Universitat de Vic
Escola Politècnica Superior

Treball Final de Carrera

Sistemes encastats en temps real

Daniel Àlvarez Sanuy

Enginyeria Tècnica Industrial, especialitat en Electrònica Industrial

Director: Moisès Serra Serra

Vic, Juny de 2009

Índex:

Índex de figures:

9. Annex A

Resum de Projecte Final de Carrera
Enginyeria Tècnica Industrial, especialitat en Electrònica Industrial

Títol: Sistemes encastats en temps real
Paraules clau: Sistemes encastats, temps real, FreeRTOS, scheduler, tasques, AT91SAM7S256.
Autor: Daniel Àlvarez Sanuy
Direcció: Moisès Serra Serra
Data: Juny de 2009

Resum

En l'actualitat, els sistemes electrònics de processament de dades són cada cop més significatius dins del sector industrial. Són moltes les necessitats que sorgeixen en el món dels sistemes d'autentificació, de l'electrònica aeronàutica, d'equips d'emmagatzemament de dades, de telecomunicacions, etc. Aquestes necessitats tecnològiques exigeixen ser controlades per un sistema fiable, robust, totalment dependent amb els esdeveniments externs i que compleixi correctament les restriccions temporals imposades per tal de que realitzi el seu propòsit d'una manera eficient.

Aquí és on entren en joc els sistemes encastats en temps real, els quals ofereixen una gran fiabilitat, disponibilitat, una ràpida resposta als esdeveniments externs del sistema, una alta garantia de funcionament i una àmplia possibilitat d'aplicacions.

Aquest projecte està pensat per a fer una introducció al món dels sistemes encastats, com també explicar el funcionament del sistema operatiu en temps real FreeRTOS; el qual utilitza com a mètode de programació l'ús de tasques independents entre elles. Donarem una visió de les seves característiques de funcionament, com organitza tasques mitjançant un scheduler i uns exemples per a poder dissenyar-hi aplicacions.

Final Project Summary
Technical Industrial Engineering, specialized in Industrial Electronics

Title: Real-time embedded systems
Keywords: Embedded systems, real-time, FreeRTOS, scheduler, tasks, AT91SAM7S256.
Author: Daniel Àlvarez Sanuy
Managing: Moisès Serra Serra
Date: June of 2009

Summary

Currently electronic data processing systems are becoming more and more influential within the industrial sector. Many are the necessities arising in the world of authentification systems, aeronautical electronics, data storage equipment and telecommunications. These technological needs have to be controlled by a reliable system which is robust, totally dependent on external events and which complies with temporally imposed restrictions so that its purpose may be carried out efficiently.

This is where real-time embedded systems come into play. These systems provide reliability, availability, a fast response to the external system events, a high functional guarantee and a broad range of applications.

This project is thought to do an introduction on the embedded systems world, as well as explain the performance of the FreeRTOS real-time operating system, which uses a method of programming via independent tasks. We will talk about his performance characteristics, how it organizes tasks through a scheduler and we will explain some examples in order to design applications on it.

CAPÍTOL 1: Introducció

En aquest capítol introductori iniciarem i donarem a conèixer l'abast del projecte. Explicarem en què consisteix, en quin àmbit de la enginyeria està situat i quines aplicacions podria tenir de cara a la indústria electrònica.

També s'enumeren els diferents objectius sobre els quals es basa el projecte. Es dóna una clara primera idea al lector sobre el contingut del projecte, els seus diferents apartats, en què consisteixen i com es desenvoluparà al llarg de la memòria.

1. Introducció:

En aquest projecte es farà una explicació dels sistemes encastats, més coneguts com a "embedded". Es pretén donar informació sobre els tipus de sistemes i plataformes que hi ha, com es programen, quin software o sistema operatiu suporten (ja sigui Windows CE, Linux Embedded, LynxOS, FreeRTOS o MaRTE OS entre molts d'altres). Es farà un estudi de mercat per saber els tipus de sistemes encastats que hi ha, amb les seves respectives característiques i finalment indicarem algunes de les seves aplicacions.

Donat que és un treball sobre noves tecnologies, hi ha poca informació al respecte. Per tant, també es farà una explicació per tal d'introduir els conceptes teòrics que aniran sortint durant el desenvolupament del treball.

Aquest projecte treballarà sobre una placa Atmel en sistema encastat en temps real. S'explicaran les característiques de funcionament, ports d'entrada i sortida, sistemes operatius de codi obert que suporta, operacions que permet fer (interrupcions, prioritats entre tasques, etc), entorn de programació i finalment es faran uns exemples sobre la placa per tal d'entendre el seu correcte funcionament.

El nombre creixent d'aplicacions resulten de la necessitat de dissenyar tecnologies que suportin el disseny de sistemes encastats. Actualment, les tecnologies i eines disponibles encara tenen limitacions importants. Per exemple, encara hi ha la necessitat d'especificar llenguatges de programació millors, eines de generació d'implementacions de les especificacions, verificadors de temps, sistemes operatius en temps real, tècniques de disseny de baix consum, tècniques de disseny per sistemes dependents, etc.

Les aplicacions dels sistemes encastats són moltes, es poden aplicar en el món de automòbils electrònics, en trens, en el món militar, en sistemes de comunicació, en electrònica de consum entre molts d'altres. Per tant, és important tenir en compte aquests tipus de sistemes en l'actualitat, on l'avanç tecnològic creix a un ritme considerable.

D'acord amb una sèrie de previsions, el mercat dels sistemes encastats aviat serà més gran que el mercat dels sistemes basats en PC. A més, es preveu que la quantitat de software utilitzat en els sistemes encastats incrementi.

1.1. Objectius del treball:

Els objectius d'aquest treball són els següents:

- Introducció als sistemes encastats
- Veure els diferents tipus de sistemes encastats
- Funcionament de la nostra placa amb un sistema operatiu en temps real
- Disseny d'aplicacions que ens permet el nostre sistema operatiu
- Execució del seu funcionament

Podem afirmar que l'objectiu principal d'aquest projecte és el disseny d'una aplicació que s'adapti a les característiques suportades i l'explicació durant el seu desenvolupament, però també és important fer una introducció als sistemes encastats i veure les seves aplicacions en l'actualitat.

1.2. Estructuració de la memòria:

La memòria s'estructura en els següents apartats:

- Introducció:
  S'explica en què consistirà el projecte i fa una breu introducció als sistemes encastats i en quins àmbits de l'enginyeria es poden aplicar.

- Fonaments teòrics:
  S'explica detalladament què són els sistemes encastats, així com els seus requisits, funcionalitats i aplicacions en l'àmbit industrial.

- Estat de l'art dels sistemes encastats:
  S'explica l'estat actual del mercat dels diferents tipus de sistemes encastats, les seves característiques corresponents, el sistema operatiu que utilitzen, etc. Bàsicament en diferenciarem tres tipus, ordenats de major a menor tamany del sistema operatiu.

- Plataforma ARM7:
  Es fa un breu estudi sobre la placa en la qual treballarem indicant els ports, entrades i sortides que conté. També s'explica com està organitzada i dividida la memòria flash.

- Entorn de treball:
  S'expliquen les eines de codi obert que s'utilitzen per al desenvolupament de l'aplicació i uns esquemes de funcionament de la compilació i debugació de tot aquest entorn.

- Sistema operatiu en temps real FreeRTOS:
  Ampli estudi sobre el funcionament del sistema operatiu FreeRTOS. Inclou el seu mètode de planificació de tasques, els seus diferents estats, prioritats entre tasques, subrutines, funcions API que permet, el programa de recuperació SAM-BA, comunicació sèrie, etc. També es defineixen altres sistemes operatius i altres mètodes d'scheduling existents, incloent el del FreeRTOS, per saber altres maneres de planificació de tasques.

- Conclusions i resultats:
  En aquest darrer apartat s'expliquen les conclusions a les que hem arribat, és a dir, si hem complert els objectius, de què ens ha servit el projecte, com es podria ampliar, etc.

CAPÍTOL 2: Fonaments teòrics

Un cop introduïts els conceptes bàsics sobre la estructuració de la memòria, procedirem a explicar la base teòrica sobre la que treballarem. Així doncs, en aquest capítol ens centrarem en un estudi dels sistemes encastats, en la que farem una breu referència històrica, definirem l'abast dels sistemes encastats i algunes de les seves aplicacions.

Introduirem conceptes importants com ara els requisits que es demanen en els sistemes encastats, les exigències que han de controlar, les restriccions temporals que han de complir, com han de ser perquè es considerin eficients, etc.

També introduirem el terme de la computació ubiqua, i com aquesta està relacionada amb els sistemes encastats i el disseny de noves tecnologies.

Per últim, enumerarem i explicarem amb més detall que el capítol anterior quines són les seves àrees d'aplicació dins de l'enginyeria, així com uns quants exemples del seu ús.

2. Fonaments teòrics:

En aquest apartat farem una explicació sobre els sistemes encastats i les seves aplicacions per tal de comprendre el contingut del treball.

2.1. Importància dels sistemes encastats:

Els sistemes encastats poden ser definits com a sistemes de processament d'informació incrustats a dins de productes com ara automòbils, equips de telecomunicació o de fabricació. Aquests sistemes tenen un gran nombre de d'aplicacions determinants, incloent obligacions en temps real i dependència, com també requisits d'eficiència. La tecnologia dels sistemes encastats és essencial per poder proveir informació ubiqua, una de les seves fites de la tecnologia d'informació moderna.

Seguint a l'èxit de la tecnologia d'informació per a aplicacions d'oficina, les aplicacions dels sistemes encastats són considerades de les més importants en l'àrea de tecnologia d'informació durant aquests darrers anys. Degut a aquesta expectativa neix el terme "era Post-PC". Aquest terme denota el fet que, en el futur, els PC estàndards seran un tipus de hardware menys dominant. Els processadors i el software serà utilitzat en sistemes molt més petits i en alguns cops pot arribar a ser invisible. És obvi que molts productes tècnics han de ser tecnològicament avançats per despertar el interès del consumidor. Cotxes, càmeres, conjunts de TV, telèfons mòbils, etc. poden ser difícilment venuts si tenen poca utilitat i un software senzill.

El nombre de processadors en els sistemes encastats ja supera el nombre de processadors d'un PC, i s'espera que aquesta tendència continuarà creixent. Segons els pronòstics, el tamany del software embedded també anirà creixent a llarg termini.

2.2.Terminis i abast:

Fins a finals dels vuitanta, el processament de la informació estava associat amb grans ordinadors centrals i grans unitats de disc. Durant els noranta, aquest desplaçament cap al processament de la informació estava associat als ordinadors personals o PCs. Aquesta tendència cap a la miniaturització es preveu que segueixi continuant i que els sistemes més petits formin part de sistemes de control més grans.

La seva presència en aquests productes més grans, com ara equips de telecomunicació, serà menys evident que la del PC. A pesar d'això, amb aquesta nova tendència l'ordinador no desapareixerà, sinó que seguirà estant present a tot arreu.

Aquí cal fer referència a la computació ubiqua, la qual es centra en l'objectiu a llarg termini de subministrament de: "informació en qualsevol moment, a tot arreu", una metodologia innovadora que es centra una mica més sobre els aspectes pràctics i l'explotació de la tecnologia ja disponible.

Els sistemes encastats són sistemes de processament d'informació que han estat incrustats a dins d'un producte superior i que no sóc directament visibles per a l'usuari. Exemples de sistemes encastats inclouen sistemes de processament d'informació en equips de telecomunicació, en sistemes de transport, en equips de fabricació i en l'electrònica de consum.

Les característiques més comunes d'aquests sistemes són les següents:

- Freqüentment, els sistemes encastats estan connectats a un entorn físic a través de sensors recollint informació sobre l'entorn i els actuadors que el controlen.
- Els sistemes encastats han de ser dependents.

  Molts sistemes encastats són essencials per a la seguretat i, per tant, han de ser totalment dependents. Les plantes nuclears són un exemple de sistemes extremament crítics que són parcialment controlats per software. La dependència és, tanmateix, molt important en altres sistemes, com ara automòbils, trens, avions, etc. Una de les raons clau per ser crítics per la seguretat és que aquests sistemes estan directament connectats a l'entorn i tenen un impacte immediat en ell.

  La dependència abasta els següents aspectes d'un sistema:

  1) Fiabilitat: La fiabilitat és la probabilitat de que un sistema no falli.
  2) Mantenibilitat: La mantenibilitat és la probabilitat de que un sistema erroni pugui ser reparat dins d'un termini determinat.
  3) Disponibilitat: La disponibilitat és la probabilitat de que els sistemes estiguin disponibles. Tant la fiabilitat com la mantenibilitat han de ser altes per poder tenir una alta disponibilitat.
  4) Seguretat: Aquest terme descriu la propietat que un sistema erroni no causi cap mal.
  5) Garantia: Aquest terme descriu la propietat de que dades confidencials continuïn confidencials i aquesta autèntica comunicació és assegurada.

- Els sistemes encastats han de ser eficients per poder assegurar un bon resultat. Les següents directrius poden ser utilitzades per avaluar l'eficiència dels sistemes encastats:

  1) Energia: Molts sistemes encastats són sistemes mòbils que obtenen la seva energia a través de bateries. Hem de tenir present que els requisits de computació van incrementant a un ritme elevat (especialment en les aplicacions multimèdia) i els clients esperen una llarga vida de les seves bateries. Per tant, l'energia elèctrica disponible s'ha d'utilitzar de manera molt eficient.
  2) Tamany de codi: Tot el codi que s'ha de fer anar en un sistema encastat ha d'estar integrat en el sistema. Habitualment, no hi ha disc durs on hi pugui ser emmagatzemat. Afegir dinàmicament codi addicional segueix sent una excepció i es limita a casos com ara mòbils Java i dispositius de recepció de senyal.

     Debut a altres limitacions i a que el codi ha de consumir el mínim de recursos de hardware possibles, el tamany del codi ha de ser el més petit possible per a cada aplicació. Aquesta característica és especialment certa per als "sistemes al xip" (SoCs), sistemes pels quals tota la informació de processament dels circuits està inclosa en un sol xip. Si la instrucció de la memòria ha d'estar integrada en aquest xip, aquest ha de ser utilitzat de manera molta eficient.

3) Eficiència en el temps d'execució: S'hauria d'implementar la funció requerida amb el nombre mínim de recursos. Nosaltres hem de poder conèixer les limitacions temporals utilitzant el nombre mínim de recursos de hardware i energia. Amb la finalitat de reduir el consum d'energia, les freqüències de rellotge i els voltatges d'alimentació han de ser el més reduïts possibles. També només haurien d'estar presents els components de hardware necessaris. Els components que no milloren el temps d'execució (com ara alguns "caché" o unitats de gestió de memòria) poden ser omesos.

4) Pes: Tots els sistemes portàtils han de pesar poc. Sovint, aquesta característica és un argument de pes de cara el client per a comprar un determinat sistema.

5) Cost: Per als sistemes encastats d'alt emmagatzematge, especialment en clients electrònics, la competivitat en el mercat és extremadament crucial, i l'ús eficient del pressupost per al desenvolupament de components de hardware i software és, en moltes ocasions, una peça clau.

- Aquests sistemes estan dedicats a una determinada aplicació.

  Per exemple, el software incorporat en els processadors d'un automòbil o un tren utilitzaran sempre aquell software, i no hi haurà cap intenció de fer-hi anar un joc d'ordinador o un programa de full de càlcul o qualsevol altra aplicació en el mateix processador. Hi ha dues raons que ho justifiquen:

  1) Fer anar programes addicionals fa que aquests sistemes siguin menys dependents i no es puguin centrar amb la seva verdadera tasca.

  2) Només és viable fer anar programes addicionals si els recursos com ara la memòria no s'utilitzen. Els recursos no utilitzats no haurien d'estar presents en un sistema eficient.

- La majoria dels sistemes encastats no utilitzen teclat, ratolí o una pantalla d'ordinador per la seva interfície amb l'usuari. En comptes d'això, hi ha una interfície amb l'usuari dedicada que consta de polsadors, elements lluminosos, pedals, etc. Per aquest motiu, l'usuari reconeix que no està involucrat en el tractament de la informació. Per tant, podem veure que l'usuari té al seu abast una interfície clara i entenedora.

- Molts sistemes encastats han de tenir restriccions en temps real. No realitzar els càlculs dins d'un termini de temps pot desencadenar una gran pèrdua de la qualitat proporcionada pel sistema (per exemple, si la qualitat d'àudio o vídeo n'és afectada) o pot provocar malentesos a l'usuari (per exemple, si els cotxes, trens, o avions no operen de la manera prevista).

  Molts dels sistemes de processament d'informació actuals estan utilitzant tècniques per incrementar la velocitat de processament d'informació per intentar igualar l'actual. Per exemple, les "caché" milloren el rendiment mig d'un sistema. En altres casos, una comunicació fiable s'aconsegueix per la repetició de certes transmissions. Per exemple, els protocols d'Internet generalment depenen de reenviar els missatges en cas de que els missatges originals es perdin. En general, aquestes repeticions provoquen una petita pèrdua de rendiment, encara que per un determinat missatge de la comunicació es poden retardar ordres de magnitud major que la demora normal. Si parlem de sistemes encastats en temps real, no es poden tolerar els arguments sobre rendiment mig o retard en el processament de dades.

- Molts dels sistemes encastats són sistemes híbrids en el sentit que inclouen les parts analògiques i digitals. Les parts analògiques utilitzen valors continus de senyals en el temps, en canvi les parts digitals utilitzen valors discrets de senyals en el temps.

- Normalment, els sistemes encastats són sistemes reactius. Els sistemes reactius poden ser com estar en un determinat estat, a l'espera d'una entrada. Per a cada entrada, duen a terme alguns càlculs i generen una sortida i un nou estat. Per tant, els autòmats són molt bons models d'aquests sistemes. Les funcions matemàtiques, les quals descriuen el problema resolt per molts algorismes, serien un model inapropiat.
- Els sistemes encastats són insuficientment representats en l'ensenyament i en els debats públics. Un dels problemes en l'ensenyament del disseny dels sistemes encastats és l'equip necessari per fer el tòpic interessant i pràctic. També, els sistemes encastats reals són molt complexos i, en conseqüència, difícils d'explicar.

Degut a aquest recull de característiques (excepte l'última), analitzarem els enfocaments més comuns per a dissenyar sistemes encastats, com també un repàs a les diferents àrees d'aplicació.

En realitat, no tots els sistemes encastats tindran totes les característiques anteriors. Nosaltres també podem definir el terme "sistema encastat" de la següent forma: *Sistemes de processament d'informació que compleixin la majoria de les característiques enumerades anteriorment.* A pesar de que aquesta definició sigui una mica imprecisa, sembla que ser impossible i innecessari eliminar aquestes imprecisions.

Moltes de les característiques dels sistemes encastats poden trobar-se en un nou tipus de computació introduïda recentment: computació ubiqua o generalitzada, també anomenada intel·ligència ambiental. L'objectiu d'aquest tipus de computació és fer possible que la informació estigui disponible sempre i a tot arreu.

La següent figura mostra una representació gràfica sobre com la computació ubiqua està influenciada pels sistemes encastats i la tecnologia de la comunicació:

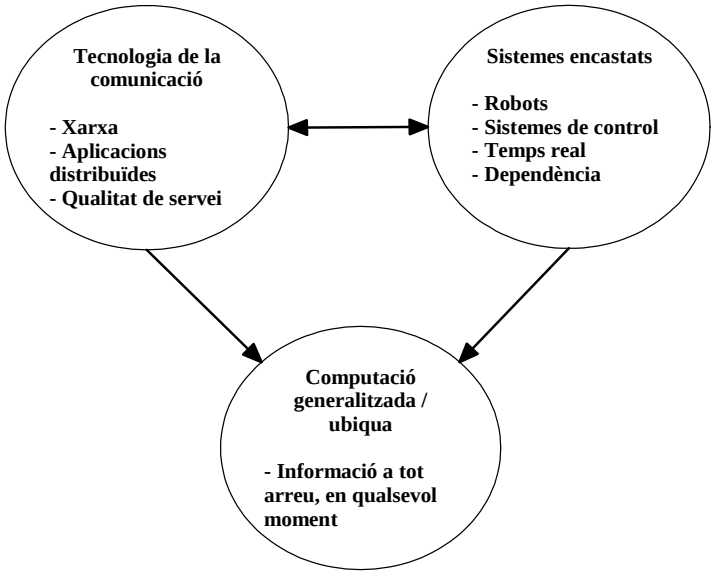

Figura 2.2.1. - *Ubiqüitat*

Per exemple, la computació ubiqua ha de conèixer els requisits de temps real i dependència dels sistemes encastats tot utilitzant tècniques fonamentals de tecnologia de la comunicació, com ara per exemple la xarxa.

