	WE-CBF_0805
RTX	**RTX4100**	IC	U1	1	RTX4100
FAIRCHILD SEMICONDUCTOR	**NC7SZ157P6X**		U2	1	NC7SV157
MICROCHIP	**MCP1725-3002E/SN**		U3	1	MCP1725
TEXAS INSTRUMENTS	**CD74HC4050M**		U4	1	4050D
TEXAS INSTRUMENTS	**SN74AHC1G125DBVR**		U11	1	74AHC1G125
Hirose Electric Co Ltd	**DM3AT-SF-PEJM5(11)**	Memorizer	SD1	1	PC card,right angle
DIODES INC.	**ZXMN3A01F**	Transistors	Q1, Q2, Q5, Q9	4	MOSFET-NCHANNEL
VISHAY	**SI2305CDS-T1-GE3**		Q3	1	MOSFET-PCHANNEL
TE Connectivity	**4-1437565-1**	Sensors	S1, S2	2	SWITCH-MOMENTARY-2
SENSIRION	**SHT21**		S3	1	Temp/Humidity Sensor
DIALIGHT	**598-8150-107F**	LED	LED1	1	YELLOW
DIALIGHT	**598-8191-107F**		LED2	1	BLUE
DIALIGHT	**598-8160-107F**		LED3	1	GREEN
HARWIN	**M20-7830346**	Pin Header	SPI	1	PCB Socket2.54（2*3）180°
FCI	**69190-410HLF**		PWMH	1	Headers & Wire Housings 10P
FCI	**69190-403**		JP1, JP2	2	Headers & Wire Housings 3P
FCI	**68000-408HLF**		ADCL, COMMUNICATION, POWER, PWML	4	Headers & Wire Housings 8P

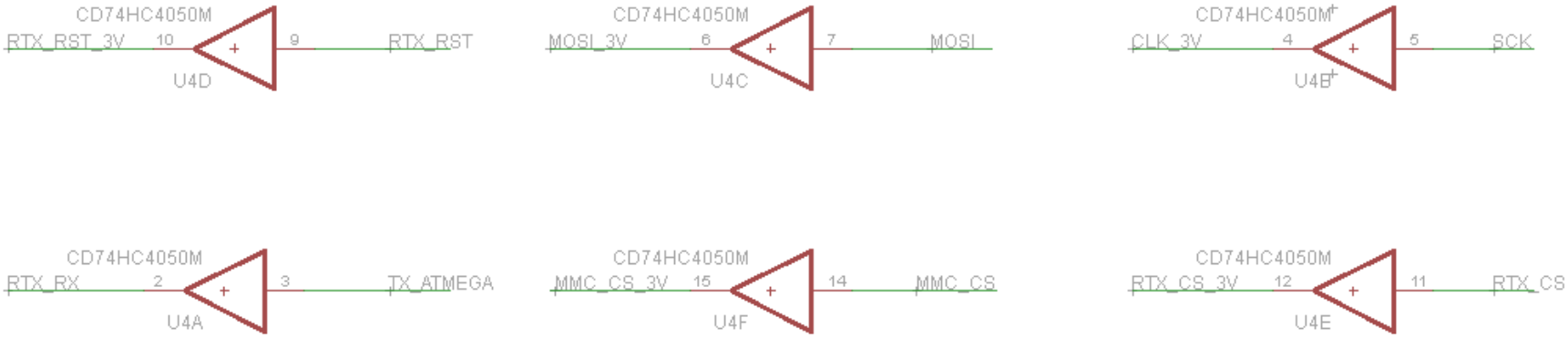
Electronica Arduino DUE



Electronica DVK




VCC
C1
100n
GND
SELECT
6
S
VCC
5
LEU1_TX
3
I0
GND
Z
4
RX_ATMEGA
US1_TX
1
I1
U2
2
GND
VCC
1
VDD
C3
100n
4P
8
VSS
GND
MODE1
MODE2
LED2
BLUE
LED3
GREEN
R17
1k
R18
1k
GND
GND
S3
VCC
L1
MH2029-300Y
C10
100n
GND
5
VDD
SDA
1
SDA
SCL
6
SCL
4
NC_2
3
NC
2
VSS
7
EPAD
GND
SHT21
VCC
POWER_I2C
R19
10k
R21
10k
SDA
SDA1
Q5
ZXMN3A01F
VCC
POWER_I2C
R20
10k
R22
10k
SCL
SCL1
Q6
ZXMN3A01F


CD74HC4050M
RTX_RST_3V
10
9
RTX_RST
U4D
CD74HC4050M
MOSI_3V
6
7
MOSI
U4C
CD74HC4050M
CLK_3V
4
5
SCK
U4B
CD74HC4050M
RTX_RX
2
3
TX_ATMEGA
U4A
CD74HC4050M
MMC_CS_3V
15
14
MMC_CS
U4F
CD74HC4050M
RTX_CS_3V
12
11
RTX_CS
U4E

Annex

- **Documentació de codi**-

Code for RTX4100 WiFi Module

Generated by Doxygen 1.8.8

Thu Jun 11 2015 12:48:02

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 AppDataWebType Struct Reference

Struct for store info (app:AppWeb)

`#include <rtx_AppWeb.h>`

Data Fields

- rsuint8 * **RX_Ptr**
- rsuint8 * TX_Ptr
- rsuint16 RX_Length
- rsuint16 TX_Length
- rsuint8 data [256]
- type_method method
- rsuint8 * ptr_url
- rsuint8 * ptr_cmd

3.1.1 Detailed Description

Struct for store info (app:AppWeb)

Definition at line 90 of file rtx_AppWeb.h.

3.1.2 Field Documentation

3.1.2.1 rsuint8 data[256]

Length for TX data

Definition at line 95 of file rtx_AppWeb.h.

3.1.2.2 type_method method

Buffer for temp data

Definition at line 96 of file rtx_AppWeb.h.

3.1.2.3 rsuint8* ptr_cmd

Pointer to url

Definition at line 98 of file rtx_AppWeb.h.

3.1.2.4 rsuint8* ptr_url

Method of query

Definition at line 97 of file rtx_AppWeb.h.

3.1.2.5 rsuint16 RX_Length

Pointer to TX data

Definition at line 93 of file rtx_AppWeb.h.

3.1.2.6 rsuint16 TX_Length

Length for RX data

Definition at line 94 of file rtx_AppWeb.h.

3.1.2.7 rsuint8* TX_Ptr

Pointer to RX data

Definition at line 92 of file rtx_AppWeb.h.

The documentation for this struct was generated from the following file:

- rtx_AppWeb.h

3.2 DnsType Struct Reference

Struct for store info (app:AppDns)

`#include <rtx_AppDns.h>`

Data Fields

- rsuint16 **id**
- rsuint16 flags
- rsuint16 qdcount
- rsuint16 ancount
- rsuint16 nscount
- rsuint16 arcount
- rsuint8 bodysection [200]

3.2.1 Detailed Description

Struct for store info (app:AppDns)

Definition at line 70 of file rtx_AppDns.h.

3.2.2 Field Documentation

3.2.2.1 rsuint16 ancount

question count

Definition at line 75 of file rtx_AppDns.h.

3.2.2.2 rsuint16 arcount

authority record count

Definition at line 77 of file rtx_AppDns.h.

3.2.2.3 rsuint8 bodysection[200]

additional record count

Definition at line 78 of file rtx_AppDns.h.

3.2.2.4 rsuint16 flags

identifier

Definition at line 73 of file rtx_AppDns.h.

3.2.2.5 rsuint16 nscount

answer record count

Definition at line 76 of file rtx_AppDns.h.

3.2.2.6 rsuint16 qdcount

flags

Definition at line 74 of file rtx_AppDns.h.

