Universitat de Vic
Escola Politècnica Superior

Treball Final de Carrera

Desenvolupament d'un sistema de transferència de software per una placa entrenadora

Guillem Soler i Franquesa

Enginyeria Tècnica Industrial especialitat Electrònica Industrial

Director: Jordi Serra i Serra

Vic, juny de 2008

ÍNDEX

Universitat de Vic
Escola Politècnica Superior

Resum de Treball Final de Carrera
Enginyeria Tècnica Industrial especialitat Electrònica Industrial

Títol: Sistema de transferència de software per una placa entrenadora
Paraules clau: Programador, bootloader, microcontrolador, placa entrenadora, comunicació sèrie.
Autor: Guillem Soler i Franquesa
Direcció: Jordi Serra i Serra
Data: Juny de 2008

Resum

El departament d'electrònica i telecomunicacions de la Universitat de Vic ha dissenyat un conjunt de plaques entrenadores amb finalitat educativa. Perquè els alumnes puguin utilitzar aquestes plaques com a eina d'estudi, és necessari disposar d'un sistema de gravació econòmic i còmode. La major part dels programadors, en aquest cas, no compleixen amb aquests requeriments.

L'objectiu d'aquest projecte és dissenyar un sistema de programació que utilitzi la comunicació sèrie i que no requereixi d'un hardware ni software específics. D'aquesta manera, obtenim una placa autònoma i un programador gratuït, de muntatge ràpid i simple d'utilitzar.

El sistema de gravació dissenyat s'ha dividit en tres blocs. Per una banda, un programa que anomenem "programador" encarregat de transferir codi de programa des de l'ordinador al microcontrolador de la placa entrenadora. Per altra banda, un programa anomenat "bootloader", situat al microcontrolador, permet rebre aquest codi de programa i emmagatzemar-lo a les direccions de memòria de programa corresponents. Com a tercer bloc, s'implementa un protocol de comunicació i un sistema de control d'errors per tal d'assegurar una correcta comunicació entre el "programador" i el "bootloader".

Els objectius d'aquest projecte s'han complert i per les proves realitzades, el sistema de programació ha funcionat correctament.

Universitat de Vic
Escola Politècnica Superior

Degree Final Project Summary
Technical Industrial Engineering in Industrial Electronics

Title: Software transfer system for a development board
Key words: Programmer, boot loader, microcontroller, development board, serial communication.
Author: Guillem Soler i Franquesa
Direction: Jordi Serra i Serra
Date: June 2008

Summary

The electronic and telecommunication department of Universitat de Vic has designed a set of development boards with educational purpose. It is necessary to have an economic and comfortable programming system, thus, the students may use the boards as a tool of study. In this case, most programmers don't comply with these requirements.

The objective of this project is to design a programming system that use a serial communication and doesn't require a specific hardware and software. Thus, we get an autonomic board and a free programmer, with a fast montage and easy to use.

The programming system is divided in three parts. First of all, a program called "programmer" transfers program code from the computer to the development board. Then, a program called "boot loader", placed in the microcontroller of board, allows to receive this program code and store it in corresponding program memory directions. At last, communication protocol and error system controls are implemented to ensure an appropriate communication between "programmer" and "bootloader".

The objectives of this project have been completed .We have obtained a programming system that allows to download software to the development board correctly

1. INTRODUCCIÓ

El departament d'electrònica i telecomunicacions de la Universitat de Vic ha decidit crear un conjunt de plaques entrenadores amb finalitat educativa. Aquestes plaques, basades en el microcontrolador ATMEGA16 de la companyia ATMEL, permetran als alumnes provar els seus programes d'una forma ràpida, ja que disposaran d'un conjunt de perifèrics fàcilment accessibles. D'aquesta manera, l'alumne, no haurà de reconstruir circuits sobre protoboards per provar les seves aplicacions cada cop que hagi de connectar una interfície o un perifèric amb el microcontrolador. Aquest fet suposa un estalvi de temps considerable que podrà ser dedicat a altres aspectes més importants de l'aprenentatge de l'assignatura.

El problema sorgeix a l'hora de descarregar un programa al microcontrolador de la placa entrenadora, ja que adquirir un programador té certs inconvenients. La majoria de programadors tenen un alt cost econòmic. Per altre banda, el seu us és limitat, ja que la majoria de vegades, està format per un hardware i un software específic. Això fa que sigui poc pràctica la programació de les plaques entrenadors si el programador ha de ser usat per molts usuaris. Tot i que podríem evitar aquesta restricció adquirint tants programadors com plaques hi hagi, això suposaria una gran despesa econòmica: el preu d'un programador pot arribar a doblar o més el preu d'una placa entrenadora.

Per no haver d'adquirir un programador, és necessari implementar un sistema de programació pel microcontrolador més àgil, econòmic, i que pugui estar a l'abast de tothom. Aquest sistema ha de ser de lliure accés i les eines requerides per utilitzar-lo han de poder ser adquirides fàcilment. D'aquesta manera s'obtindrà un mètode de gravació àgil i pràctic. Mitjançant aquest sistema, els alumnes podran programar les plaques fora dels laboratoris.

D'aquí doncs, surt la idea d'aquest treball final de carrera, que consistirà en el disseny i construcció d'aquest sistema de gravació. Aprofitant que l'ATMEGA16 disposa de memòria de programa no volàtil i comunicació sèrie, es dissenyarà un sistema compost per un bootloader i un programador comunicats via sèrie.

1.1 Objectius

L'objectiu principal és dissenyar un sistema de programació per la placa entrenadora que no requereixi d'un hardware específic, sigui econòmic i fàcilment accessible per la majoria d'usuaris. D'aquesta manera, podrem obtenir una placa autònoma, que no hagi de dependre de hardware extern. Podem desglossar aquest objectiu en:

- Crear un programa per l'ordinador, que pugui llegir fitxers hexadecimals (els quals contenen instruccions que han d'anar ubicades a la memòria de programa del microcontrolador) i n'enviï el contingut al microcontrolador pel port sèrie.

- Dissenyar un programa pel microcontrolador, anomenat bootloader, que pugui rebre les instruccions enviades per l'ordinador i les gravi a la memòria de programa del microcontrolador.
- Dissenyar un protocol de comunicació, per tal de que el microcontrolador i l'ordinador transfereixin dades correctament, amb un mètode de comprovació d'errors. Aquest mètode, que estarà basat en un CRC, serà necessari per verificar que les dades s'hagin enviat correctament entre els dos dispositius.

2. BASES GENERALS DEL PROJECTE

En aquest capítol s'explicarà de forma genèrica l'estructura del projecte. Per una banda explicarem breument el funcionament i l'estructura de la placa entrenadora que utilitzarem. Per altre banda, especificarem perquè hem escollit el sistema de gravació seleccionat, quins llenguatges de programació farem servir i quin entorn de treball utilitzarem, i la metodologia que s'ha seguit per poder dissenyar el sistema de gravació. Amb aquests sub-apartats intentarem mostrar una idea més clara de com es farà el projecte i les diferents parts que el composen.

2.1 Descripció de la placa entrenadora

La placa entrenadora que s'utilitzarà ha estat creada pel departament d'enginyeria i comunicacions de la Universitat de Vic amb finalitat educativa. Aquesta placa està adreçada als alumnes perquè la puguin fer servir fora del laboratori, agilitzant així les classes de l'assignatura. La placa entrenadora està pensada per funcionar amb el microcontrolador ATMEGA16 i conté una sèrie d'interfícies i dispositius que estan connectades amb aquest[1]. Entre aquestes i trobem:

- Un sistema d'alimentació
- Un dispositiu per a programació sobre placa (SPI), pensat per a programar la memòria flash del microcontrolador o incorporar dispositius SPI
- Un sensor de temperatura SPI
- Un dispositiu d'infraroig amb emissor i receptor.
- Una pantalla de LCD
- Un conjunt de sortides analògiques (com un altaveu i una sortida MiniJack)
- Una entrada de senyal analògica.
- Un teclat matricial 3x4
- 5 leds (1 d'aquests leds disposa d'un polsador i està connectat al pin d'una interrupció externa) i interruptors
- Una interfície sèrie RS232 DB-9 (9 pins)

En aquest projecte només s'utilitzarà:

- La interfície sèrie, ja que serà el mitjà de comunicació entre la placa i l'ordinador.
- Els leds, per fer proves i comprovar que els programes funcionin, i el polsador de la interrupció externa com a mètode d'accés al bootloader.
- El dispositiu SPI, ja que s'utilitzarà per programar el microcontrolador i guardar, en la seva memòria de programa, el bootloader.

[1] A l'apartat 1.1 de l'annex està exposat l'esquema de la placa entrenadora

2.2 Selecció del sistema de gravació

A l'hora d'escollir un sistema de gravació, hem d'analitzar les característiques del microcontrolador ATMEGA16:

- Per una banda, l'ATMEGA16 té la possibilitat de ser programat sobre placa (In-System-Programming). Tot i que és un sistema de programació que pot ser utilitzat mitjançant un hardware senzill, s'ha descartat perquè volem aconseguir que la placa sigui autònoma, és a dir, que no depengui d'un hardware extern. El que es vol aconseguir és utilitzar els ports de comunicació de la placa entrenadora per programar el microcontrolador. Aquest sistema de gravació, però, es farà servir per programar la placa entrenadora durant el projecte.

- Per altra banda, tindrem en comte les capacitats de la seva memòria de programa. La memòria de programa és on s'emmagatzema tot el codi que fa referència a les instruccions de l'aplicació carregada al microcontrolador. Per tant, quan nosaltres descarreguem un programa al microcontrolador, el codi que conté aquest programa, és guardat en aquesta memòria.

Trobarem dues qualitats d'aquesta memòria que ajudaran a establir un mètode de gravació:

- És una memòria no volàtil: El seu contingut (és a dir, les instruccions de codi de programa,) es mantenen emmagatzemades encara que el microcontrolador deixi d'estar alimentat. Per tant, podrem carregar un programa a l'ATMEGA16 i quedarà memoritzat fins que reprogramem la seva memòria amb un altre programa o bé l'esborrem.

- És una memòria autoprogramable: El mateix L'ATMEGA16 pot escriure i esborrar la seva memòria de programa (mitjançant un conjunt d'instruccions concretes). Aquesta memòria flash s'escriu per blocs de 128 bytes, que prenen el nom de pàgines.

Aquestes dues qualitats fan que ens decidim per un mètode que utilitzi un bootloader

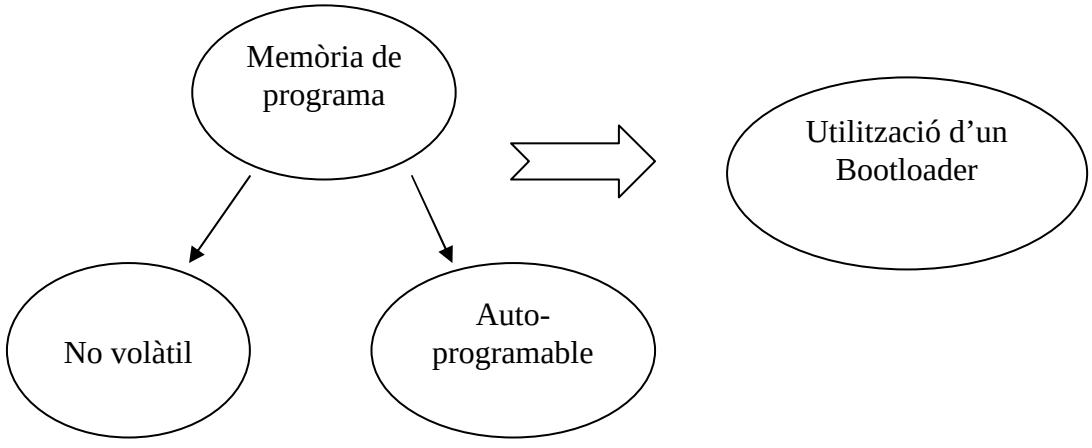


Fig. 2.2.1: Característiques de la memoria de programa de l'ATMEGA16

Un bootloader és un petit programa situat a una part de la memòria de programa del microcontrolador que permet:

- Rebre codi de programa d'una font externa
- Escriure aquest codi a les adreces inicials de la mateixa memòria de programa.

La **Figura 2.2.2** mostra el seu funcionament general; el quadre central del xip dibuixat representa la memòria de programa i el quadre gris que incorpora representa el programa del bootloader.

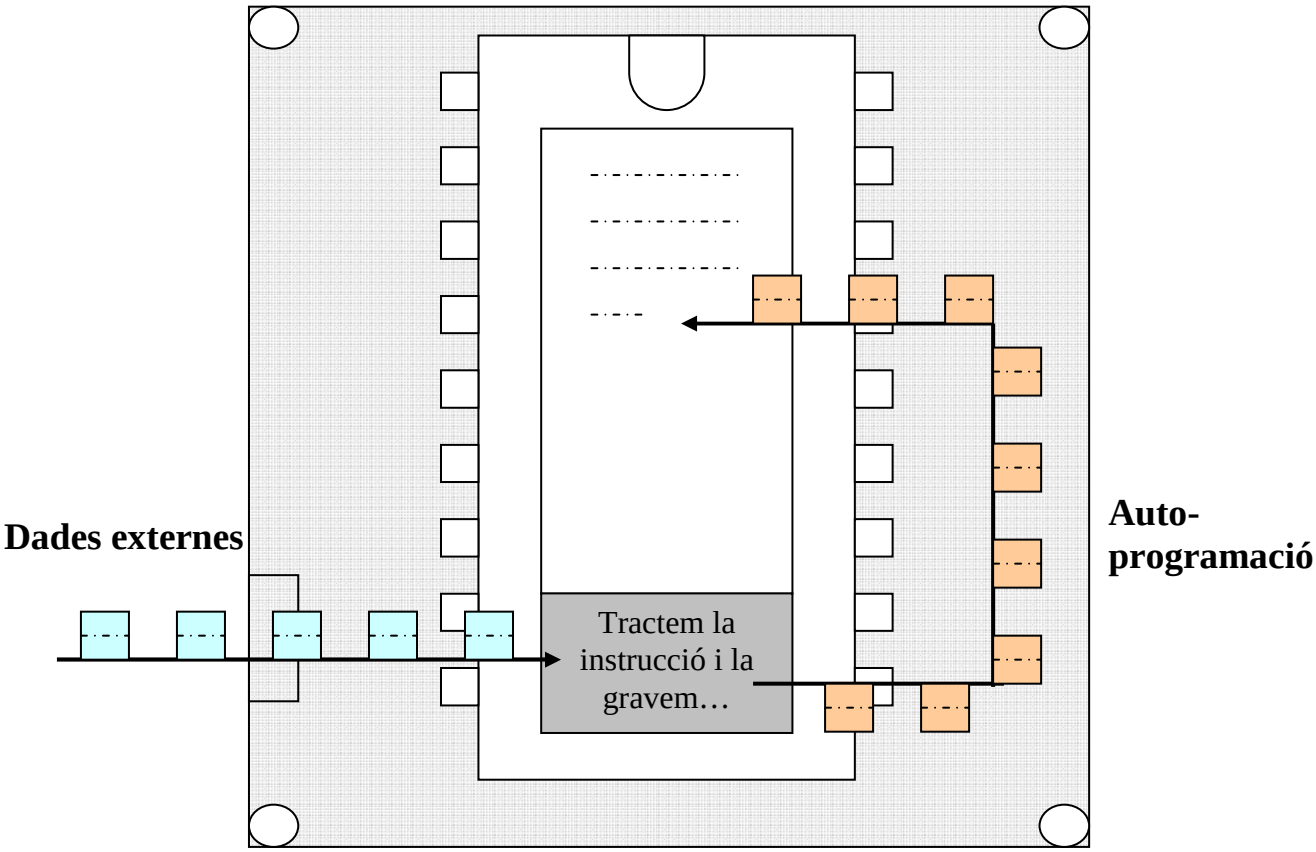


Fig. 2.2.2: Funcionament general del sistema de gravació

Les instruccions que rep el bootloader corresponen al codi del programa que volem descarregar al microcontrolador. Un cop rebudes, el bootloader les tracta i les escriu a la direcció corresponent de la memòria de programa. La font externa que envia aquestes instruccions és l'ordinador. El programa de l'ordinador que s'encarrega d'adquirir les instruccions i enviar-les cap al microcontrolador rep el nom de programador.

El programador ha de fer les següents tasques

- Accedir a un fitxer que conté el codi de programa que ha de ser enviat al microcontrolador i obtenir aquest codi. Aquest fitxer amb extensió "hex", que pot ser

llegit com a arxiu de text, és del tipus INTEL16 i conté les dades de memòria de programa organitzades d'una manera característica i en base numèrica hexadecimal. L'arxiu "hex" s'obté a partir de la compilació del programa per al microcontrolador que es fa mitjançant un software específic, en el nostre cas, l'AVR STUDIO amb el complement WinAVR. Aquest complement és necessari per poder compilar programes implementats en llenguatge C, que és el que utilitzarem.

- Enviar aquest codi al microcontrolador via sèrie.

Aquestes tasques estan representades en la següent figura:

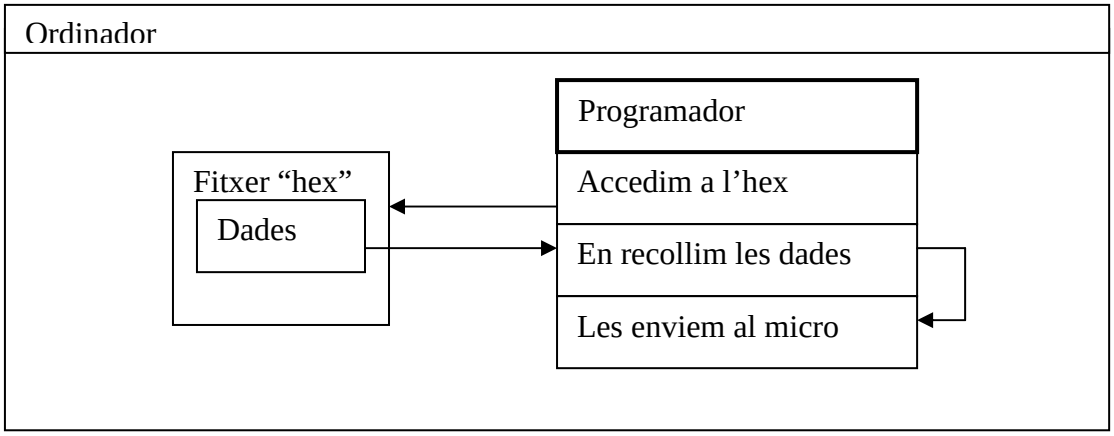


Fig. 2.2.3: Diagrama de tasques globals del programador

Per comunicar el programador amb el bootloader i poder transferir-li dades és necessari escollir un mitjà de transmissió. Aprofitant que el microcontrolador ATMEGA16 permet rebre i enviar dades via sèrie, i que la placa entrenadora té incorporada la interfície R-232 per aquest dispositiu, s'utilitzarà la comunicació sèrie asíncrona UART (Universal asyncronous reciver transmision).

Per poder establir aquesta comunicació sèrie correctament cal dissenyar un protocol, que definirà:

- una velocitat de transmissió (Baud Rate)
- un mètode de control d'errors
- una configuració per la comunicació
- un llenguatge de comunicació entre els dos dispositius perquè es sincronitzin correctament.

Així doncs, podem veure que el projecte està dividit en tres grans blocs: el bootloader, el programador i el protocol de comunicació. Per implementar aquest sistema hem dissenyat cada part per separat i finalment s'han unit les tres parts.