2.3. Àrees d'aplicació:

La següent llista comprèn les àrees clau en les que s'utilitzen els sistemes encastats:

- Automobilística electrònica: Els cotxes moderns només poden ser venuts si contenen un nombre important de components electrònics. Això inclou sistemes de control d'Airbag, sistemes de control de motors, sistemes anti-bloqueig (ABS), aire condicionat, sistemes GPS, característiques de seguretat i molts més.
- Electrònica aeronàutica: Un nombre significant del valor total dels avions és degut a l'equip de processament de senyal, com ara sistemes de control de vol, sistemes anti-col·lisió, sistemes d'informació de pilotatge, i altres. La fiabilitat és de summa importància en aquest sector.
- Trens: En els trens, la situació és similar a la dels automòbils i avions. Altra vegada, les característiques de seguretat tenen un paper molt significatiu del valor total dels trens, i la dependència és extremadament important.
- Telecomunicació: El mercat dels telèfons mòbils ha estat un dels mercats més creixents en els últims anys. Per als telèfons mòbils, el disseny de la ràdio freqüència (RF), el procés digital de senyal i el disseny de baix consum són aspectes clau.
- Sistemes mèdics: Existeix un enorme potencial i motivació per millorar el servei mèdic, aprofitant el processament de la informació dels equips mèdics.
- Aplicacions militars: El processament d'informació ha estat utilitzat en equips militars des de fa molts anys. Alguns dels primers ordinadors analitzaven senyals de ràdio militar.
- Sistemes d'autenticació: Els sistemes encastats poden ser utilitzats per propòsits d'autenticació, com ara per exemple permetre l'accés a un compte bancari, a una zona web amb privilegis, sensors d'empremtes dactilars, sistemes de reconeixement facial, etc.
- Electrònica de consum: Els equips de vídeo i àudio són un sector molt important en la indústria electrònica. El processament d'informació integrat a dins d'aquests equips és cada vegada més gran. Els nous sistemes de major qualitat s'implementen utilitzant tècniques avançades de processament de senyal digital.

Molts equips de TV, telèfons multimèdia i consoles tenen processadors i sistemes de memòria d'alt rendiment que requereixen un sistema operatiu encastat que els suporti i els controli eficientment.

- Equips de fabricació: Els equips de fabricació estan en una àrea molt tradicional en la qual els sistemes encastats han estat utilitzats durant dècades. La seguretat és molt important per aquests sistemes, i el consum d'energia és un problema de menor importància. Per exemple, la següent figura mostra un contenidor connectat a una canonada. La canonada inclou una vàlvula i un sensor. Utilitzant la lectura del sensor, un ordinador ha de controlar la quantitat de líquid que surt de la canonada.

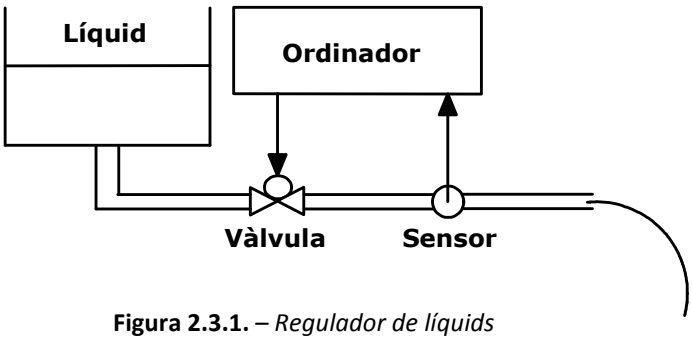

Figura 2.3.1. – *Regulador de líquids*

- Edificis intel·ligents: El processament d'informació pot ser utilitzat per incrementar el nivell de confort en els edificis, per reduir el seu consum d'energia i per proporcionar seguretat i protecció. Els subsistemes que tradicionalment no tenien relació, han d'estar connectats per a aquesta finalitat. Hi ha una tendència cap a la integració de l'aire condicionat, lluminària, control d'accés, la comptabilitat i la distribució de la informació en un únic sistema. Per exemple, es pot estalviar energia en refrigeració, calefacció i il·luminació de les habitacions buides. Les habitacions disponibles poden estar ubicades a llocs apropiats, simplificant les tasques de neteja. El soroll de l'aire condicionat pot ser reduït al nivell desitjat per les condicions actuals de funcionament. L'ús intel·ligent de les persianes pot optimitzar la lluminària i l'aire condicionat. Els nivells de tolerància dels subsistemes d'aire condicionat pot ser incrementat en les habitacions buides, i la lluminària pot ser reduïda automàticament. Es pot mostrar les llistes de les habitacions plenes a l'entrada de l'edifici en situacions d'emergència (sempre que l'energia requerida estigui disponible).

En un principi, aquests sistemes només estaran presents en edificis intel·ligents d'alta tecnologia.

- Robòtica: La robòtica també és una àrea tradicional en la qual s'utilitzen els sistemes encastats. Els aspectes mecànics són molt importants pels robots. La majoria de les característiques descrites anteriorment també s'apliquen a la robòtica. Recentment s'han dissenyat alguns nous prototips de robots, modelats en animals o humans que necessiten ser regulats per un sistema operatiu a temps real que s'adapti perfectament als seus moviments mecànics i pugui realitzar aplicacions de tot tipus.

CAPÍTOL 3: Estat de l'art dels sistemes encastats

Donat que en els capítols anteriors hem explicat què són i en què consisteixen els sistemes encastats, en aquest altre capítol farem un estudi de mercat sobre els diferents tipus de sistemes encastats que es poden trobar actualment.

Per tal d'informar al lector sobre l'actualitat, explicarem cada tipus de sistema encastat indicant en cada cas els sistemes operatius que suporten, les característiques de hardware que tenen i les aplicacions que en poden sorgir.

En aquest capítol, també justificarem l'elecció dels sistemes encastats en temps real per al desenvolupament del nostre projecte.

3. Estat de l'art dels sistemes encastats:

En aquest apartat veurem els diferents tipus de sistemes encastats o embedded que hi ha en el mercat amb les seves característiques, sistemes operatius que suporten i aplicacions que poden proporcionar.

Diferenciarem els tipus de sistemes encastats en dos grups bàsics o fonamentals segons si treballen en temps real o no:

- Sistemes encastats sense temps real
- Sistemes encastats en temps real

Farem un estudi sobre cada un d'aquests sistemes i a partir d'aquest punt determinarem i justificarem l'elecció dels sistemes STR.

3.1. Sistemes encastats sense temps real:

Diferenciarem aquests tipus de sistemes encastats en dos subapartats: plataformes amb display tàctil i plataformes ALIX i ITX.

3.1.1. Plataformes amb display tàctil:

Pertanyen en aquest grup les plataformes encastades que es basen en un display tàctil (ja sigui resistiu o capacitiu). Aquestes plataformes tenen un controlador incrustat on prèviament s'hi ha instal·lat un sistema operatiu específic; el més utilitzat és el Windows Compact Embedded o Windows CE.

Els principals sistemes operatius en sistema encastat que es troben en una plataforma d'aquest tipus són:

- Windows CE
- Windows XP Embedded
- Linux Embedded

Aquests tipus de sistemes operatius són pre-instal·lats pel fabricant del producte i no es poden intercanviar en una mateixa plataforma, això les diferencia de les plataformes d'arquitectura x86 com poden ser els ordinadors personals o les plaques ALIX i ITX que veurem més endavant.

És per aquest motiu que descartem aquest tipus de plataformes encastades per aquest projecte, ja que nosaltres pretenem implementar diferents sistemes operatius de codi obert en la mateixa plataforma.

3.1.1.1. Característiques de hardware:

Aquestes plataformes encastades han de permetre una bona interacció amb l'entorn donat que han de controlar dispositius intel·ligents, connectats i orientats a serveis de cara a l'usuari.

Per tant, les característiques de hardware en quant a velocitat de processament i memòria Flash han de ser relativament elevades per poder garantir un entorn ràpid i fiable, i es varien els ports d'entrada i sortida de dades en cada cas en funció de les necessitats de l'entorn. Per exemple, en una PDA només es podria necessitar un slot per a una targeta SD, en canvi en una plataforma connectada a diversos PLC's necessitaria diversos ports sèrie RS-232.

Les característiques més comunes d'aquestes plataformes són:

- Processador ARM o AVR de 32 bits
- Temps ràpid d'inicialització
- Rang de temperatura: -10 a +60 ºC
- Interfície controladora amb LCD Display
- Pantalla tàctil
- Sistema operatiu pre-instal·lat
- Microsoft Compact Framework 3.5 pre-instal·lat
- Suporta aplicacions Visual C++, Visual Basic i C#
- Mode debug
- Memòria i programació FLASH
- Un slot per a SD
- Ports sèrie RS-232
- Port USB per a la sincronització de la plataforma amb el PC per ActiveSync
- Port USB Client
- Sortida àudio
- Consum aproximat: 450-4500 mW
- Alimentació aproximada: 7-24 Vdc

3.1.1.2. Aplicacions:

L'aplicació més habitual d'aquestes plataformes és la seva utilització en GPS, els quals precisen d'un sistema operatiu.

Una altra clara aplicació d'aquestes plataformes són les conegudes PDA's, les quals no tenen tantes entrades i sortides ja que s'utilitzen més aviat per a ús personal. Altres exemples podrien ser mòbils multimèdia, ordinadors de bord d'un automòbil, ...

3.1.2. Pataformes ALIX i ITX:

Pertanyen en aquest grup les plaques ALIX i les plaques ITX. Una placa ALIX és un ordinador amb arquitectura i386, funcionant amb un processador d'entre 400-800 MHz, normalment amb un AMD Geode.

En canvi, les plaques ITX són una mica més potents (de l'ordre de 1-1'5GHz de processador) i són plataformes encastades que tenen la mateixa funció que un ordinador personal. Es caracteritzen per les seves reduïdes dimensions i el seu baix consum. Les diferents mides que podem trobar són:

- Mini ITX: 17x17 cm
- Nano ITX: 12x12 cm
- Pico ITX: 10x7'2 cm

3.1.2.1. Característiques de hardware:

La següent taula mostra els diferents models de plaques ALIX disponibles i les seves característiques tècniques:

Model	CPU	DRAM	LAN	MiniPCI	PCI	USB	Altres	BIOS
1D	LX800	256MB	1	1	1	2	VGA, Àudio, PS/2	Award
2D0	LX700	128MB	2	2	0	0	-	TinyBIOS
2D1	LX700	128MB	3	1	0	0	-	TinyBIOS
2D2	LX800	256MB	2	2	0	2	-	TinyBIOS
2D3	LX800	256MB	3	1	0	2	-	TinyBIOS
2D13	LX800	256MB	3	1	0	2	Bateria, I2C, COM2	TinyBIOS
3D1	LX700	128MB	1	2	0	0	-	TinyBIOS
3D2	LX800	256MB	1	2	0	2	-	TinyBIOS
3D3	LX800	256MB	1	2	0	2	VGA, Àudio	Award
6B2	LX800	256MB	2	1	0	2	MiniPCI Express	TinyBIOS

Figura 3.1.2.1.1. – *Plaques ALIX*

Aquest tipus de sistemes encastats suporten els següents principals sistemes operatius que es mostren a continuació per al seu correcte funcionament:

- Debian Linux
- Free BSD
- Damn Small Linux
- iMedia Linux
- FreeDOS, MS-DOS 5.0
- NetBSD
- OpenBSD
- ZeroShell
- Ikarus OS
- Ubuntu Linux
- Windows XP
- RouterOS

S'ha de dir que aquests tipus de sistemes encastats no porten disc dur, per tant el sistema operatiu s'ha de carregar prèviament a una Compact Flash (CF).

Com que aquests sistemes operatius també configuren automàticament els ports d'entrada i sortida del sistema encastat i no deixa gaire marge de maniobra per a fer aplicacions, descartem també aquesta opció per al nostre projecte.

3.1.2.2. Aplicacions:

Aquests sistemes serveixen bàsicament per a aplicacions Wireless. La seva aplicació més comuna és la seva utilització com a routers, però també serveixen com a firewalls o sistemes especials dedicats a aplicacions de xarxa.

3.2. Sistemes encastats en temps real:

Un sistema en temps real és un sistema que interactua activament amb un entorn amb dinàmica coneguda en relació amb les seves entrades, sortides i restriccions temporals, per a poder donar-li un correcte funcionament d'acord amb els conceptes d'estabilitat, controlabilitat i adaptabilitat. Aquests sistemes utilitzen S.O. de tamany reduït respecte els anteriors.

Els STR són sistemes informàtics que es troben en moltes aplicacions, des de l'electrònica de consum fins el control de complexos processos industrials. Estan presents en pràcticament tots els aspectes de la nostra societat com ara per exemple: telèfons mòbils, automòbils, ingenis espacials, processos automàtics de fabricació, aeronaus, producció d'energia, etc. A més, aquests sistemes estan en constant augment ja que cada vegada més màquines es fabriquen incloent un número major de sistemes controlats per computador. Un clar exemple és la indústria de l'automòbil: un turisme actual de gamma mitja inclou al voltant d'una dotzena d'aquests automatismes (ABS, airbag, etc). Un altre exemple seria els electrodomèstics de nova generació, que inclouen STR per al seu control i temporització.

La principal característica dels sistemes encastats en temps real és el temps. El correcte funcionament d'aquests dispositius no depèn tant sols del resultat lògic que retorna la computadora, sinó del temps en què es produeix aquest resultat.

Una altra característica important d'aquests sistemes és que són reals, és a dir, que la reacció del sistema a esdeveniments externs ha de succeir durant la seva evolució. El temps dels sistema (temps intern) ha de ser mesurat utilitzant la mateixa escala amb la que es mesura el temps de l'ambient controlat (temps extern).

Un STR té tres condicions bàsiques que sempre ha de complir:

- Interactua amb el món real (procés físic)
- Emet respostes correctes
- Compleix restriccions temporals

Aquestes condicions diferencia clarament els STR dels Sistemes en Línia. La funció d'aquests és la d'estar encès, disponible i generalment connectat a una xarxa de computadors i depèn de la capacitat del hardware per a atendre peticions de servei, però en cap moment està en sincronia amb el món real ni té restriccions temporals.

3.2.1. Sistemes operatius en temps real (RTOS):

Un sistema operatiu en temps real (SOTR) és un sistema operatiu que ha estat desenvolupat per a aplicacions de temps real. Com a tal, se li exigeix correcció de les seves respostes sota certes restriccions temporals, de manera que si no les respecta es diu que el sistema ha fracassat. Per garantir el correcte funcionament en el temps desitjat, es necessita que el sistema sigui predictible.

La manera més habitual de passar informació des del món exterior al programa és amb les interrupcions. Per naturalesa són imprevisibles. En un sistema de temps real aquestes interrupcions poden informar diferents esdeveniments com la presència de nova informació en un port de comunicacions, d'una nova mostra d'àudio en un equip de so o d'un nou quadre d'imatge en una vídeo-gravadora digital.

Els sistemes operatius més utilitzats per a aquests sistemes són:

- FreeRTOS
- MaRTE OS
- SOOS Project
- QNX
- LynxOS
- VxWorks
- eCos
- BeRTOS

Són de programació lliure i per tant es poden modificar els paràmetres de configuració dels ports. Al permetre també l'ús d'interrupcions i mode debug, escollim aquest tipus de sistemes encastats per al nostre projecte.

3.2.2. Característiques de hardware:

Característiques més comunes dels sistemes encastats en temps real:

- Processador de 16 o 32 bits
- Velocitat de processament entre 30-60MHz
- 256Kb de memoria flash
- Permet fer debug
- Permet interrupcions
- USB 2.0
- Alimentació 7-14Vdcc
- Ports sèrie RS-232
- Botó extern de reset
- No utilitza molta memòria
- Multi-tasca

3.2.3. Aplicacions:

Aquests sistemes operatius fan fàcil de controlar un entorn per desenvolupar tasques simultànies en temps real com ara semàfors, protocols de xarxa, servidors web, ordinadors de bord d'un automòbil, sistemes d'autentificació, robòtica, etc.

CAPÍTOL 4: Plataforma ARM7

Com que ja hem justificat l'elecció dels sistemes operatius en temps real, en aquest quart capítol explicarem l'esquema del hardware de la nostra placa així com el diagrama de blocs del seu processador encastat.

Aquí informarem sobre el mapa de memòria de la memòria flash i com està dividida en els corresponents lock bits que governen cada regió, i també de què poden ser d'utilitat aquests bits de bloqueig.

A més a més explicarem un mètode manual a seguir en cas de mal funcionament general de la memòria flash.

Així doncs, a partir d'aquest capítol donarem per entès que es treballa sobre la placa Atmel AT91SAM7S-EK amb un processador AT91SAM7S256 de 32 bits, i que transferirem i debugarem les nostres aplicacions a través de la interfície JTAG.

4. Plataforma ARM7:

La nostra placa d'avaluació Atmel AT91SAM7S-EK permet la utilització de desenvolupament de codi per a aplicacions que funcionin en un processador AT91SAM7Sxx. En el nostre cas, tenim un processador ARM7 AT91SAM7S256 de 32 bits.

Aquesta placa té els següents components:

- Port USB
- 2 Ports sèrie (UART i DBGU)
- Interfície JTAG/ICE per a debugar
- 4 Entrades analògiques
- 4 LEDs i 4 polsadors per a ús general
- 1 Polsador reset
- Connector d'expansió
- Àrea de prototips

4.1. Diagrama de blocs de la plataforma AT91SAM7S-EK:

La següent figura ens mostra un diagrama de blocs de la placa AT91SAM7S-EK amb els perifèrics que controla el microprocessador:

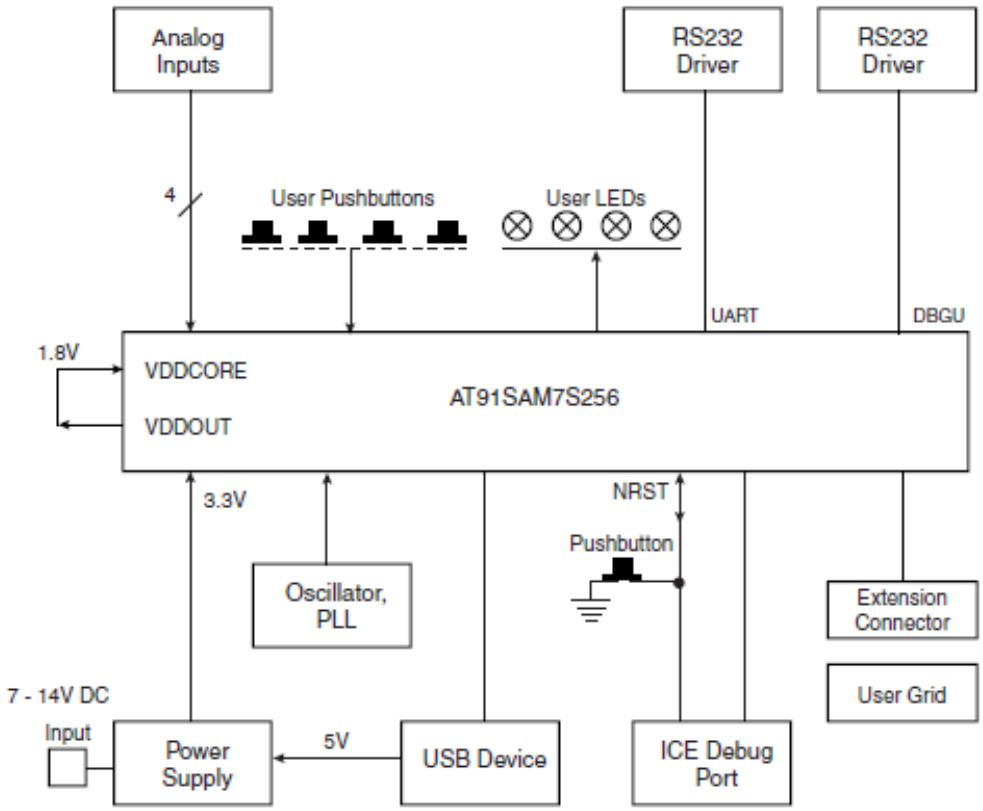

Figura 4.1.1. – *Diagrama de blocs AT91SAM7S-EK*

4.2. Processador:

El processador AT91SAM7S256 pertany a una sèrie de microprocessadors basats en un processador ARM RISC de 32 bits.