The documentation for this struct was generated from the following file:

- rtx_AppDns.h

Chapter 4

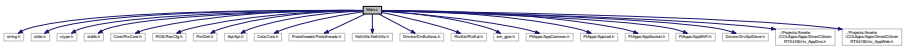
File Documentation

4.1 Main.c File Reference

Firmware for RTX4100 WiFi Module.

```
#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <Core/RtxCore.h>
#include <ROS/RosCfg.h>
#include <PortDef.h>
#include <Api/Api.h>
#include <Cola/Cola.h>
#include <Protothreads/Protothreads.h>
#include <NetUtils/NetUtils.h>
#include <Drivers/DrvButtons.h>
#include <RtxEai/RtxEai.h>
#include <em_gpio.h>
#include <PtApps/AppCommon.h>
#include <PtApps/AppLed.h>
#include <PtApps/AppSocket.h>
#include <PtApps/AppWiFi.h>
#include <Drivers/DrvSpiSlave.h>
#include <../Projects/Amelie/COLApps/Apps/SmartCitizenRTX4100/rtx_AppDns.h>
#include <../Projects/Amelie/COLApps/Apps/SmartCitizenRTX4100/rtx_AppWeb.h>
```

Include dependency graph for Main.c:



Macros

- #define **SPI_COMMUNICATION**
- #define **printf**(...) RtxEaiPrintf(ColaIf->ColaTaskId,__VA_ARGS__)
- #define **MAX_TX_POWER** 18
- #define **PRINT**(x) UartPrint((rsuint8*)x)
- #define **PRINTLN**(x) UartPrintLn((rsuint8*)x)
- #define **TMP_STR_LENGTH** 100
- #define **CMD_STR_LENGTH** TMP_STR_LENGTH

- #define **TX_BUFFER_LENGTH** 500
- #define **MAX_ARGV** 3
- #define **APP_DNS_RESOLVE_RSP_TIMEOUT** (10∗RS_T1SEC)
- #define **SPI_baudrate** 8000000 /∗∗ Baudrate for SPI ∗/
- #define **SOFT_AP_ESSID** "SCK_AP_" /∗∗ Partial essid ∗/
- #define **SOFT_AP_SECURITY_TYPE** AWST_NONE /∗∗ Default security for soft AP ∗/
- #define **SOFT_AP_COUNTRY_CODE** "ES" /∗∗ Default country for soft AP ∗/
- #define **SOFT_AP_CHANNEL** 2437 /∗∗ Default channel (6) for soft AP ∗/
- #define **SOFT_AP_INACT** 10 /∗∗ Time in min for set client as inactive ∗/
- #define **SOFT_AP_BEACON** 100 /∗∗ Time in ms, for beacon ∗/
- #define **STATIC_IP** "192.168.1.99" /∗∗ Default IP for soft AP ∗/
- #define **SUBNET_MASK** "255.255.255.0" /∗∗ Default MASK for soft AP ∗/

Functions

- int extract_substring (rsuint8 ∗dest, rsuint8 ∗orig, int orig_ptr)

  Helper function to extract a substring from a string.
- void Wifi_save_appInfo_to_NVS ()

  Saves the application info object contents to NVS.
- void Wifi_read_appInfo_from_NVS ()

  Retrieves the application info object contents to NVS.
- void get_ap_info_from_str (rsuint8 ∗ap_data, ApInfoType ∗ap_info)

  Fulfills an ApInfoType object from a string.
- static PT_THREAD (PtWifi_setup_AP(struct pt ∗Pt, const RosMailType ∗Mail, rsuint8 ∗ap_data))

  Setups an AP.
- static char PtWifi_suspend (struct pt ∗Pt, const RosMailType ∗Mail)

  Suspends the WiFi chip.
- static char PtWifi_resume (struct pt ∗Pt, const RosMailType ∗Mail)

  Resumes the suspended WiFi chip.
- void Wifi_set_power_save_profile (rsuint8 profile)

  Sets the powersave profile.
- static char PtWifi_connect (struct pt ∗Pt, const RosMailType ∗Mail)

  Associates and connects to an already configured AP.
- rsuint8 Wifi_is_connected ()

  Checks if associated and connected to the AP.
- void Wifi_TCP_close ()

  Sends a query to close the currently open TCP connection.
- void Wifi_TCP_send (rsuint16 len)

  Sends len bytes in tx_buffer using the TCP connection.
- char Wifi_TCP_receive ()

  Receive data in the RX buffer. Must be called by the user when it polls the status and sees that TCP_received is activated.
- rsuint8 Wifi_get_status ()

  Obtains the system status.
- void Wifi_set_tx_power (rsuint8 power)

  Sets the transmit wireless transmit power.
- static char PtWifi_power_on_off (struct pt ∗Pt, const RosMailType ∗Mail, char on)

  Powers on/off the WiFi chip.
- static char PtWifi_disconnect (struct pt ∗Pt, const RosMailType ∗Mail)

  Disassociates and disconnects from the WiFi AP.
- static char PtWifi_IP_config (struct pt ∗Pt, const RosMailType ∗Mail, rsuint8 ∗config)

Configures the IP connection (DHCP, static)

- static char PtWifi_DNS_resolve (struct pt ∗Pt, const RosMailType ∗Mail, rsuint8 ∗name, rsuint32 ∗o_↩ response)

  Resolves a domain name using the DNS service.
- static char PtWifi_TCP_on_connect (struct pt ∗Pt, const RosMailType ∗Mail)

  Event handled. Fired when the TCP connection has been stablished.
- static char PtWifi_TCP_start (struct pt ∗Pt, const RosMailType ∗Mail, ApiSocketAddrType addr)

  Starts a new TCP connection to the given server.
- static char PtWifi_softAP (struct pt ∗Pt, const RosMailType ∗Mail)

  Sets softAP mode.
- static char PtTest (struct pt ∗Pt, const RosMailType ∗Mail)

  Test procedure which can be called from the debug terminal.
- static char force_wifimode (struct pt ∗Pt, const RosMailType ∗Mail)

  Test WiFi mode (use with operation_mode=4 in arduino)
- static char PtMain (struct pt ∗Pt, const RosMailType ∗Mail)

  Main protothread. It controls the SPI or the debug terminal.
- void ColaTask (const RosMailType ∗Mail)

  Main CoLa task event handler.

Variables

- static AppDataType **app_data**
- static const RosTimerConfigType **PacketDelayTimer**
- static const RosTimerConfigType **DnsRspTimer**
- static char **TCP_is_connected**
- static char **TCP_received**
- static int **socketHandle**
- static rsuint8 ∗ **TCP_receive_buffer_ptr**
- static int **TCP_Rx_bufferLength**
- static rsuint8 **is_suspended**
- static rsbool **webmode** = false
- static rsbool **web_idle** = true
- static PtEntryType ∗ **Pt_AppDnsd**
- static PtEntryType ∗ **Pt_AppWebd**
- static RsListEntryType **PtList**
- rsuint8 **tx_buffer** [TX_BUFFER_LENGTH]
- rsuint8 **rx_buffer** [TX_BUFFER_LENGTH]

4.1.1 Detailed Description

Firmware for RTX4100 WiFi Module.

Date

2014, 2015

Definition in file Main.c.