En aquesta memòria, cada un d'aquests blocs serà tractat en un apartat diferent i serà desglossat en diferents subparts. Per tenir una visió general d'aquests blocs globals podeu observar la següent figura:

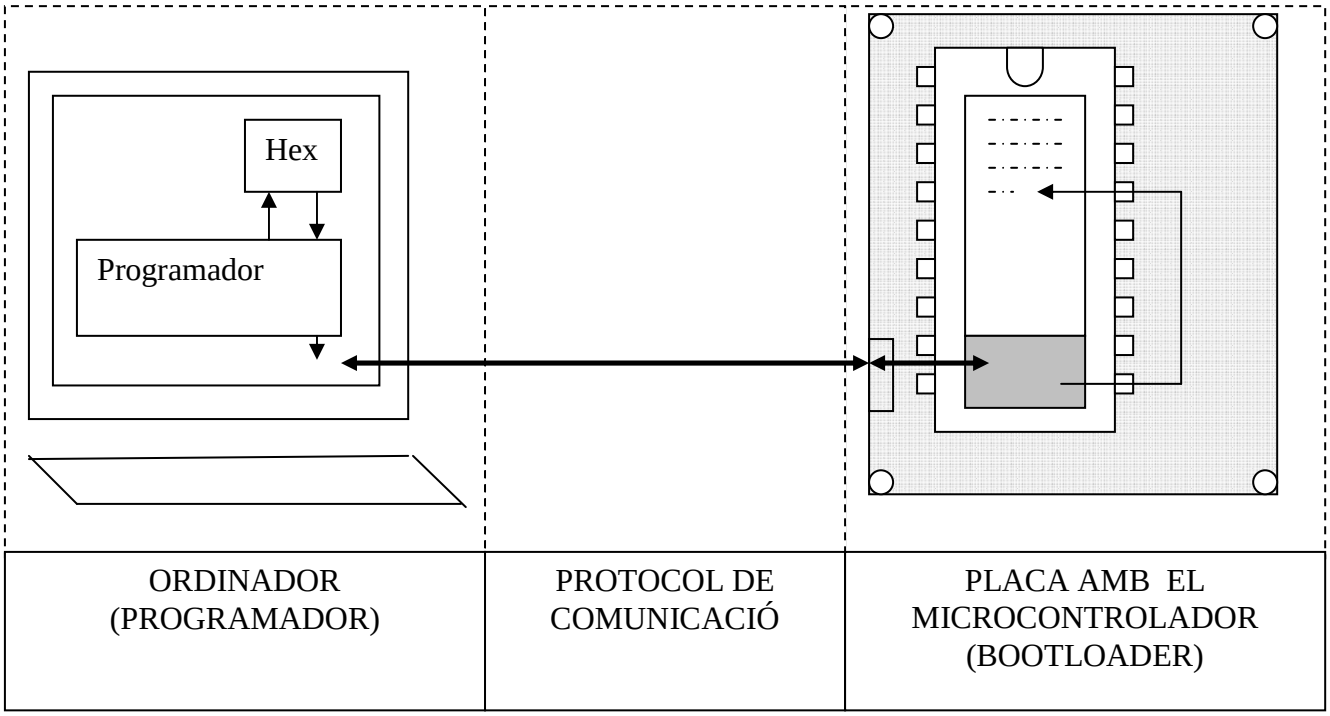


Fig. 2.2.4: Blocs del sistema de gravació

En els apartats següents especificarem el llenguatge amb que programarem el bootloader i el programador, i el software que utilitzarem.

2.3 Selecció del llenguatge i software de programació

El llenguatge de programació que s'utilitzarà tant pel programador com pel bootloader serà el C. Escollim aquest llenguatge perquè el microcontrolador només permet programar en C i en Assemblador. Tot i que utilitzarem el llenguatge assemblador per aprendre l'arquitectura de l'ATMEGA16, el C ens permetrà programar d'una manera més pràctica, ja que és un llenguatge de més alt nivell, i aconseguirem uns programes més organitzats i entenedors que en assemblador. Com que el bootloader el programarem en C, el programador també, d'aquesta manera l'aprenentatge del llenguatge ens servirà per programar en els dos dispositius.

Per programar el microcontrolador utilitzarem el software AVR Studio de la companyia ATMEL, que pot ser descarregat gratuïtament des de la seva web. També serà necessari instal·lar un complement anomenat WinAVR. Aquest complement ens permetrà compilar programes fets en llenguatge C i inclourà un conjunt de llibreries, anomenat "AVR lib-c", que contenen funcions que utilitzarem.

Per programar amb l'ordinador s'utilitzarà el software DEV-C++. S'ha escollit aquest software perquè és de lliure accés i permet programar en C. També permet incorporar un complement anomenat wxWidgets que possibilita la creació d'interfícies gràfiques, i que s'aprofitarà per dissenyar-ne una per tal oferir una visió més atractiva del

programador. WxWidgets treballa amb objectes, per tant, passaríem de treballar amb llenguatge C a llenguatge C++. El paquet d'eines de compilació que du incorporat el DEV-C++ s'anomena MingW i està basat en el paquet d'eines de compilació que utilitza la maquinaria UNIX transportada a la plataforma Microsoft Windows.

2.4 Metodologia: com s'ha realitzat el projecte

És necessari organitzar les tasques per tal de realitzar el projecte. El dividirem en tres blocs que seran treballats per separat, tenint en comte, però, que cada bloc dependrà dels altres dos. Primerament, cal analitzar bé cada bloc per saber que s'ha d'aprendre a fer abans de construir-los, com se'ns mostra en la figura següent:

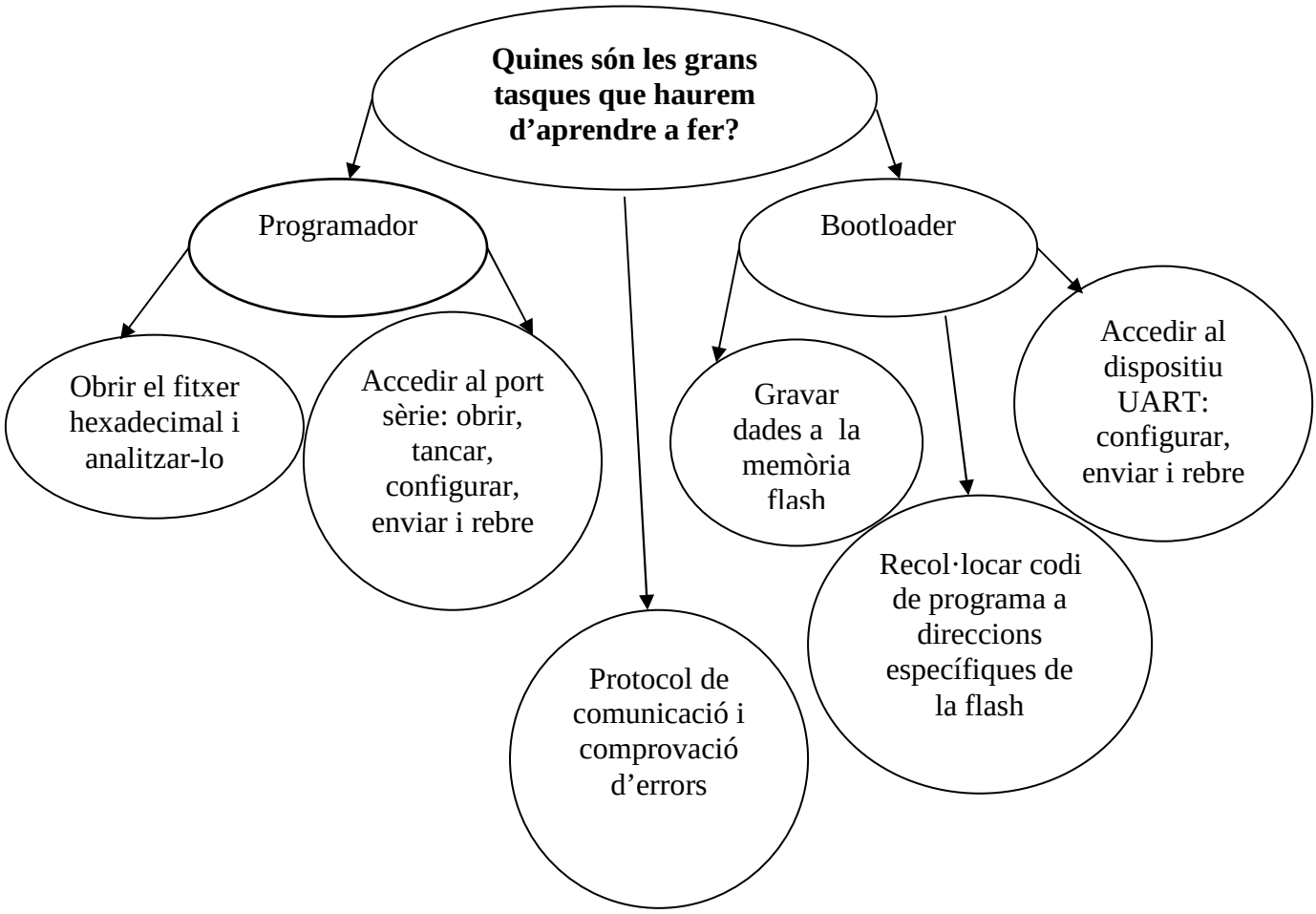


Fig. 2.4: Tasques principals del sistema de gravació

Pel que fa el protocol de comunicació, haurem de:

- Saber quins són els paràmetres que s'hauran de configurar en la comunicació per tal d'establir un protocol.

- Trobar un sistema de comprovació d'errors que ens pugui anar bé i saber-lo implementar.

Per fer les tasques del programador ens serà útil:

- Crear alguns programes en llenguatge C[2]. Tot i tenir uns coneixements mínims apresos al llarg de la carrera cursada, és necessari fer-ne memòria. D'aquests programes destacarem:
    - Programes que treballin amb fitxers, ja que el programador haurà d'accedir a l'arxiu "hex", com a arxiu de text, i extreure'n el contingut de dades necessàries.
    - Programes que es comuniquin pel port sèrie, perquè necessitarem saber com manipular aquest port.

Per fer les tasques del bootloader ens serà útil:

- Fer petits programes amb llenguatge assemblador i C, que toquin diferents recursos del microcontrolador[3]. El llenguatge assemblador és un llenguatge de baix nivell que ens permetrà fixar-nos bé com el microcontrolador treballa amb els seus registres i com fa certes operacions.

- Consultar el manual del microcontrolador, ja que ens informa sobre registres, les característiques i restriccions dels diferents recursos que té.

Aquests tres blocs estan exposats a continuació. Començarem explicant el que fa referència al protocol de comunicació. En aquest apartat intentarem aclarir el sistema de transferència de dades (com i en quina quantitat transferim les dades) i el funcionament del port sèrie. Un cop explicat el protocol, procedirem a explicar el programador, i el bootloader.

[2] La llista de programes pel programador estan descrites a l'apartat 1.2.1 de l'annex
[3] La llista de programes pel microcontrolador estan descrites a l'arpatat 1.2.2 de l'annex

3. PROTOCOL DE COMUNICACIÓ

El microcontrolador i la placa entrenadora es comunicaran via sèrie, mitjançant un cable "Null-Modem". En el cas de que l'ordinador no tingui port sèrie, es pot adquirir un adaptador de USB a sèrie. Per realitzar una correcta comunicació entre els dos dispositius cal establir un protocol. Aquest protocol és el conjunt de normes que afecten a la comunicació i que han de ser complertes per l'ordinador i el microcontrolador. Definint aquest conjunt de regles farem que els dos dispositius es puguin entendre i s'enviïn les dades correctament. Per poder descriure aquestes normes dividirem aquest capítol en diferents subparts.

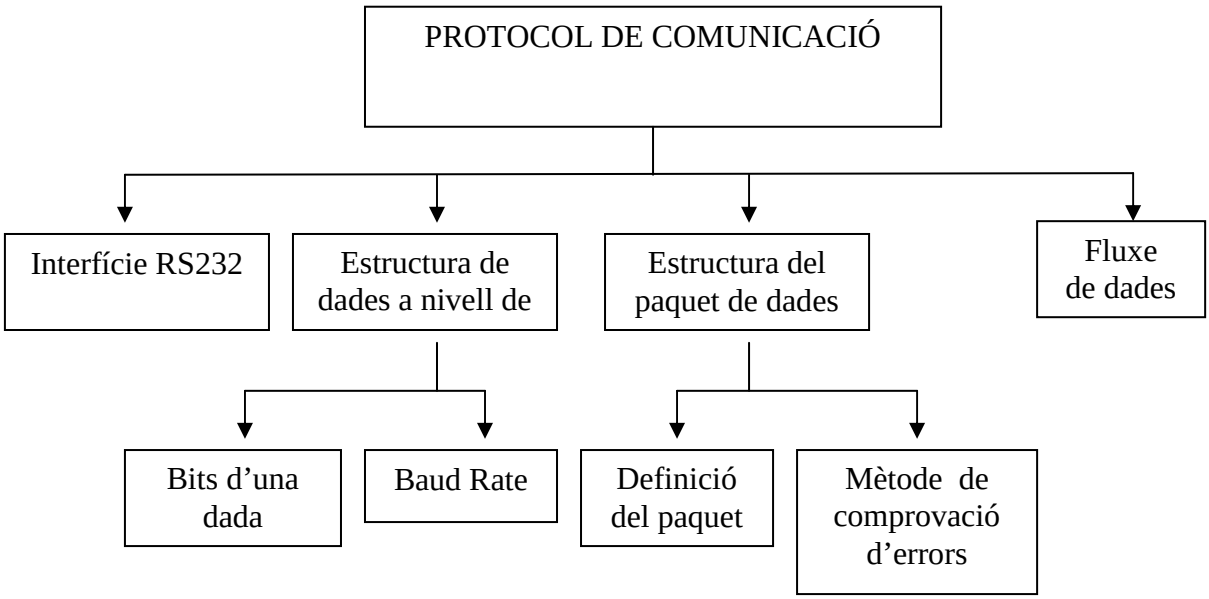


Fig. 3: Parts del protocol de comunicació

Començarem descrivint breument la capa física del port sèrie. Posteriorment, definirem l'estructura de dades a nivell de bit de la comunicació que establirem. Després descriurem l'estructura dels paquets de dades escollida i el mètode de comprovació d'errors que s'impartirà. Ja per acabar, comentarem el sistema de comandes que s'utilitza i els diagrames de flux de dades

3.1 Interfície RS-232

La capa física del port sèrie s'anomena RS-232[4]. Aquest Standard especifica un connector DB-9 de 9 pins, dels quals només n'utilitzarem dos: el pin de recepció de dades, RxD, i el pin de transmissió de dades, TxD. La connexió es realitza mitjançant un cable "Null-Modem" .

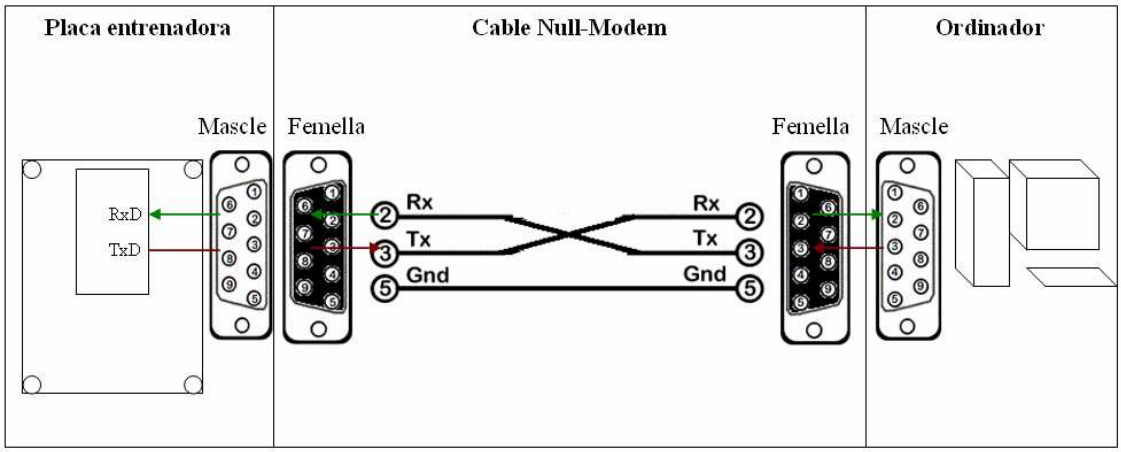


Fig. 3.1: Esquema de la connexió entre la placa entrenadora i l'ordinador.

3.2 Estructura de dades a nivell de bit

La comunicació sèrie que utilitzarem serà asíncrona. i utilitzant el dispositiu UART (Universal asyncronous reciver transmision). Per tal de sincronitzar la transferència de dades entre l'ordinador i la placa entrenadora cal establir unes normes. Podem dividir aquestes regles en dues parts:

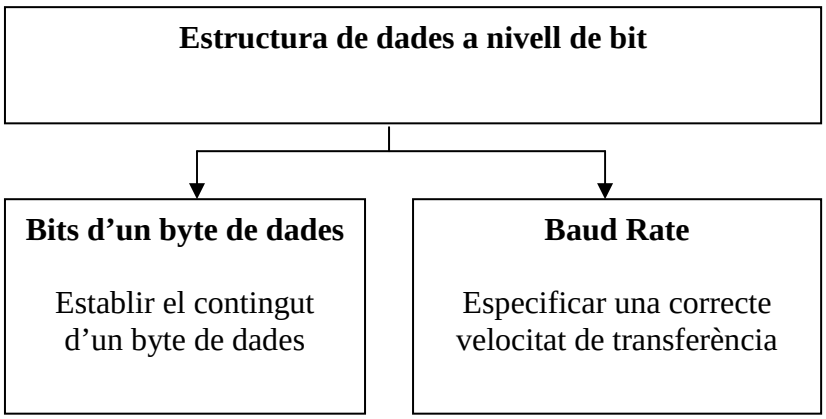


Fig.3.2: Parts de l'estructura de dades a nivell de bit

[4] A l'apartat 1.3 de l'annex hi ha l'esquema de pins dels connectors de la interfície RS-232.

3.2.1 Bits d'una byte de dades

En la comunicació asíncrona, cada dada està composta per una sèrie de bits. Aquests, en ordre d'enviament, són:

- 1 bit d'inici
- de 5 a 9 bits bits de dada
- 1 bit de paritat (opcional)
- 1 o 2 bits de parada

Exemple d'una transmissió de 5 bits de dades, un bit de paritat parell i un bit de parada:

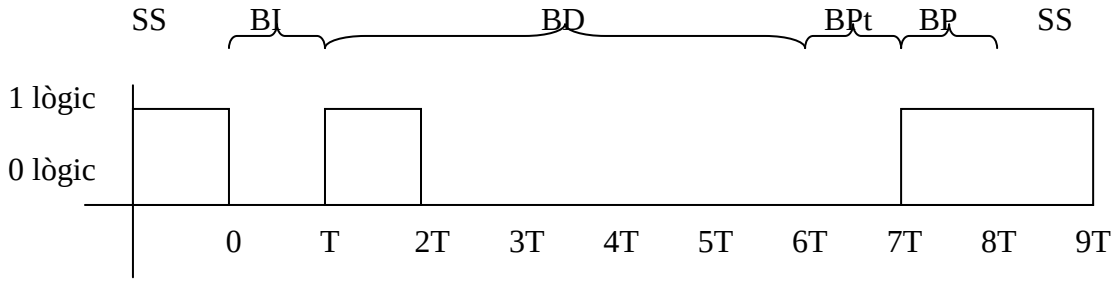


Fig.3.2.2: Gràfic d'un exemple de transferència d'una dada

SS: No s'envia cap dada.
BI: Bit d'inici
BD: Bits de dades
BPt: Bit de paritat parell
BP: Bit de parada
T: unitat de temps de recepció/enviament d'un bit.

■ El bit d'inici (Start bit): Aquest bit indica que seguidament es rebrà una dada i s'indica com una senyal de 0 lògic.

■ Bits de dades: La dada podrà contenir entre 5 i 9 bits, ja que la capacitat del buffer del microcontrolador és de 1 byte. En el nostre cas, com que podem arribar a transmetre bytes de fins amb valor màxim de 0xFF, necessitarem 8 bits de dades.

■ El bit de paritat: El bit de paritat és opcional i se situa entre l'últim bit de dades i el primer bit de stop. El bit de paritat s'utilitza com un sistema de control d'errors. En funció de la quantitat de 1 lògics o 0 lògics que hi ha en els bits de la dada el bit de paritat pren 1 lògic o 0 lògic. No utilitzarem aquest bit perquè fem servir el sistema de CRC per la comprovació d'errors

■ Els bits de parada (Stop bits): el bit/s de parada indiquen al final de la dada i s'indica com una senyal de 1 lògic. Es poden seleccionar 1 o 2 bits de parada. En el nostre cas, utilitzarem 1 sol bit de parada perquè no s'ha vist motiu per incorporar-ne 2.