Com a característiques generals té una memòria Flash de 256Kb, una memòria SRAM de 64Kb, un ampli conjunt de perifèrics, incloent un port USB 2.0, i un complet conjunt de funcions de sistema minimitzant el nombre de components externs. El dispositiu és ideal per a usuaris d'un microprocessador de 8 bits que busquen una millora de rendiment i una ample memòria.

La memòria Flash encastada pot ser programada al sistema a través de la interfície JTAG-ICE o sèrie. Està dissenyada en lock bits i un security bit que protegeixen el firmware d'escriptures accidentals, preservant així la seva confidencialitat.

El processador AT91SAM7S256 és un microprocessador per a ús general. El seu port USB integrat fa que sigui un dispositiu ideal per a aplicacions que requereixen la connexió de perifèrics a un PC o a un telèfon mòbil. El seu agressiu preu i el seu nivell d'alta integració empeny al seu àmbit d'ús a dins d'un mercat molt sensible al cost.

4.3. Mapa de memòria AT91SAM7S256. Lock bits:

El nostre processador té 256k de memòria Flash organitzada en 1024 pàgines de 256 bytes cada una. Sabent que 256 bytes = 0x100 bytes tenim que:

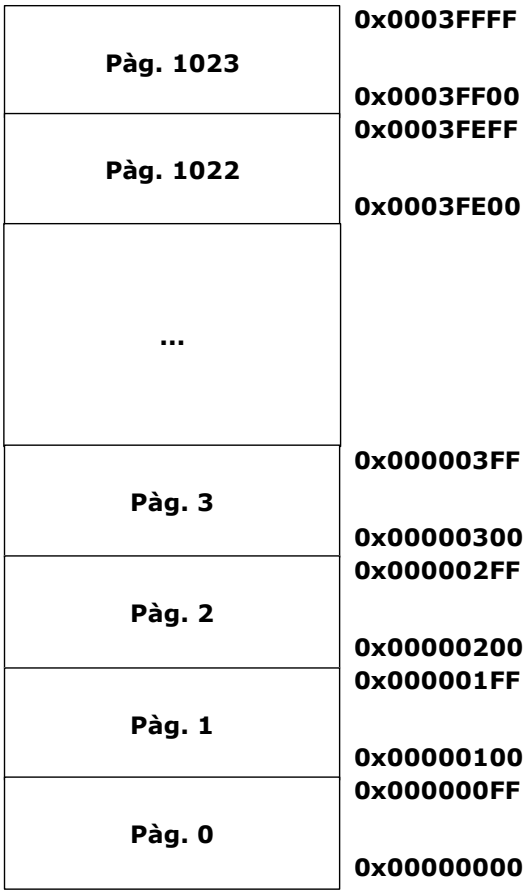

Figura 4.3.1. – Pàgines m*emòria Flash*

Per una banda sabem que una regió conté 64 *pàgines i que la memòria té un total de* 1024 *pàgines*:

$$\frac{1024\ pàg.totals}{64\ pàg.\ per\ regió} = 16\ lock\ bits.$$

Per l'altra banda sabem que cada pàgina té 256 *bytes, llavors*:

$$64\ pàg. \times 256\ bytes\ per\ regió = 0x4000\ bytes.$$

A partir d'aquí podem dir que 16 *lock bits o regions de* $0x4000$ *bytes cada una* $(0x0000 - 0x3FFF)$ *ocupen un total de*:

$$0x4000\ bytes\ per\ regió \times 16\ lock\ bits = 0x40000\ bytes = 256\ KBytes.$$

Figura 4.3.2. – *Càlculs memòria Flash*

Per altra banda, aquesta memòria està subdividida en regions “LOCK”. Hi ha 16 regions LOCK i cada una d’elles conté 64 pàgines (16384 bytes per regió). No es pot esborrar o programar cap dada en una regió “LOCKED”:

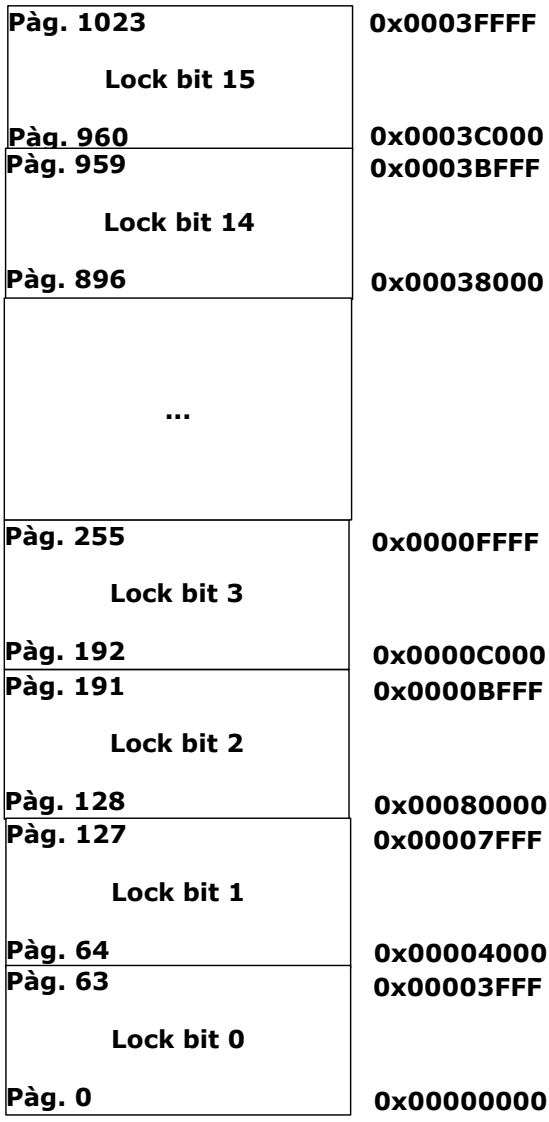

Figura 4.3.3. – *Lock bits memòria Flash*

Un cop sabem l’estructura de la memòria flash i dels lock bits, en podem trobar una utilitat per al disseny de les nostres aplicacions.

Nosaltres podem gravar a una posició de memòria i, per prevenir futurs accidents, bloquejar la regió en la que està allotjada. D’aquesta manera evitem que es pugui gravar en aquella posició de memòria preservant la confidencialitat de les dades.

En el següent apartat expliquem amb més de detall com gestionar aquestes regions.

4.4. Accés a la plataforma mitjançant comandes. Telnet:

La nostra placa AT91SAM7S256, de la mateixa manera que les de la sèrie AT91SAM7S, també es pot connectar a l'ordinador mitjançant el protocol Telnet. Això permet debugar manualment l'estat de la placa, per tant ens permet veure i editar els lock bits i esborrar sectors de la memòria flash.

Per a fer-ho, primer hem d'accedir a la placa a través del gravador JTAG:

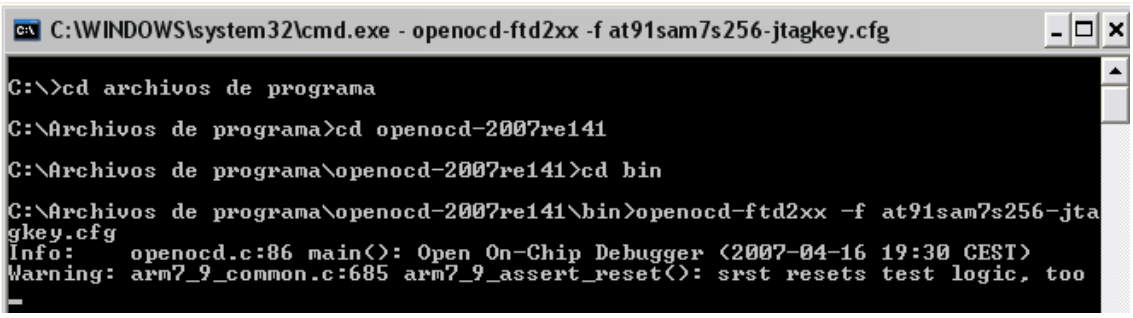

Figura 4.4.1. – *Telnet: Accés*

Un cop oberta la connexió, ja podem connectar la placa amb el OpenOCD a través de Telnet. Obrim una altra consola, posem la comanda: "telnet localhost 4444" i pressionem enter:

Figura 4.4.2. – *Telnet: Debugador*

Ara ja podem debugar a través del OpenOCD.

Tot seguit fem una mostra de les comandes que podem realitzar sobre el nostre processador AT91SAM7S256:

- flash info 0

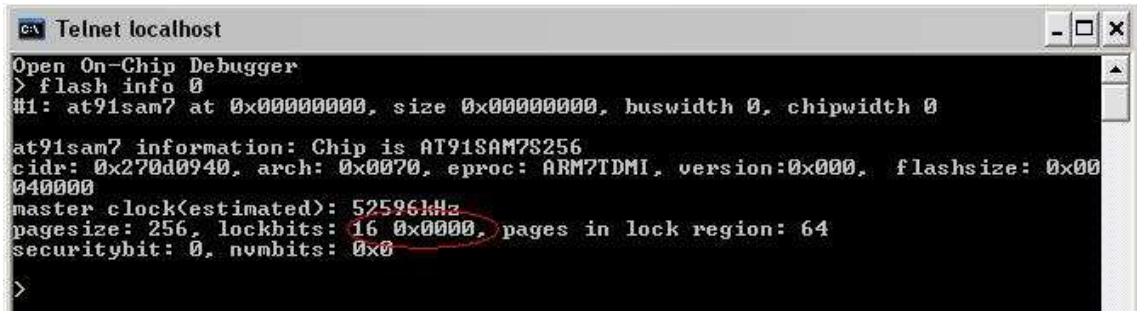

Figura 4.4.3. – *Telnet: Informació flash*

El valor hexadecimal marcat (0x0000) indica que no hi ha lock bits actius. La memòria flash està desbloquejada i a punt per a ser gravada.

- flash protect 0 0 <nombre de lock bits-1> [‘off’, ‘on’]

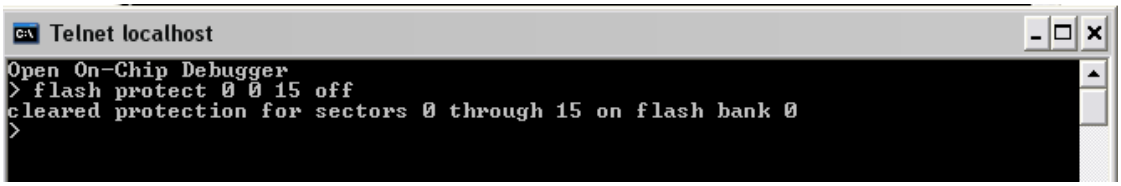

Figura 4.4.4. – *Telnet: Protecció flash*

Aquesta comanda ens ha netejat del lock bit 0 al 15. Si, pel contrari, haguéssim posat:

flash protect 0 0 15 on

Ens hauria bloquejat des del lock bit 0 fins al 15.

- flash erase 0 0 <nombre de seccions-1>

Figura 4.4.5. – *Telnet: Neteja flash*

Aquesta comanda esborra tots els 16 sectors (0-15) de la memòria flash. És útil en el cas de que quedés una part de la memòria gravada sense voler-ho i això provoqués un mal funcionament general en alguna aplicació.

- Reset

Figura 4.4.6. – *Telnet: Reset flash*

Atura el procés en l'estat actual. La comanda "reset", també pot ser complementada amb:

Reset ['run', 'halt', 'init', 'run_and_halt', 'run_and_init']

4.5. Reset manual a través del jumper.

A part dels lock bits, també hi ha el security bit. Aquest bit de seguretat, si està activat no permet gravar ni debugar sobre cap sector de la memòria de la placa. La única manera de desactivar-lo és a través d'un jumper.

Hem de fer un curtcircuit de dos pins amb l'ús d'un jumper, seguint aquests passos:

- Treure alimentació
- Introduir el jumper en els dos pins assenyalats a la figura
- Aplicar alimentació durant aproximadament vint segons
- Retirar l'alimentació
- Extreure el jumper

En la següent imatge, veiem els pins sobre els quals s'ha d'introduir el jumper:

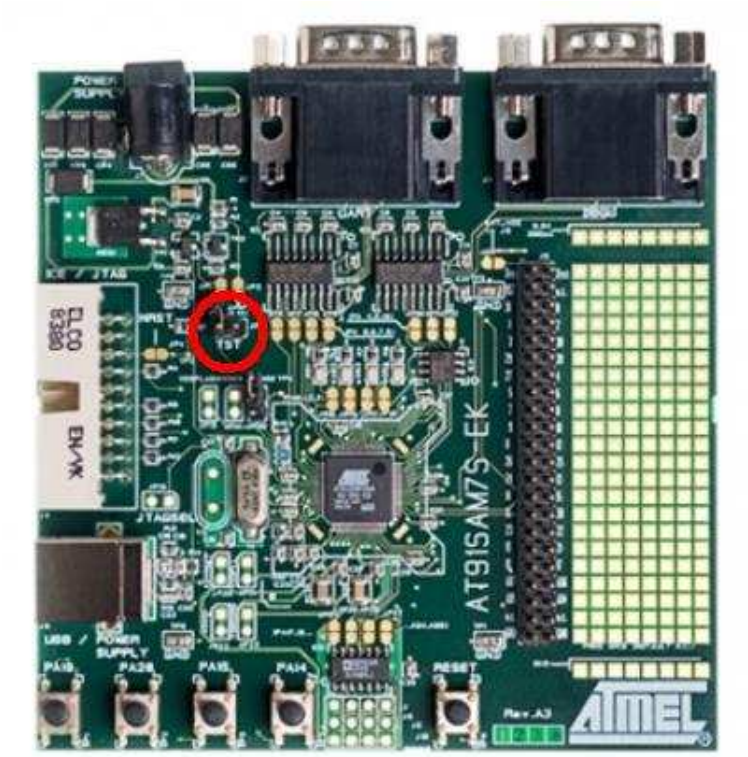

Figura 4.5.1. – *Reset manual memòria flash*

Cal dir que al fer aquest reset manual, a part de desbloquejar el security bit, també esborra tot el contingut de la memòria flash i els drivers per poder connectar-ho a l'ordinador. També bloqueja els lock bits 0 i 1.

CAPÍTOL 5: Entorn de treball

En els anteriors capítols hem anat perfilant el nostre projecte. A partir d'aquest capítol explicarem les eines necessàries per poder posar-nos a programar amb un sistema encastat en temps real i com es relacionen entre elles.

En tot moment treballarem amb eines open source (de codi obert) que ens permetran realitzar les nostres futures aplicacions sense cap mena d'impediment.

Al final del capítol, explicarem en forma de blocs el funcionament de totes les eines, les seves funcions i els protocols que utilitzen per comunicar-se. Veurem com es realitzen els processos de compilació i debugació d'una manera clara i entenedora.

Per tant, podem dir que aquest capítol és la introducció al que serà el cos del projecte, donat que hem anat definint i justificant l'elecció d'aquesta placa amb el seu determinat sistema operatiu i la metodologia de funcionament de les seves eines de software.

5. Entorn de treball:

En aquest apartat donarem a entendre quins són els requisits open source indispensables per al disseny i implementació de les nostres aplicacions.

5.1. YAGARTO, Eclipse, OpenOCD

Per a poder desenvolupar la nostra aplicació, cal conèixer les eines que necessitem. Bàsicament, farem servir tres eines Open Source per a poder fer la nostra aplicació, compilar-la i transferir-la a la placa:

- Compilador ARM GNU C/C++ (pack amb totes les eines necessàries per compilar)
- OpenOCD (programa per transferir a placa i debugar l'aplicació)
- Eclipse Integrated Development Environment (IDE) (entorn de programació)

Pel que fa a l'Eclipse i al OpenOCD farem servir les eines Open Source de YAGARTO, dissenyades expressament per aquest tipus d'aplicacions. En canvi, no farem servir el compilador de YAGARTO ja que és incomplet, en el seu lloc farem servir el pack del WinARM.

A partir d'aquestes eines, podrem crear diverses aplicacions, editar-les i veure'n el seu funcionament, tant a la placa com en el mode Debug que explicarem més endavant.

5.1.1. Compilador ARM:

Per a poder compilar, necessitarem una col·lecció d'eines GNU per la família ARM de processadors a l'entorn Windows anomenada: WinARM. Totes les eines necessàries estan incloses al pack de distribució.

5.1.2. OpenOCD:

El OpenOCD JTAG server que farem servir, és la solució open-source per transferir les nostres aplicacions i debugar-les a la placa. Permet el seu ús en processadors ARM7 o ARM9, entre d'altres.

Permet fer ús del debugador GNU GDB compilat per les arquitectures ARM. El OpenOCD és compatible amb qualsevol entorn IDE, ja sigui l'Eclipse, l'IAR o l'Emacs.

Per poder fer funcionar correctament aquest debugador JTAG, primer s'han d'instal·lar els seus drivers a l'ordinador.

5.1.3. Eclipse

Per a dissenyar les nostres aplicacions utilitzarem l'entorn de programació Eclipse el qual permet editar, compilar, linkar, descarregar i debugar. Aquest entorn resulta ser molt clar i entenedor, permetent d'una manera força visual veure les diferents aplicacions que tenim, els fitxers a compilar i transferir, com també una bona configuració del programa per transferir a placa (en aquest cas, el OpenOCD) i un bon menú de debugació.

Per a poder utilitzar aquest entorn, primer s'ha de tenir instal·lada la plataforma Java JRE.

5.2. Compilació:

El compilador C GNU de codi obert (open source) que fem servir proporciona una velocitat de processament de codi i rendiment molt semblant als millors compiladors professionals per a ARM, Keil, Hitex, IAR entre d'altres.

Aquest compilador i assemblador (versió ARM) estan preparats per compilar els diferents arxius. Les sortides del compilador i l'assemblador són arxius "object". Aquests arxius no contenen l'adreça de la memòria; d'això se n'encarrega posteriorment el linkador, permetent a l'usuari carregar l'aplicació al lloc desitjat de la memòria.

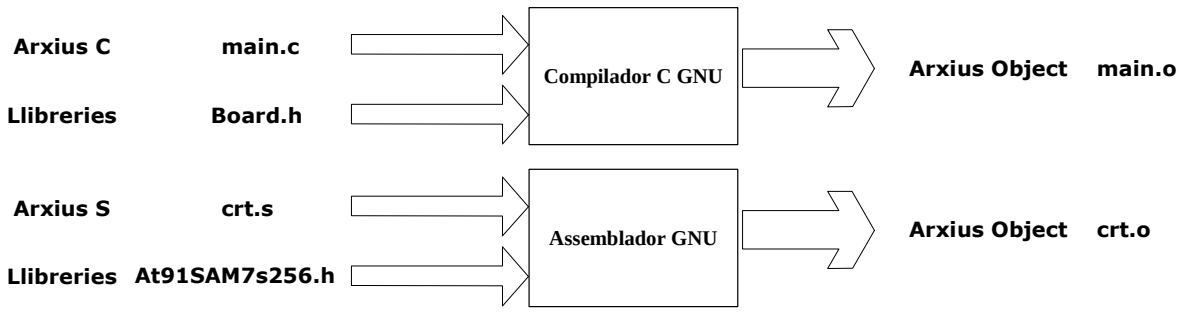

Figura 5.2.1. – *Compilador GNU*

Acte seguit, el linkador GNU és utilitzat per a recollir els arxius "object" creats anteriorment, més altres mòduls definits des de llibreries, en resol totes les adreces i ho combina tot a un sol fitxer de sortida descarregable d'extensió ".out".

Figura 5.2.2. – *Linkador GNU*

Per últim, si es vol gravar l'aplicació a la memòria Flash, es necessita un arxiu binari d'extensió ".bin" que pugui ser enviat a la memòria a través del debugador JTAG o la utilitat SAM-BA. Aquest arxiu és creat cridant la utilitat GNU "ObjCopy".