4.1.2 Function Documentation

4.1.2.1 void ColaTask (const RosMailType ∗ *Mail*)

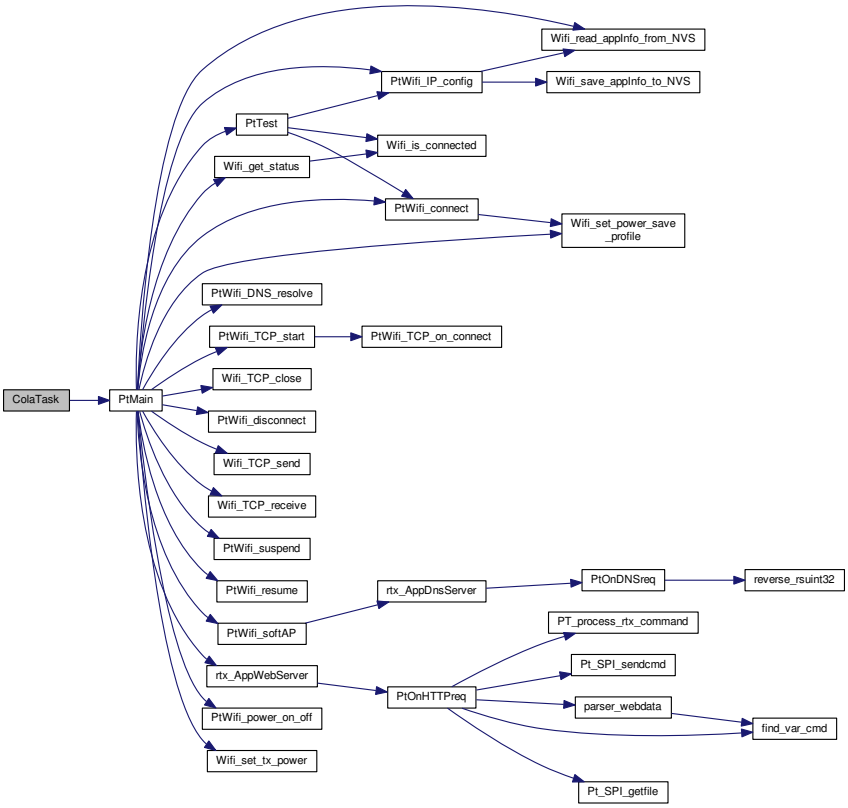
Main CoLa task event handler.

Parameters

Mail	: protothread mail

Definition at line 1510 of file Main.c.

Here is the call graph for this function:



4.1.2.2 int extract_substring (rsuint8 * *dest*, rsuint8 * *orig*, int *orig_ptr*)

Helper function to extract a substring from a string.

Parameters

dest	: extracted substring pointer
orig	: input string pointer
orig_ptr	: offset at the input string to extract next string

Definition at line 335 of file Main.c.

4.1.2.3 static char force_wifimode (struct pt * *Pt*, const RosMailType * *Mail*) [static]

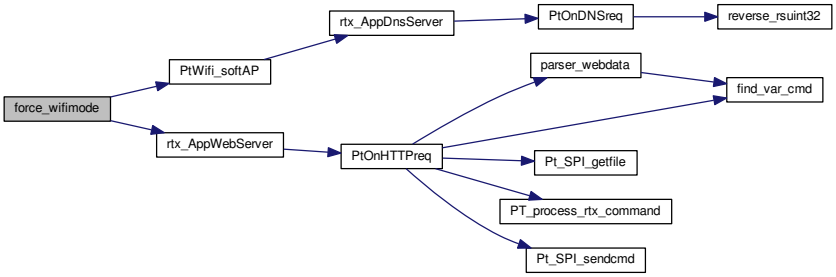
Test WiFi mode (use with operation_mode=4 in arduino)

Parameters

Pt	: current protothread pointer
Mail	: protothread mail

Definition at line 1123 of file Main.c.

Here is the call graph for this function:



4.1.2.4 void get_ap_info_from_str (rsuint8 * *ap_data,* ApInfoType * *ap_info*)

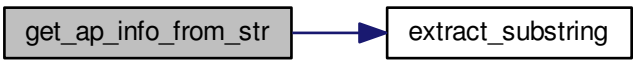
Fulfills an ApInfoType object from a string.

Parameters

ap_data	: input string
ap_info	: AP info object to fulfill

Definition at line 371 of file Main.c.

Here is the call graph for this function:



4.1.2.5 static PT_THREAD (PtWifi_setup_AP(struct pt *Pt, const RosMailType *Mail, rsuint8 *ap_data)) [static]

Setups an AP.

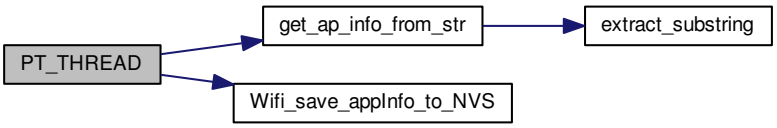
Parameters

Pt	: current protothread pointer
Mail	: protothread mail

ap_data	: input configuration string. If NULL, it will use the data stored at the NVS to configure the AP

Definition at line 465 of file Main.c.

Here is the call graph for this function:



4.1.2.6 static char PtMain (struct pt * *Pt*, const RosMailType * *Mail*) [static]

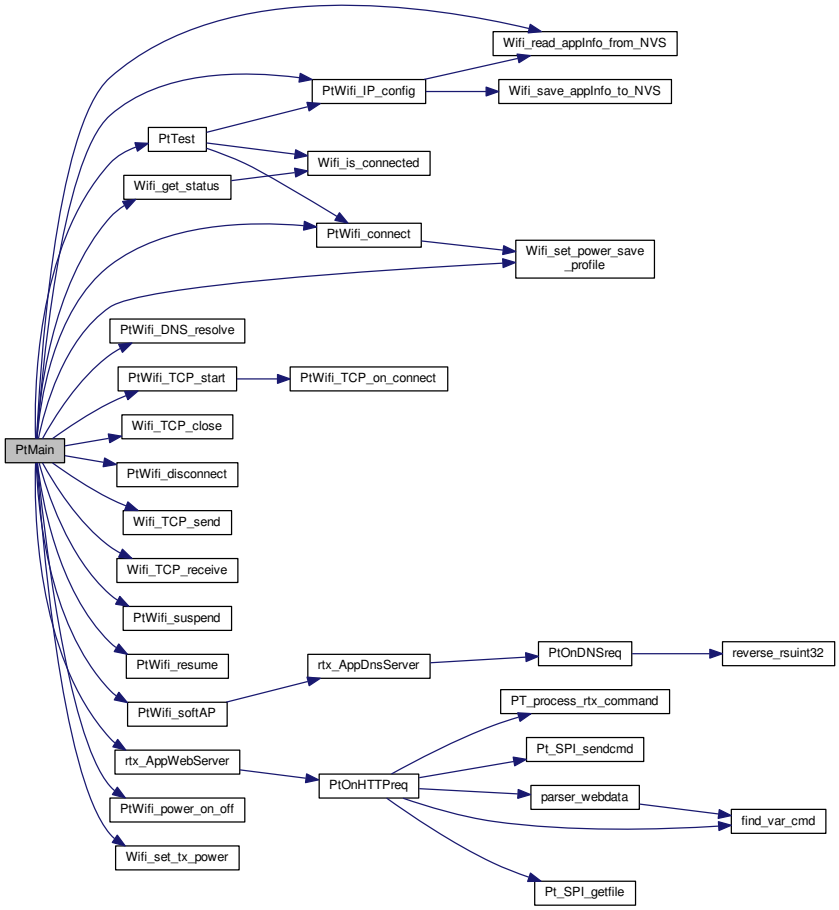
Main protothread. It controls the SPI or the debug terminal.

Parameters

Pt	: current protothread pointer
Mail	: protothread mail

Definition at line 1142 of file Main.c.

Here is the call graph for this function:



4.1.2.7 static char PtTest (struct pt * *Pt*, const RosMailType * *Mail*) [static]

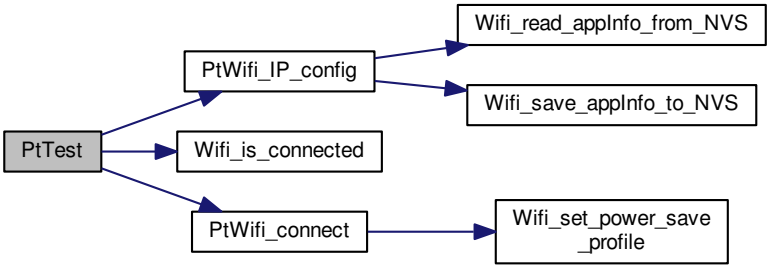
Test procedure which can be called from the debug terminal.

Parameters

Pt	: current protothread pointer
Mail	: protothread mail

Definition at line 1031 of file Main.c.