Així doncs, cada dada transmesa estarà formada per 10 bits. Els bits d'aquestes dades s'enviaran cada cert temps de la mateixa manera que es recolliran cada cert temps. El paràmetre que especifica aquest interval de temps és el Baud Rate. .

3.2.2 Baud Rate

El Baud Rate és la velocitat en que es transfereixen les dades. Les seves unitats de mesura són "bps" (bits per segon). El Baud Rate ha de ser el mateix per els dos dispositius comunicats en sèrie, i especificarà en quin interval de temps s'envien i es reben els bits que es transmeten.

A l'hora d'escollir un Baud Rate haurem de tenir en comte:

- S'haurà de tenir en comte el temps que tarden les operacions a produir-se entre dades rebudes, ja que els dos dispositius han d'estar preparats per enviar-les i rebre-les.

- Els Bauds Rates disponibles. A diferència del microcontrolador, les API's de Windows només permeten uns valors determinats de Baud Rate.

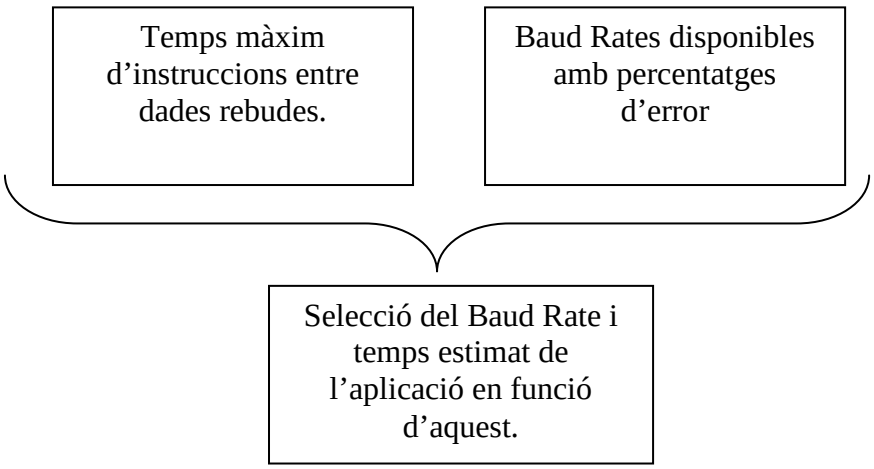


Fig. 3.2.2.1: Aspectes a analitzar per determinar el Baud Rate

- Temps màxim d'instruccions entre dades rebudes.

  Ara cal saber quin temps màxim poden arribar a trigar les operacions que es troben entre rebre una dada i una altra. Això és necessari perquè si utilitzem una velocitat molt elevada, pot ser que després de rebre una dada i mentre l'estem tractant, n'arribi una altra i no pugui ser tractada a temps. Per fer això:

  - Analitzem codi del microcontrolador
  - Trobem les operacions que tarden més a efectuar-se entre rebre una dada i una altra
  - Calculem els cicles màquina d'aquestes operacions: 152 cm
  - Calculem el temps que tarda a efectuar-se un cicle màquina en funció de la freqüència d'oscil·lació.

    1 cm / 14.7456 Mhz = 6.78·10-8 segons

  - Calculem el temps que tarda a efectuar-se el conjunt d'operacions:

    6.78·10-8 segons x 152 cm = 10.308 us

  - Calculem el Baud Rate màxim que podem utilitzar:

    1 bit / 10.308·10-6 ms = **97.01 Kbps**

- Baud Rates disponibles amb percentatge d'errors

  Ara consultem les velocitats possibles que podem utilitzar. Els Baud Rates que pot prendre el microcontroaldor tenen un percentatge d' error. Aquest percentatge d'error es deu a que el valor del registre del microcontrolador que especifica el Baud Rate (UBRR) a de ser un número enter. Si calculant aquest valor obtenim un nombre amb decimals, aquests representaran un error.

Baud Rate disponibles ordinador (bps)	Baud Rate microcontrolador (bps)	Probabilitat d'error (%)
300	300	0.0
600	600	0.0
1200	1200	0.0
2400	2400	0.0
4800	4800	0.0
9600	9600	0.0
14.4 K	14.4 k	0.0
19.2 K	19.2 k	0.0
-	28.8 k	0.0
38.4 K	38.4 k	0.0
56 K	56 K	-2.8
57.6 K	57.6 k	0.0
-	76.8 k	0.0
115.2 K	115.2 k	0.0
128 K	128 k	-2.8
-	230.4 k	0.0
-	460.8 k	0.0
-	921.6 k (màx. possible)	-7.8

Taula 3.2.2.2: Baud Rates de les API's de Windows, microcontrolador i percentatge d'error

Com que necessitem agafar un Baud Rate inferior a 97 Kbps podríem utilitzar 76.8 Kbps que té 0% de percentatge d'error. Però com que aquest valor no està disponible a les API's de Windows haurem d'agafar l'inferior.

- Selecció del Baud Rate i temps estimat de l'aplicació en funció d'aquest.

Com a resultat de l'anàlisi anterior, observem que hem d'agafar el Baud Rate de 57600 bps. Després de fer l'estudi sobre el temps de transferència i gravació del codi de programa[5] obtenim els resultats representats en la taula següent. El temps total serà igual al temps de transferència de les dades i el temps de gravació.

Baud Rate (Kbps)	**57.6**
Temps total estimat màxim per la transferència i gravació d'una pàgina.	**35,68 +- 1,4 milisegons**
Temps total estimat màxim per la transferència i gravació de totes les 120 pàgines	**3,83 +- 1 segons**

Taula 3.2.2.2: Temps de transferència i gravació del sistema.

[5] Aquest estudi està descrit a l'apartat 1.7 de l'annex

Observant l'estudi també veiem que:

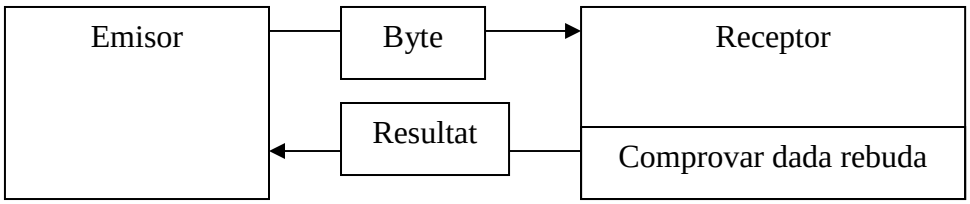
Temps gravació = 30.8% - 37.9% Temps de transferència.

Aquest temps serà aproximat perquè hi ha una quantitat de codi (referent a inicialitzacions i altres operacions) que no considerem del programador i del bootloader.

3.3 Estructura dels paquets de dades

Aquest apartat definim una estructura de paquets de dades per poder enviar-les correctament, amb un mètode de comprovació d'errors. Podem enviar les dades de dues maneres: per blocs o paquets, o byte a byte. Hem de considerar, però, que després de cada enviament hem de fer una comprovació d'errors.

Enviar dades byte a byte



Enviar dades en una trama

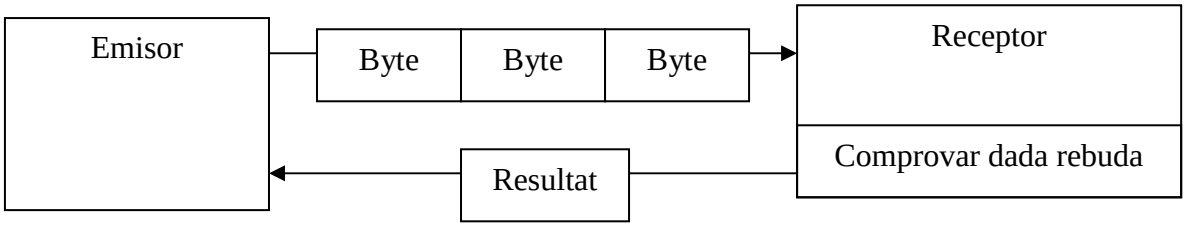


Figura 3.3: Forma d'enviar les dades

La forma en que s'ha decidit enviar les dades a anat sorgint a mesura que s'han anat creant els algoritmes del programador i del bootloader[6]. Hem partit d'un algoritme inicial, que s'ha anat modificant fins a arribar al algoritme definitiu.

[6] Aquests algoritmes estan exposats a l'apartat 1.5 de l'annex.

3.3.1 Selecció de l'estructura dels paquets de dades

- Primer algoritme

Característiques:

- Seguia un sistema de comandes, on l'ordinador donava una ordre al microcontrolador. Aquestes comandes eren:
    - Ordenar que es prepari per rebre la direcció de la dada que s'enviarà posteriorment.
    - Ordenar que es prepari per rebre una dada (un sol byte)
    - Ordenar que s'escrigui una pàgina,
    - Ordenar que es finalitzi l'aplicació

Problema:

- El sistema funcionava enviant les dades byte a byte i això tenia els següent inconvenients:
    - Per cada dada rebuda s'havia d'enviar el caràcter ACK per informar al programador que s'havia rebut correctament
    - S'hauria d'establir un control d'errors per byte rebut, si fos un CRC es rebria un byte més per cada dada i s'acabarien enviant el doble de bytes necessaris.

Solució:

- Reduir les comandes
- Enviar trames de dades reduint d'aquesta manera els bytes de comprovació d'errors.

- Segon algoritme

Característiques:

- Creació de trames per enviar els dades. Aquestes trames eren compostes per:
    - l'operador (el valor de la comanda)
    - la direcció inicial del primer byte
    - el conjunt de bytes de dades de la pàgina
    - un CRC de 16 bits: un "checksum" per fer una comprovació d'errors.

- Les comandes passaven a ser dues:
    - Ordenar que es prepari per rebre una trama de dades

    - Ordenar que es finalitzi l'aplicació

Problema:

- No es tenia en compte quan una pàgina només estava escrita pel principi i pel final, deixant un tros buit al mig. El que feia aquest algoritme era col·locar tots els bytes de dades seguits i per tant, de forma errònia.

Possibles solucions:

- Enviar trames completes de 128 bytes (pàgines completes), omplint amb els valors 0xFF(valor que pren la direcció quan no hi ha cap dada) els espais de la pàgina que estan buits.
    - Avantatges
        - Podríem treure la comanda escriure pàgina ja que després d'enviar la pàgina completa ja pot ser gravada
        - És molt més simple d'implementar
    - Desavantatges
        - Enviem valors (els valors buits) que no són necessaris d'enviar, per tant el temps de transferència serà més llarg i el percentatge d'error serà major
- Enviar trames de bytes consecutius però no pàgines complertes
    - Avantatges
        - Enviem només les dades necessàries més el CRC de 16 bits per trama.
        - No malgasta la vida de la flash inútilment (grava els valors mínims necessaris) i el temps de gravació és molt inferior al mètode inferior.
    - Desavantatges:
        - S'ha d'incloure la comanda d'escriure pàgina.
        - La llargada de la trama ha de ser indicada al microcontrolador

Solució escollida:

- Enviar trames de bytes consecutius però no pàgines complertes. S'escull aquest sistema perquè es reduirà el nombre de bytes transmesos. Al enviar només els bytes necessaris el sistema serà més ràpid.

Solucionant aquest problema obtenim l'estructura definitiva del programador.

Contingut de la trama

Cal especificar la llargada de la trama al microcontrolador per tal de que aquest sàpiga quantes dades rebrà, abans de rebre-les. Per fer això, inclourem el valor de la llargada a la mateixa trama, en una posició anterior a les posicions dels bytes de dades.

Vaig decidir organitzar la trama de la següent manera:

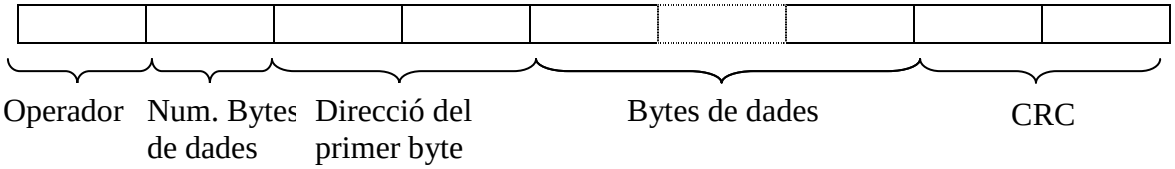


Figura 3.3.1: Organització d'una trama de dades

D'aquesta manera el bootloader, al rebre l'operador corresponent a la comanda "rebre una trama", la rep en el següent ordre:

- el número de bytes de dades,
- dos bytes de direcció,
- la quantitat de bytes de dades especificada en el número de bytes de dades rebut
- dos bytes més de CRC

Aquests bytes de CRC, que encara no hem explicat, fan referència a un sistema de comprovació d'errors. Aquest sistema es basa en calcular un valor a partir del valor dels bytes de la trama i adjuntar-lo a la trama. El microcontrolador, farà el mateix càlcul i comprovarà el valor rebut amb el valor calculat. Aquest valor s'anomena CRC i forma part del sistema de comprovació d'errors que farem servir.

3.3.2 Sistema de control d'errors

Per assegurar una bona recepció de les dades enviades pel programador, cal establir un sistema de control d'errors. Aquest sistema fa un control sobre les dades les dades que es reben i informa a l'usuari si s'han rebut correctament o no. Hi ha un gran ventall de sistemes basats en control d'errors[7]. El que es farà servir en aquest projecte serà el sistema de CRC o control de redundància cíclica perquè és un sistema senzill d'implementar i efectiu.

El mètodes de CRC es basa en afegir al final de les dades a enviar un nou valor de "n" bytes que representi la redundància cíclica d'aquestes. Aquest valor s'obté utilitzant un polinomi generador que pot ser de 12,16 o 32 bits en funció del tipus de CRC. La següent taula mostra els CRC més utilitzats:

[7] Alguns d'aquests sistemas els podeu trobar explicats a l'apartat 1.6 de l'annex

Tipus de CRC	Característiques
CRC 16 (16 bits)	- Polinomi generador : (x16+x15+x2+1) - S'utilitza generalment en controladors de hardware (drive-disk controllers. - Utilitzat a principalment a EUA. - Per transmissions d'entre 128 i 256 bytes
CRC CCITT (16 bits)	- Polinomi generador: (x16+x12+x5+1) - S'utilitza normalment en transferències IrDa (infraroig) i PPP. - Utilitzat a Europa principalment - Per transmissions d'entre 128 i 256 bytes
XMODEM (16 bits)	- Polinomi generador: (x16+x12+x5+1) - S'utilitza en el protocol XModem (Comunicacions amb mòdems) - Per transmissions d'entre 128 i 256 bytes
CRC12 (12 bits)	- Polinomi generador: x12+x11+x3+x2+x1 - Per transmissions de 6 bits
CRC32 (32 bits)	- Polinomi generador: $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$ - Per a transmissions molt llargues, de més de 256 bytes

Taula 3.3.2: Tipus de CRC

Com que les trames que enviarem seran de aproximadament d'uns 130 bytes s'utilitzarà un CRC de 16 bits. He escollit aquest CRC perquè buscant informació he trobat que el CRC que utilitza el protocol XMODEM és de 16 bits. Aquest protocol envia paquets de 128 bytes, una quantitat semblant al màxim de bits que pot arribar a transmetre el programador.

El CRC16 es podria utilitzar però com que el CRC CCITT i el XMODEM són més comuns a Europa, el descarto. La diferència de l'XMODEM i el CCITT només es troba en com són desplaçats els bits del CRC dins l'algorisme d'actualització del CRC. XMODEM desplaça i tracta primer el byte de més pes mentre que CCITT tracta primer el byte de menys pes. He escollit el CCITT com podria haver escollit XMODEM. El CRC allargarà la trama un 1,6 % aproximadament en cas d'una trama completa.

La idea principal de l'algorisme CRC és crear un valor per afegir al final de la trama de dades a enviar de tal manera que el polinomi corresponent a la trama més el CRC sigui divisible per el polinomi generador. D'aquesta manera, el receptor només haurà dividir la trama rebuda pel polinomi generador i verificar si és correcta o no depenent del residu.

Els passos teòrics a seguir per obtenir el CRC d'una dada són els següents [8]:

1- Afegir tants 0 a la dreta de la dada a enviar com el grau del polinomi generador
2- Dividir en mòdul 2 la dada amb els 0 afegits entre el polinomi generador obtingut i obtenir el residu
3- El residu és el CRC que es suma a la trama

Exemple:

Dada a transmetre: 1110001010 (x9+x8+x7+x3+x)
Polinomi generador de 4 bits :1111 (x3+x2+x1+1)

Afegim tants 0 com el grau del polinomi generador a la dreta de la dada
1- 1110001010-000
Dividim la dada pel generador en mòdul 2
2- 1110001010000 / 1101

```
1110001010000
1111
 0010
    1001
    1111
     1100
     1111
       1110
       1111
            1000
            1111
             111
```

3 – Restem, en mòdul 2, el missatge amb els 0 de més amb el residu de l'operació
4 – 1110001010000-111 = 1110001010111
5 – Enviem el resultat, que serà la dada més el CRC

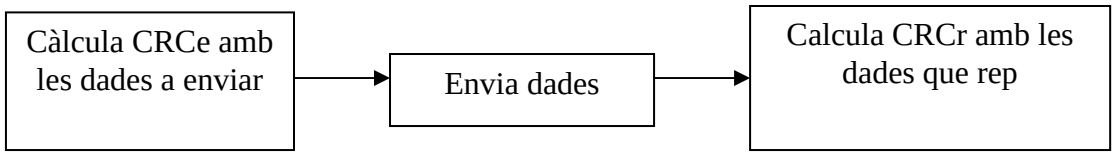
El codi del receptor ha de dividir en mòdul 2 la dada rebuda pel polinomi generador i si el residu és 0 la dada s'haurà enviat correctament.

En el nostre cas, per calcular el CRC implementarem una funció extreta de la llibreria "crc.h" del paquet de llibreries "AVR lib-c". Aquesta funció actualitza el valor del crc em cada dada que se li passa. La comprovació d'aquest CRC per part del microcontrolador serà diferent a l'explicació teòrica. Com que el microcontrolador rebrà trames de aproximadament 130 bytes com a màxim, no podem pretendre unificar aquests bytes i dividir el número resultant en mòdul 2, ja que serà molt llarg. Per tant, el que farem serà calcular el crc per les dades rebudes utilitzant la mateix

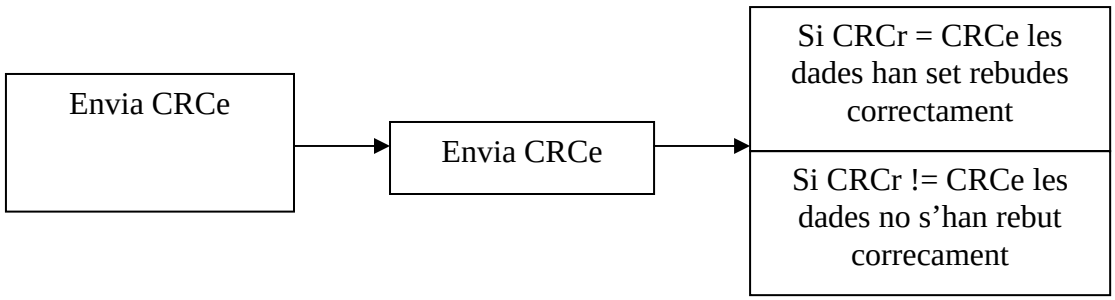
[8] Informació torbada en la direcció num. 11 de la bibliografia, capítol 7.

funció de la llibreria “crc.h”. Un cop calculat compararem el CRC rebut amb el CRC enviat. Els passos seran els següents:

1)



2)



3.4 Flux de dades: sistema de comandes

Un altre aspecte important que haurem de definir és el flux de dades. Aquest flux serà el llenguatge de comunicació entre l’ordinador i el microcontrolador i consistirà en unes preguntes i unes respostes.