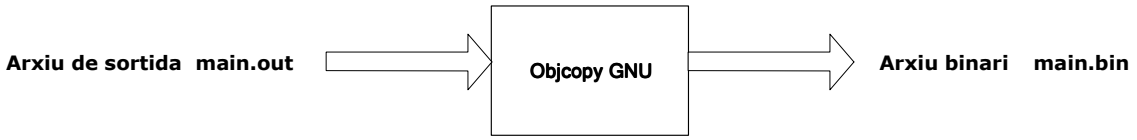

Figura 5.2.3. – *Objcopy GNU*

Totes les operacions descrites anteriorment es poden fer entrant comandes a una consola de Windows. Tanmateix, l'Eclipse automatitza tot aquest procés per agilitzar el temps de compilació de manera més eficient i rendible.

5.3. Debugació:

La debugació és complicada donat que la plataforma a debugar és una placa exterior separada del PC.

La unitat de debugació GNU GDB està completament integrada a l'entorn Eclipse per proporcionar una debugació animada amb breakpoints, execució pas per pas i una sofisticada inspecció de les variables i estructures de l'aplicació.

Quan es comença el debugador de l'Eclipse, s'executa automàticament un el debugador GNU GDB (arm-elf-gdb.exe). L'Eclipse es comunica amb aquest programa utilitzant el protocol GDB/MI.

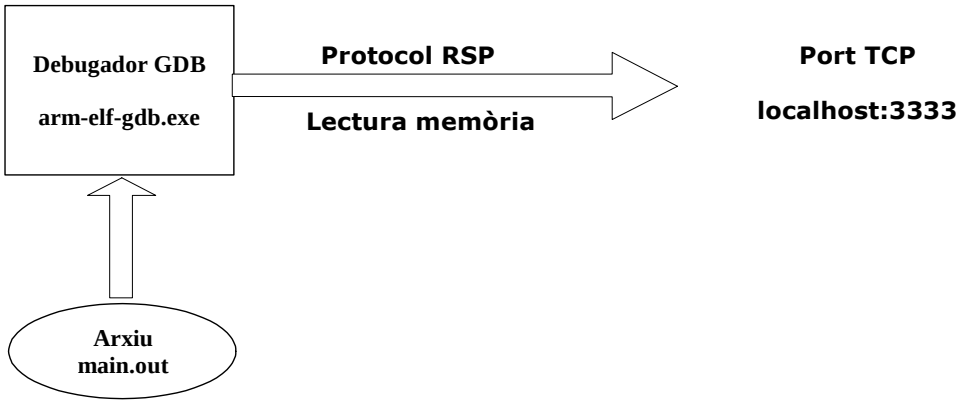

Figura 5.3.1. – *Connexió debugador*

Es necessita un servidor daemon (un programa que es va repetint de fons esperant comandes) per acceptar les comandes de debugació del protocol RSP, i convertir-les en comandes compatibles amb el xip ARM encastat de la placa.

Aquest servidor daemon (OpenOCD o J-Link GDB Server) ha d'anar abans de que comenci el GDB.

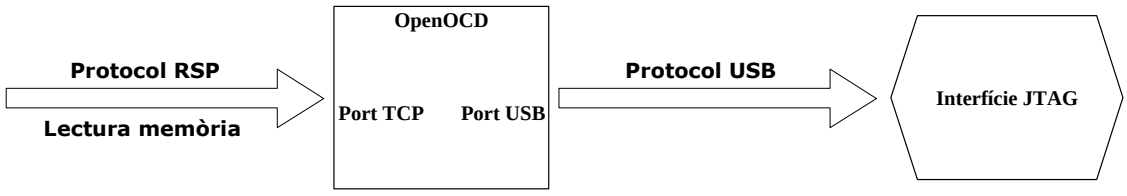

Figura 5.3.2. – *Connexió OpenOCD*

El següent diagrama de blocs mostra tot el procés de debugació. Començant per l'Eclipse, a través del GDB i el OpenOCD podem debugar la nostra placa a través del dispositiu JTAG. Els resultats són bidireccionals, és a dir, des de l'Eclipse es poden veure el valor de les variables, dels registres, llegir el contingut de la memòria, etc.

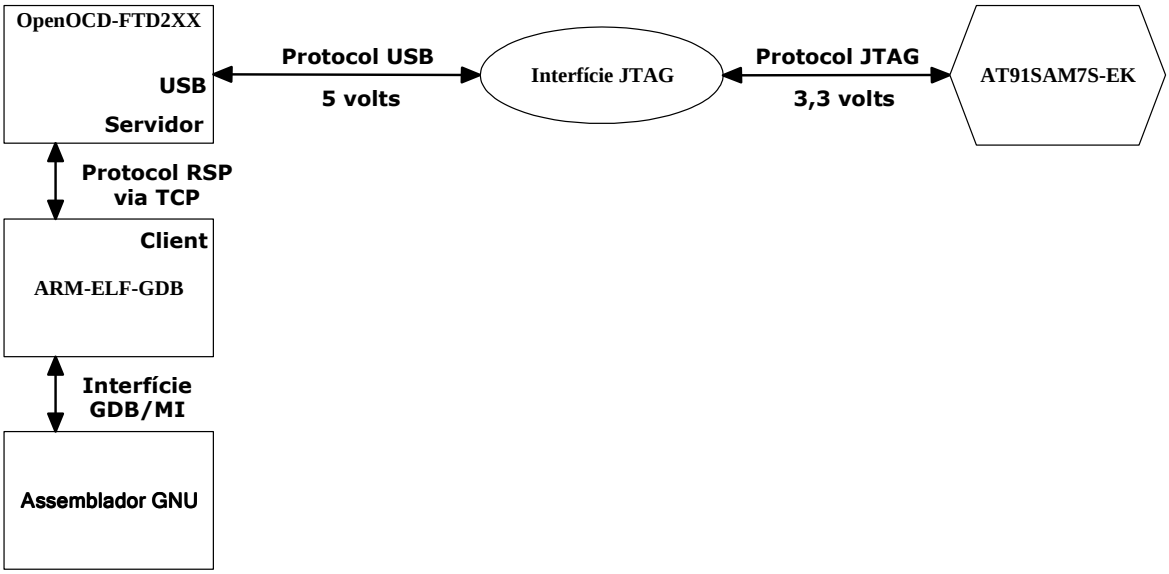

Figura 5.3.3. – *Diagrama complet*

El resultat de tota aquesta coordinació de software és un entorn amigable de debugació. Si per exemple posem el cursor sobre el nom d'una variable, l'Eclipse cridarà el debugador GDB demanant pel seu valor. A continuació el GDB sol·licitarà una lectura de memòria a l'adreça de memòria apropiada, veient finalment el resultat a l'entorn Eclipse.

CAPÍTOL 6: Sistema operatiu en temps real FreeRTOS

Un cop arribats en aquest punt, ja ens podem introduir en el nostre sistema operatiu en temps real. Es tracta del FreeRTOS, un S.O. totalment lliure que generalment utilitza tasques per a desenvolupar les seves aplicacions.

En aquest capítol explicarem com gestiona les tasques i les subrutines el FreeRTOS, tot fent un diagrama d'estats i definint com i on es creen les tasques i subrutines, tot i que generalment treballarem sobre tasques degut a que permeten més joc en les aplicacions.

Després veurem com està configurat el port sèrie de la placa per tal de poder enviar i rebre dades, veure-les a l'ordinador per tal de fer un bon seguiment de l'aplicació en tot moment, i per últim explicarem el procés de recuperació del programa SAM-BA.

També farem alguns exemples per donar a conèixer la utilitat de gestionar múltiples tasques i la forma en què es comuniquen entre elles (tenint en compte que són totalment independents l'una de l'altra).

A continuació explicarem com tracta els errors aquest sistema operatiu, per tal de poder controlar les errades del sistema.

També farem un llistat de les funcions API més comunes que permet, que poden servir de cara a futures aplicacions més completes que les que es tracten en aquest projecte.

Tractarem sobre altres mètodes d'organització de tasques (scheduling) i altres sistemes operatius en temps real. Veurem com i en què es basen per gestionar el processos i ens adonarem de que al canviar aquest procediment, es pot canviar de forma considerable l'estructura de funcionament de l'aplicació.

Per tant, podem afirmar que aquest capítol és el cos del projecte i que a partir d'aquestes bases hem de ser capaços d'implementar aplicacions en altres sistemes operatius en temps real similars.

6. Sistema operatiu en temps real FreeRTOS:

El sistema operatiu que farem servir per a les nostres aplicacions és el FreeRTOS. Es tracta d'un sistema operatiu en temps real adreçat als petits sistemes encastats en temps real, el qual utilitza el kernel per gestionar les diferents tasques. Atès que la majoria del codi ha estat escrit amb el llenguatge C de programació, aquest SOTR ha estat implementat en diferents plataformes. El fet de que la seva grandària sigui inferior a la majoria dels altres SOTR i que, en general, sigui senzill de fer anar, fa viable la seva aplicació a molts sistemes encastats.

Aquest SOTR de codi obert està dissenyat per ser simple, portable i concís.

Un ús típic d'aquest SOTR és quan sorgeix la necessitat d'estar comprovant periòdicament el valor d'un sensor o l'estat d'algun bit d'alta prioritat mentre de rerefons es va executant una tasca continuada de prioritat més baixa. D'aquesta manera, en tot moment s'estarà executant el procés de baixa prioritat i quan hi hagi una interrupció interna (timer) o externa (canvi de valor d'un pin) el SOTR ho detectarà i passarà a executar la tasca d'interrupció.

El FreeRTOS mesura el temps a partir de "ticks". Internament, crea una variable "TickCount" que sempre s'està incrementant a ritme de rellotge a dins d'un bucle infinit. Quan es vol fer ús d'un esdeveniment de temps (interrupció) o bloquejar una tasca, es fa servir el valor d'aquesta variable. Actualment, un retard de tasca de 1000 unitats amb la funció "vTaskDelay(1000)" provoca un retard d'un segon en temps real.

Aquest SOTR porta predefinides tota una sèrie de funcions que serveixen per facilitar el disseny de l'aplicació a desenvolupar.

6.1. Tasques:

El model més tradicional d'estructurar un sistema operatiu en temps real (SOTR) és amb l'ús de tasques independents. Cada tasca executa el seu propi procés sense cap mena de dependència amb altres tasques en el sistema o dins l'scheduler. L'scheduler és l'encarregat de decidir quina tasca executar en cada moment.

En les nostres aplicacions, es farà ús del SOTR anomenat FreeRTOS, el qual utilitza tasques independents per al seu desenvolupament. Les característiques més importants de les tasques i l'scheduler són:

- Només es pot executar una tasca en cada moment.
- Cada tasca és independent.
- L'scheduler ha de repartir el temps de processament entre les diferents tasques.
- L'scheduler ha de començar i finalitzar repetidament cada tasca tal com indiqui l'aplicació.
- Una tasca no té cap mena de coneixement de l'activitat de l'scheduler.
- L'scheduler té la responsabilitat de guardar l'estat actual d'una tasca al moment de rellevar-la per una altra, i de retornar-li el mateix valor quan es torni a fer servir.
- Cada tasca s'emmagatzema en el seu propi stack.

6.1.1. Estats de les tasques:

Les tasques poden tenir un dels següents estats:

- Execució:
  Quan una tasca s'està executant es diu que està en estat d'execució. Això significa que en aquest moment utilitza el processador.

- Disponible:
  Les tasques que poden ser executades (és a dir, no bloquejades o suspeses) s'anomenen disponibles. Actualment no s'executen perquè hi ha una altra tasca d'igual o superior prioritat en estat d'execució.

- Bloqueig:
  Una tasca està bloquejada quan està esperant un esdeveniment temporal o extern. Per exemple, si una tasca crida vTaskDelay() aquesta es bloquejarà fins que el temps d'espera hagi finalitzat. Les tasques també es poden bloquejar esperant en la cua (queue) i en esdeveniments de semàfor. Les tasques en estat de bloqueig sempre tenen un període de temps després del qual la tasca serà desbloquejada. Aquest estat de tasques no està disponible per a l'scheduler.

- Suspensió:
  Aquest altre estat de tasques tampoc està disponible per a l'scheduler. Les tasques entraran i sortiran en aquest estat només quan hi hagi una comanda explícita que ho sol·liciti. En aquest cas, la comanda d'entrada és: vTaskSuspend() i la de sortida xTaskResume(). En aquest estat no es pot definir un període de temps per poder finalitzar l'estat.

Cal dir que es poden crear tasques dins de tasques, i canviar l'estat d'una tasca a dins d'una altra o unes altres. Això permet un ampli ventall de possibilitats al dissenyar l'aplicació.

En la següent figura es mostra l'esquema de funcionament entre els diferents estats de les tasques:

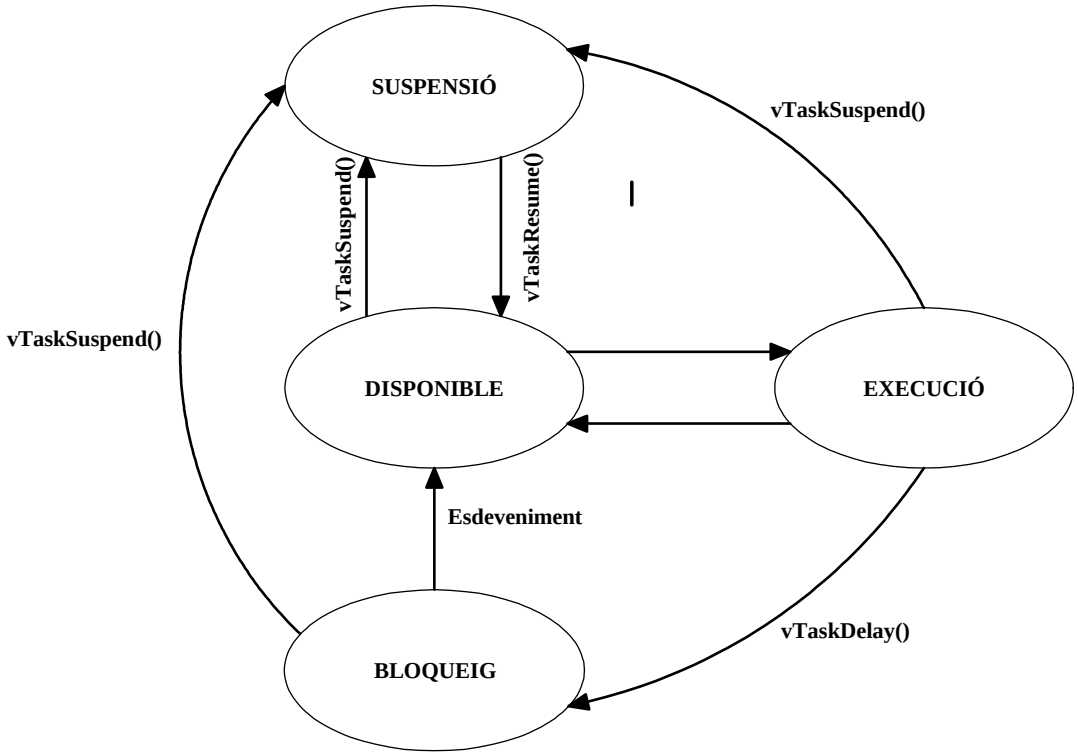

Figura 6.1.1.1. – *Estats de les tasques*

En aquest esquema gràfic es pot veure com, a partir de l'estat d'execució, es poden alternar les diferents tasques. Des de l'estat d'execució, veiem com una tasca que està actualment en procés només pot ser rellevada per una altra tasca que estigui en estat disponible i tingui més prioritat sobre ella. També, es pot donar el cas de que es cridi a la funció vTaskDelay() i llavors la tasca actual passa en estat de bloqueig durant un temps definit per l'usuari.

Des de qualsevol estat, si es crida la funció vTaskSuspend() es suspèn la tasca en qüestió i no torna a sortir-ne fins que es crida la funció vTaskResume().

Podem eliminar completament una tasca a partir de la funció vTaskDelete().

També és important adonar-se de que qualsevol tasca que no estigui en procés d'execució, i estigui capacitada per fer-ho, ha de passar forçosament per l'estat de disponibilitat.

6.1.2. Funcionament de les tasques:

El funcionament d'execució de tasques és simple. Només es pot executar una tasca en cada moment, però en un interval de temps qualsevol s'executen totes, en moments diferents. Donat que la velocitat del processador és elevada, no es pot apreciar a simple vista com va canviant entre tasques, donant la sensació que totes estiguin actives. La següent figura mostra aquesta peculiaritat:

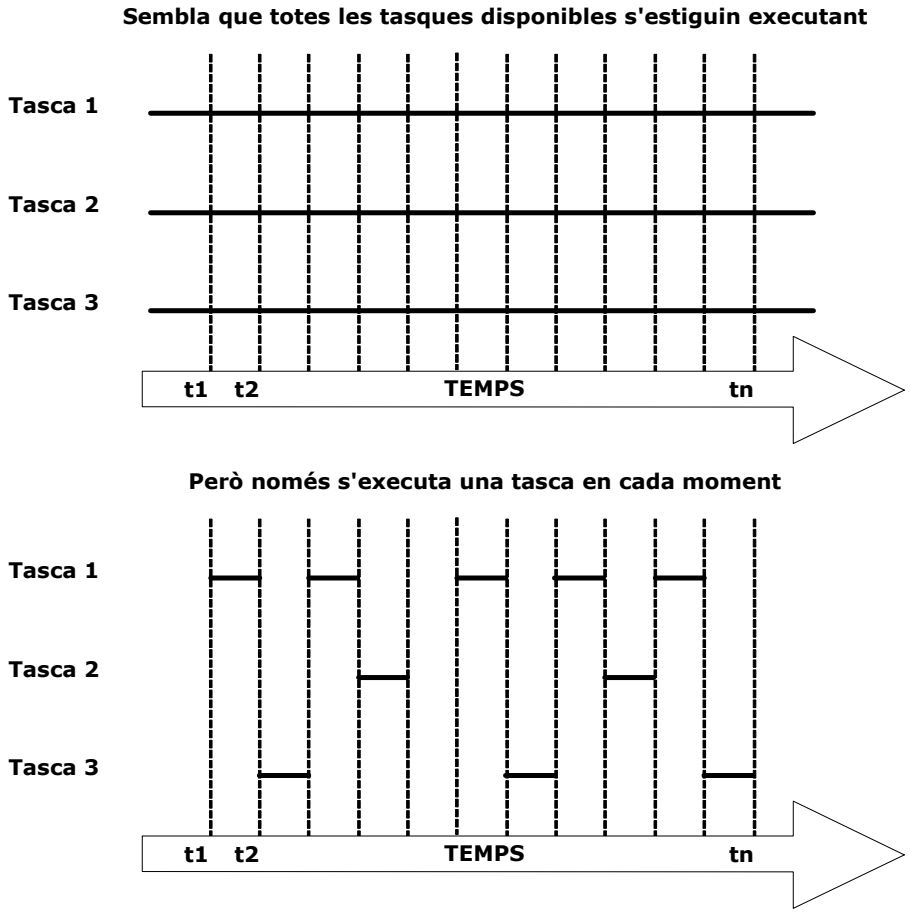

Figura 6.1.2.1. – *Funcionament de les tasques*

S'ha de tenir en compte que, perquè aquest esquema es compleixi, no podem posar una tasca d'interrupció (timer o interrupció externa per part de l'usuari) amb una prioritat més elevada que una tasca de rerefons, que es vulgui estar executant sempre. Si ho fem, aquesta tasca de rerefons només s'executarà un cop cada quan hi hagi una interrupció, i això no ens interessa.

Per al seu bon funcionament, la tasca de rerefons ha de tenir una prioritat més gran o igual que la de la interrupció.

6.1.3. Estructura de les tasques:

Quan definim una tasca, ha de tenir la següent estructura:

```
void Tasca ( void *pvParameters )
{
    for( ;; )
    {
        // Codi de l'aplicació.
    }
}
```

Quan creem una tasca, s'han de passar els següents paràmetres:

```
portBASE_TYPE    xTaskCreate(
                 pdTASK_CODE pvTaskCode,
                 const portCHAR * const pcName,
                 unsigned portSHORT usStackDepth,
                 void *pvParameters,
                 unsigned portBASE_TYPE uxPriority,
                 xTaskHandle *pvCreatedTask
                 );
```

On:

"pvTaskCode" Apuntador a la tasca. Les tasques han d'estar dissenyades per no retornar mai (bucle infinit).
"pcName" Nom descriptiu per a la tasca.
"usStackDepth" Tamany de l'stack de la tasca (nombre de variables que pot retenir).
" *pvParameters"* Apuntador que serà utilitzat com a paràmetre de la tasca a crear.
"*uxPriority"* Prioritat que li assignem a la tasca.
"*pvCreatedTask"* Utilitzat per passar un handle segons el qual la tasca creada es pugui referenciar.