Here is the call graph for this function:



4.1.2.8 static char PtWifi_connect (struct pt * *Pt*, const RosMailType * *Mail*) [static]

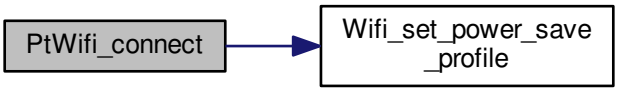
Associates and connects to an already configured AP.

Parameters

Pt	: current protothread pointer
Mail	: protothread mail

Definition at line 575 of file Main.c.

Here is the call graph for this function:



4.1.2.9 static char PtWifi_disconnect (struct pt * *Pt*, const RosMailType * *Mail*) [static]

Disassociates and disconnects from the WiFi AP.

Parameters

Pt	: current protothread pointer
Mail	: protothread mail

Definition at line 776 of file Main.c.

4.1.2.10 static char PtWifi_DNS_resolve (struct pt * *Pt*, const RosMailType * *Mail*, rsuint8 * *name*, rsuint32 * *o_response*) [static]

Resolves a domain name using the DNS service.

Parameters

Pt	: current protothread pointer
Mail	: protothread mail
name	: domain name to resolve
o_response	: resolved IP address

Definition at line 881 of file Main.c.

4.1.2.11 static char PtWifi_IP_config (struct pt * *Pt*, const RosMailType * *Mail*, rsuint8 * *config*) [static]

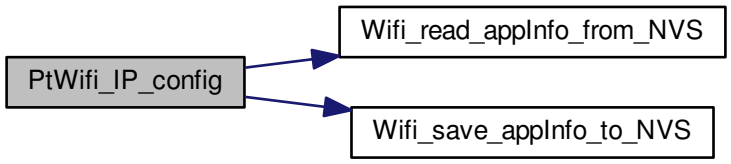
Configures the IP connection (DHCP, static)

Parameters

Pt	: current protothread pointer
Mail	: protothread mail
config	: configuration string. NULL to read configuration from the NVS

Definition at line 791 of file Main.c.

Here is the call graph for this function:



4.1.2.12 static char PtWifi_power_on_off (struct pt * *Pt*, const RosMailType * *Mail*, char *on*) [static]

Powers on/off the WiFi chip.

Parameters

Pt	: current protothread pointer
Mail	: protothread mail
on	: true for poweron, false for poweroff

Definition at line 754 of file Main.c.

4.1.2.13 static char PtWifi_resume (struct pt * *Pt*, const RosMailType * *Mail*) [static]

Resumes the suspended WiFi chip.

Parameters

Pt	: current protothread pointer

Mail	: protothread mail

Definition at line 540 of file Main.c.

4.1.2.14 static char PtWifi_softAP (struct pt * *Pt*, const RosMailType * *Mail*) [static]

Sets softAP mode.

Parameters

Pt	: current protothread pointer
Mail	: protothread mail

Definition at line 988 of file Main.c.

Here is the call graph for this function:

4.1.2.15 static char PtWifi_suspend (struct pt * *Pt*, const RosMailType * *Mail*) [static]

Suspends the WiFi chip.

Parameters

Pt	: current protothread pointer
Mail	: protothread mail

Definition at line 522 of file Main.c.

4.1.2.16 static char PtWifi_TCP_on_connect (struct pt * *Pt*, const RosMailType * *Mail*) [static]

Event handled. Fired when the TCP connection has been stablished.

Parameters

Pt	: current protothread pointer
Mail	: protothread mail

Definition at line 935 of file Main.c.

4.1.2.17 static char PtWifi_TCP_start (struct pt * *Pt*, const RosMailType * *Mail*, ApiSocketAddrType *addr*) [static]

Starts a new TCP connection to the given server.

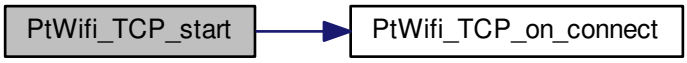
Parameters

Pt	: current protothread pointer
Mail	: protothread mail

addr	: server IP address and TCP port

Definition at line 959 of file Main.c.

Here is the call graph for this function:



4.1.2.18 rsuint8 Wifi_get_status ( )

Obtains the system status.

Returns

bit wise system status. Bit 0: Wifi connected, bit 1: TCP connected, bit 2: TCP data received, bit 3: WiFi suspended, bit 4: Soft AP enabled, bit 5: Server Web enabled

Definition at line 727 of file Main.c.

Here is the call graph for this function:

4.1.2.19 rsuint8 Wifi_is_connected ( )

Checks if associated and connected to the AP.

Returns

True if associated and connected to the AP. False otherwise

Definition at line 656 of file Main.c.

4.1.2.20 void Wifi_set_power_save_profile (rsuint8 *profile*)

Sets the powersave profile.

Parameters

profile	: powersave profile, 0: low power, 1: medium power, 2: high power, 3: max power

Definition at line 556 of file Main.c.

4.1.2.21 void Wifi_set_tx_power (rsuint8 *power*)

Sets the transmit wireless transmit power.

Parameters

power	: wireless transmit power

Definition at line 742 of file Main.c.

4.1.2.22 char Wifi_TCP_receive ( )

Receive data in the RX buffer. Must be called by the user when it polls the status and sees that TCP_received is activated.

Parameters

len	: number of bytes to send

Definition at line 689 of file Main.c.

4.1.2.23 void Wifi_TCP_send (rsuint16 *len*)

Sends len bytes in tx_buffer using the TCP connection.

Parameters

len	: number of bytes to send

Definition at line 674 of file Main.c.

4.1.3 Variable Documentation

4.1.3.1 const RosTimerConfigType DnsRspTimer [static]

Initial value:

```
=
  ROSTIMER(COLA_TASK, APP_DNS_RSP_TIMEOUT,
  APP_DNS_RSP_TIMER)
```

Definition at line 133 of file Main.c.

4.1.3.2 const RosTimerConfigType PacketDelayTimer [static]

Initial value:

```
=
  ROSTIMER(COLA_TASK, APP_PACKET_DELAY_TIMEOUT,
  APP_PACKET_DELAY_TIMER)
```

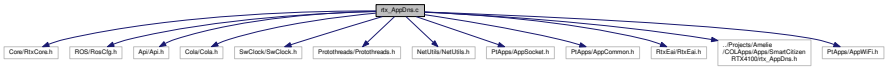
Definition at line 129 of file Main.c.

4.2 rtx_AppDns.c File Reference

Very simple DNS Server.

```
#include <Core/RtxCore.h>
#include <ROS/RosCfg.h>
#include <Api/Api.h>
#include <Cola/Cola.h>
#include <SwClock/SwClock.h>
#include <Protothreads/Protothreads.h>
#include <NetUtils/NetUtils.h>
#include <PtApps/AppSocket.h>
#include <PtApps/AppCommon.h>
#include <RtxEai/RtxEai.h>
#include <../Projects/Amelie/COLApps/Apps/SmartCitizenRTX4100/rtx_AppDns.h>
#include <PtApps/AppWiFi.h>
```

Include dependency graph for rtx_AppDns.c:



Macros

- #define **DNS_SERVER_PORT** 53
- #define **printf**(...) RtxEaiPrintf(ColaIf->ColaTaskId,__VA_ARGS__)

Functions

- static void reverse_rsuint32 (rsuint32 *data)
  Helper function to reverse data order.
- static char PtOnDNSreq (struct pt *Pt, const RosMailType *Mail)
  Protothread used to handle the data received.
- PtEntryType * rtx_AppDnsServer (RsListEntryType *PtList)
  Init function for DNS Server.

4.2.1 Detailed Description

Very simple DNS Server.

Author

Jordi Casas

Date

4/2015

Definition in file rtx_AppDns.c.

4.2.2 Function Documentation

4.2.2.1 static char PtOnDNSreq (struct pt * *Pt*, const RosMailType * *Mail*) [static]

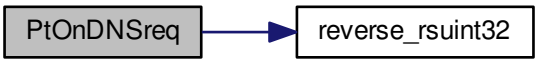
Protothread used to handle the data received.