Flux de dades

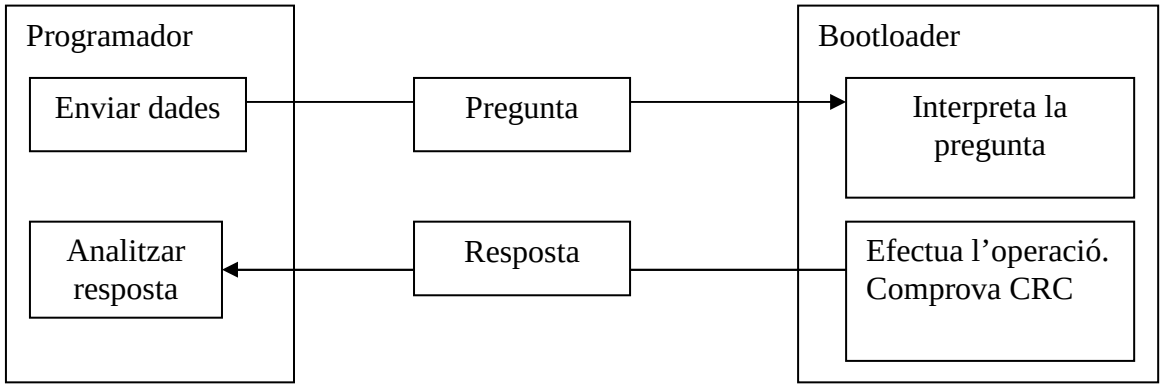


Figura 3.4.1: Funcionament del flux de dades

El sistema que s’ha creat per seguir aquest flux està basat en un sistema de comandes. És a dir, que un dels dispositius farà d’ordenant i l’altre d’operador. Després de que l’ordenant doni l’ordre a l’operador, esperarà una resposta. S’ha decidit assignar la tasca d’ordenant al ordinador i la d’operador al bootloader pels següents motius:

- L'ordinador treballa a més freqüència que el microcontrolador, la qual cosa fa que executi molt més ràpidament l'aplicació i pugui realitzar moltes més operacions que el microcontrolador en el mateix temps.
- Com més petita sigui la mida del bootloader, disposarem de més memòria de programa lliure. Això farà que puguem carregar programes de més longitud.

Ordres

Ordre	Caràcter	Descripció
Rebre la trama	STX (0x02)	El microcontrolador anirà rebent la trama pel port sèrie i l'anirà tractant
Escriure pàgina	ETX (0x03)	El microcontrolador escriurà la pàgina del buffer a la memòria flash
Fi de programa	EOT (0x04)	El micrcontrolador aturarà l'aplicació i saltarà a la direcció inicial de memòria de programa

Taula 3.4.2: Comandes o ordres del sistema de gravació

Decidir un valor o un altre per les comandes és indiferent, tot i que no podem utilitzar el valor 0x00 perquè és el valor que pren el buffer del microcontrolador si no es rep cap caràcter i confondria el sistema. S'ha intentat assignar valors d'acord a els que s'utilitzen normalment en comunicacions entre dispositius (STX, ETX i EOT). Aquests valors poden ser consultats en la taula de caràcters ASCII[9].

Resposta

Resposta	Caràcter	Descripció
Correcte	ACK (0x06)	Tot a anat correcte i la comprovació d'errors és correcte, es pot continuar l'aplicació
Incorrecte	NAK (0x15)	S'han rebut malament les dades, cal procedir segons el cas.

Taula 3.4.3: Respostes a les comandes del sistema de gravació

La resposta funcionarà de la següent manera:

- Si rep les ordres 0x03 o 0x04, opera i envia la resposta ACK.
- Si es rep una trama de dades (conjuntament amb l'ordre 0x02) se'n fa la comprovació d'errors. Depenent del resultat de la comprovació enviarem:
    - ACK: S'han rebut les dades correctament
        - Conseqüència: procedeix l'aplicació del programador

[9] A l'apartat 7 de l'annex hi trobareu una taula de caràcters ASCII

- NAK: S'han rebut les dades malament.
  - Conseqüència: El programador torna a enviar la trama fins a tres intents de rebre ACK. Si el tercer intent falla, finalitzem l'aplicació.

El valor de les respostes, s'ha tret de la taula ASCII.

Així doncs, el bootloader serà el servidor que esperarà constantment caràcters per procedir a efectuar operacions, mentre que el programador s'ocuparà de la major part de les tasques.

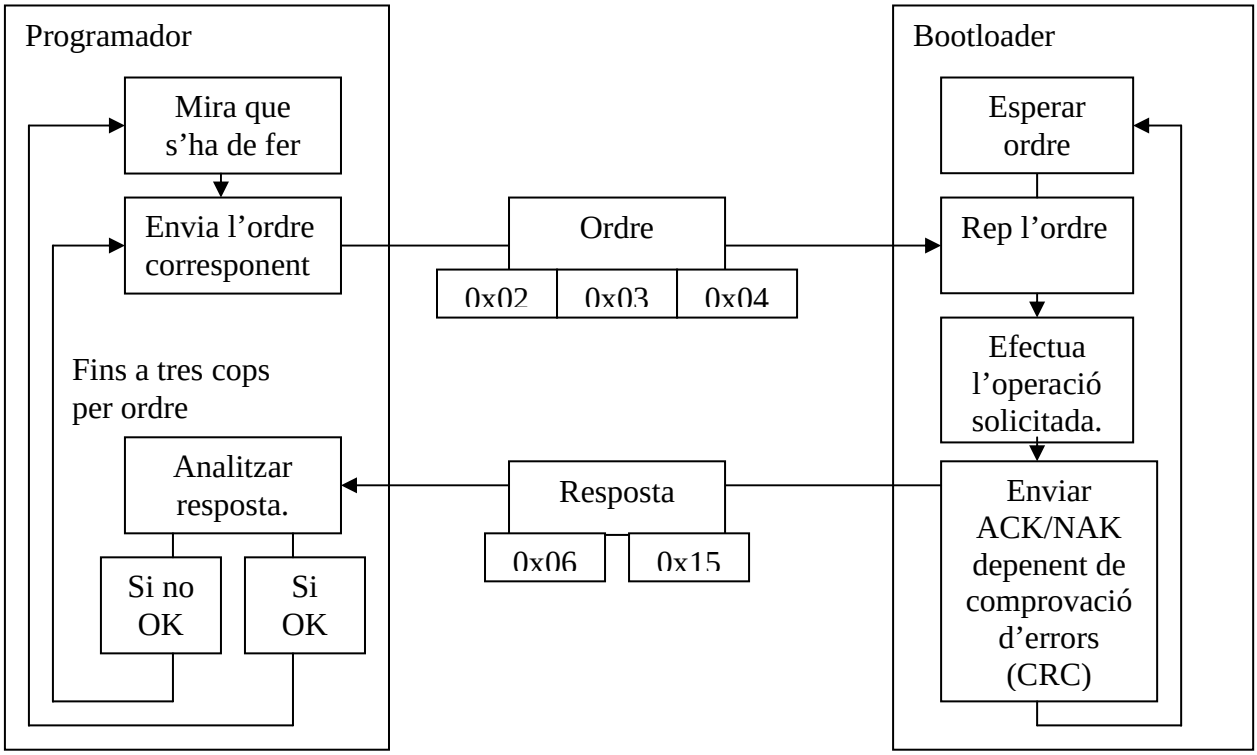


Diagrama 3.4.4: Funcionament del sistema de comandes

Ara que ja hem comentat tot el funcionament de la comunicació sèrie juntament amb tots els paràmetres i mètodes que intervindran, procedirem a començar el segon capítol. Aquest segon capítol tracta sobre el programador, el seu funcionament i la seva estructura.

4. EL PROGRAMADOR

El programador és el programa situat a l'ordinador que analitzarà un arxiu "hex", que conté el codi de programa del microcontrolador, i n'enviarà el seu contingut al microcontrolador utilitzant una comunicació sèrie asíncrona[10]. Aquestes dues tasques són les que farà el programador amb l'ajut de tot un sistema que haurem de dissenyar i estructurar. Podem dividir l'estructura del programador en diferents blocs.

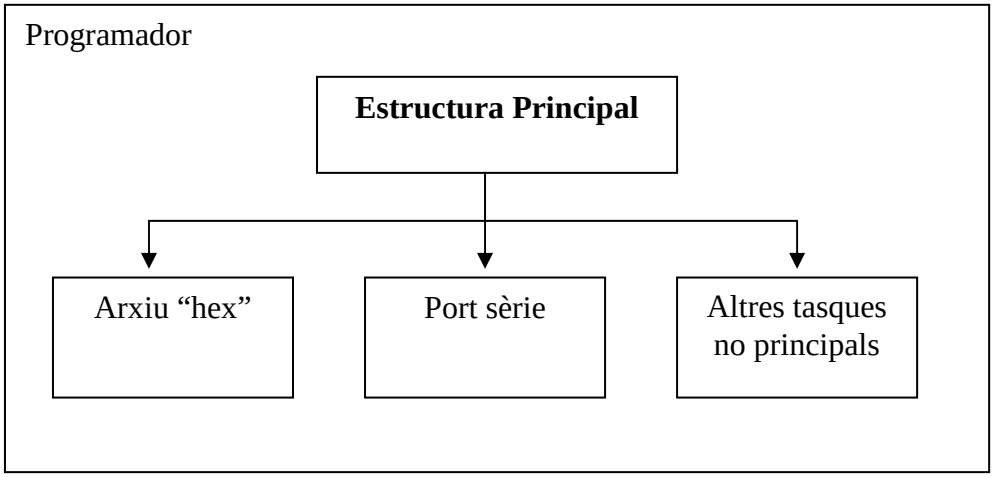


Figura 4: Blocs globals del programador

- **Arxiu "hex"** : Obrir i tancar l'arxiu hexadecimal, un sistema d'anàlisi del seu contingut, així com un control de possibles errors que s'hi puguin trobar.

- **Port sèrie**: Obrir, configurar i tancar el port sèrie, i enviar i rebre les dades per aquest.

- **Altres tasques no principals**: També trobarem un altre conjunt de tasques diverses, que faran altres operacions, potser no tan importants, però necessàries perquè el programa funcioni. Per exemple, fer el càlcul del CRC, operacions per preparar trames de dades,...

- **Estructura principal**: L'algoritme principal unificarà totes les parts i les cridarà en l'ordre corresponent.

Totes les operacions que faci el programador, juntament amb els errors i les dades que es transfereixin seran mostrades per la sortida "stdout" (Per defecte, és la pantalla de la consola). L'usuari podrà optar per un informe extens o breu.

[10] El codi de programa del programador està exposat a l'apartat 1.9 de l'annex

S'ha creat una interfície gràfica pel programador, utilitzant el complement WxWidgets de Dev-C que podeu consultar en la secció 4.4. Tot i no ser necessària pel programador, si que en facilita l'ús i mostra una visió més atractiva del sistema de gravació.
A l'apartat 4.5 enumerem els passos per executar el programador des de la consola[11].

Així doncs, primer de tot, cal conèixer el contingut de l'arxiu "hex", per tal de poder-lo analitzar i saber quines són les dades que hem d'enviar i quines no. Posteriorment , s'haurà d'esbrinar la forma d'accedir i manipular el port sèrie de l'ordinador. Per últim, haurem d'implementar tota l'estructura principal amb un algoritme que funcioni correctament.

4.1 Arxiu hexadecimal: anàlisi i estructura

L'arxiu "hex" es crearà quan compilem un programa pel microcontrolador, utilitzant l'AVR Studio de ATMEL. Aquest tipus d'arxiu pren el nom de INTEL16 i descriu, de forma organitzada i característica, el codi de programa que ha d'anar emmagatzemat a la memòria de programa del microcontrolador.

Dividirem aquest apartat en dues parts: una per explicar el contingut d'aquest arxiu, i l'altre per descriure la manera que utilitzarem per analitzar-lo.

4.1.1 Composició de l'arxiu hexadecimal

Per tal de saber que conté aquest arxiu hexadecimal, l'obrim com si es tractés d'un arxiu de text. Un cop obert, observarem que el seu contingut, que serà una cosa semblant a això:

:1000000012C02BC02AC029C028C027C026C025C0C6
:1000100024C023C022C021C020C01FC01EC01DC0DC
:100020001CC01BC01AC011241FBECFE5D4E0DEBF28
:10003000CDBF10E0A0E6B0E0ECE9F0E002C0059032
:100040000D92A036B107D9F710E0A0E6B0E001C0EC
:100050001D92A036B107E1F701C0D2CFCAE5D4E0C6
:10006000DEBFCDBFA7E0B0E00BD0802DFE01319602
:10007000A0E0B0E085E0182E0BD080E090E00DC04D
:10008000E199FECFBFBBAEBBE09A11960DB20895C9
:0C009000F7DF01921A94E1F70895FFCF0A
:00000001FF

Cada línia d'aquest codi representa un conjunt de bytes, representats amb valors hexadecimals. Cada fila està composta per diversos paràmetres formats per un conjunt de bytes cada un, i de la següent manera (exemple):

[11] Per més informació, podeu consultar el manual d'usuari del programador a l'annex 2.

: 10 – 0040 – 00 – 0D92A036B107D9F710E0A0E6B0E001C0 – EC

Aquests paràmetres són, en ordre d'esquerra a dreta

- **Byte d'inici:** ":" – Caràcter que especifica el principi de línia
- **Direcció:** "10" – El primer byte de la sèrie especifica el nombre de bytes de dades que conté la fila. En aquest cas són 16 (numeració decimal).

- **Adreça:** "0040" – Els següents dos bytes identifiquen la direcció de la memòria de programa on resideix el primer byte de dades de la fila. La direcció dels següents bytes de dades són consecutives.

- **Tipus de gravació: "**00" –Aquest byte especifica el tipus de gravació que es durà a terme. Hi ha fins a 5 tipus de gravació però els que ens interessen a nosaltres són:
    - "00" – Gravació de dades amb adreça de 16 bits: Conté dades de codi de programa.
    - "01" – Final de gravació. Normalment la fila pren el valor de ":00000001FF": Ens indicarà el final del fitxer hexadecimal

  El valor 2 només surt en la primera fila de programes compilats en assemblador amb l'AVRStudio. Aquesta fila conté valors nuls i alguns compiladors l'incorporen al principi d'un arxiu hexadecimal. Nosaltres la depreciarem.

  :020000020000FC

- **Bytes de dades: "**0D92A036B107D9F710E0A0E6B0E001C0" – Seqüència consecutiva de bytes de dades. Cada byte li correspondrà una direcció, que serà la consecutiva de la direcció del byte anterior.

- **CRC:** "EC" – És un "checksum" del conjunt dels bytes de la fila. Per obtenir-lo es resta de 0x100 el byte baix del resultat de la suma de tots els bytes de la fila
    - Exemple: :030245FFXX -> 03+02+45+FF = 149 -> 100 – 49 = B7
        - :030245FF**B7**

4.1.2 Anàlisi de l'arxiu hexadecimal

Per analitzar l'arxiu es crearà una funció que, un cop cridada, analitzi una fila i en retorni tots els paràmetres comentats anteriorment. Com que no és possible retornar sis variables (una variable per paràmetre) d'una funció, hem decidit situar-los en sis variables globals pels següent motius:

- Era simple d'implementar i pràctic.
- L'ordinador, a diferència del microcontrolador, disposa de memòria suficient per emmagatzemar variables
- Executa les funcions més ràpid, ja que no se'ls hi passen paràmetres

Tot i així, en programes complexos és aconsellable no utilitzar variables globals, ja que, si es produeix un error a causa d'una variable global, és més difícil trobar el punt del problema

Altres opcions que s'havien plantejat van ser:

- Retornar una taula o una matriu amb els sis paràmetres inclosos,
- Crear un objecte i atribuir-li sis atributs diferents (llavors ja utilitzaríem C++)
- Crear una estructura
- Passar els paràmetres per referència

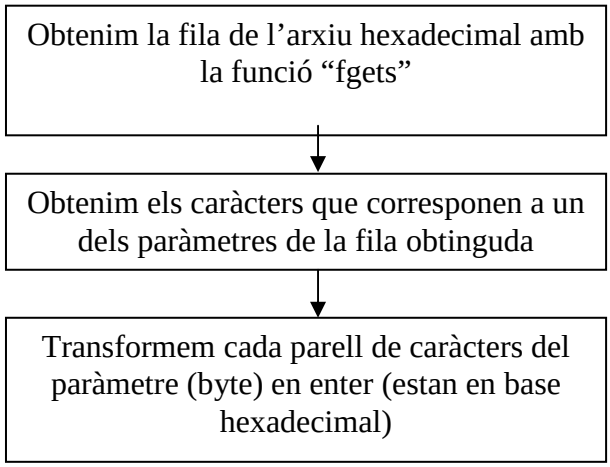
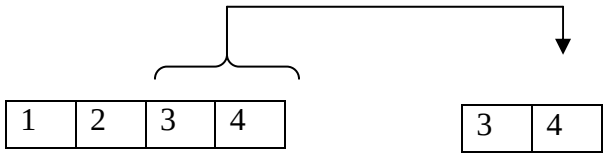


Diagrama 4.1.2.1: Obtenció dels paràmetres de l'arxiu hexadecimal

La següent taula mostra els passos seguits per obtenir els paràmetres del fitxer hexadecimal

Funció emprada	Funcionament	Inconvenient trobat
Getchar	Recollir cada caràcter i ajuntar cada parell de caràcters consecutius (ja que seria un byte)	Possible però era molt rebuscat
Funció pròpia	Retallava la fila per les posicions indicades en la funció	Buscant més es va trobar la tercera opció on es podia utilitzar una funció ja existent.
Strncpy	Copia un número de caràcters especificat d'una cadena a una altra. Com que la cadena a retallar se li com a apuntador, només cal incrementar aquest per retallar des de la posició desitjada.	

Taula 4.1.2.2: Passos seguits per obtenir el mètode d'obtenció de les dades de l'arxiu hexadecimal



Cal transformar aquesta cadena de caràcters en un valor numèric. La següent taula mostra les funcions que es van implementar per trobar-ne una de adequada:

Funció emprada	Funcionament	Inconvenient trobat
Atoi	Converteix caràcter ASCII en un valor enter en base 10	Les dades de l'arxiu hexadecimal estan en base 16 i la conversió no era possible
Strol	Converteix caràcter ASCII en una valor enter en la base indicada	Retorna un valor de tipus "long" i amb la tercera opció és possible retornar variables d'un amida inferior.
Sscanf	Llegeix d'una font(cadena de caràcters en el nostre cas) i la copia a una altre(caràcter sense signe) amb la conversió seleccionada ("%X" per convertir a hexadecimal)	

Taula 4.1.2.3: Passos seguits per obtenir el mètode de conversió de les dades de l'arxiu hexadecimal

La conversió però, era errònia per una simple raó. Faltava incorporar el caràcter de final de fila "\0" al final de la cadena copiada i llavors funcionava. La funció que en va resultar, doncs, retornava un enter equivalent a la conversió d'una cadena de caràcters a número en base 16.

S'aplica aquesta funció per cada paràmetre que volem obtenir de cada fila del fitxer hexadecimal.

4.2 Accés al port sèrie utilitzant les API'S de Windows

Accedir al port sèrie significa: iniciar-lo i configurar-lo, enviar dades, rebre dades i conèixer els errors que es poden produir.

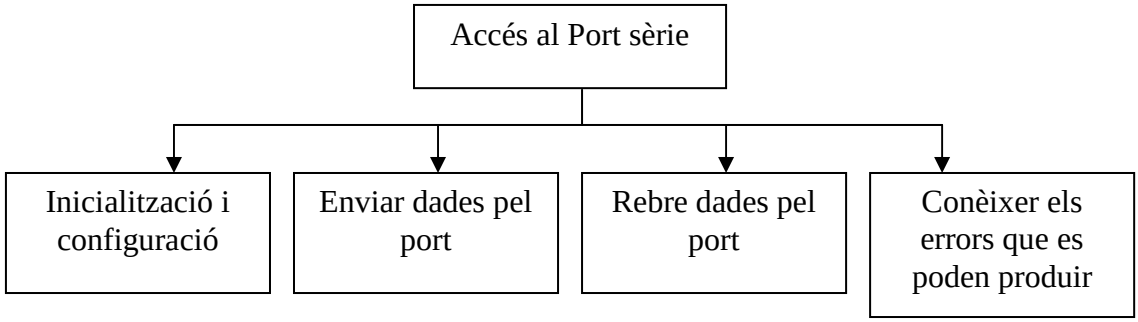


Figura 4.2: Accés al port sèrie

Abans d'explicar com accedim al port sèrie sobre el sistema operatiu Microsoft Windows, el qual farem servir, exposarem una petita introducció.