6.1.4. Prioritat entre tasques:

Cada tasca té associada una prioritat que va de 0 a <configMAX_PRIORITIES - 1>. El valor de configMAX_PRIORITIES està definit al fitxer "FreeRTOSConfig.h" i pot ser editat en qualsevol moment. S'ha de tenir en compte que, quan més elevat sigui aquest nombre, més RAM consumirà el kernel del sistema operatiu (en aquest cas, el FreeRTOS).

Les tasques amb un nombre baix denoten que tenen baixa prioritat, basant-se en que la prioritat per defecte està definida per tskIDLE_PRIORITY amb el valor zero.

L'scheduler s'assegurarà que cada tasca en l'estat disponible tingui preferència en els temps de processament davant d'una altra tasca de prioritat inferior en el mateix estat. En altres paraules, la tasca a la que s'hi dediquin més recursos de processament sempre serà la tasca amb més prioritat per poder passar a l'estat d'execució.

6.1.5. Exemple multitasca:

En el següent exemple, veurem un dels grans avantatges que comporta l'ús de múltiples tasques dins de la nostra aplicació. Solucionarem el problema dels rebots, és a dir, farem que quan es produeixi una interrupció externa per part de l'usuari (es premi un polsador) el sistema només executi un cop la ordre corresponent, però permetent en tot moment l'ús d'altres interrupcions, si es produeixen.

Amb programació C, podem fer el següent esquema:

```
if ((pPIO->PIO_PDSR & SW1) == 0)                // Polsador 1 activat.
{
        // Activar sortida.
        while ((pPIO->PIO_PDSR & SW1) == 0);  // Espera a que el polsador 1 es desactivi.
}
if ((pPIO->PIO_PDSR & SW2) == 0)                // Polsador 2 activat.
{
        // Activar sortida.
        while ((pPIO->PIO_PDSR & SW2) == 0);  // Espera a que el polsador 2 es desactivi.
}
...
```

El problema és que si ho fem així i per exemple el polsador 1 es manté premut, l'aplicació mai activarà la interrupció del polsador 2 encara que aquesta es produeixi.

En canvi, amb un sistema multitasca podem fer el següent:

```
Creació_Tasca_Espera(...);
if ((pPIO->PIO_PDSR & SW1) == 0) // Polsador 1 activat.
{
        vTaskResume(Handle_Tasca_Espera);  // Reprenem la tasca.
}
if ((pPIO->PIO_PDSR & SW2) == 0) // Polsador 2 activat.
{
        vTaskResume(Handle_Tasca_Espera);  // Reprenem la tasca.
}
...
```

On "Creació_Tasca_Espera(...)" és:

```
for ( ;; )
{
        vTaskSuspend(Handle_Tasca_Espera);    // Suspenem la tasca perquè només s'executi
                                              // un cop per cicle.
        while ((pPIO->PIO_PDSR & SWn) == 0); // Espera a que el polsador n es desactivi.
}
```

D'aquesta manera, quan es produeixi una interrupció la tasca d'espera evitarà que es torni a executar la mateixa tasca, i a més no aturarà la tasca global d'interrupcions permetent continuar amb el seu procés.

6.2. Port Sèrie:

El FreeRTOS també porta incorporat la configuració del port sèrie a través del port RS232 UART de la placa. Aquest port, ens permet rebre (interrupció) i enviar informació bit a bit connectant-lo a ordinadors, microprocessadors i a tot tipus de perifèrics.

En el nostre cas, ens connectarem al port sèrie COMx mitjançant el programa "HyperTerminal" que porta incorporat Windows i un cable null-modem. D'aquesta manera podrem veure en tot moment el que ens retorna el port sèrie.

En el FreeRTOS, aquest port està configurat d'una manera molt simple d'utilitzar des del punt de vista de l'usuari. Un cop compilat el fitxer "simple_serial.c" que conté la velocitat Baud Rate a la que volem transmetre, i cridem la funció "uart0_init()", que té definits els pins d'entrada i sortida de dades (Tx i Rx), com també la velocitat del rellotge, els bits que volem transmetre, si volem o no el bit de paritat i quants "stop bits" volem, ja podem fer ús del port sèrie.

Per a fer-ho, podem utilitzar les següents funcions:

- uart0_putc();

  Envia un sol caràcter a través del port sèrie.

- uart0_puts();

  Envia un string a través del port sèrie. També permet enviar un vector de caràcters.

- uart0_kbhit();

  Funció que retorna verdader si hi ha un caràcter en el buffer receptor.

- uart0_getc();

  Llegeix un caràcter del port sèrie.

Per a que al efectuar les funcions "uart0_putc()" i "uart0_puts()" ens retorni automàticament el valor emmagatzemat al buffer receptor i veure'l a la nostra consola, necessitem una funció que vagi llegint constantment el port sèrie. D'això ja se n'ocupa el fitxer "syscalls.c" que, dins d'un bucle o loop infinit, va comprovant constantment si es rep un caràcter, i en aquest cas el transmeti.

6.3. Comunicació entre tasques:

Hi ha diferents maneres de comunicar-se entre tasques per poder enviar-se informació, missatges, etc. Aquestes són les cinc formes diferents de comunicació:

- Cues
- Semàfors binaris
- Semàfors comptadors
- Mutexes
- Mutexes recursius

6.3.1. Cues:

Les cues són la forma més bàsica de comunicació entre tasques. El seu funcionament és simple; donades dues tasques "A" i "B", on "A" és la tasca emissora i "B" la tasca receptora, cada vegada que la tasca "A" envia una dada aquesta s'afegeix a la cua d'un registre FIFO (First In First Out). Quan la tasca "B" crida una dada, rebrà la primera que s'hagi enviat.

Quan es crea una cua, es pot definir el tamany màxim de cada dada i el nombre de dades que permet. Normalment, les dades a enviar solen ser de poc tamany, en el cas de voler enviar dades de més tamany és preferible utilitzar apuntadors per referir-se a elles.

Les funcions API del FreeRTOS permeten definir un temps de bloqueig. Aquest temps és útil en aquest dos casos:

- Es vol fer lectura d'una cua buida. Aleshores aquest temps indica el nombre de "ticks" que ha d'esperar per tornar a fer la lectura.
- Es vol fer l'escriptura d'una cua plena. Aleshores aquest temps indica el nombre de "ticks" que ha d'esperar per tornar a fer l'escriptura.

Exemple:

En l'exemple que s'adjunta a continuació farem servir només una cua, tenint en compte que els primers tres polsadors enviaran els ítems (en aquest cas, strings) 1, 2 i 3 respectivament i el polsador 4 llegirà de la cua, si li és possible. La tasca "Enviar_Cua" farà d'emissora d'un valor del tipus unsigned portlong, i la tasca "Rebre_Cua farà de receptora.

Tasques d'enviament i recepció:

```
unsigned portLONG Val;

volatile AT91PS_PIO     pPIO = AT91C_BASE_PIOA;

xQueueHandle Cua;

void Enviar_Cua( void *pvParameters )
{
        bool Creada=false;
        int i=0;
        for ( ;; )
        {
                vTaskSuspend(Env);
                if (Creada!=true)
                {
                        Cua = xQueueCreate( 10, sizeof( unsigned portLONG ) );
                        Creada=true;
                }
                if ((pPIO->PIO_PDSR & SW1) == 0)
                        Val="Item 1";   // Enviarem l'ítem 1
                if ((pPIO->PIO_PDSR & SW2) == 0)
                        Val="Item 2";   // Enviarem l'ítem 2
                if ((pPIO->PIO_PDSR & SW3) == 0)
                        Val="Item 3";   // Enviarem l'ítem 3

                if( Cua != 0 )
                {
                        if( xQueueSend( Cua, ( void * ) &Val, ( portTickType ) 0 ) == pdPASS )
                        {
                                uart0_puts("Valor enviat a la cua: ");
                                uart0_puts(Val);
                                uart0_puts("\r\n");
                        }
                        else
                                uart0_puts("Cua plena o no disponible\r\n");
                }
                else
                        uart0_puts("Error: Cua no creada\r\n");
        }
}
```

```
void Rebre_Cua( void *pvParameters )
{
	for ( ;; )
	{
		vTaskSuspend(Reb);

		if( Cua != 0 )
		{
			if(xQueueReceive( Cua, ( void * ) &Val, ( portTickType ) 0 ) == pdPASS)
			{
				uart0_puts("Valor rebut de la cua: ");
				uart0_prints(Val);
				uart0_puts("\r\n");
				vParTestToggleLED(3);  // Només encendrà el LED4 si rep
				vTaskDelay(250);       // correctament de la cua
				vParTestToggleLED(3);
			}
			else
				uart0_puts("Cua buida o no disponible\r\n");
		}
		else
			uart0_puts("Error: Cua no creada\r\n");
	}
}
```

Descripció:

En la tasca “Enviar_Cua” primer creem la cua on emmagatzemar els ítems. Ho fem a dins d’un condicional per poder provocar l’error de lectura “Error: Cua no creada” de l’altra tasca.

Després de decidir quin ítem enviar, cridem a “xQueueSend” i si és possible afegim el primer ítem al final de la cua, a punt per ser el primer rebut. Per a fer això, passem com a primer paràmetre la cua que utilitzem i com a segon, un apuntador a la posició de memòria on es troba localitzada la variable. En aquest exemple no hem definit un “block time”, passant-li com a tercer paràmetre un 0 donat que ja suspenem la tasca a l’inici i la reprenem quan es prem algun dels primers tres polsadors.

En el cas de que la cua no s’hagi creat correctament, o hagi omplert el nombre màxim de posicions (en aquest cas 10), el port sèrie ens mostrarà l’error adient.

Després d’haver enviat almenys un ítem ja estem preparats per rebre’l amb l’altra tasca.

En la tasca de recepció “Rebre_Cua” seguim un procediment similar a l’anterior. Si la cua està correctament creada i es pot llegir, adquirirem els ítem en el mateix ordre en què han estat enviats. Tampoc hem definit un temps de bloqueig pel mateix motiu. En aquest exemple, hem activat durant un temps un led per saber, d’una manera visual, quan rebem un ítem. En el cas de que la cua no estigui creada, estigui buida o no disponible no s’encendrà.

Veure el posterior apartat de “Llistat de funcions API” per a saber més funcions per utilitzar en les cues.

6.3.2. Semàfors:

Una altra forma de comunicació són els semàfors. Aquest altre mètode està pensat per sincronitzar una tasca amb una altra tasca o amb una interrupció, de manera que quan es produeixi un esdeveniment a la rutina d'interrupció, la tasca estigui habilitada per fer una determinada funció.

El funcionament és simple; quan creem un semàfor l'hem "d'agafar" amb la funció "xSemaphoreTake()" per tal de bloquejar l'accés i no permetre que una altra tasca el pugui agafar. Per la seva banda, una tasca que vulgui accedir a un determinat recurs no ho podrà fer fins que el semàfor no estigui disponible. Quan es produeixi una interrupció, es cridarà la funció "xSemaphoreGive()", alliberant-lo i permetent que pugui ser "agafat" per aquesta tasca. De seguida que passi això, la tasca que volia accedir-hi automàticament l'agafarà i realitzarà el recurs. Un cop realitzat, la tasca ha de tornar a alliberar el semàfor.

Nosaltres farem servir el semàfor binari, el qual conté només una cua de llargada 1. En canvi els semàfors comptadors contenen una cua de més llargada (màxim recomanat 10), cosa que els permet tenir un funcionament similar a 10 semàfors binaris. Cada vegada que una tasca agafa el semàfor, es decrementa el seu comptador i cada cop que l'allibera, s'incrementa.

En el cas dels semàfors i dels mutexes que veurem a continuació, hem de tenir present que el seu contingut no ens interessa, només hem de saber si està disponible o no.

Exemple:

En el següent exemple veurem el funcionament bàsic d'un semàfor binari. Després de crear i agafar el semàfor a la mateixa funció, intentarem accedir a ell a través d'una tasca per a poder realitzar una determinada acció. Veurem com, després de que el semàfor s'hagi alliberat per la tasca, tornarà a ser agafat per una altra tasca sense que aquesta executi cap acció, només el tingui per bloquejar l'accés a la tasca fins que es torni a produir una altra interrupció o esdeveniment de temps.

Funcions de creació de semàfor, tasca que hi intenta accedir i tasca d'interrupció:

```
volatile AT91PS_PIO     pPIO = AT91C_BASE_PIOA;

xSemaphoreHandle xSemafor;

void Creacio_Semafor ( void )
{
        vSemaphoreCreateBinary( xSemafor );
        if( xSemafor != NULL )
        {
                uart0_puts("Semafor creat\r\n");
                xSemaphoreTake( xSemafor, 0 );
        }
   else
        uart0_puts("Error: Semafor no creat\r\n");
}

void Tasca ( void * pvParameters )
{
        for ( ;; )
        {
                if( xSemafor != NULL )
                {
                        if( xSemaphoreTake( xSemafor, ( portTickType ) 5000 ) == pdTRUE )
                        {
                                uart0_puts("S'ha entrat correctament al recurs\r\n");

                                        // Acció a realitzar

                                xSemaphoreGive(xSemafor);
                                uart0_puts("S'ha sortit del recurs\r\n");
                        }
                        else
                                uart0_puts("No s'ha pogut accedir al recurs\r\n");

                }
        }
}
```

```
void Int ( void * pvParameters )
{
        for ( ;; )
        {
                if ((pPIO->PIO_PDSR & SW1) == 0)
                        xSemaphoreGive( xSemafor );
                xSemaphoreTake( xSemafor, 0 );
        }
}
```

Descripció:

Tal i com hem dit abans, la tasca intentarà accedir al semàfor. En aquest cas, i a diferència de la cua, farem que la tasca es bloquegi un determinat temps quan fa la crida a "xSemaphoreTake()" i vagi mostrant periòdicament un missatge d'error en el cas de que el semàfor no estigui creat o disponible.

En el moment que creem el semàfor a la funció "Creacio_Semafor()" automàticament l'agafem i evitem que es pugui realitzar la tasca. Quan es produeixi la interrupció, detectada per la tasca "Int", la tasca "Tasca" podrà accedir al recurs que estava esperant i un cop aquest finalitzat l'ha d'alliberar, cosa que provoca que l'agafi la tasca "Int" que mantindrà el semàfor ocupat.

D'aquesta manera, ens podem assegurar que es sincronitzi correctament una tasca i una interrupció i que només es realitzi cada cop que es produeixi una interrupció.

6.3.3. Mutexes:

Els mutexes són una variant dels semàfors i de funcionament semblant. Aquests tenen la particularitat de que tenen definit un sistema de prioritats, de manera que si el "mutex" està agafat per una tasca de baixa prioritat i intenta agafar-lo una altra tasca de prioritat més gran, aquesta l'obté i la que el tenia inicialment es queda bloquejada.

Aquests són els mutexes sobre els que farem l'exemple, però també hi ha els mutexes recursius, els quals es poden agafar repetides vegades amb "xSemaphoreTakeRecursive()" per la mateixa tasca, de manera que per poder ser agafat per una altra tasca abans, obligatòriament, s'ha d'alliberar amb "xSemaphoreTakeRecursive()" el mateix nombre de vegades.

Només les versions més recents del FreeRTOS suporten els semàfors comptadors i aquests dos tipus de mutexes.

Exemple:

En el següent exemple crearem un mutex i accedirem a ell a través de la tasca "A". Al seu torn, provocarem que la tasca "B" de prioritat més gran que "A" agafi el semàfor prèviament adquirit per "A". De la mateixa manera que amb el semàfor binari, si cap de les dues demana el semàfor aquest serà agafat per la tasca "Int".

Funció Creacio_Mutex, TascaA, TascaB i tasca Int:

```
volatile AT91PS_PIO      pPIO = AT91C_BASE_PIOA;

xSemaphoreHandle Mutex;

void Creacio_Mutex ( void )
{
        Mutex=xSemaphoreCreateMutex();

        if( Mutex != NULL )
        {
                uart0_puts("Mutex creat\r\n");
                xSemaphoreTake( Mutex, 0 );
        }
   else
        uart0_puts("Error: Mutex no creat\r\n");
}

void TascaA ( void * pvParameters )
{
        for ( ;; )
        {

                vTaskSuspend(Ta);

                uart0_puts("Inici Tasca A\r\n");

                if( Mutex != NULL )
                {
                        if( xSemaphoreTake( Mutex, ( portTickType ) 5000 ) == pdTRUE )
                        {
                                uart0_puts("Recurs compartit\r\n");

                                xSemaphoreGive(Mutex);
                        }
                        else
                                uart0_puts("No s'ha pogut accedir al recurs compartit\r\n");

                }
                else
                        uart0_puts("No s'ha pogut accedir al recurs compartit\r\n");
                uart0_puts("Fi Tasca A\r\n");
        }
}
```

```
void TascaB ( void * pvParameters )
{
	for ( ;; )
	{
		vTaskSuspend(Tb);

		uart0_puts("Inici Tasca B\r\n");

		if( Mutex != NULL )
		{
			if( xSemaphoreTake( Mutex, ( portTickType ) 5000 ) == pdTRUE )
			{
				uart0_puts("Recurs compartit\r\n");

				xSemaphoreGive(Mutex);
			}
			else
				uart0_puts("No s'ha pogut accedir al recurs compartit\r\n");

		}
		else
			uart0_puts("No s'ha pogut accedir al recurs compartit\r\n");
		uart0_puts("Fi Tasca B\r\n");
	}
}

void Int ( void * pvParameters )
{
	for( ;; )
	{
		if ((pPIO->PIO_PDSR & SW1) == 0)
			vTaskResume(Ta);
		if ((pPIO->PIO_PDSR & SW2) == 0)
			vTaskResume(Tb);
		if ((pPIO->PIO_PDSR & SW3) == 0)
			xSemaphoreGive(Mutex);
		xSemaphoreTake( Mutex, 0 );
	}
}
```

Descripció:

Després de crear el semàfor, premem el botó 1 per reprendre la tasca A. Al no poder accedir al mutex, premem el botó 3 i l'alliberem, d'aquesta manera quan tornem a prémer el botó 1 la tasca A podrà accedir-hi. Un cop a dins del recurs compartit, i abans no torni a alliberar el mutex, fem que la tasca B es reprengui. D'aquesta manera provoquem que la tasca A es bloquegi i es passi a executar la B.

6.4. Subrutines:

En les nostres aplicacions també podem fer ús de les subrutines. Aquestes tenen una estructura molt similar a la de les tasques però tenen el seu propi scheduler.

6.4.1. Estats de les subrutines:

Les subrutines poden tenir un dels següents estats:

- Execució:
  Quan una subrutina s'està executant es diu que està en estat d'execució. Això significa que en aquest moment utilitza el processador.

- Disponible:
  Les subrutines que poden ser executades (és a dir, no bloquejades) s'anomenen disponibles. Actualment no s'executen perquè hi ha una altra subrutina d'igual o superior prioritat en estat d'execució o bé hi ha una tasca en estat d'execució de prioritat més gran que tskIDLE_PRIORITY.

- Bloqueig:
  Una subrutina està bloquejada quan està esperant un esdeveniment temporal o extern. Per a bloquejar una subrutina, s'ha de cridar a la funció crDELAY(). Les subrutines en estat de bloqueig sempre tenen un període de temps després del qual seran desbloquejades. Aquest estat de tasques no està disponible per a l'scheduler.

El seu esquema de funcionament és el següent:

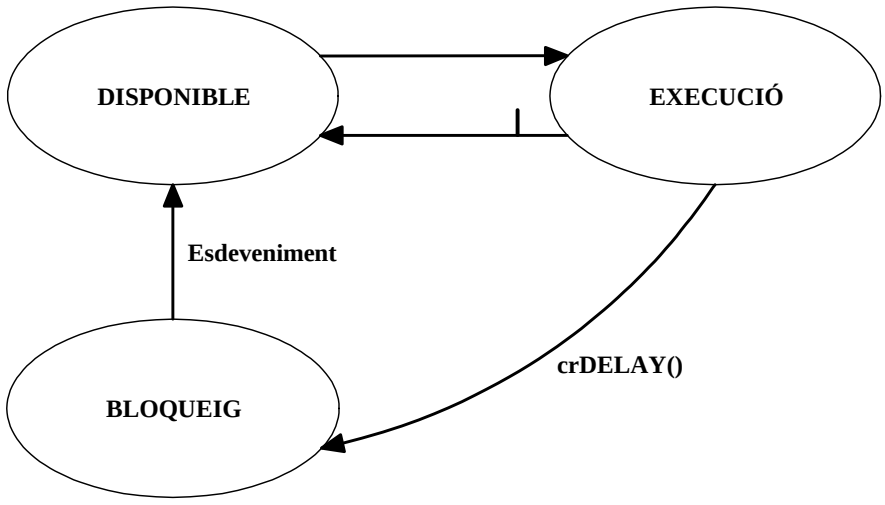

Figura 6.4.1.1. – *Estats de les subrutines*

Tal i com es pot veure, és un esquema més senzill que el de les tasques donat que no hi ha l'estat de suspensió.