Parameters

Pt	: pointer to protothread (for yield it)
Mail	: pointer to mails

Definition at line 114 of file rtx_AppDns.c.

Here is the call graph for this function:



4.2.2.2 static void reverse_rsuint32 (rsuint32 * *data*) [static]

Helper function to reverse data order.

Parameters

data	: pointer to data

Definition at line 92 of file rtx_AppDns.c.

4.2.2.3 PtEntryType* rtx_AppDnsServer (RsListEntryType * *PtList*)

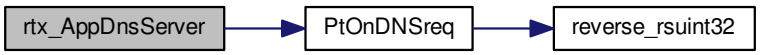
Init function for DNS Server.

Parameters

PtList	: pointer to Protothread control

Definition at line 204 of file rtx_AppDns.c.

Here is the call graph for this function:



4.3 rtx_AppDns.h File Reference

Very simple DNS Server (header)

Data Structures

- struct DnsType

Struct for store info (app:AppDns)

Functions

- PtEntryType * rtx_AppDnsServer (RsListEntryType *PtList)
  Init function for DNS Server.

Variables

- static const rschar **dns_h_flags** [] = {0x81,0x80}
- static const rschar dns_h_ancount [] = {0x00,0x01}
- static const rschar dns_h_nscount [] = {0x00,0x00}
- static const rschar dns_h_arcount [] = {0x00,0x00}
- static const rschar dns_a_name [] = {0xc0,0x0c}
- static const rschar dns_a_type [] = {0x00,0x01}
- static const rschar dns_a_class [] = {0x00,0x01}
- static const rschar dns_a_ttl [] = {0x00,0x00,0x03,0x84}
- static const rschar dns_a_sizeaddr [] = {0x00,0x04}

4.3.1 Detailed Description

Very simple DNS Server (header)

Author

Jordi Casas

Date

4/2015

Definition in file rtx_AppDns.h.

4.3.2 Function Documentation

4.3.2.1 PtEntryType* rtx_AppDnsServer (RsListEntryType * *PtList*)

Init function for DNS Server.

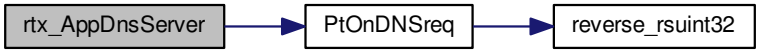
SizeAddr = 4 (ipv4 -> 32b -> 4B)

Parameters

PtList	: pointer to Protothread control

Definition at line 204 of file rtx_AppDns.c.

Here is the call graph for this function:



4.3.3 Variable Documentation

4.3.3.1 const rschar dns_a_class[ ] = {0x00,0x01} [static]

Type = A

Definition at line 94 of file rtx_AppDns.h.

4.3.3.2 const rschar dns_a_name[ ] = {0xc0,0x0c} [static]

Force to 0 additional record

Definition at line 92 of file rtx_AppDns.h.

4.3.3.3 const rschar dns_a_sizeaddr[ ] = {0x00,0x04} [static]

TTL (900s=15min)

Definition at line 96 of file rtx_AppDns.h.

4.3.3.4 const rschar dns_a_ttl[ ] = {0x00,0x00,0x03,0x84} [static]

Class = IN

Definition at line 95 of file rtx_AppDns.h.

4.3.3.5 const rschar dns_a_type[ ] = {0x00,0x01} [static]

Name

Definition at line 93 of file rtx_AppDns.h.

4.3.3.6 const rschar dns_h_ancount[ ] = {0x00,0x01} [static]

Flags activate: Response & Recursive

Definition at line 86 of file rtx_AppDns.h.

4.3.3.7 const rschar dns_h_arcount[ ] = {0x00,0x00} [static]

Force to 0 authority record

Definition at line 88 of file rtx_AppDns.h.

4.3.3.8 const rschar dns_h_nscount[ ] = {0x00,0x00} [static]

Force to 1 answer

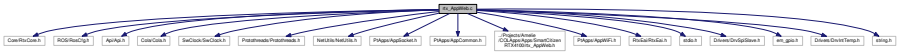
Definition at line 87 of file rtx_AppDns.h.

4.4 rtx_AppWeb.c File Reference

Simple Web Server.

```
#include <Core/RtxCore.h>
#include <ROS/RosCfg.h>
#include <Api/Api.h>
#include <Cola/Cola.h>
#include <SwClock/SwClock.h>
#include <Protothreads/Protothreads.h>
#include <NetUtils/NetUtils.h>
#include <PtApps/AppSocket.h>
#include <PtApps/AppCommon.h>
#include <../Projects/Amelie/COLApps/Apps/SmartCitizenRTX4100/rtx_AppWeb.h>
#include <PtApps/AppWiFi.h>
#include <RtxEai/RtxEai.h>
#include <stdio.h>
#include <Drivers/DrvSpiSlave.h>
#include <em_gpio.h>
#include <Drivers/DrvIntTemp.h>
#include <string.h>
```

Include dependency graph for rtx_AppWeb.c:



Macros

- #define **printf**(...) RtxEaiPrintf(ColaIf->ColaTaskId,__VA_ARGS__)
- #define **printc**(...) RtxEaiLogComment(ColaIf->ColaTaskId,__VA_ARGS__)

Functions

- void find_var_cmd ()

  Set ptr_cmd to ptr of cmd (if exist)
- void parser_webdata (rsuint8 *ptr_data)

  Get method & url from petition.
- static char Pt_SPI_getfile (struct pt *Pt, const RosMailType *Mail, AppSocketDataType *pInst, rsuint8 *file)

  Protothread used to download file from arduino and send it to client.
- static char Pt_SPI_sendcmd (struct pt *Pt, const RosMailType *Mail, AppSocketDataType *pInst, rsuint8 *cmd)

  Protothread used to send command to arduino, get result and send it to client.
- static char PT_process_rtx_command (struct pt *Pt, const RosMailType *Mail, AppSocketDataType *pInst, rschar *command)

  Protothread used to execute the command and send result to client.
- static char PtOnHTTPreq (struct pt *Pt, const RosMailType *Mail)

  Protothread used to handle the data received.
- PtEntryType * rtx_AppWebServer (RsListEntryType *PtList)

  Init function for Web Server.

Variables

- static const RosTimerConfigType **PacketDelayTimer**
- static AppDataWebType **AppData**

4.4.1 Detailed Description

Simple Web Server.

Author

Jordi Casas

Date

4/2015, 5/2015, 6/2015

Definition in file rtx_AppWeb.c.

4.4.2 Function Documentation

4.4.2.1 void parser_webdata (rsuint8 * *ptr_data*)

Get method & url from petition.

Parameters

ptr_data	Pointer to data (web request)

Definition at line 127 of file rtx_AppWeb.c.

Here is the call graph for this function:

4.4.2.2 static char PT_process_rtx_command (struct pt * *Pt*, const RosMailType * *Mail*, AppSocketDataType * *pInst*, rschar * *command*) [static]

Protothread used to execute the command and send result to client.

Parameters

Pt	: pointer to protothread (for yield it)
Mail	: pointer to mails
pInst	Pointer to instance (socket connection)
command	Pointer to command

Definition at line 436 of file rtx_AppWeb.c.

4.4.2.3 static char Pt_SPI_getfile (struct pt * *Pt*, const RosMailType * *Mail*, AppSocketDataType * *pInst*, rsuint8 * *file*) [static]

Protothread used to download file from arduino and send it to client.

Parameters

Pt	: pointer to protothread (for yield it)
Mail	: pointer to mails
pInst	Pointer to instance (socket connection)
url	Pointer to file

Definition at line 160 of file rtx_AppWeb.c.

4.4.2.4 static char Pt_SPI_sendcmd (struct pt * *Pt,* const RosMailType * *Mail,* AppSocketDataType * *pInst,* rsuint8 * *cmd*) [static]

Protothread used to send command to arduino, get result and send it to client.

Parameters

Pt	: pointer to protothread (for yield it)
Mail	: pointer to mails
pInst	Pointer to instance (socket connection)
cmd	Pointer to command

Definition at line 359 of file rtx_AppWeb.c.