4.2.1 Introducció a les API's de Windows

Quan arranquem l'ordinador, la BIOS realitza una cerca de dispositius connectats al sistema i els hi assigna una direcció determinada en funció de l'ordre en que els ha trobat. Normalment les direccions dels ports sèries prenen els valors representats a la taula:

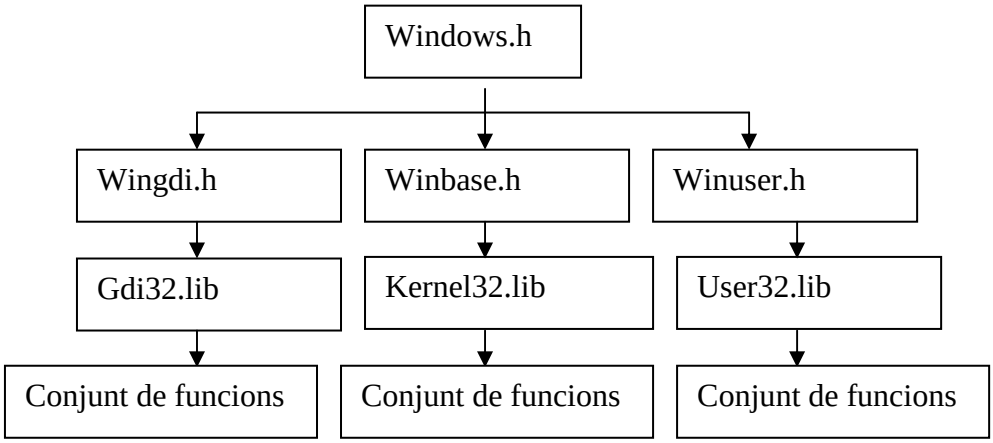
Port	Direcció en hexadecimal
COM1	3F8 – 3FF
COM2	2F8 – 2FF
COM3	3E8 – 3EF
COM4	2E8 – 2EF

Taula 4.2.1.1: Direccions dels ports sèrie

Tot i poder saber amb exactitud la direcció d'un port en concret (observant l'apartat corresponent del port en l'administrador de dispositiu de Microsoft Windows), el sistema operatiu Microsoft Windows no permet accedir-hi a través de les seva direcció.
La única forma de manipular i treballar el port és utilitzant funcions que utilitzin un conjunt de llibreries característiques de Microsoft Windows anomenades API's.

Les API's (Application Programming Interficie) de Windows són un conjunt de funcions d'aquest mateix sistema que permeten executar una aplicació sota la plataforma de Windows i accedir als diferents recursos que aquest disposa. Aquestes funcions prenen el nom de WinAPI (concretament Win32 per les versions de Windows95 o posteriors) i estan agrupades en llibreries dinàmiques (dll). Aquestes

llibreries estan referenciades per llibreries estàtiques com user32.lib, gdi32.lib i kernel32.lib. A l'hora de programar no és necessari incloure per separat cada llibreria, sinó que podem usar capçaleres que les agrupen com wingdi.h, winbase.h i winuser.h. Totes aquestes capçaleres de la API de Windows també estan agrupades en una única capçalera que s'anomena windows.h. Així doncs, només cal incloure windows.h al programa per tal de poder utilitzar totes les funcions de l'API de Windows. Tot i així, les funcions referents a la manipulació i configuració del port sèrie es troben a la llibreria Winbase.h.



Taula 4.2.1.2: Estructura de les API's de Windows

Per buscar informació sobre les funcions que hem utilitzant d'aquestes API's sobre el port sèrie ens hem adreçat a dues pàgines web:

- "Winapi Conclase"[12]. Aquesta web ens ha set de molta utilitat ja que, a part de contenir informació de les API's, també incorpora un curs en C i descriu una gran varietat de funcions.
- Web oficial de Microsoft Developer Network[13]. Conté tota la informació referent a les API's de Windows.

Feta aquesta introducció, explicarem com manipular el port sèrie de Windows i quines funcions s'han d'utilitzar

4.2.2 Comunicació sèrie via API's

La forma de manipular el port sèrie utilitzant les API's de Windows és tractant-lo com si fos un arxiu. Per això, el primer que s'ha de fer és obrir aquest "arxiu". Després caldrà configurar-lo per tal de que la comunicació sigui igual que la acordada amb el protocol. Llavors podem procedir a enviar(escriure) o rebre (llegir) dades pel port. I finalment, abans de tancar l'aplicació, tancarem el port. A part d'això, també es poden produir uns errors en concret quan utilitzem alguna funció referent al port, que caldrà analitzar.

[12] Per més informació podeu consular la direcció 3 de la bibliografia, capítol 7
[13] Per més informació podeu consultar la direcció 1 de la bibliografia, capítol 7

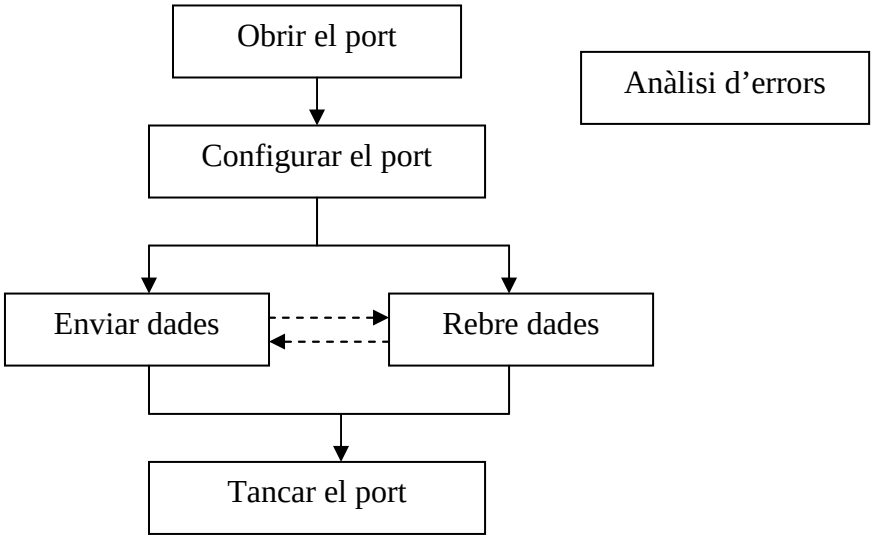


Diagrama 4.2.2.1: Diagrama de blocs de l'accés al port sèrie

Obrir el port

El port sèrie s'ha de definir com una variable de tipus "Handle". Un "Handle" no és res més que una variable que fa referència a un objecte de Windows.

Ara cal assignar la variable creada amb el dispositiu correcte, per exemple amb el port sèrie COM1. Per fer això s'utilitza la funció "CreateFile", que retorna una variable de tipus Handle. La funció "CreateFile" retornarà el valor INVALID_HANDLE_VALUE si no s'ha creat el port correctament. Per tant s'haurà de comprovar que no retorni aquest valor.

Configurar el port

Configurar el port sèrie és necessari per tal d'establir la comunicació asíncrona acordada, amb el Baud Rate determinat, amb la quantitat de bits de parada que s'ha decidit utilitzar, sense utilitzar el sistema de paritat,... . Però a part d'això també és possible configurar uns esdeveniments que indiquen quan el port rep dades noves, quan n'ha enviat, i moltes altres accions. Per altre banda, també caldrà definir uns temps màxims per rebre i enviar les dades. Dividirem aquesta secció en tres apartats: el primer farà referència als paràmetres que configurarem per establir la comunicació, el segon per especificar els esdeveniments i el tercer per definir els "time-outs".

- Configuració dels paràmetres de la comunicació sèrie

  Per definir els paràmetres de la comunicació hem de crear una estructura tipus DCB. Posteriorment, assignar a aquesta estructura la

configuració actual del port sèrie representat en el "handle" utilitzant la funció "GetCommState". La funció "GetCommState" rep els paràmetres del "handle" i la direcció on es troba l'estructura DCB que hem creat. Retorna 1 si s'ha fet l'assignació correctament i 0 en cas contrari, per tant, haurem de comprovar que la càrrega s'hagi efectuat amb èxit.

Un cop carregada la configuració del port a la variable DCB en podem modificar els seus paràmetres. Només destacar per sobre que serà necessari especificar el Baud Rate a 57600, el nombre de bits d'un byte de dades a 8, que només utilitzarem sols un bit de parada i desactivar el bit de paritat, així com el seu control.

La configuració quedarà de la següent manera:

configuracio.BaudRate = CBR_9600;
configuracio.ByteSize = 8;
configuracio.Parity = NOPARITY;
configuracio.fParity = FALSE;
configuracio.StopBits = ONESTOPBIT;

Per últim, cal assignar la nova configuració al port utilitzant la funció SetCommState i passant-li com a paràmetres el handle del port i la direcció de l'estructura DCB que hem configurat.

- Configuració dels esdeveniments

  És possible monitoritzar certs esdeveniments del port sèrie amb la funció SetCommMask. Aquesta funció ha de rebre el "Handle" del port i la màscara de l'esdeveniment a monitoritzar com a paràmetres.

  Aquesta funció retorna vertader si s'ha configurat la màscara correctament o per el contrari retorna fals, per tant s'haurà de comprovar el valor retornat

  La màscara "EV_ERR" no s'utilitzarà perquè ja es farà un control d'errors constantment en les operacions que en puguin causar. "EV_CTS", "EV_DSR", "EV_RING", "EV_RLSD" fan referència al *handshakng* per hardware, per tant és inútil utilitzar-les. Com que no fem servir cap caràcter especial d'esdeveniments tampoc és necessari "EV_RXFLAG". Tampoc és necessari l'esdeveniment per saber que s'ha enviat l'últim caràcter del buffer de sortida.

  Només interessaria utilitzar la màscara "EV_RXCHAR" per saber quan rebem una dada pel port sèrie. Comentarem més endavant la forma de detectar aquest esdeveniment. Tot i haver-lo utilitzat al llarg del projecte, al final, es va eliminar per poder utilitzar els "time-out" i poder implementar un control d'error per la lectura del port.

- Configuració dels time-outs

  Definir uns temps màxims d'espera de dades pel port sèrie ens serà molt útil en el cas de rebre el caràcter de comprovació que ens enviarà el microcontrolador un cop li haguem enviat una ordre. I per altre banda, també podem posar un "time-out" pels caràcters que enviem per seguretat. Per definir aquests "time-outs" crearem una estructura de tipus "COMMTIMEOUT". Aquesta estructura conté els objectes de la següent taula:

Paràmetre	Descripció	Valor	Perquè?
ReadIntervalTimeout	Temps màxim d'espera entre bytes en milisegons	Per defecte	En el nostre programa només rebem un byte (ACK) i per tant no té sentit establir un valor determinat per aquest paràmetre
ReadTotalTimeoutMultiplier	Temps màxim d'espera per un byte. La funció ReadFile esperarà aquest temps multiplicat pel número de bytes a rebre i retornarà (si no ha rebut les dades abans).	1000	Donarem un marge d'un segon però podríem donar fins a 100 milisegons (el byte que rebem el rebríem en un milisegon o menys)
ReadTotalTimeoutConstant	Temps extra d'espera de ReadFile (se suma amb el temps anterior per byte que es rebrà)	0	No ens interessa posar un temps extra perquè només rebem un caràcter i volem agilitzar el programa.
WriteTotalTimeoutMultiplier	Temps màxim d'espera per un byte. La funció WriteFile esperarà aquest temps multiplicat pel número de bytes a enviar i retornarà (sinó a enviat les dades abans).	1	Cada byte (10 bits) tardarà en transmetre's 0,17ms (a 57,6 Kbps). El mínim valor que pot prendre aquesta variable és 1 milisegon.
WriteTotalTimeoutConstant	Temps extra d'espera de WriteFile(se suma amb el temps anterior per byte que s'enviarà)	0	Si el valor pren 0 no hi haurà aquest temps d'espera extra.

Taula 4.2.2.2: Objectes de l'estructura COMMTMEOUT

Un cop definits aquests paràmetres per l'estructura "COMMTIMEOUT" creada, utilitzarem la funció SetCommTimeout per guardar la configuració d'aquesta estructura al port sèrie.

Enviar dades

Per enviar una dada o conjunt de dades pel port sèrie s'utilitza la funció "WriteFile". Aquesta funció retorna fals si s'ha produït un error en l'escriptura o cert en cas contrari.

La Següent taula mostra els paràmetres que se li han de passar consecutivament, la descripció i el valor que prendran.

Paràmetre	Descripció	Valor	Perquè?
hFile	Handle a on s'escriurà	Port	És el "handle" on volem escriure
lpBuffer	Apuntador a la variable que conté les dades a escriure. S'anirà incrementant en funció de t	Trama	És la cadena de caràcters que volem enviar pel port sèrie
nNumerofBytesToWrite	Número de bytes a enviar	t	t serà la variable referent a la quantitat de caràcters (bytes) de la trama
lpNumberOfBytesWritten	Apuntador a la variable de número de bytes enviats	&n	n serà un DWORD que mostrarà la quantitat de caràcters que s'han enviat un cop retorni la funció
lpOverlapped	Estructura que permet desplaçar la posició inicial d'escriptura en el Handle	NULL	En dispositius de comunicació no té sentit. Encara que el valor de desplaçament fos especificat s'ignoraria

Taula 4.2.1.1: Paràmetres de la funció WriteFile

Definint la funció amb aquests paràmetres s'escriuran les dades al buffer de sortida i quan estigui ple, o no n'hi hagin més per enviar, s'enviaran.

També és possible enviar la trama utilitzant un llaç que es repeteixi t vegades (t bytes a enviar) i que vagi enviant els valors de les posicions de la trama cridant consecutivament la funció "WriteFile". Aquest mètode escriuria la dada al buffer i l'enviaria abans de rebre una altra dada.

El primer mètode és el més correcte i el que es fa servir en l'aplicació final. El segon mètode m'ha anat molt bé per fer proves de transmissió (enviar només uns quants caràcters de la cadena o posar un retard entre els caràcters enviats utilitzant la funció "Sleep").

En la nostre aplicació, crearem una funció anomenada "Enviar" que rebrà com a paràmetre l'apuntador inicial a la cadena de dades a enviar i la quantitat de dades que es volen enviar.

Rebre dades

Els únics valors per sèrie que haurà de rebre el programador seran els valors de comprovació del CRC. La funció "ReadFile" ens permet llegir una sèrie de dades del port. Com la funció "WriteFile", retorna fals si es produeix algun error i vertader si fa la lectura correctament.

Paràmetre	Descripció	Valor	Perquè?
hFile	Handle que es llegirà	Port	És el "handle" que volem llegir
lpBuffer	Apuntador a la variable on s'emmagatzema la dada provinent del buffer d'entrada. S'anirà incrementant en funció de nNumerofBytesToRead	valor	Cadena de caràcters on guardarem els bytes rebuts
nNumerofBytesToRead	Número de bytes a que volem rebre	1	Només necessitarem rebre un caràcter
lpNumberOfBytesRead	Apuntador a la variable de número de bytes rebuts	&n	n serà un DWORD que mostrarà la quantitat de caràcters que s'han rebut un cop retorni la funció
lpOverlapped	Estructura que permet desplaçar la posició inicial de lectura en el Handle	NULL	En dispositius de comunicació no té sentit. Encara que el valor de desplaçament fos especificat s'ignoraria

Taula 4.2.2.4: Paràmetres de la funció ReadFile

- Utilització d'esdeveniments en el port sèrie

  Tot i que no hem acabat utilitzant el sistema d'esdeveniments l'explicarem breument perquè l'hem usat al llarg del projecte. Com ja hem comentat, l'esdeveniment "EV_RXCHAR" ens permet detectar quan arriba una dada nova en el port sèrie. Per fer aquesta detecció, hem

de cridar la funció "WaitCommEvent". Quan es produeix un esdeveniment, aquesta funció retorna i guarda en una variable el valor de l'esdeveniment que s'ha produït. Llavors cal mirar si aquest valor és igual a "EV_RXCHAR" per confirmar que s'ha rebut una dada nova al port i procedir a fer-ne la lectura si és el cas.

Tancar el port

Per tancar el port sèrie només és necessari cridar la funció "CloseHandle", passant-li com a paràmetre el handle obert, que fa referència al port sèrie. La funció retornarà fals, en cas de que es produís un error (per exemple, si el port no està obert), i cert si es tanca correctament.

Totes les funcions comentades en els apartats de manipulació del port sèrie retornen fals si es produeix un error i per tan, s'haurà d'informar a l'usuari de l'error produït.

Anàlisi d'errors

Quan es produeix un error en les operacions comentades anteriorment, la funció corresponent retorna fals i l'error es guarda en un registre de la memòria. Aquest error serà substituït per els errors que es produeixen posteriorment per tant, cal analitzar-lo al moment que es produeixi. La funció per consultar l'últim error que s'ha produït és "GetLastError" i retorna el valor d'un error. La llista d'errors que poden aparèixer (descartant els errors procedents del port paral·lel, referents a l'ús d'impressores, i l'error de paritat) estan descrits en la taula següent:

Valor
CE_BREAK
CE_FRAME
CE_IOE
CE_MODE
CE_OVERRUN
CE_TXFULL
CE_RXOVER

Taula 4.2.2.5: Errors en la comunicació sèrie

Ara que hem descrit com manipular el port sèrie i com analitzar el fitxer hexadecimal procedim a explicar l'estructura principal del programador.

4.3 Estructura del programador

El programador ha de seguir una estructura o algoritme que permeti complir l'objectiu principal del programador; analitzar l'arxiu hexadecimal i enviar-ne el contingut pel port sèrie. Per arribar a l'algorisme final s'han passat per uns algorismes inicials. En l'apartat 3.3 s'expliquen els problemes que es van trobar en aquests algoritmes i les solucions que es van aportar per arribar a l'algoritme final.

Recordem un parell d'aspectes importants del programador per aclarir el seu funcionament:

- Les dades (referents al codi de programa de l'arxiu hexadecimal) s'envien en trames de bytes consecutius i que formen part de la mateixa pàgina.
- S'utilitza un sistema de comandes basat en tres ordres: Enviar una trama, enviar l'ordre d'escriure una pàgina i enviar l'ordre de finalitzar l'aplicació

Els passos globals que ha de seguir el programador són:

1. Inicialitza les variables i configura el port sèrie
2. Extreu una fila de l'arxiu hexadecimal
3. Analitza les instruccions i la seva direcció en grups de dos per saber si formen part de la mateixa pàgina
   - Si s'hi troben, enviem la comanda de gravar pàgina i esperem la resposta
   - Sinó, comprovem si són consecutives:
     - Si són consecutives les afegim al final d'una trama
     - Si no són consecutives enviem la trama amb la comanda d'enviar trama i esperem la resposta
4. Tornem al punt 1, extraient la següent fila del fitxer hexadecimal i repetim el procés fins al final del fitxer.
5. Si s'arriba al final del fitxer hexadecimal enviem la comanda de finalitzar l'aplicació i esperem la resposta

Dividirem el programador en 4 blocs. Aquests blocs estan compostos per un altre conjunt de blocs, explicats en els apartats posteriors en forma de pseudocodi. Per fer referència als noms d'algunes funcions les posarem entre parèntesis en la línia on hi ha l'operació que efectuen.