6.4.2. Estructura de les subrutines:

Quan creem una subrutina, s'han de passar els següents paràmetres:

```
portBASE_TYPE xCoRoutineCreate(
                      crCOROUTINE_CODE pxCoRoutineCode,
                      unsigned portBASE_TYPE uxPriority,
                      unsigned portBASE_TYPE uxIndex
                     );
```

On:
"pxCoRoutineCode" Nom de la subrutina a crear.
"uxPriority" Prioritat que li assignem.
"uxIndex" Variable que li podem passar a la funció.

Quan definim una subrutina, ha de tenir la següent estructura:

```
void Ex_Subrutina( xCoRoutineHandle xHandle, unsigned portBASE_TYPE uxIndex )
{
        crSTART( xHandle );
        for( ;; )
        {
                // Codi de l'aplicació.
        }
        crEND();
}
```

És obligatori que cada subrutina comenci amb crSTART() i acabi amb crEND(). S'ha de tenir en compte que no es poden bloquejar subrutines externes, només la pròpia.

L'scheduler de les subrutines és diferent que el de les tasques. Aquest s'anomena vCoRoutineSchedule() i s'ha d'estar executant contínuament per tal de que les subrutines funcionin correctament. El millor lloc per situar-lo és a dins de la funció vApplicationIdleHook(), la qual sempre que pot s'executa (amb prioritat zero) com si es tractés d'una tasca de rerefons. Aquesta tasca és creada automàticament per l'scheduler un cop aquest és iniciat, però per poder fer-ne ús s'ha de posar a "1" el paràmetre "configUSE_IDLE_HOOK" en el fitxer "FreeRTOSConfig.h":

```
void vApplicationIdleHook( void )
{
        for( ;; )
        {
                vCoRoutineSchedule( void );
        }
}
```

6.4.3. Exemple d'una subrutina:

El següent exemple crea quatre subrutines que activen repetidament els quatre leds durant un petit període de temps, a una freqüència determinada, a través del paràmetre "uxIndex":

A dins del main posem:

```
for (int i=0;i<4;i++)
        xCoRoutineCreate( Ex_4LEDS, 1, i );
```

A dins d'un altre fitxer a compilar hem de posar:

```
void Ex_4LEDS( xCoRoutineHandle xHandle, unsigned portBASE_TYPE uxIndex )
{

crSTART( xHandle );

    int Freq[4]={ 50, 100, 200, 400 };

    for( ;; )
    {
        vParTestToggleLED(uxIndex );
        crDELAY( xHandle, Freq[ uxIndex ] );
    }

    crEND();
}
```

D'aquesta manera els quatre leds s'aniran encenent i apagant a la freqüència que decidim.

És possible l'ús simultani de subrutines i tasques, però sempre una tasca de prioritat més gran que 0 tindrà més prioritat que qualsevol subrutina. En aquest estat de concurrència, el comportament de les subrutines anirà molt condicionat per les tasques així que, en la mesura del possible, s'ha de procurar utilitzar el mínim nombre de subrutines possible.

Certament es poden crear subrutines dins d'altres subrutines (igual que amb les tasques) però no tasques dins de subrutines.

6.5. SAM-BA:

El SAM-BA (SAM Boot Assistant) és un programa que, a través d'un sistema de recuperació, permet restaurar la memòria flash de la placa tornant-la al seu estat inicial. Aquest procediment és molt útil en el cas de tenir una mala configuració de la memòria flash i no saber com recuperar-la.

Aquest procediment també és efectiu en el cas de que el security bit estigui activat.

Un cop instal·lat el programa al PC, el procediment és el següent:

1) Desconnectar la placa de l'alimentació.
2) Posar el jumper al pin ERASE (JP28).
3) Alimentar la placa durant, almenys, 50ms.
4) Desconnectar la placa de l'alimentació, treure el jumper del pin ERASE.
5) Fer el mateix amb el pin TST (JP5), alimentant la placa durant, almenys, 10 segons.
6) Alimentar la placa.
7) Prémer reset.

En el cas de que no hi hagi cap problema a la memòria flash i senzillament es vulgui accedir a la placa a través del SAM-BA, només cal fer els passos 5 i 6.

Al fer aquest procés, a la memòria flash s'hi carrega l'aplicació SAM-BA i els lock bits 0, 1 queden automàticament activats. Un cop fet això, podem executar el SAM-BA des de l'Eclipse a través del port USB (\usb\ARM0) o del port sèrie (COMx).

Cal tenir en compte que, degut a un conflicte de prioritats, no es pot utilitzar correctament el port sèrie alimentant la placa a través de l'USB, sinó que s'ha de fer a través d'una alimentació externa (7-12Vdc). En el cas de voler transferir un programa a través del port sèrie, s'ha de fer servir el port DBGU de la placa.

Un cop a dins del programa, podem gravar a la memòria flash un fitxer "*.bin" o extreure'n l'actual sense haver d'utilitzar el debugador JTAG ni el OpenOCD.

Una altra de les seves utilitats és que ens permet esborrar el contingut de la memòria flash, com també l'ús del protocol TCL a través de la seva línia de comandes:

Figura 6.5.1. – *Comunicació SAM-BA*

Per tant, podem concloure que gravar a través del SAM-BA és molt útil en el cas de no disposar de debugador, i només puguem utilitzar el port USB o el port sèrie (amb alimentació externa), com també activar el security bit i enviar comandes a través del protocol TCL. En canvi, el seu principal inconvenient és que per gravar cada fitxer "*.bin" s'han de fer els passos 5, 6 i 7.

6.6. Errors:

El sistema operatiu FreeRTOS també té definit un mètode per controlar els errors. Així doncs, quan es produeix un error (ja sigui enviant un ítem a una cua plena, que una funció no retorni el valor esperat, etc), el sistema executa automàticament una tasca d'error que, en aquest cas, ens encén i apaga el led 4 a una determinada freqüència apagant la resta de leds. Un cop entrats en aquesta tasca, que li hem d'assignar la prioritat més alta possible, només en podrem sortir fent un reset. El codi de la tasca és el següent:

```
for ( ;; )
{
	// Temps de retard que ens defineix el temps en què ha de pampalluguejar.
	vTaskDelay( xDelayPeriod );
	// Comprova que cap de les tasques en execució no tingui cap error.
		if( prvCheckOtherTasksAreStillRunning() != pdPASS )
		{
			// Error detectat.
			pPIO2->PIO_SODR = LED_MASK; // Apaga tots els leds.
			vParTestToggleLED( mainCHECK_TASK_LED ); // Encén el led 4.
		}
}
```

Podem veure que, un cop creada aquesta tasca, no interferirà en cap procés de cap altra tasca mentre no hi hagi cap error. Quan se'n produeixi un, al ser aquesta la tasca amb la prioritat més alta, romandrà aquí fins que es premi reset.

Per la seva banda, la funció prvCheckOtherTasksAreStillRunning() fa el següent:

```
{
	portLONG lReturn = ( portLONG ) pdPASS;

	// Comprova totes les tasques per verificar que no hi hagi cap error. Si se'n produís un,
	automàticament la funció retornaria pdFAIL.

	if( xAreIntegerMathsTaskStillRunning() != pdTRUE )
	{
		lReturn = ( portLONG ) pdFAIL; // Error detectat.
	}

	if( xArePollingQueuesStillRunning() != pdTRUE )
	{
		lReturn = ( portLONG ) pdFAIL; // Error detectat.
	}
	...
	if( xAreDynamicPriorityTasksStillRunning() != pdTRUE )
	{
		lReturn = ( portLONG ) pdPASS; // Aquest cas no es considera un error donat
					que es produeix quan es crida vTaskSuspend();
	}
	return lReturn;
}
```

6.7. Llistat funcions API:

En aquest altre apartat numerarem les funcions API més importants que permet utilitzar el FreeRTOS, com també una breu descripció del seu significat.

- Creació de tasques:
  - portBASE_TYPE xTaskCreate();

    Crea una tasca i l'afegeix a la cua de tasques disponibles.
  - void vTaskDelete();

    Elimina completament una tasca, estigui a l'estat que estigui. Allibera memòria.
- Control de tasques:
  - void vTaskDelay();

    Bloqueja una tasca durant els "ticks" que defineixi l'usuari.
  - void vTaskDelayUntil();

    Especifica un temps absolut (exacte) durant el qual la tasca es bloqueja. Aquesta funció difereix de vTaskDelay() en que, al poder definir la freqüència, el temps és més precís.
  - unsigned portBASE_TYPE uxTaskPriorityGet();

    Obté la prioritat de qualsevol tasca.
  - void vTaskPrioritySet();

    Estableix la prioritat de qualsevol tasca.
  - void vTaskSuspend();

    Suspèn qualsevol tasca. Un cop suspesa, en cap cas consumeix processador.
  - void vTaskResume();

    Reprèn una tasca prèviament suspesa.
- Utilitats de les tasques:
  - xTaskHandle xTaskGetCurrentTaskHandle();

    Retorna el handle de la tasca que s'estigui executant en aquell moment.
  - volatile portTickType xTaskGetTickCount();

    Retorna el nombre de ticks des de que l'scheduler ha començat.

- portBASE_TYPE xTaskGetSchedulerState();

Pot retornar una de les següents tres opcions:

- taskSCHEDULER_NOT_STARTED: Scheduler no iniciat.
- taskSCHEDULER_RUNNING : Scheduler actualment en procés.
- taskSCHEDULER_SUSPENDED: Scheduler suspès.

- unsigned portBASE_TYPE uxGetNumberOfTasks();

Funció que retorna el nombre total de tasques bloquejades, suspeses i en execució. Pot ser que també compti una tasca eliminada però que encara no s'hagi alliberat de la memòria.

➢ Control del kernel:

- void vTaskStartScheduler();

Inicia el procés de control de totes les tasques creades.

- void vTaskEndScheduler();

Atura el procés de l'scheduling, eliminant totes les tasques creades i parant el procés multitasca.

- void vTaskSuspendAll();

Suspèn totes les tasques, sense eliminar-les. Manté el comptatge del tick.

- void xTaskResumeAll();

Reprèn totes les tasques suspeses anteriorment per vTaskSuspendAll();

➢ Cues:

- unsigned portBASE_TYPE uxQueueMessagesWaiting();

Retorna el nombre de missatges emmagatzemats a la cua.

- xQueueHandle xQueueCreate();

Crea una nova cua. També retorna el handle de la cua creada.

- void vQueueDelete();

Elimina completament una cua, juntament amb el seu contingut.

- portBASE_TYPE xQueueSend();

Envia un ítem al final de la cua. Enviament per còpia, no per referència.

- portBASE_TYPE xQueueSendToBack();

Funció equivalent a l'anterior.

- portBASE_TYPE xQueueSendToFront();

Envia un ítem a l'inici de la cua. Enviament per còpia, no per referència.

- portBASE_TYPE xQueueReceive();

Rep un ítem de la cua. Recepció per còpia.

- portBASE_TYPE xQueuePeek();

Rep un ítem de la cua sense eliminar-lo de la mateixa. Recepció per còpia.

- Semàfors/Mutexes:

- vSemaphoreCreateBinary();

Crea un semàfor binari utilitzant el mecanisme de les cues. La llargada de la cua és un.

- xSemaphoreHandle xSemaphoreCreateCounting();

Crea un semàfor comptador utilitzant el mecanisme de les cues.

- xSemaphoreHandle xSemaphoreCreateMutex();

Crea un semàfor mutex utilitzant el mecanisme de les cues.

- xSemaphoreHandle xSemaphoreCreateRecursiveMutex();

Crea un semàfor mutex recursiu utilitzant el mecanisme de les cues.

- xSemaphoreTake();

Obté un semàfor prèviament creat.

- xSemaphoreGive();

Allibera un semàfor prèviament creat i obtingut.

- xSemaphoreTakeRecursive();

Obté un semàfor de tipus mutex prèviament creat.

- xSemaphoreGiveRecursive();

Allibera un semàfor de tipus mutex prèviament creat i obtingut.

- Subrutines:

- portBASE_TYPE xCoRoutineCreate();

Crea una nova subrutina i l'afegeix a la llista de subrutines disponibles.

- void crDELAY();

Retarda una subrutina el temps definit per l'usuari.

- void vCoRoutineScheduler();

Controla i gestiona les subrutines amb el mètode priority-based.

6.8. Altres schedulers i sistemes operatius en temps real:

En aquest apartat numerarem els diferents tipus d'schedulers que hi ha, inclòs el que utilitza el FreeRTOS, per tal de veure diverses maneres de gestionar les tasques.

6.8.1. Scheduler:

El FreeRTOS té un simple scheduler basat en prioritats. Aquest scheduler fa que s'executi la tasca amb la prioritat més alta. Això s'aconsegueix mitjançant el TickCounter, el qual decideix si la tasca actual ha de ser canviada per una altra. Per a fer això, es requereix que a cada tasca se li assigni una prioritat, podent així executar la que la tingui més alta. Les tasques amb la mateixa prioritat s'executaran per ordre racional (mètode Round-Robin).

6.8.2. Tipus d'schedulers:

A continuació veurem de manera resumida els principals mètodes d'scheduling que existeixen:

- First-Come-First-Served (FCFS)

  Aquest scheduling té un simple mètode de funcionament molt similar als registres FIFO. S'executen les tasques per l'ordre en què han estat creades.

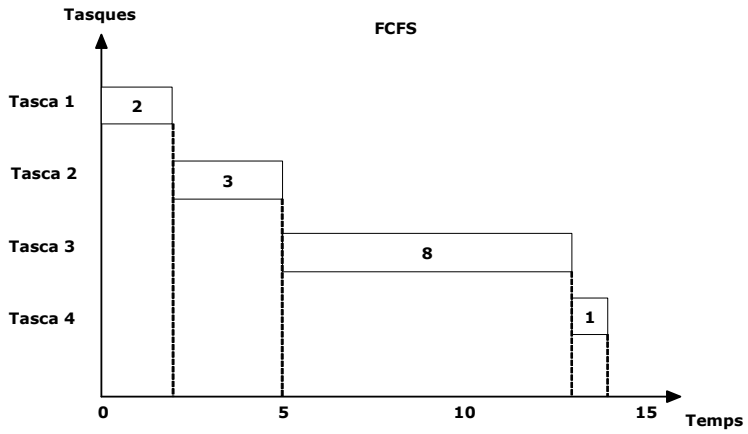

Figura 6.8.2.1. – *FCFS*

- Shortest Job First (SJF)

  Aquest altre scheduler intenta executar les tasques per ordre creixent de durada. La seva major dificultat és saber estimar la duració futura de cada tasca.

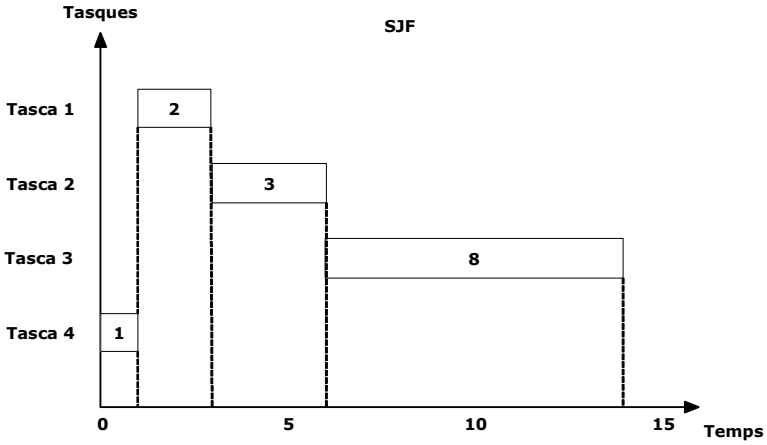

Figura 6.8.2.2. – *SJF*

- Round-Robin (RR)

  Es tracta d'un mètode simple de planificació de tasques. Organitza de manera equitativa i racional les diferents tasques, i les executa en conseqüència en cada cicle.

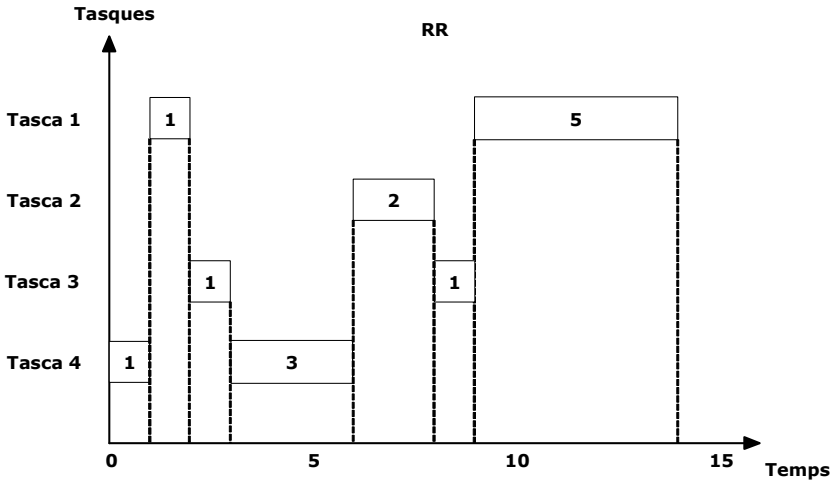

Figura 6.8.2.3. – *Round-Robin*

- Priority-based

  Sistema de planificació de tasques basat en prioritats. És el que utilitza el FreeRTOS.

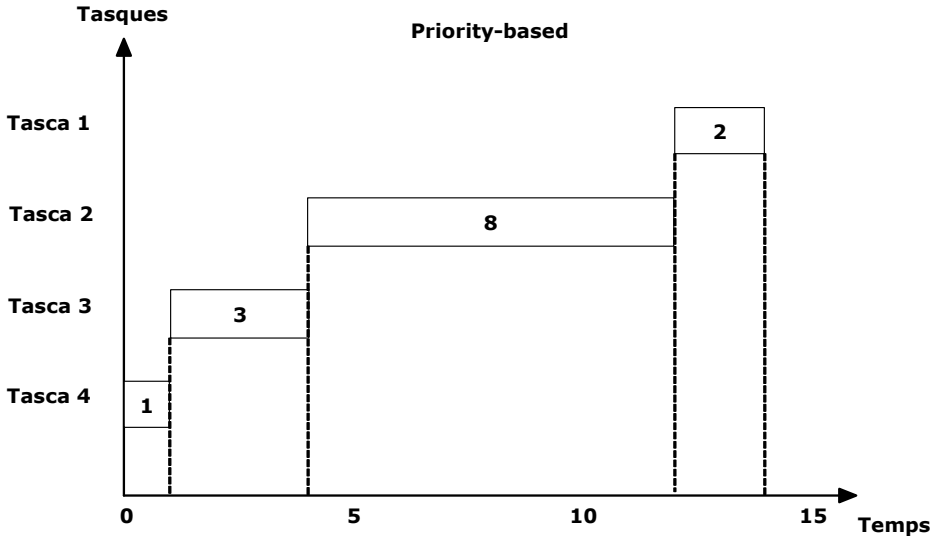

Figura 6.8.2.4. – *Priority-based*

- Bitmap

  Aquest mètode d'scheduling és molt similar a l'anterior. Consisteix en una cua de threads, on cada thread té associada una prioritat. Hi ha 32 nivells diferents de prioritats disponibles i cada nivell només pot ser associat a un thread. El nivell de proritat més alt és el 0, i el més baix el 32. Això limita al nombre total de threads a 31 (un és el thread idle). L'scheduler sempre executa el thread amb la prioritat més alta. Si entra un thread a la cua de prioritat més alta que el que s'està executant, aquest passa a executar-se rellevant l'anterior.