4.4.2.5 static char PtOnHTTPreq (struct pt * *Pt,* const RosMailType * *Mail*) [static]

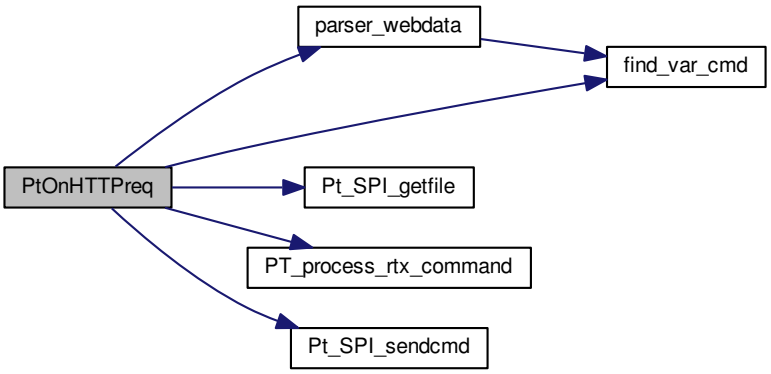
Protothread used to handle the data received.

Parameters

Pt	: pointer to protothread (for yield it)
Mail	: pointer to mails

Definition at line 662 of file rtx_AppWeb.c.

Here is the call graph for this function:



4.4.2.6 PtEntryType* rtx_AppWebServer (RsListEntryType * *PtList*)

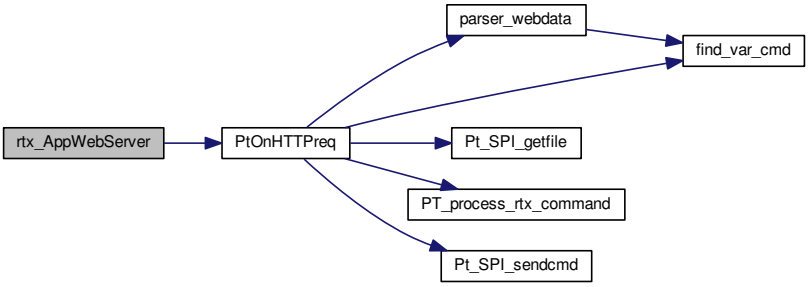
Init function for Web Server.

Parameters

PtList	: pointer to Protothread control

Definition at line 812 of file rtx_AppWeb.c.

Here is the call graph for this function:



4.4.3 Variable Documentation

4.4.3.1 const RosTimerConfigType PacketDelayTimer [static]

Initial value:

```
=
  ROSTIMER(COLA_TASK, APP_PACKET_DELAY_TIMEOUT,
  APP_PACKET_DELAY_TIMER)
```

Definition at line 84 of file rtx_AppWeb.c.

4.5 rtx_AppWeb.h File Reference

Simple Web Server (header)

Data Structures

- struct AppDataWebType

  Struct for store info (app:AppWeb)

Macros

- #define **HTTP_DEFAULT_WEBPAGE** "/WEB/INDEX.HTM"
- #define **HTTP_SERVER_PORT** 80
- #define **web_optioncmd** "cmd=" /∗∗ Option for send the command ∗/
- #define **web_optioncmd_RTX** "rtx_" /∗∗ Prefix for RTX function ∗/
- #define **web_sdmask** "/WEB/" /∗∗ Prefix for file in SD ∗/
- #define **web_cmd_separator** "%3A" /∗∗ Separator for parameters in cmd ∗/
- #define **maxBufferTX** 512 /∗∗ Max buffer for SPI∗/
- #define **SPI_baudrate** 8000000 /∗∗ Baudrate for SPI ∗/

Enumerations

- enum type_method { **GET**, POST, UNK }
  Enum for supported methods.

Functions

- PtEntryType * rtx_AppWebServer (RsListEntryType *PtList)
  Init function for Web Server.

Variables

- static const char web_msg_template []
- static const char web_msg_template_file []
- static const char web_msg_template_cmd []
- static const char web_error404 []
- static const char web_error501 []

4.5.1 Detailed Description

Simple Web Server (header)

Author

Jordi Casas

Date

4/2015, 5/2015, 6/2015

Definition in file rtx_AppWeb.h.

4.5.2 Enumeration Type Documentation

4.5.2.1 enum type_method

Enum for supported methods.

Enumerator

POST Get (vars in URI)

UNK Post (vars in body) Method not supported

Definition at line 81 of file rtx_AppWeb.h.

4.5.3 Function Documentation

4.5.3.1 PtEntryType* rtx_AppWebServer (RsListEntryType * *PtList*)

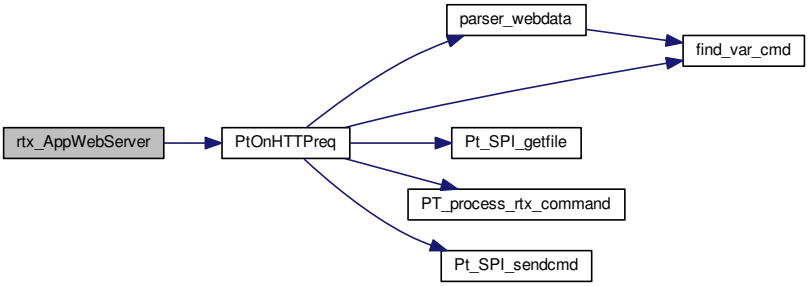
Init function for Web Server.

Parameters

PtList	: pointer to Protothread control

Definition at line 812 of file rtx_AppWeb.c.

Here is the call graph for this function:



4.5.4 Variable Documentation

4.5.4.1 const char web_error404[] [static]

Initial value:

```
= "<h1>FILE NOT FOUND.</h1><br>Redirect in 3s to mainpage"
"<meta http-equiv=\"Refresh\" content=\"3; url=/\">"
```

Template for file not exist

Definition at line 127 of file rtx_AppWeb.h.

4.5.4.2 const char web_error501[] [static]

Initial value:

```
= "<h1>METHOD NOT IMPLEMENTED.</h1><br>Redirect in 3s to mainpage"
"<meta http-equiv=\"Refresh\" content=\"3; url=/\">"
```

Template for method not implemented

Definition at line 131 of file rtx_AppWeb.h.

4.5.4.3 const char web_msg_template[] [static]

Initial value:

```
= "HTTP/1.1 %s\r\n"
"Connection: close\r\n\r\n"
"<!DOCTYPE html><html><head><title>%s</title></head>"
"<body>%s</body></html>\r\n"
```

Template for send generic request.

Definition at line 105 of file rtx_AppWeb.h.

4.5.4.4 const char web_msg_template_cmd[ ] [static]

Initial value:

```
= "HTTP/1.1 %s\r\n"
"Connection: close\r\n\r\n"
"<!DOCTYPE html><html><head><title>%s</title>"
"<meta name=\"viewport\" content=\"width=device-width,height=device-height,initial-scale=1.0\"/></head>"
"<body>Result command executed: <b>%s</b>"
"<br><br><a href=’/’>Return to main page</a>"
"<meta http-equiv=\"Refresh\" content=\"10; url=/\">"
"</body></html>\r\n"
```

Template for send result cmd. (Includes link&timeout redirects)

Definition at line 117 of file rtx_AppWeb.h.

4.5.4.5 const char web_msg_template_file[ ] [static]

Initial value:

```
= "HTTP/1.1 200\r\n"
"Connection: close\r\n"
"Cache-Control: public, max-age=3600\r\n"
"Content-Length: %i\r\n\r\n"
```

Template for send file. (Includes cache-control param in header)

Definition at line 111 of file rtx_AppWeb.h.