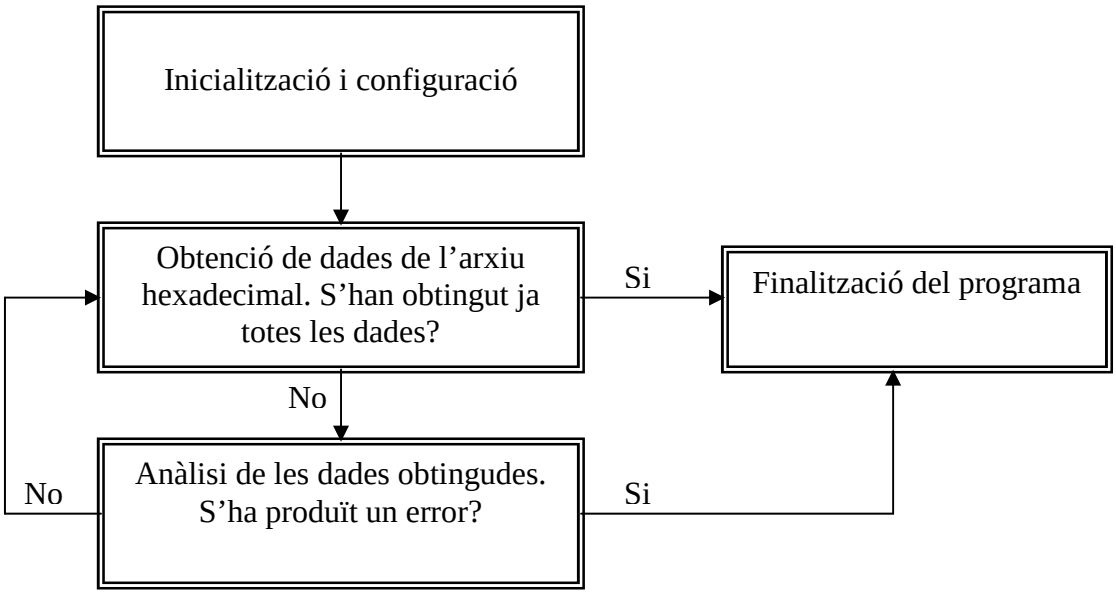


Diagrama 4.3: Diagrama de blocs general del programador

- Inicialització i configuració: En aquest bloc hi haurà el conjunt d'operacions que fan referència a la inicialització de variables i configuració del port sèrie
- Obtenció de dades de l'arxiu hexadecimal: En aquest bloc hi trobarem les operacions que extreuen els paràmetres de cada fila de l'arxiu hexadecimal.
- Anàlisi de les dades obtingudes: Aquest apartat serà el més extens. Analitzarem les dades per saber si són consecutives i pertanyen a la mateixa pàgina. En funció d'aquest anàlisi

4.3.1 Inicialització i configuració

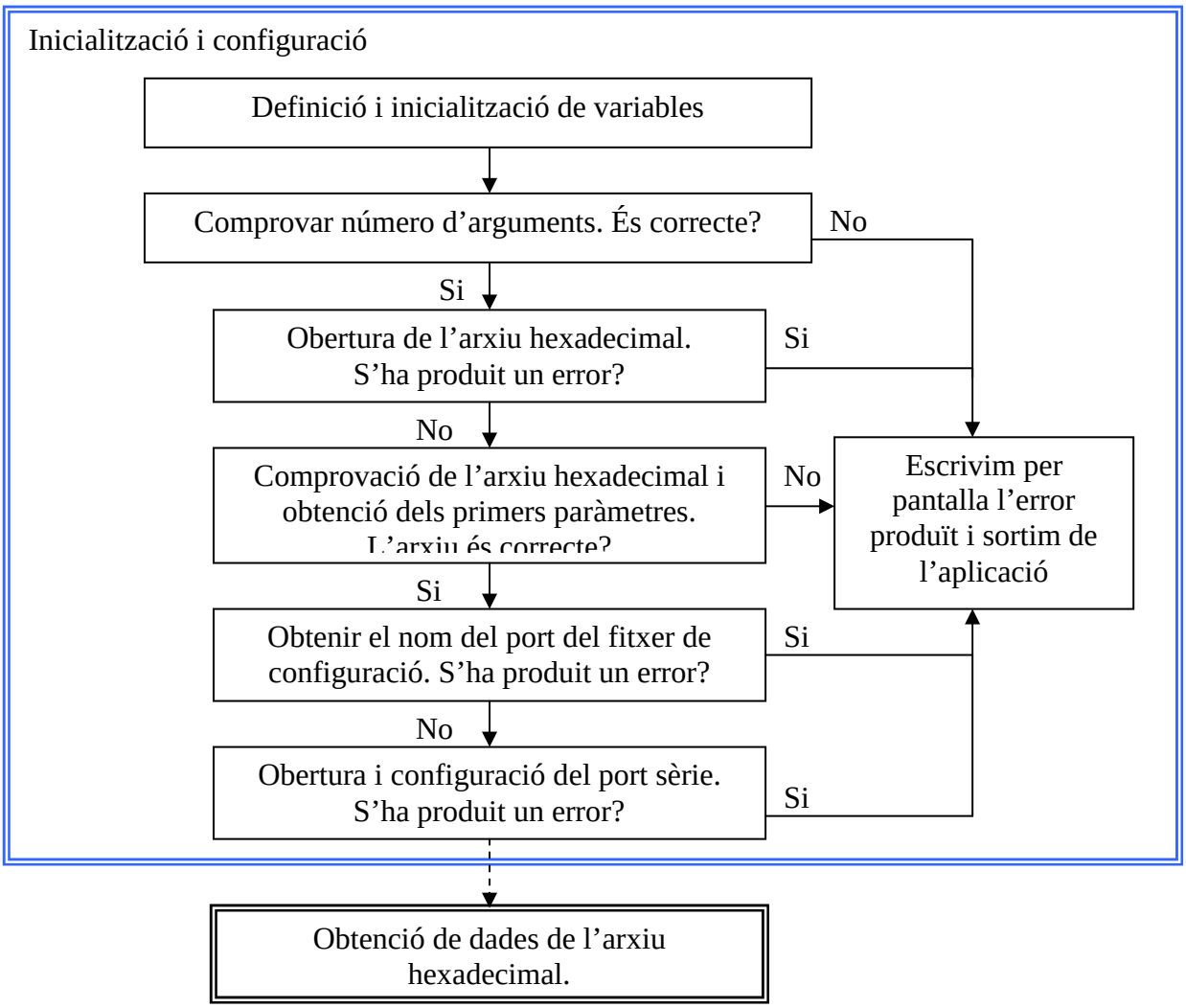


Diagrama 4.3.1: Diagrama de blocs de la inicialització i configuració del programador

- Comprovar número d'arguments

  Aquesta comprovació es fa per assegurar-nos que s'ha passat la direcció de l'arxiu hexadecimal com a paràmetre. El programa ha de tenir dos arguments: el número d'arguments i la direcció de l'arxiu hex.

  - *Si número d'arguments passats = 1 llavors*
    - *Informem de l'error*
    - *Finalitzem l'aplicació*
  - *Fisi*

Aquí també es comprovarà si l'usuari vol un informe extens. L'usuari haurà passat al programador un segon paràmetre en aquest cas (no importa el valor del paràmetre). Les escriptures en pantalla que només apareixen en l'informe detallat, estan impreses sobre un fitxer amb la funció "fprintf". Aquest fitxer serà igual a "stdout" (sortida per defecte) en cas de que es vulgui crear un informe extens. En cas contrari, serà un fitxer temporal, creat amb la funció "tmpfile", que s'esborrarà un cop finalitzi el programador.

- Si número d'arguments passats < 3 llavors
- informe = fitxer temporal (tmpfile)
- Sinó
- informe = stdout

- Obertura de l'arxiu hexadecimal

- Obrim el fitxer hexadecimal en mode lectura ("fopen")
- Si obertura incorrecta llavors
- Informem de l'error
- Finalitzem l'aplicació
- Sinó
-Retornem

Quan cridem la funció d'obrir l'arxiu hexadecimal li passem la variable que conté la direcció i nom de l'arxiu hexadecimal.

- Comprovació de l'arxiu "hex" i obtenció dels primers paràmetres

Aquesta funció s'ha creat pels següents motius:

- No podem permetre descarregar un programa que ocupi zona reservada per el bootloader, ja que gravaríem sobre aquest i ens espatllaríem el sistema.
- És necessari obtenir les dades de la primera direcció vàlida de l'arxiu hexadecimal, ja que posteriorment comparem les dades consecutives.

En aquesta funció trobarem dues tasques:

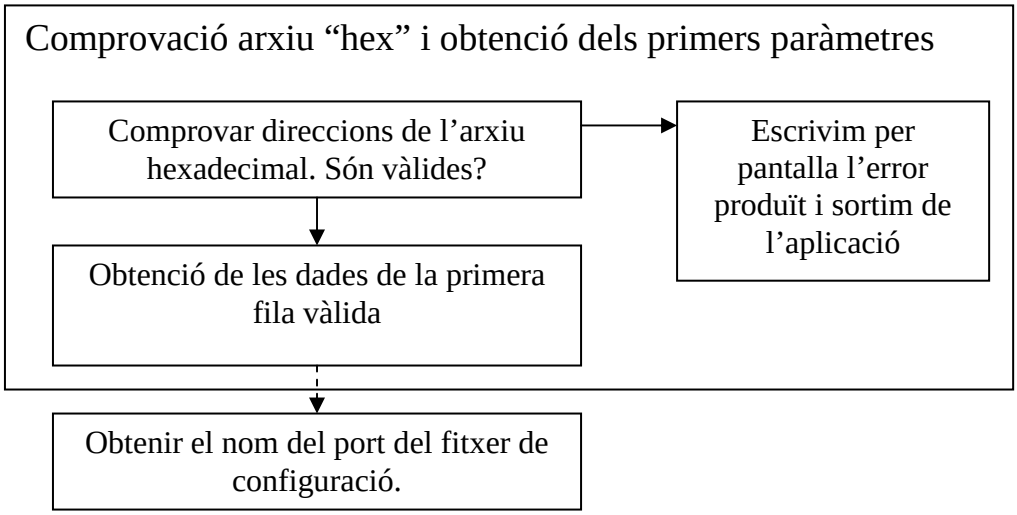


Diagrama 4.3.1.1: Diagrama de blocs de la comprovació de l'arxiu hexadecimal

- Comprovació de les direccions

  Mirem la última direcció de cada fila i donem error en cas de que superi l'adreça límit.

  - Mentre obtenir fila(fgets) != Final de fitxer (Null) fer
  - longitud = longitud + 1
  - Llegir fila(fgets)
  - Obtenir paràmetres de la fila
  - i = i+1
  -Si direcció + numero de bytes >= 0x3C00 llavors
  - Informem de l'error
  - Finalitzem l'aplicació
  Fisi
  - Fi mentre

- Obtenció de les dades de la primera fila

  La primera fila amb valors vàlids contindrà el valor 0x00 en el paràmetre de tipus de gravació. Aquest paràmetre li hem posat el nom de "final". Depenent del compilador que ha creat l'arxiu hexadecimal, aquest pot contenir una fila inicial no vàlida. Aquesta fila conté el valor 0x02 en el paràmetre de tipus de gravació.

  - Fer
  - Obtenir fila (fgets)
  - Obtenir paràmetres de la fila

- Mentre final != 0x00
- Rebobinar apuntador de l'arxiu (rewind)
- direcció anterior = direcció;
- Retornem de la funció

El que es vol és obtenir la direcció inicial per fer la comparació del primer byte amb ell mateix.

- Obtenir el nom del port del fitxer de configuració

Per obrir el port primer és necessari saber quin es vol obrir i quin nom pren. El nom del port està ubicat en el fitxer de configuració. Un cop haguem obtingut la cadena que conté el nom del port cal substituir l'últim caràcter, salt de línia (\n), pel caràcter final de cadena (\0) perquè no ens produeixi error a l'hora de llegir-lo.

- Obrim el fitxer de configuració en mode lectura ("fopen")
- Si obertura incorrecta llavors
- Informem de l'error
- Finalitzem l'aplicació
- Sinó
- Anem llegint els caràcters fins a trobar el caràcter '='
- Copiem el nom del port que bé a continuació (fgets) a una variable global
- Afegim el caràcter '/0' a l'ultima posició del nom del port
-Retornem

- Obertura i configuració del port sèrie.

Obertura del port sèrie:

- Obrim el port utilitzant el nom obtingut ("CreateFile")
- Si obertura incorrecta llavors
- Informem de l'error
- Finalitzem l'aplicació
-Fisi

Configuració dels paràmetres de comunicació

- Carreguem configuració del port sobre una estructura DCB (GetCommState)
Si apareix error en la obtenció de la configuració llavors
- Informem de l'error
- Finalitzem l'aplicació
-Fisi
- Configurem l'estructura DCB amb els paràmetres següents:

DCB.BaudRate = CBR_57600;
DCB.ByteSize = 8
DCB.Parity = NOPARITY
DCB.fParity = FALSE
DCB.StopBits = ONESTOPBIT

- Guardem la nova configuració del port ("SetCommSTATE")
- Si apareix error al guardar la configuració llavors
- Informem de l'error
- Finalitzem l'aplicació
-Fisi

Configuració dels "time-outs"

- Obtenim configuració dels "time-outs" d'escriptura/lectura del port sobre una estructura COMMTIMEOUT(GetCommTimeouts)
Si apareix error en la obtenció de la configuració llavors
- Informem de l'error
- Finalitzem l'aplicació
-Fisi
-Configurem l'estructura COMMTIMEOUT amb els paràmetres següents:
COMMTIMEOUT..WriteTotalTimeoutMultiplier = 1;
COMMTIMEOUT.WriteTotalTimeoutConstant = 0;
COMMTIMEOUT.ReadTotalTimeoutMultiplier = 1000;
COMMTIMEOUT.ReadTotalTimeoutConstant = 0;
- Guardem la nova configuració del port ("SetCommTimeOuts")
- Si apareix error al guardar la configuració llavors
- Informem de l'error
- Finalitzem l'aplicació
-Fisi

4.3.2 Obtenció de dades de l'arxiu hexadecimal

En aquest apartat hi inclourem el conjunt de funcions i operacions que fan les tasques de:

- Obtenir una fila de dades de l'arxiu hexadecimal
- Desagrupar la fila obtinguda en els sis paràmetres
- Comprovar que la fila contingui codi vàlid per analitzar-la
- Enviar comanda de finalitzar l'aplicació

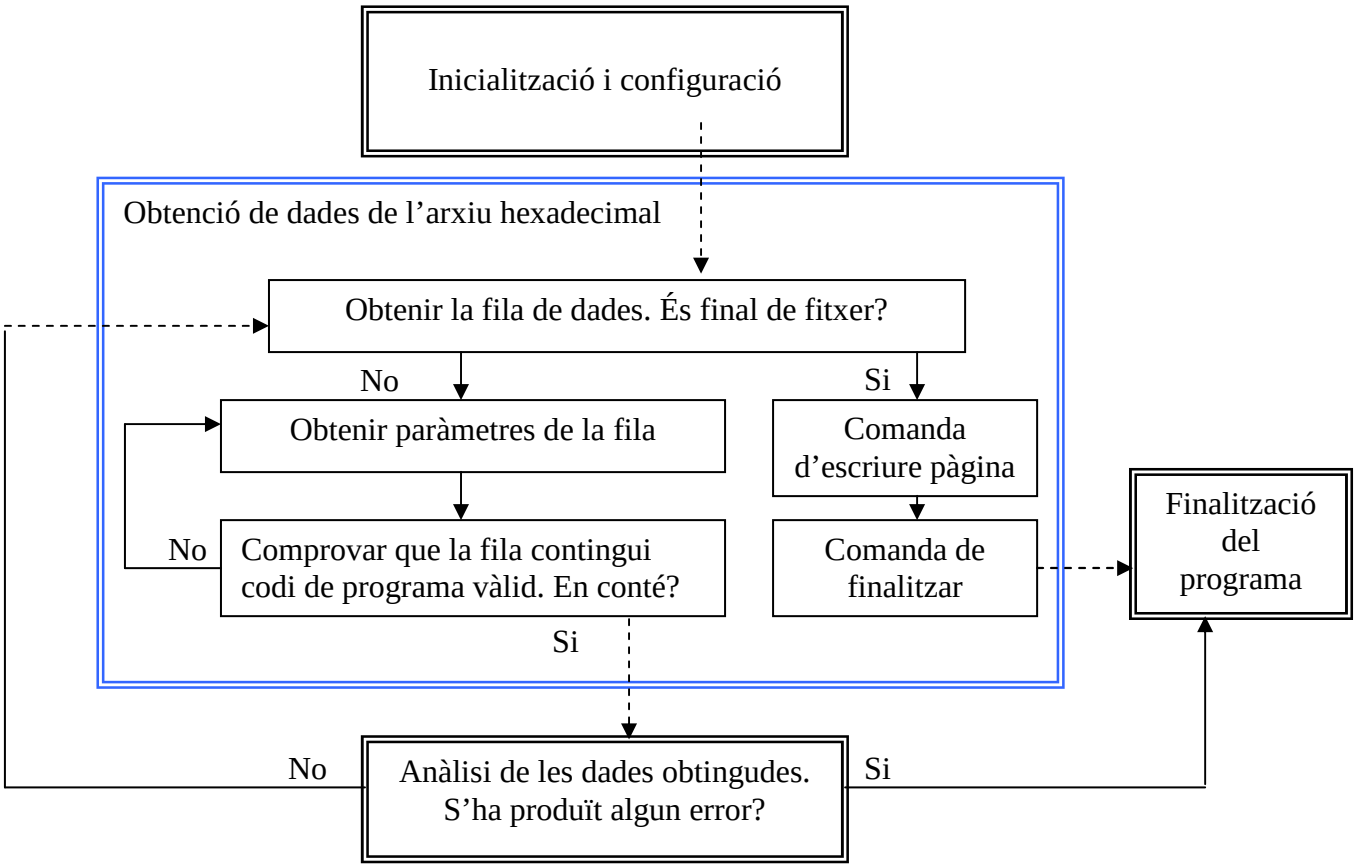


Diagrama 4.3.2: Diagrama de blocs de l'obtenció de dades de l'arxiu hexadecimal

- Obtenir la fila de dades.

  Cada cop que cridem la funció per obtenir una fila de l'arxiu hexadecimal n'obtindrem una de nova, ja que l'apuntador s'incrementa automàticament.

- Obrim la fila de l'arxiu hexadecimal (fgets)
- Si fila obtinguda = Final de fitxer (NUL) llavors

- Enviem Comanda "Finalitzar aplicació"
(Comanda_micro(0x04))
- Anem al bloc "Finalització de programa"
-Sinó
-Procedim a "Obtenir paràmetres de la fila"

- Obtenir la fila de dades.

La forma d'obtenir cada dada està explicada en l'apartat 4.1.2

- Obtenim número de bytes en variable "num_bytes"
- Obtenim direcció part alta en variable "dirh"
- Obtenim direcció part baixa en variable "dirl"
- Ajuntem les dues parts de la direcció en la variable "direcció"
- Obtenim tipus de gravació en variable "final"

Obtenim la cadena de bytes de codi de programa de la fila

- Mentre i < num_bytes
- Obtenim parella de bytes de codi de programa i els posem a la cadena "bytes"
- FiMentre

- Comprovar que la fila contingui codi de programa vàlid

Sempre que haguem d'analitzar una fila hem de comprovar que conté codi de memòria de programa. Això, en el nostre cas, suposa que el tipus de gravació de la fila pren el valor 0.

- Si "final"= 0 llavors
- Anem al bloc "Anàlisi de les dades obtingudes"
-Sinó
- Anem a inici del bloc "Obtenció de dades de l'arxiu hexadecimal

4.3.3 Anàlisi de les dades obtingudes

Aquest bloc és el més extens i tracta sobre l'anàlisi dels bytes de codi de programa obtinguts de cada fila del fitxer hexadecimal. El conjunt d'operacions que efectuen les tasques següents:

- Comprovar si:
  - Els bytes de dades es troben a la mateixa pàgina
  - Són consecutius

- Procedir a preparar una trama, enviar la trama o enviar la comanda d'escriure pàgina en funció de les comprovacions anteriors

Per aclarir possibles confusions, cal comentar que quan parlem de “byte actual” ens referirem al byte que s’està tractant en el moment. El “byte anterior” fa referència al byte tractat en el cicle anterior.