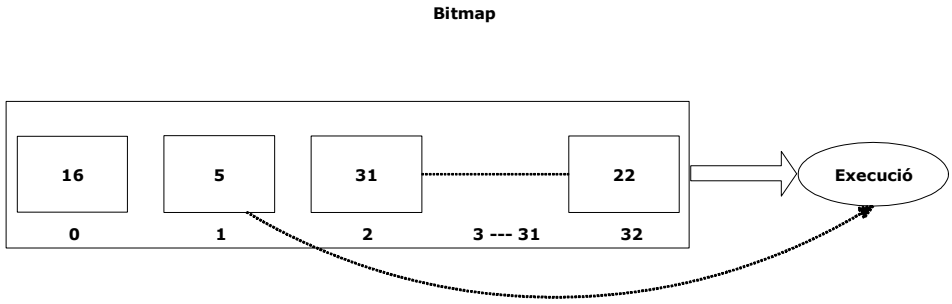

Figura 6.8.2.5. – *Bitmap*

- Multi-level queue (MLQ)

  Aquest tipus d'scheduler és més sofisticat i permet separar els diferents sistemes de treball en estat disponible i aplicar els algorismes d'scheduling més adients en cada cas. El diferencia del bitmap en el fet de que utilitza un conjunt de cues, on cada cua conté un nombre definit de threads, tots amb la mateixa prioritat. Cada cua té el seu propi algorisme d'scheduling.

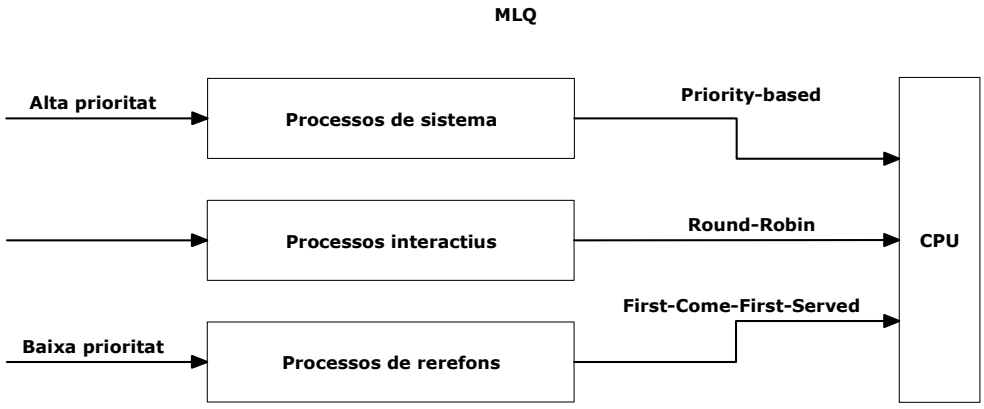

Figura 6.8.2.6. – *MLQ*

- Multi-level feedback queue

  En aquesta variant del MLQ, els processos no estan assignats permanentment a una cua quan entren al sistema. Tenint en compte aquest enfocament, si un procés exhaureix el seu temps "quantum" (per exemple si s'ha encallat), automàticament és transferit a una altra cua amb més temps "quantum" i més baixa prioritat. L'últim nivell utilitza l'algorisme FCFS.

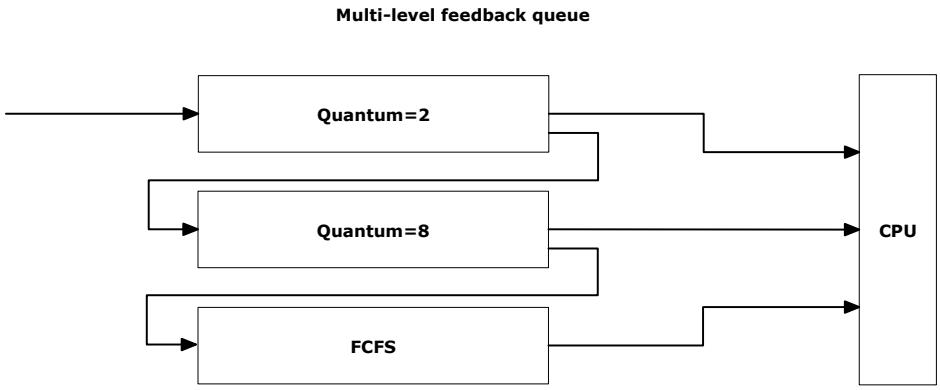

Figura 6.8.2.7. – *Multi-level feedback queue*

6.8.3. Altres sistemes operatius en temps real:

En aquest apartat explicarem breument altres sistemes operatius en temps real:

- eCos:

Aquest sistema operatiu en temps real està dissenyat per a dur a terme aplicacions en sistemes encastats. La seva naturalesa permet a l'usuari configurar-lo i personalitzar-lo per poder complir amb els requisits demanats per a l'aplicació, executant les tasques de la millor manera possible i optimitzant els recursos del hardware.

- RTEMS - The Real-Time Executive for Multiprocessor Systems:

El RTEMS és un sistema operatiu en temps real encarat a la qualitat comercial que fa ús dels sistemes encastats. És una alternativa de codi obert que suporta sistemes amb multiprocessadors. El RTEMS està dissenyat per suportar aplicacions amb les exigències més altes en temps real essent, alhora, compatible amb eines de codi obert.

- BeRTOS:

El BeRTOS és un altre SOTR adequat a les plataformes encastades. Funciona en molts microprocessadors i microcontroladors, des de CPUs de 8 bits a 32, inclús a PCs.

- TinyOS:

El TinyOS és un altre sistema operatiu especialitzat en sensors de xarxa encastats. Conté una arquitectura base que fa possible una ràpida innovació i implementació tot minimitzant el tamany del codi; requisit indispensable en moltes aplicacions que treballen sobre la memòria dels sensors de xarxa.

- NicheTask:

El NicheTask és un SOTR basat en l'scheduling de Round-Robin que principalment s'utilitza en sistemes operatius d'interiors d'edificis, utilitzat per InterNiche Technologies. És un sistema operatiu petit, eficient i flexible que permet l'execució de tasques i és totalment lliure.

- RTOS UH:

Aquest SOTR es va dissenyar per facilitar la tasca de programació per a processos de control automàtics que demanessin uns requisits específics. Està basat en la família de processadors MC 68xxx, MC 683xx, com també amb el PowerPC. Aquest sistema operatiu ofereix el mateix entorn de programació i característiques de temps real que utilitza el hardware.

- TNKernel RTOS**:**

Aquest darrer SOTR és compacte i molt ràpid per a aplicacions basades en microprocessadors encastats de 32/16 bits. Utilitza l'scheduling "Priority-based" amb possibilitat de canviar a "Round-Robin" en cas de tasques amb la mateixa prioritat.

CAPÍTOL 7: Conclusions

En aquest darrer capítol tractarem sobre les conclusions a les que hem arribat després de realitzar el projecte. Parlarem sobre si hem complert els objectius proposats, quines conclusions en podem treure, si el sistema operatiu treballat era fàcil o complicat d'entendre i fer anar, si hem pogut treure profit dels coneixements adquirits, etc.

També esmentarem la opinió personal del treball, i sobre possibles millores si volguéssim continuar en aquests sistemes.

En definitiva, amb aquest capítol tanquem el projecte i animen a continuar a possibles futurs lectors interessats en el món dels sistemes encastats a partir dels coneixements adquirits en aquest.

7. Conclusions i resultats:

En aquest darrer apartat explicaré les conclusions a les que he arribat i els resultats que en puc extreure un cop fet el projecte.

7.1. Conclusions:

En aquest treball he fet tot un estudi sobre què eren els sistemes encastats, quins tipus hi havia, les seves característiques bàsiques de funcionament, què necessitem per posar-nos a dissenyar una aplicació en temps real i uns exemples demostratius.

Per tant, puc dir que he complert els objectius plantejats al principi del treball:

- Introducció als sistemes encastats
- Veure els diferents tipus de sistemes encastats
- Funcionament de la nostra placa amb un sistema operatiu en temps real
- Disseny d'aplicacions que ens permet el nostre sistema operatiu
- Execució del seu funcionament

A més a més, he afegit un apartat d'scheduling, el qual hem de tenir en compte si volem utilitzar altres sistemes operatius en temps real que utilitzin uns mètodes diferents de gestió de tasques. Per exemple, el S. O. anomenat "eCos" permet dos tipus d'schedulers, el bitmap i el MLQ.

He explicat la funcionalitat d'un sistema operatiu en temps real, definint com s'estructura, quins mètodes fa servir per a coordinar les tasques i diferents maneres d'organitzar-les. Per tant, he fet un estudi complet per saber les seves característiques de funcionament i les seves funcions bàsiques per al disseny de qualsevol aplicació.

Així mateix, ha estat una experiència positiva el fet de treballar amb el FreeRTOS, ja que m'ha permès entendre bé el principi i les bases d'aquest tipus de programació multitasca i m'ha animat per poder treballar en altres sistemes similars.

En el meu cas, la part més laboriosa ha estat la recerca d'informació teòrica. Malgrat que ja fa uns quants anys que aquests sistemes estan inventats, no hi ha molta informació fiable al respecte.

El que sí he trobat ben documentat han estat els exemples sobre el processador en qüestió. Buscant una mica, es poden trobar i entendre exemples de tot tipus: gestió de tasques, prioritats, exemples pel port sèrie, aplicacions útils, ... Aquesta ha estat la part més gratificant.

7.2. Resultats:

Com a resultats del projecte, puc dir que no he fet cap aplicació extremadament complicada però si una mica de tot. És a dir, en aquest projecte he intentat explicar i fer funcionar les diferents funcionalitats que permet la placa (tasques, subrutines, comunicació sèrie, SAM-BA, ...) més que fer una aplicació complexa i completa sobre el control d'un tren, per exemple.

La veritat és que si tingués ocasió de continuar aquest projecte, voldria fer i dirigir tot un sistema de robòtica, donat que és una part interessant de l'electrònica industrial.

Pel que fa a aquest sistema operatiu puc afirmar que va bé per aprendre però que és més aviat petit. Per exemple, l'apartat del control d'errors podria estar molt més ben definit. En el cas que intentem accedir a una posició de memòria inexistent mitjançant un apuntador, tant bon punt es realitza l'acció el sistema es queda travat.

El FreeRTOS requereix força memòria RAM. Només pel fet de crear una tasca, una cua i cridar l'scheduler ens consumeix més de 350 bytes de RAM, i cal tenir present que només disposem de 64 Kbytes amb un processador de 32 bits. Aquesta característica suposa un gran inconvenient per a processadors de 8 o 16 bits.

També he trobat un petit entrebanc amb les eines per compilar i debugar. Al principi, utilitzava un compilador ARM incomplet que, per a aplicacions demostratives petites anava bé però a l'hora de compilar el FreeRTOS donava problemes. És per aquest motiu que el compilador ARM ha estat l'única eina que no he utilitzat del YAGARTO.

Aquest projecte tracta de sistemes que actualment estan força integrats en la nostra societat, tant el l'àmbit industrial (robòtica, aeronàutica, emmagatzemament de dades, ...) com en l'ús quotidià (sistemes de vigilància, caixers automàtics, registre de partides, ...) i segurament en el futur tindran un paper important.

Per acabar, aconsellaria als lectors a fer el disseny d'aplicacions més complexes, a provar altres sistemes operatius en temps real i explorar les seves capacitats i aplicacions.

Bibliografia

8. Bibliografia:

Adreces d'Internet:

- http://www.siwawi.arubi.uni-kl.de/avr_projects/arm_projects/ (15/03/09)
- http://www.freertos.org/ (10/04/09)
- http://www.pcengines.ch/ (20/02/09)

Llibre:

- *MARWEDEL, Peter*. Embedded Systems Design. Universitat de Dortmund, Alemanya: Editorial Springer, 2006.

Annex A

9. Annex A:

En aquest annex explicarem en detall com configurar l'entorn de treball.

9.1. Vistes de l'Eclipse:

La següent imatge ens mostra les diferents parts en les que està dividida l'Eclipse:

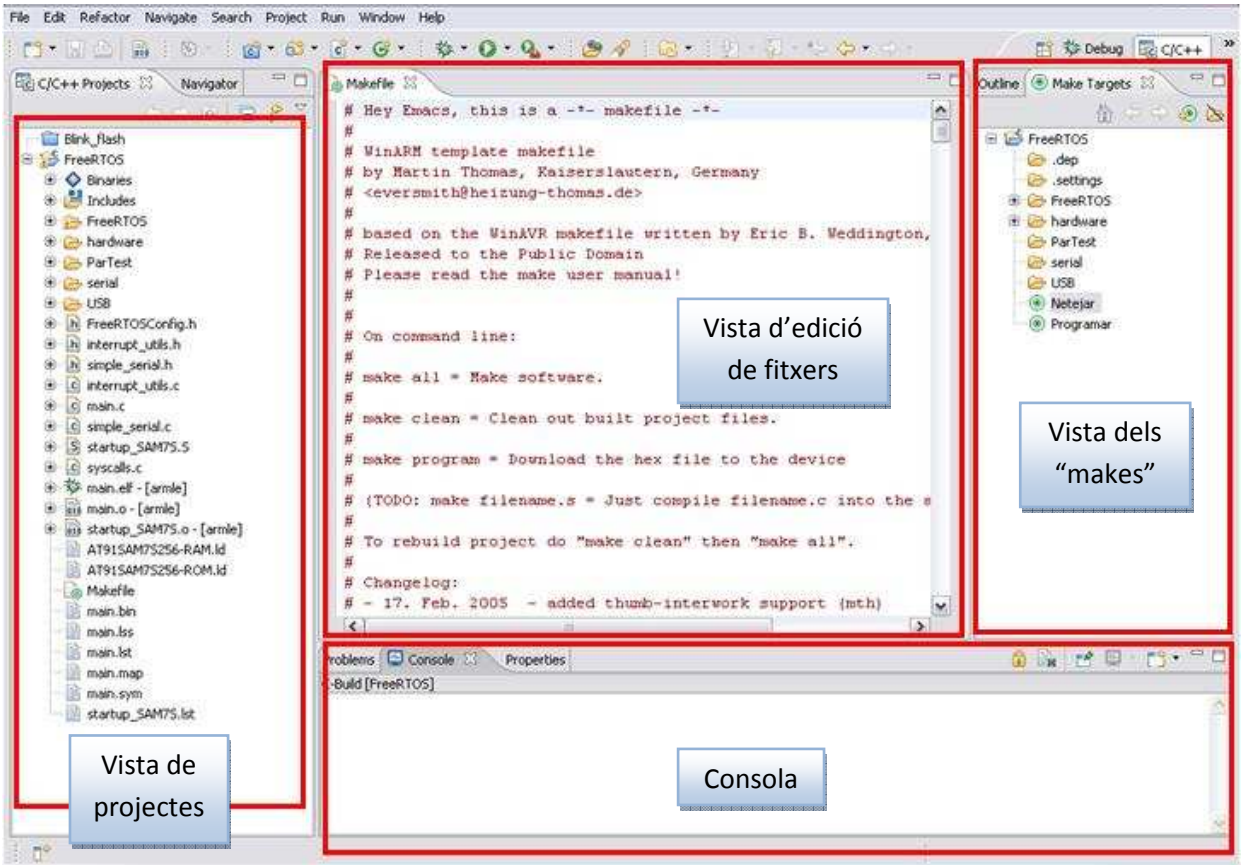

Figura 9.1.1. – *Vistes de l'Eclipse*

- Vista de projectes:

  Vista en la que veiem els nostres projectes actuals, i ens permet obrir-los per fer-ne ús o tancar-los per utilitzar-los més endavant.

- Vista d'edició de fitxers:

  Vista en la que editarem el codi dels nostres fitxers.

- Vista dels "makes":

  Vista en el que veurem els "make targets" disponibles. Es poden crear, modificar i eliminar.

- Consola:

  Vista on veurem el procés de compilació, linkatge i transferència de les nostres aplicacions. Si apareix algun error durant el procés, es veurà reflectit aquí.

9.2. Configuració de l'Eclipse:

Aquí explicarem com es configura l'Eclipse per tal de poder fer ús del compilador GNU, el OpenOCD i els make targets que ens permetran compilar i transferir les nostres aplicacions.

Primer de tot, hem de posar els arxius "at91sam7s256-jtagkey.cfg" i "at91sam7s256-jtagkey-flash-program.cfg" dins de la carpeta /bin del directori d'instal·lació de l'OpenOCD, i també cal posar l'arxiu "script.ocd" a l'arrel de la carpeta de treball, que en aquest cas l'anomenarem "Workspace".

Ara ja podem obrir l'Eclipse. Un cop haguem entrat al programa, després d'haver definit el directori de treball (Workspace), anem al menú → Window → Open Perspective → Other… i després clicar a C/C++ per obrir la perspectiva C/C++, donat que les nostres aplicacions són en llenguatge C.

A continuació procedirem a definir el directori del OpenOCD. Anem al menú → External Tools → External Tools... tal i com s'indica a següent figura:

Figura 9.2.1. – *External Tools*

I indiquem el directori del OpenOCD i del seu arxiu de configuració "at91sam7s256-jtagkey.cfg" que ens permet obrir el dispositiu:

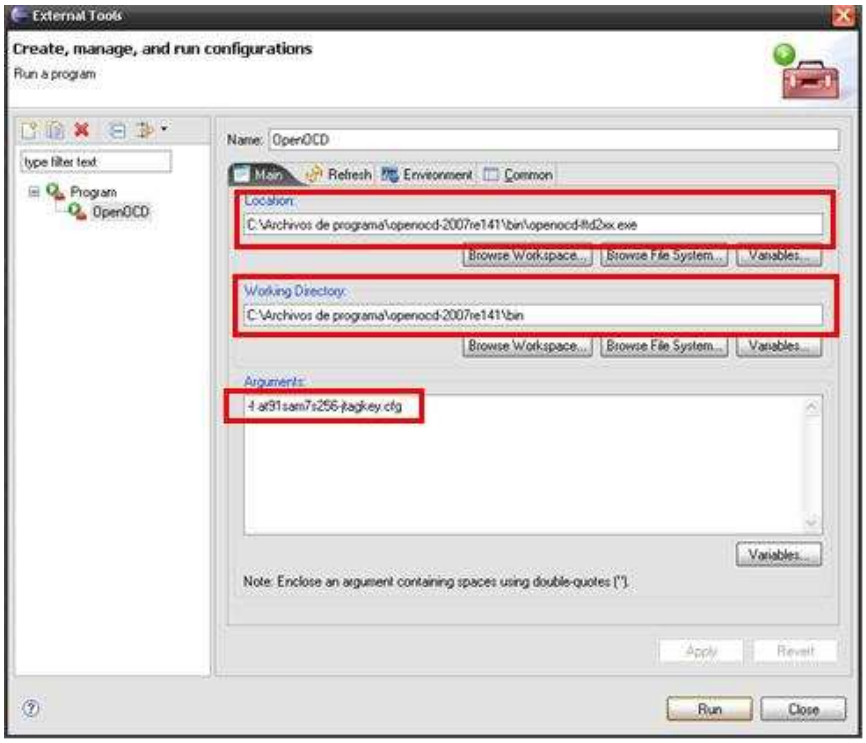

Figura 9.2.2. – *Configuració OpenOCD*

Al primer requadre, hem de posar la ruta del OpenOCD. En aquest cas, fem servir l'executable "openocd-ftd2xx.exe" i no el "openocd-pp.exe" donat que el nostre dispositiu JTAGKey és un dispositiu USB FTDI, i ha de fer servir aquest executable.

Al treball de directori, senzillament s'hi ha de posar el mateix directori del OpenOCD.

Per últim, a Arguments, cal posar aquest arxiu de configuració amb el paràmetre "–f" a davant per tal d'obrir el dispositiu JTAG per poder iniciar la transferència o el mode debug.