Index

Code for Arduino Due

Generated by Doxygen 1.8.8

Thu Jun 11 2015 12:48:48

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 SCDVK Class Reference

`#include <SCDVK.h>`

Public Member Functions

- void begin ()

  Initialize variables, pin modes, etc.
- void execute ()

  executed program (loop)
- void normal_mode ()

  Normal execution (without softap)
- void softap_mode ()

  Send API(SPI) to start SoftAP.
- void webserver_mode ()

  Send API(SPI) to start WebServer and prepare for it.
- void webserver_exit ()

  Send API(SPI) and exit webserver mode.
- float getTemperature ()

  Used for get temperature.
- float getHumidity ()

  Used for get humidity.
- void echo ()

  Send data btwn serials(1&2)
- void test_serverweb ()

  Force serverweb state (without start it)
- void main ()

  main for default mode
- void rtxserver ()

  Process API rcv by RTX (webserver mode)
- void exec_cmd (char *cmd, char *data)

  Send ALL data btwn serials(1&2)
- uint16_t readSHT21 (uint8_t type)

  Used for read data from SHT21.

3.1.1 Detailed Description

Definition at line 8 of file SCDVK.h.

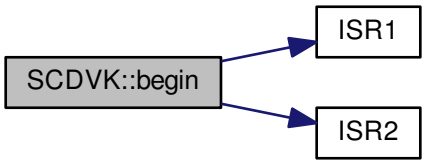
3.1.2 Member Function Documentation

3.1.2.1 void SCDVK::begin ( )

Initialize variables, pin modes, etc.

Definition at line 144 of file SCDVK.cpp.

Here is the call graph for this function:



3.1.2.2 void SCDVK::echo ( )

Send data btwn serials(1&2)

Definition at line 271 of file SCDVK.cpp.

3.1.2.3 void SCDVK::exec_cmd (char * *cmd*, char * *data*)

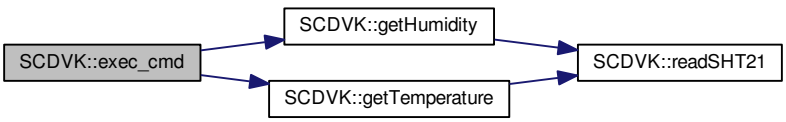
Send ALL data btwn serials(1&2)

Parameters

cmd	Ptr to cmd to send
data	Ptr to save result

Definition at line 340 of file SCDVK.cpp.

Here is the call graph for this function:

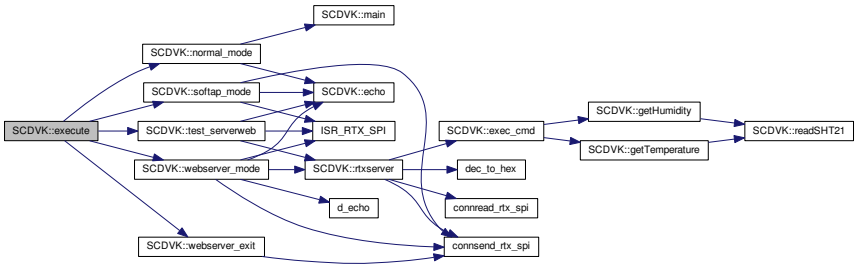


3.1.2.4 void SCDVK::execute ( )

executed program (loop)

Definition at line 297 of file SCDVK.cpp.

Here is the call graph for this function:



3.1.2.5 float SCDVK::getHumidity ( )

Used for get humidity.

Returns

humidity in %

Definition at line 562 of file SCDVK.cpp.

Here is the call graph for this function:

3.1.2.6 float SCDVK::getTemperature ( )

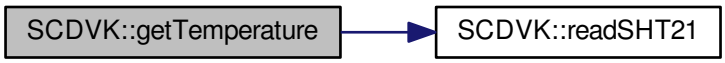
Used for get temperature.

Returns

temperature in Cº

Definition at line 553 of file SCDVK.cpp.

Here is the call graph for this function:



3.1.2.7 void SCDVK::main ( )

main for default mode

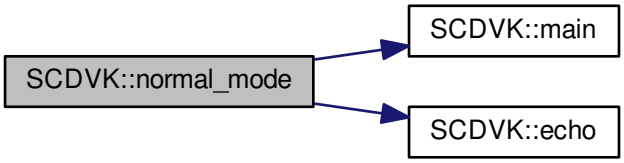
Definition at line 321 of file SCDVK.cpp.

3.1.2.8 void SCDVK::normal_mode ( )

Normal execution (without softap)

Definition at line 226 of file SCDVK.cpp.

Here is the call graph for this function:



3.1.2.9 uint16_t SCDVK::readSHT21 (uint8_t *type*)

Used for read data from SHT21.

Parameters

type	Type data called

Returns

number returned by SHT21

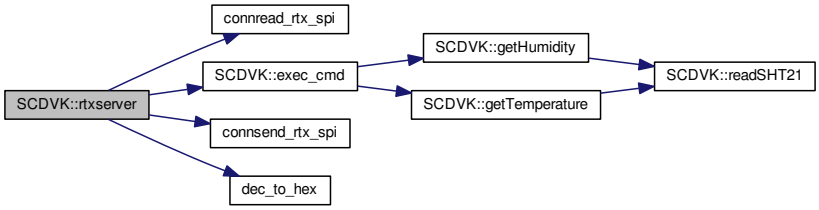
Definition at line 534 of file SCDVK.cpp.

3.1.2.10 void SCDVK::rtxserver ( )

Process API rcv by RTX (webserver mode)

Definition at line 368 of file SCDVK.cpp.

Here is the call graph for this function:

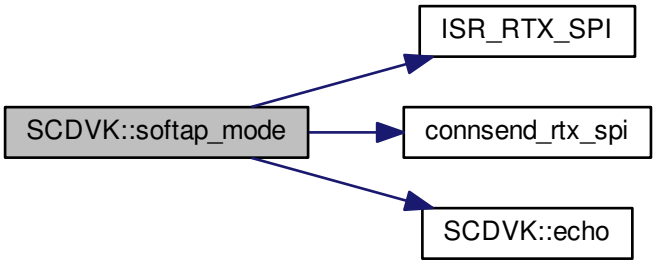


3.1.2.11 void SCDVK::softap_mode ( )

Send API(SPI) to start SoftAP.

Definition at line 240 of file SCDVK.cpp.

Here is the call graph for this function:

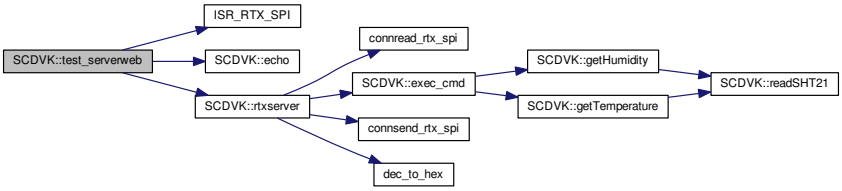


3.1.2.12 void SCDVK::test_serverweb ( )

Force serverweb state (without start it)

Definition at line 282 of file SCDVK.cpp.

Here is the call graph for this function:

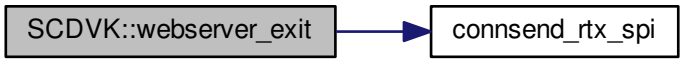


3.1.2.13 void SCDVK::webserver_exit ( )

Send API(SPI) and exit webserver mode.

Definition at line 209 of file SCDVK.cpp.

Here is the call graph for this function:

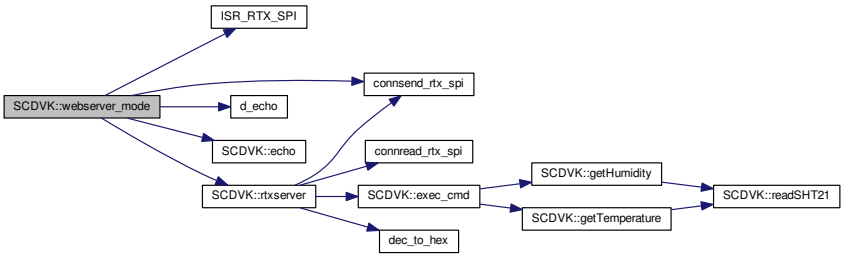


3.1.2.14 void SCDVK::webserver_mode ( )

Send API(SPI) to start WebServer and prepare for it.