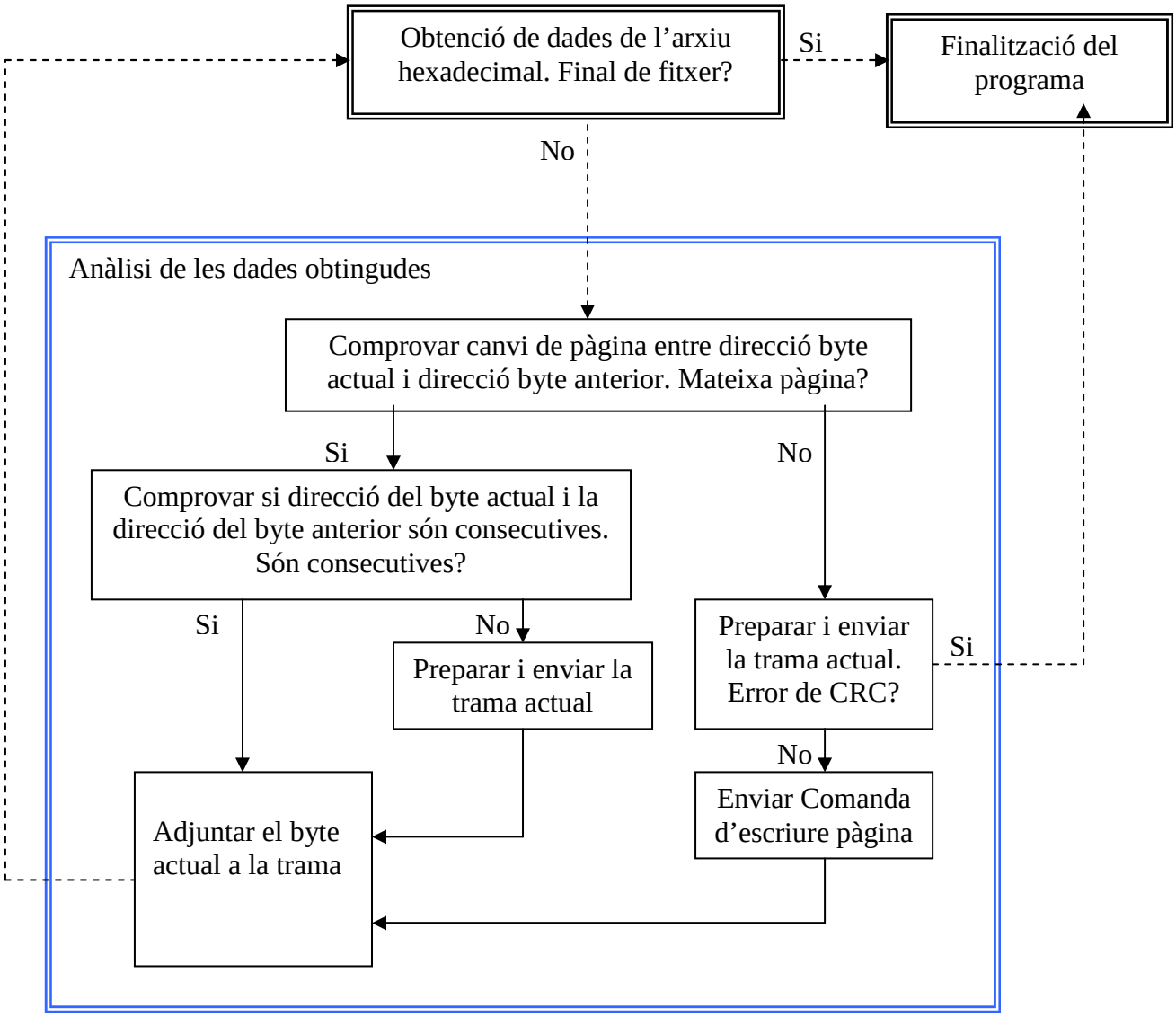


Diagrama 4.3.3: Diagrama de blocs de l’anàlisi de les dades obtingudes

- Comprovar canvi de pàgina entre les direccions del byte actual i anterior.

  Fent aquesta comprovació veurem si els bytes es troben a la mateixa pàgina. En cas que no es trobin a la mateixa pàgina haurem de procedir a enviar la trama guardada fins aquell moment, enviar l’ordre de gravar pàgina i començar una trama nova. Per observar la pàgina en que es troba una direcció, dividim aquesta pel número de bytes d’una pàgina, 0x80.

La funció que fa la comprovació és la següent i rep com a paràmetres les dues direccions que ha de comparar:

-Dividim direcció del byte actual per 0x80 (div)
-Dividim direcció del byte anterior per 0x80 (div)
-Comprovem si els dos resultats són iguals
- Si no són iguals llavors
- Procedim a Preparar i enviar la trama actual i a enviar la comanda d'escriure pàgina
- Sinó
- Comprovem si les direccions són consecutives
- Fisi

- Comprovar si les direccions dels bytes actual i anterior són consecutives

Si les direccions del bytes són consecutives significarà que, aquests bytes , han de ser enviats dins la mateixa trama. En cas contrari, hem d'enviar la trama guardada en aquell moment i començar-ne una de nova.

La funció que compara si les direccions són consecutives és la següent i rep com a paràmetres les dues direccions que ha de comparar:

- Si direcció byte actual = direcció byte anterior + 1 llavors
- Són consecutives, procedim a adjuntar el byte a la trama
- Sinó
- No són consecutives, procedim a enviar la trama
- Fisi

- Preparar i enviar la trama actual

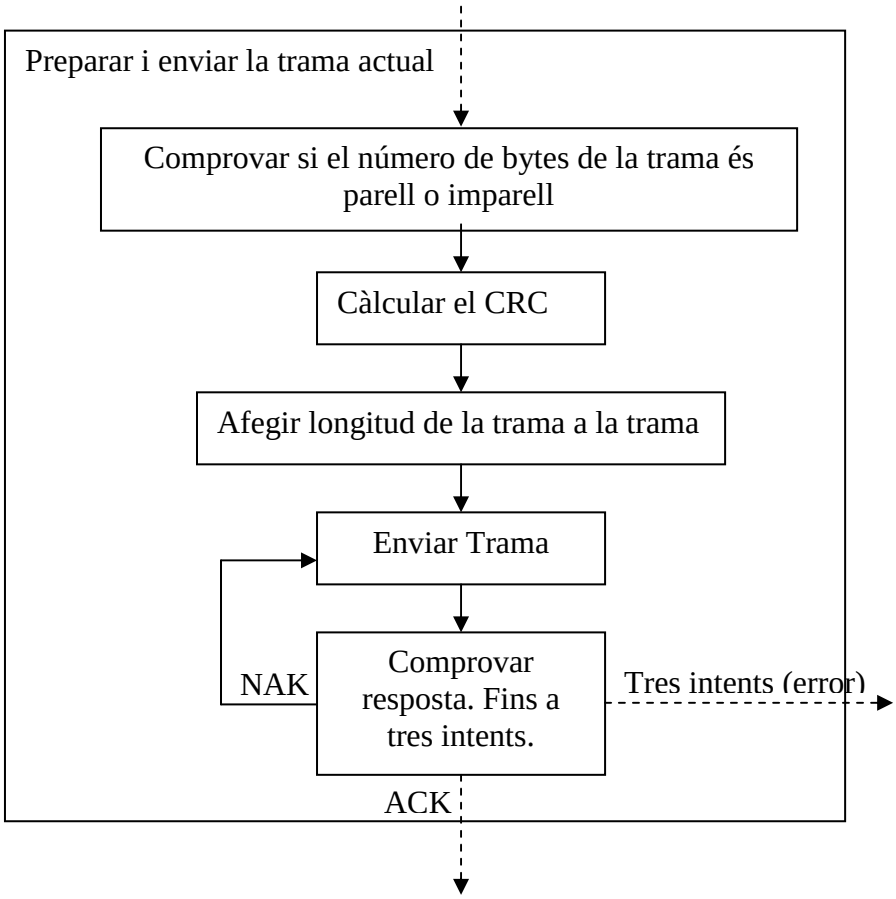


Diagrama 4.3.3.1: Diagrama de blocs de les operacions per preparar i enviar una trama

o Comprovar si el número de bytes de la trama és parell o imparell

Aquesta comprovació és necessària perquè necessitem enviar un número parell de bytes de codi al bootloader. El bootloader gravarà les pàgines de dos en dos bytes. Si el número és imparell caldrà afegir un valor 0xFF(buit) a la trama. Tot i així, les operacions de codi de programa estan compostes per parells de bytes. Per tan, normalment la trama tindrà ja un nombre parell de bytes de dades. Però per seguretat fem aquesta comprovació

- Adjuntem número de bytes de codi de la trama a la posició 1
- Dividim núm. de bytes de codi de la trama per 0
- Si el residu != 0 llavors
- Afegim 0xFF a última posició de la trama
- Fisi

- Calcular el CRC

Per calcular el CRC cridarem la funció "CRC_update" i li passarem com a paràmetre el CRC i una dada de la trama. El codi d'aquesta funció es pot trobar dins l'apartat de CRC de les llibreries "AVR lib-c"

- CRC = 0
- i = 0
- *Mentre i<longitud de la trama*
  - *CRC = CRCc_update(crc,Trama[i])*
  - *i = i+1;*
- *FiMentre*
- *Adjuntem crc a les dues últimes posicions de la trama*

- Enviar la trama

A part d'enviar la trama, comprovem també s'hi s'han enviat totes les dades.

- *Enviem la trama (WriteFile)*
- *Si apareix error al enviar la trama llavors*
  - *Informem d'error*
  - *Anem al bloc de Finalitzar l'aplicació*
- *Fisi*
- *Si número bytes enviats < número bytes de la trama*
  - *Informem de l'error*
  - *Anem al bloc de Finalitzar l'aplicació*
- *Fisi*

- Comprovar la resposta: ACK o NACK

La variable "reset", inicialitzada a 0, porta el comte d'intents per rebre el caràcter ACK.

- *Llegim del port*
- *Si arriba una nova dada l'analitzem llavors*
  - *Si dada != 0x06(NCK) llavors*
    - *Si dada = 0x15 (ACK) llavors*
      - *Informem de mala rebuda de dades*
    - *Sinó*
      - *Informem que a passat el temps d'espera*
    - *Fisi*
    - ***r**eset = reset + 1*
    - *Si reset = 3 llavors*

- Informem de que el tercer intent a fallat
-Anem a bloc "Finalitzar Aplicació"
- Fisi
- Tornem al bloc "Enviar Trama"
- Sinó
- reset = 0
- Continuem l'aplicació

- Enviar comanda d'escriure pàgina

La comanda "escriure pàgina" i la comanda "finalitzar aplicació" s'envien dins la mateixa funció "Enviar_comanda". A aquesta funció li passem el valor de la comanda que es vol. El mateix sistema de CRC també s'aplica al enviar una comanda per assegurar-nos que es rep correctament.

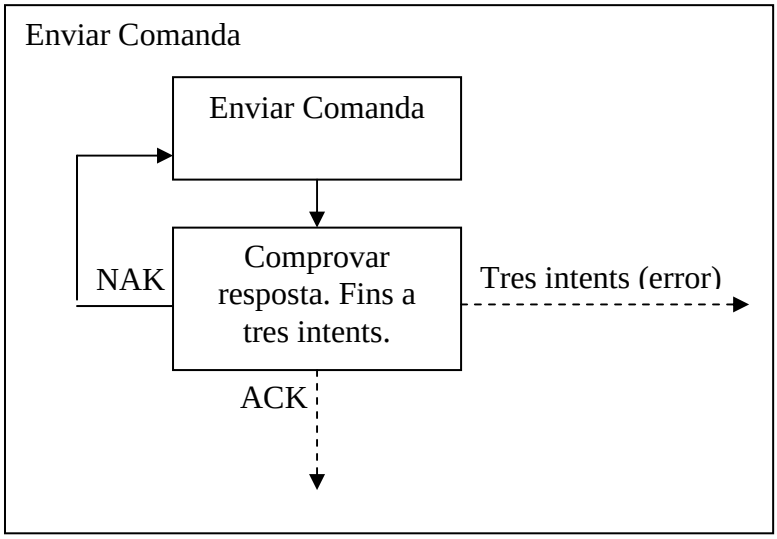


Diagrama 4.3.3.2: Diagrama de blocs per enviar una comanda

- Si valor passat = 0x03 llavors

- Fer
- Enviem la comanda d'escriure pàgina (WriteFile)
- Mentre Comprovar_resposta != ACK
- Sinó
Fer
- Enviem comanda finalitzar aplicació (WriteFile)
- Mentre Comprovar_resposta != ACK

- Adjuntar el byte actual a la trama

- Trama[última posició trama] = bytes[posició del byte actual]
- direcció anterior = direcció actual
- Incrementem "última posició trama"
- Incrementem "posició del byte actual" per tal d'analitzar el següent byte.

4.3.4 Finalització del programa

La funció que pertany a aquest bloc tancarà l'aplicació correctament. Cridarem aquesta funció passant-li el valor 1 com a paràmetre, si es finalitza a causa d'un error, o 0 en cas contrari.

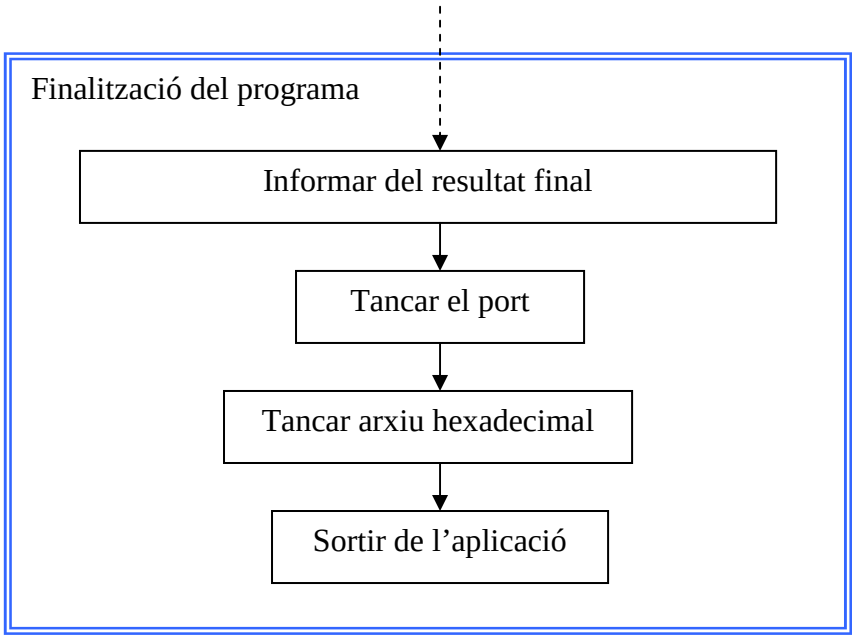


Diagrama 4.3.4: Diagrama de blocs de la finalització del programa

- Informar del resultat final

- Si paràmetre rebut = 1 llavors
- Informar de que el programa a finalitzat a causa d'error
- Sinó
- Informar que el programa a finalitzat correctament
- Fisi

- Tancar el port

- Tanquem el port sèrie (CloseHandle)
- Si no s'ha pogut tancar llavors
- Informem de l'error
- Fisi

- Tancar el fitxer hexadecimal

- Tanquem el fitxer hexadecimal (CloseFile)
- Si no s'ha pogut tancar llavors
- Informem de l'error
- Fisi

- Tancar el fitxer hexadecimal

Per sortir de l'aplicació utilitzarem la funció "exit". El valor que li passarem serà 0, si s'ha finalitzat correctament, o 1, si s'ha finalitzat a causa d'un error. Aquest valor que retorna el programador serà analitzat per la interfície gràfica.

- exit(valor)

Passem ara a descriure la construcció de la interfície gràfica pel programador.

4.4 Interfície gràfica

La construcció d'una interfície gràfica pel programador permetrà a l'usuari utilitzar-lo d'una forma més intuïtiva i donarà un aspecte visualment més atractiu al programa. Tot i així és un aspecte complementari del projecte, ja que la funció del programador i el seu funcionament, seran els mateixos.

Per dissenyar aquesta interfície utilitzarem un complement pel software Dev-c anomenat WxWidgets. Aquests objectes poden ser botons, formularis, caixetes de text,... o també poden contenir un conjunt de funcions determinades. WxWdigets també incorpora unes plantilles per la creació de nous projectes que permeten començar a programa amb un formulari ja creat. En el nostre cas crearem un projecte nou utilitzant una plantilla per crear una finestra de tipus "Frame" en comtes de "Dialog":

- Un "Frame" serà un objecte que representarà una finestra considerada com a principal,
- Un "Dialog", també és un finestra, però pensada per ser secundaria (sempre estarà per sobre de un "Frame" quan apareix-hi).

Al crear el projecte es crearan diferents arxius que són:

- Les capçaleres de l'aplicació i de l'objecte "Frame"
- Els arxiu C++ corresponents de les dues capçaleres
- Un arxiu de tipus "frm" on trobarem l'aspecte visual del "Frame" que hem creat i les barres d'eines per inserir nous objectes com botons, barres, caixetes de

text,... . L'editor que incorpora ens permet configurar les diferents opcions dels objectes creats i del formulari de forma interactiva. D'aquesta manera, al codi referent al paràmetre modificat de l'objecte és inserit automàticament pel programa.

El programa resultant de la interfície gràfica acaba sent en llenguatge C++ perquè s'utilitzen objectes. Tot i així, en el nostre cas només es crearan els objectes creats en l'editor i per tan, no s'aprofundirà en la programació orientada a objectes.

En el pròxim apartat, mostrem les pantalles de la interfície gràfica amb els seus objectes i explicarem com hem relacionat el programador amb aquesta interfície gràfica.

4.4.1 Formularis de la interfície gràfica

Finestra principal

Quan executem la interfície gràfica, ens apareixerà la següent pantalla principal

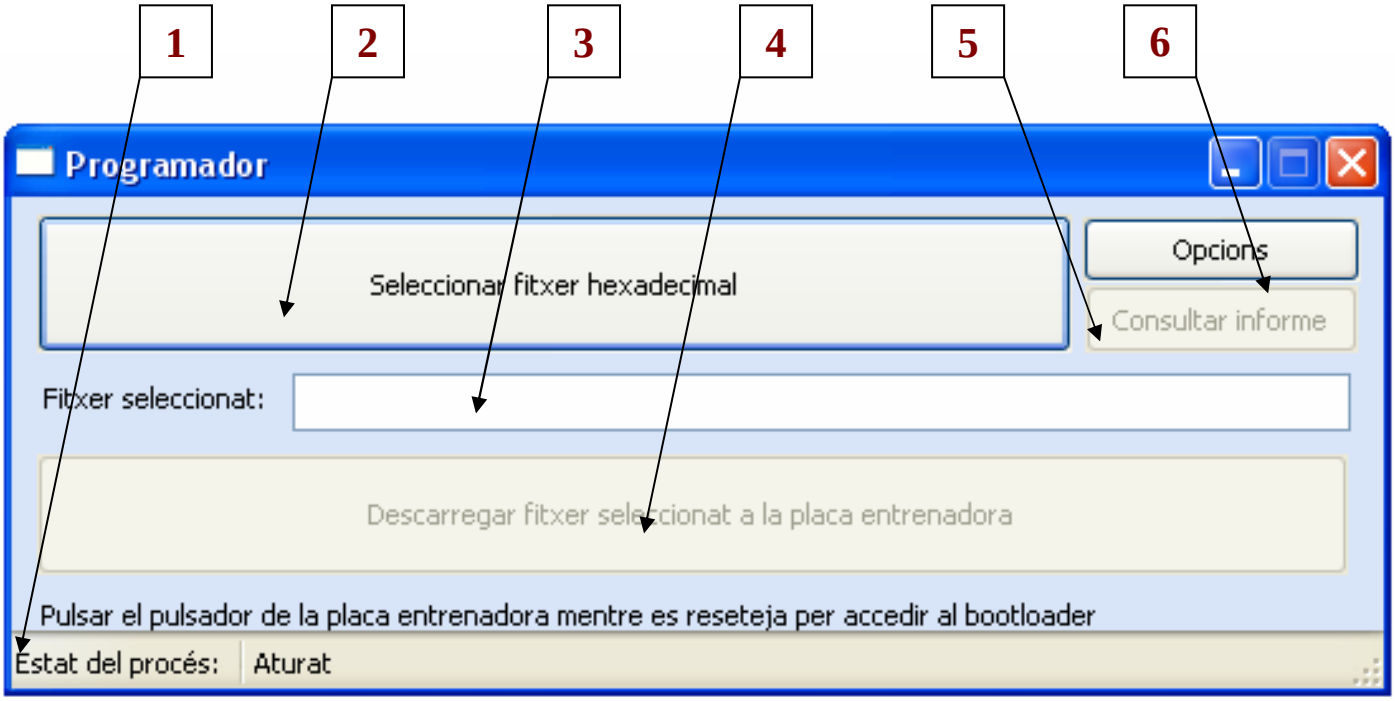


Figura 4.4.1.1: Finestra principal de la interfície gràfica

1 La barra d'estat servirà per indicar a l'usuari de la situació de la interfície gràfica. Quan executem la descarrega de l'arxiu hexadecimal, la finestra quedarà inhabilitada però es mostrarà el missatge "Efectuant descarrega..." en aquesta barra.

2 Botó per seleccionar l'arxiu "hex" a descarregar al microcontrolador. Crida un objecte anomenat "OpenFileDialog" que fa apareixer una finestra d'obertura de fitxers. El fitxer hex obert serè el fitxer seleccioat. També es necessari configurar el paràmetre corresponent perquè només es mostrin els arxiu amb extensió "hex".