9.2.3. Creació d'un nou projecte:

Per a crear un nou projecte, només cal dirigir-se al menú i fer click sobre:

Al següent menú, triarem l'opció "Standard Make Project". La diferència entre aquest i el "Managed Make Project" és que aquest últim ens crea automàticament el fitxer "makefile" (sense extensió), però com que ja ens ve inclòs a l'aplicació no és necessari crear-lo:

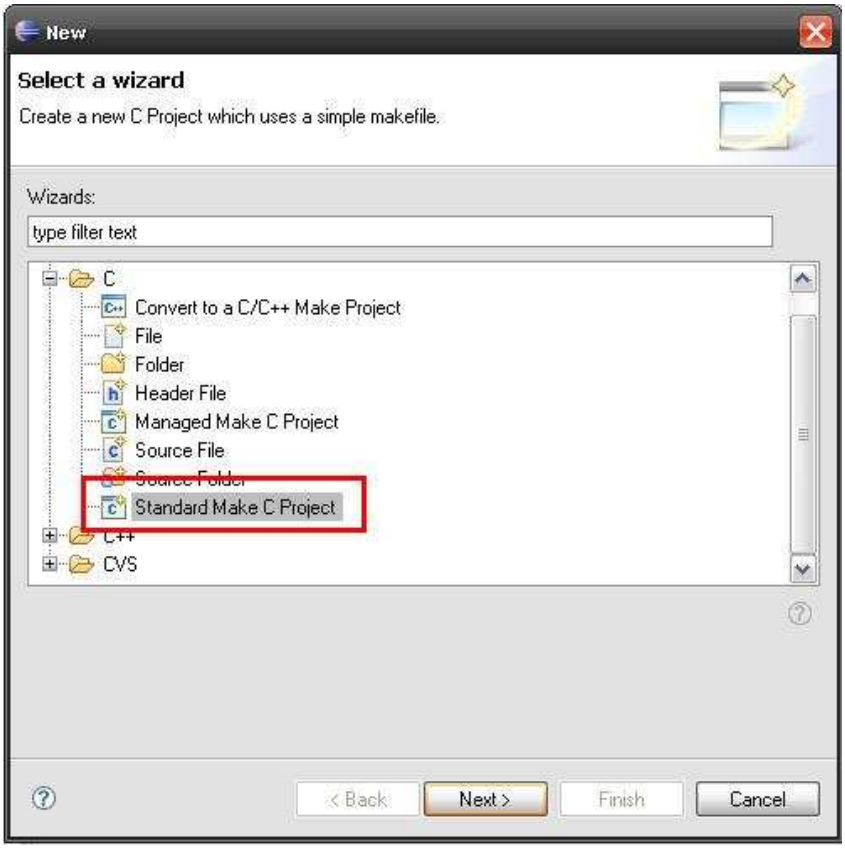

Figura 9.2.3.1. – *Projecte C nou*

A continuació introduïm el nom del projecte i se'ns crearà una carpeta al Workspace amb el mateix nom, i també un nou projecte a la "vista de projectes" de l'Eclipse.

Un cop creat al projecte al directori de treball, hem d'importar els arxius de l'aplicació al nostre projecte. Per a fer-ho, fem click amb el botó dret a sobre el nostre projecte i a continuació "Import...":

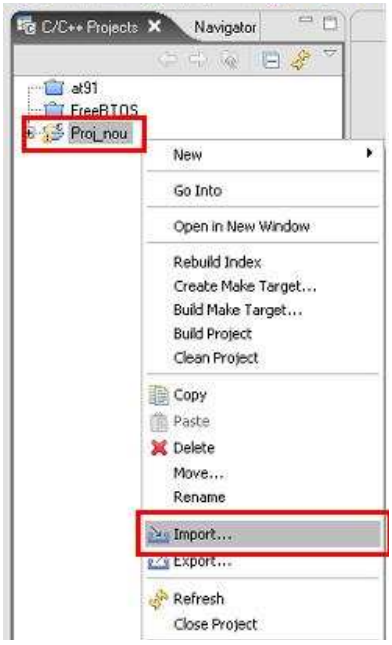

Figura 9.2.3.2. – *Importació fitxers*

I a continuació triem la opció: File System. D'aquesta manera importarem tot el contingut de la carpeta desitjada:

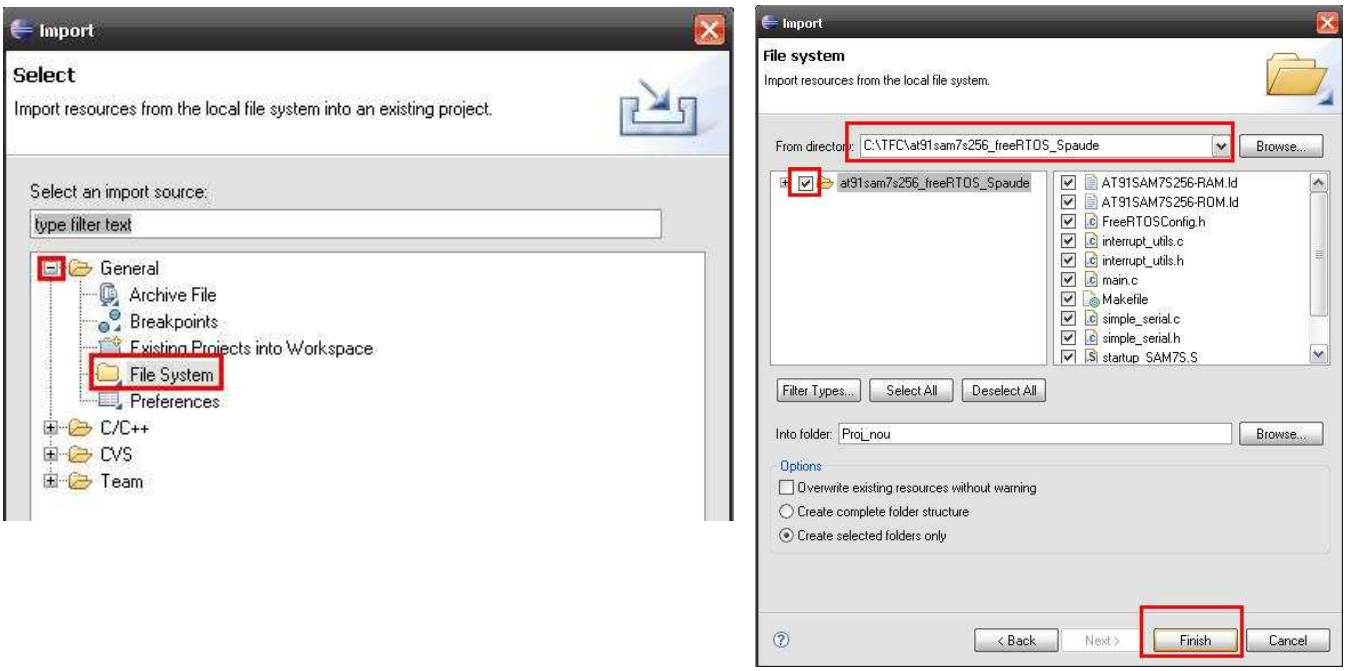

Figura 9.2.3.3. – *Fitxers a importar*

Un cop importat el contingut de la carpeta, en podrem editar el seu contingut a la vista d'edició de fitxers de l'Eclipse.

9.3. Make targets:

Aquí explicarem com compilar i transferir l'aplicació a través dels Make targets. Obrim la vista dels makes si és que encara no la teníem oberta:

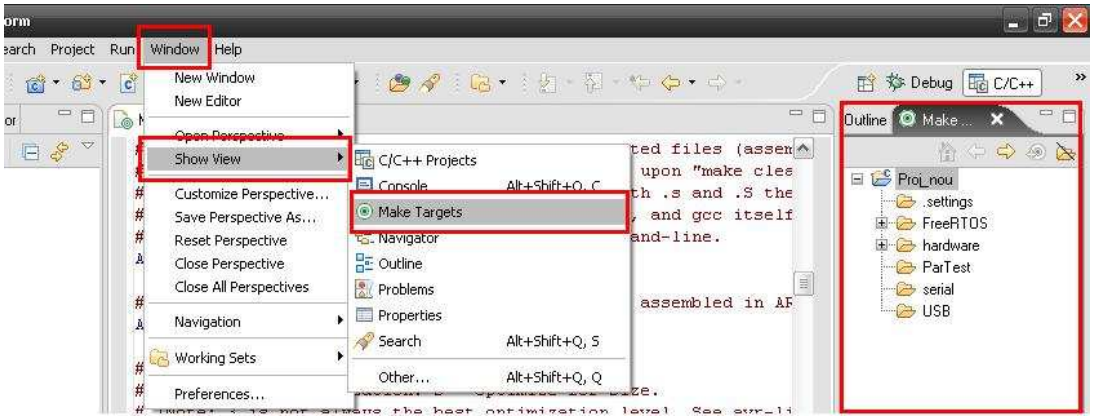

Figura 9.3.1. – *Make targets*

Ara fem click amb el botó dret sobre el nom del nostre projecte o aplicació i li diem: Add Make Target:

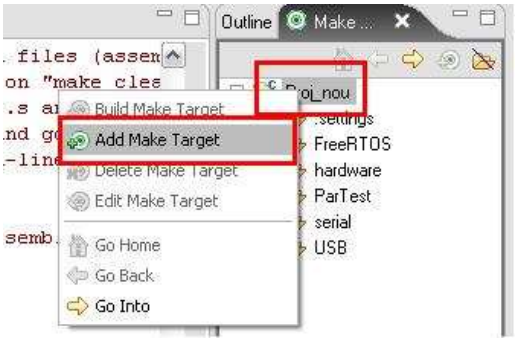

Figura 9.3.2. – *Afegir target*

A continuació li definirem dos make targets: Netejar i Programar. El primer netejarà el projecte actual i el compilarà de nou, i el segon el transferirà a placa:

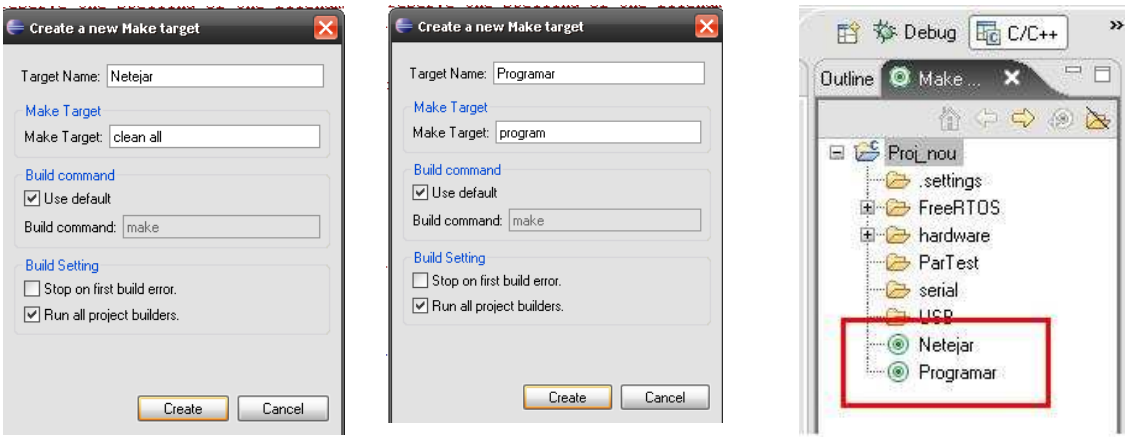

Figura 9.3.3. – *Creació targets*

Un cop fet això, ja estarem a punt per crear i transferir la nostra aplicació. Ara veurem com debugar-la.

9.4. Mode Debug:

Primer, obrim la perspectiva Debug amb el mateix procediment que l'anterior. Un cop a dins cal configurar el debugador, per a fer-ho anem a Debug → Debug...

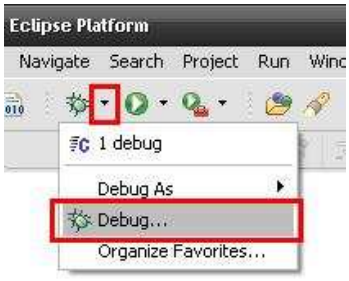

Figura 9.4.1. – *Debug*

Després, fem click sobre "Embedded Debug (Native)" i seleccionem el nostre projecte:

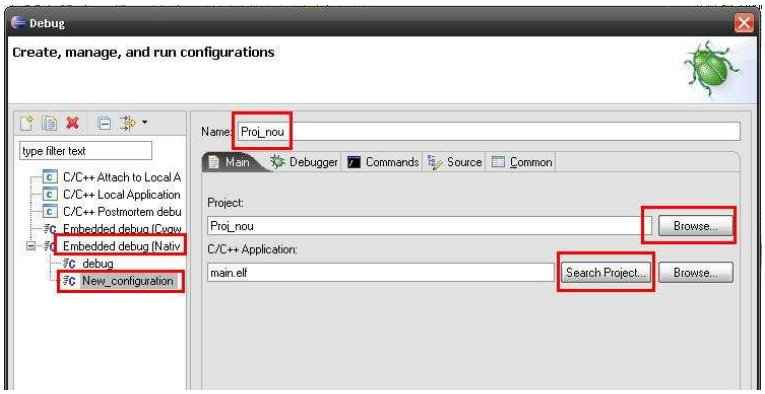

Figura 9.4.2. – *Configuració debug*

Després anem a la pestanya "Debugger" i seleccionem el debugador arm-elf-gdb.exe:

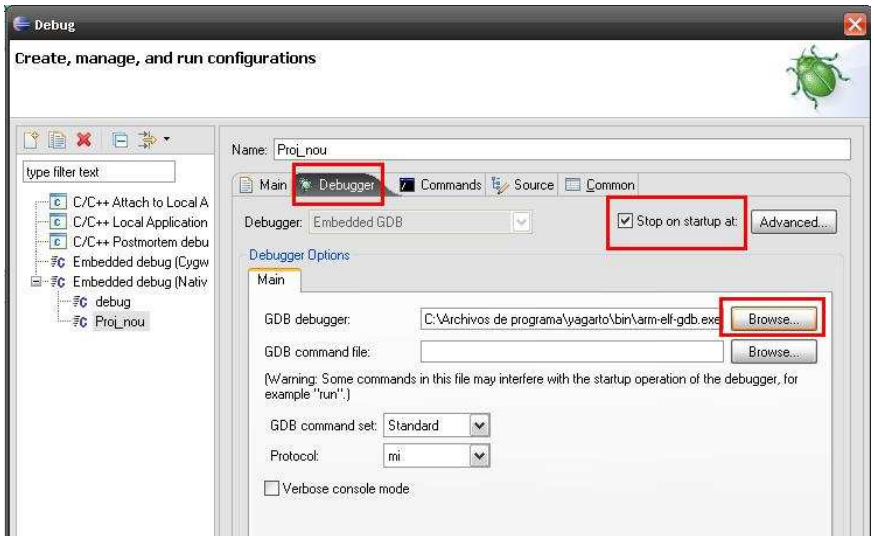

Figura 9.4.3. – *Configuració debugació*

Per últim, anem a la pestanya "Commands" i hi introduïm la comanda "target remote localhost:3333" al requadre superior i les següents comandes al requadre inferior:

```
monitor soft_reset_halt
monitor armv4_5 core_state arm
monitor mww 0xffffff60 0x00320100
monitor mww 0xfffffd44 0xa0008000
monitor mww 0xfffffc20 0xa0000601
monitor wait 100
monitor mww 0xfffffd08 0xa5000401
monitor mww 0xfffffc2c 0x00480a0e
monitor wait 200
monitor mww 0xfffffc30 0x7
monitor wait 100
set remote memory-write-packet-size 1024
set remote memory-write-packet-size fixed
set remote memory-read-packet-size 1024
set remote memory-read-packet-size fixed
monitor mww 0xfffffd00 0xa5000004
monitor mww 0xffffff00 0x01
monitor reg pc 0x00000000
monitor arm7_9 sw_bkpts enable
load
continue
```

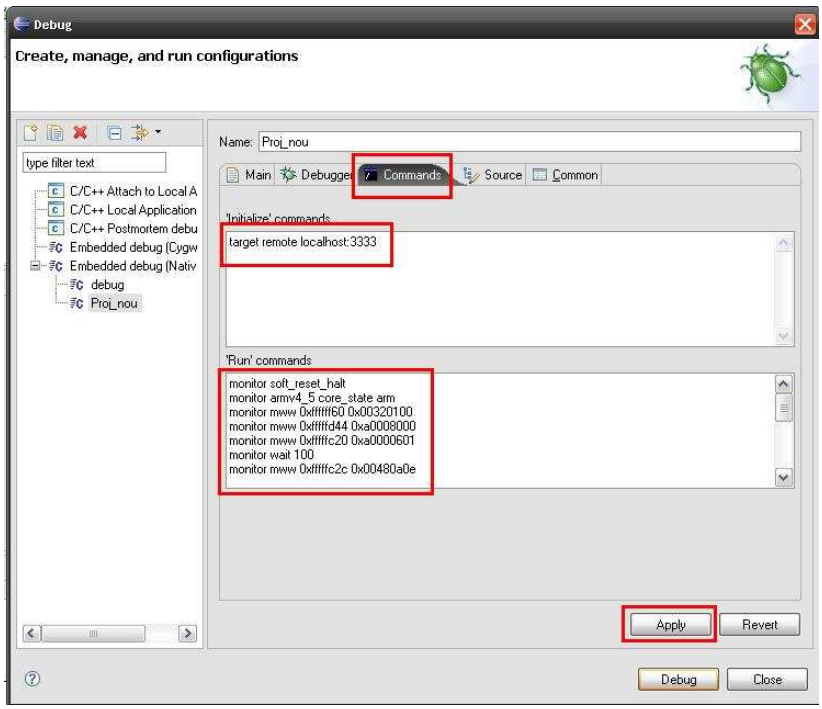

Figura 9.4.4. – *Comandes debugació*

Ara ja podem debugar. Per a fer-ho, primer cal executar el OpenOCD des de external tools i posteriorment el debugador:

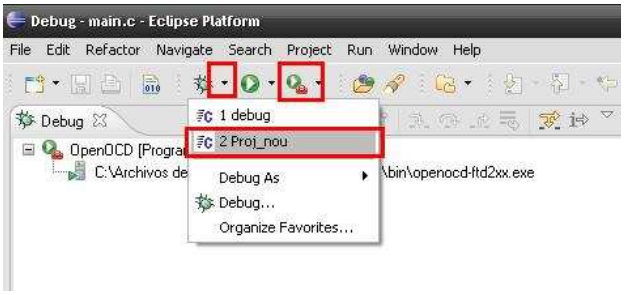

Figura 9.4.5. – *Obrir debugació*

Un cop fet, ja podem procedir a la seva debugació a partir de la barra d'eines que ens proporciona aquest mode:

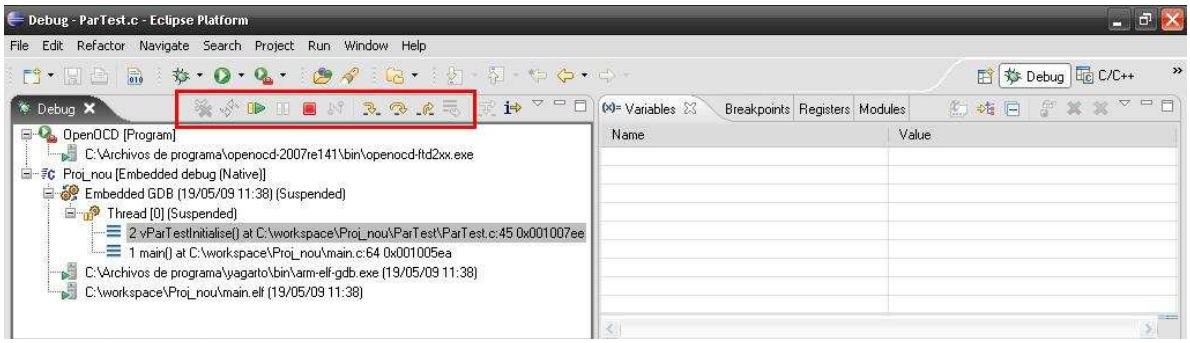

Figura 9.4.6. – *Ús de la debugació*

Aquestes eines ens permeten pausar i reprendre la debugació, parar-la completament, i fer un seguiment a través de les diverses funcions. També podrem veure l'estat de les variables, breakpoints i registres en tot moment en el requadre lateral.

Cal destacar que hi ha dos mètodes de debugació:

- Debugació Flash:

  Permet debugar l'aplicació localitzada en la memòria flash. Com a característica principal, utilitza breakpoints de hardware i només admet dos breakpoints. La debugació és relativament lenta.

- Debugació Ram:

  Permet debugar l'aplicació localitzada en la memòria ram. A diferència de l'altre tipus de debugació, aquesta utilitza breakpoints de software i, per tant, en permet molts. La debugació és més aviat ràpida, però té l'inconvenient de tenir només 64 Kb d'espai, enfront dels 256 kb de la memòria Flash.