Definition at line 179 of file SCDVK.cpp.

Here is the call graph for this function:



The documentation for this class was generated from the following files:

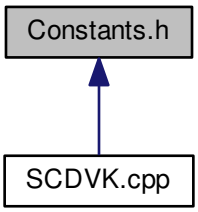
- SCDVK.h
- SCDVK.cpp

Chapter 4

File Documentation

4.1 Constants.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define RESET 8
- #define MUX 2
- #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 RTX_IRQ_SPI 22
- #define SHT21_ADDRESS 0x40

4.1.1 Macro Definition Documentation

4.1.1.1 #define BLUE A0

Definition at line 10 of file Constants.h.

4.1.1.2 #define GREEN A1

Definition at line 11 of file Constants.h.

4.1.1.3 #define MMC_CS 10

Definition at line 12 of file Constants.h.

4.1.1.4 #define MUX 2

Definition at line 8 of file Constants.h.

4.1.1.5 #define POWER 6

Definition at line 14 of file Constants.h.

4.1.1.6 #define PROG_MODE A2

Definition at line 9 of file Constants.h.

4.1.1.7 #define RESET 8

Definition at line 7 of file Constants.h.

4.1.1.8 #define RTX_CS 9

Definition at line 13 of file Constants.h.

4.1.1.9 #define RTX_IRQ_SPI 22

Definition at line 17 of file Constants.h.

4.1.1.10 #define SELECT_MODE_1 7

Definition at line 15 of file Constants.h.

4.1.1.11 #define SELECT_MODE_2 3

Definition at line 16 of file Constants.h.

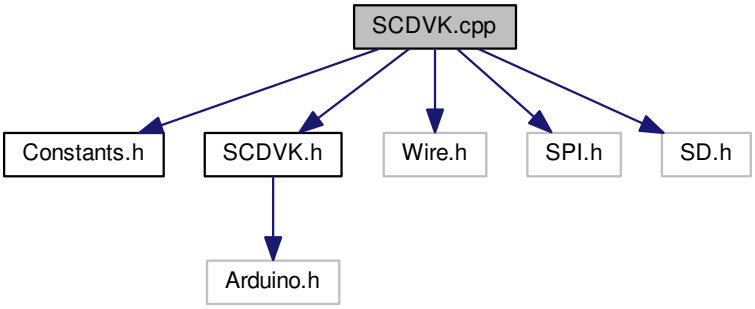
4.1.1.12 #define SHT21_ADDRESS 0x40

Definition at line 19 of file Constants.h.

4.2 SCDVK.cpp File Reference

```
#include "Constants.h"
#include "SCDVK.h"
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
```

Include dependency graph for SCDVK.cpp:



Macros

- #define debug
- #define d_println(x) SerialUSB.println(x); d_echo(); delay(3)
- #define d_print(x) SerialUSB.print(x)
- #define SPI_TX_BUFFER 1024
- #define SPI_RX_BUFFER 256
- #define SPI_SPEED 1000000
- #define SPI_TIMEOUT 3000

Functions

- void ISR1 ()
- void ISR2 ()
- void ISR_RTX_SPI ()
- byte dec_to_hex (byte *dst, long dec_num)

  Convert a long var to an array of bytes.
- void connread_rtx_spi (int l)

  Read N bytes from SPI.
- void connsend_rtx_spi (byte *data, int len)

  Send N bytes to SPI.
- void d_echo ()

  Send ALL data btwn serials(1&2)

Variables

- int operation_mode = 0
- volatile bool rtx_signal_off = false
- volatile bool rtx_signal_on = false
- byte SPI_tx [SPI_TX_BUFFER]
- byte SPI_rx [SPI_RX_BUFFER]
- byte SPI_rx_count = 0
- bool SD_INIT = false
- bool softap_enabled = false
- bool webserver_enabled = false
- bool timeout_error = false
- unsigned long timeisr = 0
- unsigned long time_timeout = 0

4.2.1 Macro Definition Documentation

4.2.1.1 #define d_print(*x*) SerialUSB.print(x)

Definition at line 11 of file SCDVK.cpp.

4.2.1.2 #define d_println(*x*) SerialUSB.println(x); d_echo(); delay(3)

Definition at line 10 of file SCDVK.cpp.

4.2.1.3 #define debug

Definition at line 7 of file SCDVK.cpp.

4.2.1.4 #define SPI_RX_BUFFER 256

Definition at line 18 of file SCDVK.cpp.

4.2.1.5 #define SPI_SPEED 1000000

Definition at line 19 of file SCDVK.cpp.

4.2.1.6 #define SPI_TIMEOUT 3000

Definition at line 20 of file SCDVK.cpp.

4.2.1.7 #define SPI_TX_BUFFER 1024

Definition at line 17 of file SCDVK.cpp.

4.2.2 Function Documentation

4.2.2.1 void connread_rtx_spi (int *l*)

Read N bytes from SPI.

Parameters

l	Number bytes to read

Definition at line 106 of file SCDVK.cpp.

4.2.2.2 void connsend_rtx_spi (byte * *data*, int *len*)

Send N bytes to SPI.

Parameters

data	Ptr to array
len	Number bytes to send

Definition at line 125 of file SCDVK.cpp.

4.2.2.3 void d_echo ( )

Send ALL data btwn serials(1&2)

Definition at line 570 of file SCDVK.cpp.

4.2.2.4 byte dec_to_hex (byte * *dst*, long *dec_num*)

Convert a long var to an array of bytes.

Parameters

dst	Ptr to array of bytes
dec_num	Number to convert

Returns

number of bytes required

Definition at line 87 of file SCDVK.cpp.

4.2.2.5 void ISR1 ( )

Definition at line 50 of file SCDVK.cpp.

4.2.2.6 void ISR2 ( )

Definition at line 60 of file SCDVK.cpp.

4.2.2.7 void ISR_RTX_SPI ( )

Definition at line 69 of file SCDVK.cpp.

4.2.3 Variable Documentation

4.2.3.1 int operation_mode = 0

Definition at line 23 of file SCDVK.cpp.

4.2.3.2 volatile bool rtx_signal_off = false

Definition at line 30 of file SCDVK.cpp.

4.2.3.3 volatile bool rtx_signal_on = false

Definition at line 31 of file SCDVK.cpp.

4.2.3.4 bool SD_INIT = false

Definition at line 40 of file SCDVK.cpp.

4.2.3.5 bool softap_enabled = false

Definition at line 41 of file SCDVK.cpp.

4.2.3.6 byte SPI_rx[SPI_RX_BUFFER]

Definition at line 35 of file SCDVK.cpp.

4.2.3.7 byte SPI_rx_count = 0

Definition at line 36 of file SCDVK.cpp.

4.2.3.8 byte SPI_tx[SPI_TX_BUFFER]

Definition at line 34 of file SCDVK.cpp.

4.2.3.9 unsigned long time_timeout = 0

Definition at line 45 of file SCDVK.cpp.

4.2.3.10 unsigned long timeisr = 0

Definition at line 44 of file SCDVK.cpp.

4.2.3.11 bool timeout_error = false

Definition at line 43 of file SCDVK.cpp.

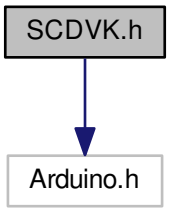
4.2.3.12 bool webserver_enabled = false

Definition at line 42 of file SCDVK.cpp.

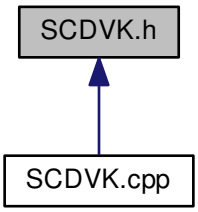
4.3 SCDVK.h File Reference

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

Include dependency graph for SCDVK.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SCDVK

Functions

- void d_echo ()

  Send ALL data btwn serials(1&2)

4.3.1 Function Documentation

4.3.1.1 void d_echo ( )

Send ALL data btwn serials(1&2)

Definition at line 570 of file SCDVK.cpp.