3 Mostrarà el nom de l'arxiu seleccionat. La caixeta de text, ha de tenir el paràmetre READ_ONLY a true per tal de que no si pugui escriure.

4 Botó per cridar el programa principal del programador. Al clicar-hi executarem el programador passant-li la direccio de l'arxiu hexadecimal com a paràmetre. Aquesta direcció ha d'anar col·locada entre cometes perquè ens agafi tota la cadena com a un sol paràmetre. L'habilitarem només quan hi hagi seleccionat un arxiu "hex" (Per no causar errors).

5 Un cop l'executable del programador hagi finalitzat, habilitarem aquest botó. Aquest botó obrirà l'arxiu de text que s'haurà creat i que especificarà tots els passos que haurà anat seguint el programador juntament amb els errors, si n'hi ha hagut algún.

6 Clicant aquest botó accedirem a la finestra d'opcions (executarem un "Dialog") on es podrà seleccionar el número de port sèrie i l'executable del programador.

Finestra de configuració

Quan cliquem el botó "Opcions" de la pantalla principal accedirem a la finestra de configuració (**Figura 4.4.1.2**). Els paràmetres possibles a configurar són:

- El nom del port a utilitzar
- El nom de d'executable del programador
- L'opció de crear un fitxer informe i el nom que prendrà.

Tots aquests paràmetres estaran descrits en un fitxer de text anomenat "Programador.conf". Quan executem la finestra d'opcions carregarem aquests paràmetres del fitxer i els col·locarem a les caselles corresponents. Si cliquem "Acceptar" se sobreescriurà el fitxer "Programador.conf" amb els paràmetres seleccionats que hi hagi en aquell moment a la finestra d'opcions i la tancarem. Si, per contra, cliquem "Cancelar" tancarem la finestra sense reescriure el fitxer de configuració.

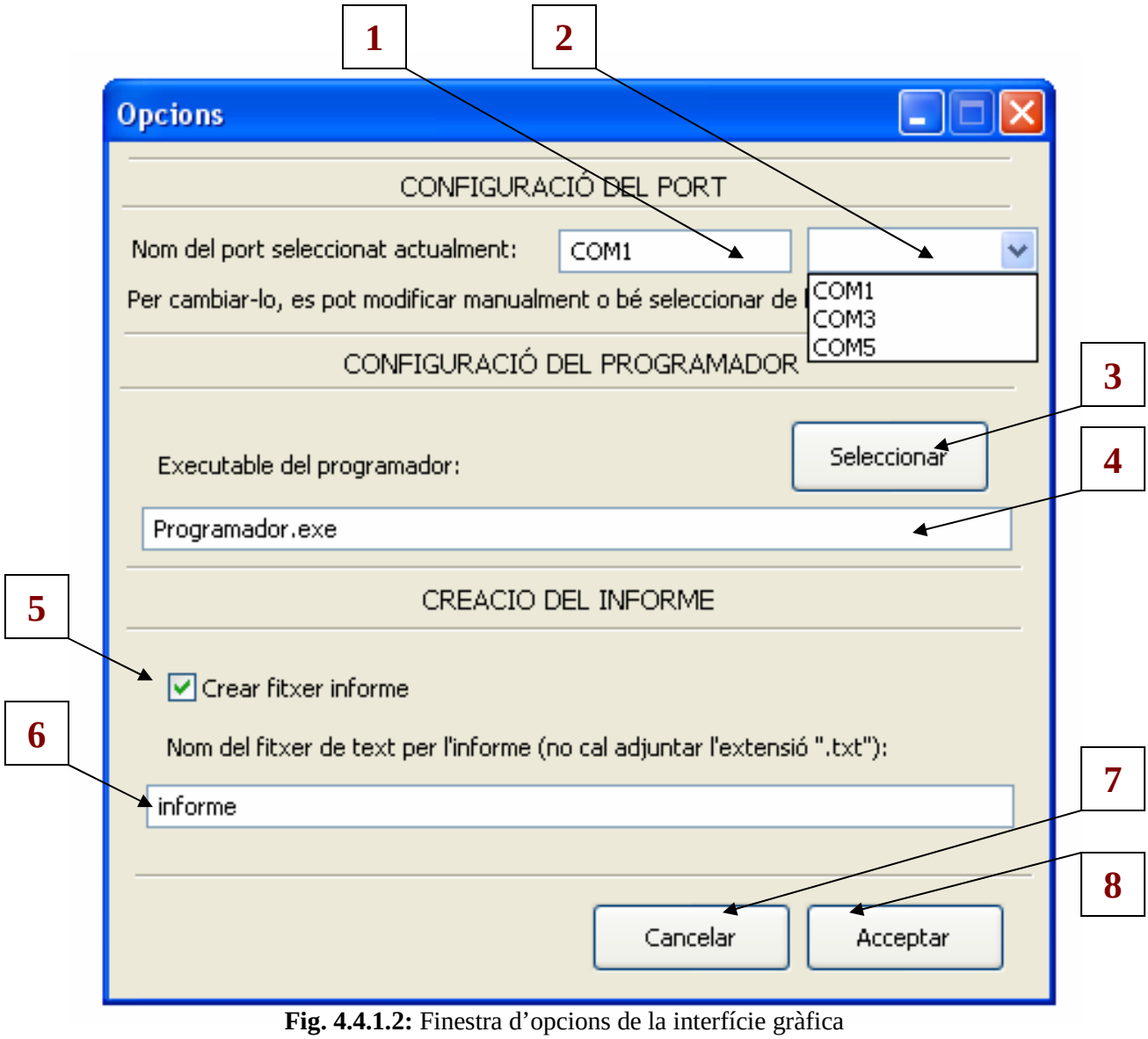


Fig. 4.4.1.2: Finestra d'opcions de la interfície gràfica

1 En aquesta casella es mostrarà el nom del port especificat en el fitxer de configuració "Programador.conf". Com que els noms dels ports sèrie poden ser modificats, farem que aquesta casella sigui modificable.

2 Aquesta llista mostra els ports disponibles. Es van plantejar diferents mètodes per trobar els ports sèrie del sistema. L'últim mètode que apareix a la **Taula 4.4.1.3** és el que utilitzem. L'inconvenient que té se soluciona deixant la casella del nom del port coma modificable. D'aquesta manera si l'usuari a cambiat el nom d'un port, i sap que hi és, el pot especificar.

Funció	Descripció	Inconvenient
GetPortNames	Retorna una llista amb els nom dels ports sèrie del sistema. Es troba a "System.IO.Ports.SerialPort"	No es va trobar la manera d'accedir a aquesta funció.
mode	Cridada des de de la consola del sistema. Mostra per stdout el nom dels ports del sistema i les seves característiques	Poc pràctic perquè hauríem d'analitzar aquesta sortida i ens apareixen tots els ports.
OpenFile	Intentem obrir sis ports sèrie amb el nom que prenen per defecte (ComX). Si s'obre el tanquem i el mostrem com a disponible.	Els noms del ports que apareguin seran els que els hi posa el sistema per defecte

Fig. 4.4.1.2: Mètodes per obtenir els ports sèrie del sistema

3 Obre la finestra de selecció per escollir l'executable del programador. Aquest ha d'estar dins la mateixa carpeta que la interfície gràfica per tal d'assegurar-nos que podrà obrir el fitxer de configuració. No obtindrem la direcció de l'executable escollit, només n'obtindrem el nom,

4 Apareix el nom del programador especificat. S'actualitza inicialment amb la direcció indicada en el fitxer de configuració. Aquesta casella no es modificable per tal d'assegurar que l'executable escollit existeixi.

5 Si està activat, quan l'executable del programador retorni, (S'hagi efecutat la descarrega de l'arxiu hexadecimal o s'hagi produit un error) es crearà un fitxer de text on s'especificaran tots els passos que ha anat seguint el programador. Això ho fem perquè l'usuari tingui la possibilitat de saber totes les operacions que s'han anat efectuant si ho desitja.

6 En aquesta casella mostrarem el fitxer informe especificat en l'arxiu de configuració. També serà modificable per tal de triar el nom d'aquest arxiu. Serà necessari incloure l'extensió per tal de que pugui ser obert correctament.

7 El botó "Cancelar" tancarà la finestra d'opcions deixant de banda si s'han modificat o no els paràmetres de les Caselles.

8 El botó "Aceptar" re-escriurà el fitxer de configruació amb els paràmetres que en aquell moment estiguin especificats en la finestra de configuració

4.4.2 Prevenció d'errors

Per evitar possibles errors en la interfície hem pres diverses mesures. Les mesures referents a la pantalla principal són:

- Habilitar el botó "Descarregar arxiu 'hex' només quan s'hagi seleccionat un arxiu hexadecimal.
- Habilitar el botó "Consultar Informe" només quan la descarrega hagi finalitzat i l'opció de crear el fitxer de l'informe estigui seleccionada.
- Informar amb un missatge a l'usuari si es clica el botó "Descarregar arxiu 'hex' " i no existeix el fitxer de configuració. En aquest cas retornem sense executar el programador.
- La selecció del fitxer hexadecimal la fem mitjançant la finestra d'obertura de fitxers per tal d'assegurar que l'arxiu existeixi.

Les mesures que s'han pres en la pantalla d'opcions són:

- Al entrar a la finestra de configuració i en cas de que el fitxer de configuració no existeixi, s'informarà a l'usuari de la creació d'aquest fitxer. Aquest contindrà uns paràmetres per defecte.

- La selecció de l'executable del programador la fem mitjançant la finestra d'obertura de fitxers per tal d'assegurar que aquest executable existeixi.

4.4.3 Relació interfície - programador

Hem decidit que la interfície gràfica i el programador fossin executables diferents pels següents motius:

- El programador es pot utilitzar per consola i senes dependre de la interfície gràfica. D'aquesta manera podem fer, d'una manera més àgil:
    a. redireccionar la sortida al fitxer que es vulgui
    b. passar l'arxiu hexadecimal manualment utilitzant la consola
- Sempre es pot crear una altra interfície gràfica que s'adapti millor al programador.

Tot i així també es va intentar inserir el codi del programador dins la interfície gràfica. Al cridar la funció del programa principal del programador ens vam trobar amb els següents problemes:

- El formulari quedava penjat fins que finalitzava.
- Si es cridava com a un subprocés (Thread), a l'hora de finalitzar teníem problemes perquè no quedaven totes les funcions del programador tancades.

Execució del programador i anàlisi del resultat

Quan es polsi el botó de descarrega, s'efectuaran les operacions representades en el següent diagrama:

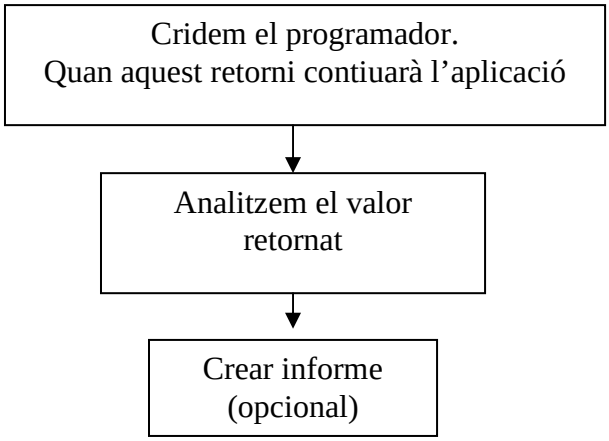


Diagrama. 4.4.3: Diagrama global de blocs de la interfície gràfica

- Crida del programador

  La funció que ens permet cridar l'executable del programador és "wxExecute" i rep els paràmetres següents (**Figura 4.4.3.2)**:

  - direcció del programador + direcció de l'arxiu hexadecimal: Les direcció que li passem (Tan de l'executable del programador com del fitxer hexadecimal) han d'anar entre cometes per ser interpretades com un sol paràmetre en cas que continguin espais en blanc.

  - Cadena de cadenes de caràcters on es farà la redirecció de la sortida "stodout" del programador

  - El valor "wxEXEC_SYNC": servirà per indicar que volem esperar que finalitzi el programador abans de continuar amb l'aplicació de la interfície.

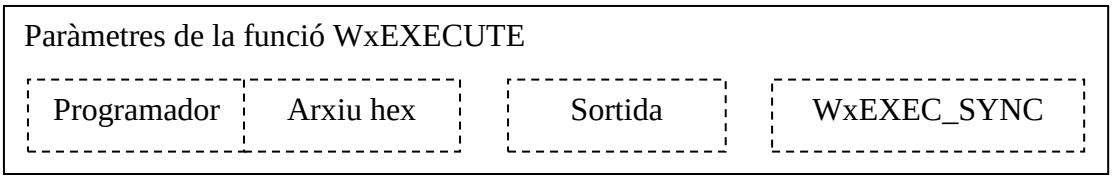


Fig. 4.4.3.2: Paràmetres de la funció WxExecute

L'aplicació continuarà un cop el programador hagi finalitzat i retorni el valor 0 o 1.

- Anàlisi del valor retornat

Quan rebem el valor de retorn del programador, el qual serà un valor que especificarà si hi ha hagut algun error(valor cert) o no(valor fals). En cas d'error caldrà analitzar quin és observant la variable on s'ha direccionat la informació. Com que la informació sobre errors la situem darrera el caràcter '#', només haurem de extreure les cadenes que continguin aquest caràcter en primera posició.

-Si valor retornat = 0
- Informe que la descarrega s'ha efectuat amb èxit
-Sinó
- i = 0
- Mentre i < longitud de la variable "sortida" fer
- Si sortida[i[1]] = '#'
- Adjuntar l'error a una cadena
- Fisi
- i = i+1
- FiMentre
- Informar dels errors mostrant la cadena creada
- Fisi

- Crear informe

Ja per últim, si s'ha activat l'opció de crear un informe, crearem un fitxer on hi copiarem el contingut de la variable de sortida.

- Si opció de crear un informe està activada llavors
- Crear fitxer informe en mode escriptura (fopen)
- i=0
- Mentre i<longitud de la variable "sortida" fer
- Escriure cadena "sortida[i]" al fitxer (fprintf)
- i = i+1
-FiMentre
- FiSi

4.5 EXECUSIÓ DEL PROGRAMADOR

En aquest apartat explicarem els passos per executar el programador des de la consola.

1. Obrir el fitxer de configuració i especificar el nom del port a utilitzar.
2. Executar la consola (Menú inicio -> Ejectuar... -> cmd)
3. Cridar el programador passant-li com a paràmetre la direcció de l'arxiu hexadecimal. Si la direcció de l'arxiu hexadecimal o del programador, conté espais en blanc, situar-la entre cometes. Les cometes són necessàries per identificar la cadena de caràcters com a un sol paràmetre. Exemple:

```
C:>Programador.exe "C:\Documents and Settings\usuari\FitxerHex.hex"
```

L'informa que apareixerà per pantalla un cop s'executi el programador serà breu. Si es vol visualitzar un informe extens cal afegir un segon paràmetre (No importa quin valor tingui). Per exemple:

```
C:>Programador.exe "C:\Documents and Settings\usuari\FitxerHex.hex" llarg
```

4. Es pot redireccionar l'informe que surt per pantalla en un fitxer. Per això, cal afegir l'operador ">" (sobreescriure fitxer) o ">>" (afegir a final de fitxer) seguit del nom del fitxer. Exemple:

```
C:>Programador.exe "C:\Documents and Settings\usuari\FitxerHex.hex" llarg > info.txt
```

5. EL BOOTLOADER

El bootloader serà el programa situat en les direccions inferiors de la memòria de programa del microcontrolador que crearem per poder implementar el sistema de gravació correctament[14]. És possible utilitzar el bootloader gràcies a dos aspectes:

- El microcontrolador ATMEGA16 incorpora instruccions per escriure i esborrar la seva memòria de programa
- La memòria de programa de l'ATMEGA16 és no volàtil, o altrament dit, es tracta d'una memòria flash. Això evita que en perdrem el contingut un cop tallem l'alimentació del microcontrolador.

La complexitat del programa del bootloader és inferior a la del programa del programador, ja que, com hem mencionat anteriorment, el bootloader treballa com a operand del programador. Aquest fet, fa que la quantitat d'operacions que efectua siguin mínimes, permeten que ocupi menys espai de memòria i ens permeti disposar de més espai per emmagatzemar programa.

5.1 Objectius i bases del bootloader

Les tasques principals del bootloader són dues:

- Rebre correctament les dades que provenen del programador via sèrie i utilitzant d'interfície que proporciona la placa entrenadora.
- Col·locar aquestes dades rebudes a les direccions de la memòria de programa corresponent.

Mitjançant aquestes tasques, el bootloader guardarà a la memòria de programa el codi de programa que li hagi enviat el programador.

Per tal de treure el màxim profit del bootloader i de tot el sistema de càrrega en general, ens interessa que el booloader ocupi el mínim d'espai possible. Tot i que el nombre de bytes màxim que podria ocupar el bootloader és de 2047 bytes, hem d'intentar minimitzar el màxim l'espai que ocupi. Això farà que disposem de més memòria lliure per carregar-hi programes. Per reduir la mida del bootloader podem:

- Minimitzar les operacions i el codi de programa
- Aplicar l'opció d'optimització que ens ofereix el compilador de l'AVRStudio

La utilització de l'optimització farà un gran servei. En el nostre cas, podrem passar d'un programa que ocupa 1200 bytes a aproximadament 700.

[14] El codi de programa del bootlloader està exposat a l'apartat 1.8 de l'annex

5.2 Funcionament del bootloader

El funcionament del bootloader es mostra en la **Figura 5.2**

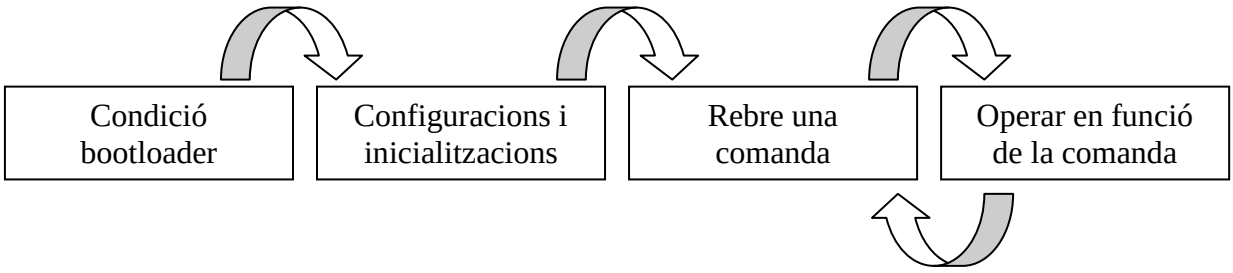


Fig. 5.2: Funcionamet global del bootloader

1. Condició bootloader: La primera cosa que ha de fer el bootloader és saber si el que vol l'usuari, és quedar-se a la secció del bootloader (per fer una descarrega) o anar a la secció d'aplicació. Per fer tal tasca s'utilitza una condició determinada per un bit. Depenent de si aquest està activat o no saltem al programa carregat a les direccions inicials de la memòria o continuar en el bootloader.
2. Configuracions i inicialitzacions: Configurem el port sèrie i inicialitzem les variables.
3. Esperem una comanda
4. Quan arribi la comanda, l'analitzem per saber quina és i efectuem les operacions corresponent. Un cop finalitzada la comanda tornem al punt 3.

En el següent apartat, explicarem breument els avantatges d'utilitzar aquest sistema amb el microcontrolador ATMEGA16, ja que està preparat per implementar-lo. També es resumirà l'estructura de la memòria de programa flash per tal de aclarir certs dubtes que es puguin produir en els apartats posteriors.

5.3 Estructura de la memòria de programa

La memòria de programa és on s'emmagatzemen les instruccions, en forma de valors, que conté el programa carregat al microcontrolador. Per conèixer l'estructura i funcionament de la memòria dividirem aquest apartat en tres blocs, que tracten sobre la composició, la modificació i la divisió d'aquesta memòria.

5.3.1 Composició de la memòria

La memòria de programa de l'ATMEGA16 està composta per un total de 16382 bytes o millor dit, 8191 "words". Parlo de "words" perquè cada instrucció, d'aquesta memòria de programa, està formada per 2 bytes. Aquests "words" estan agrupats en blocs de 64. Cada bloc d'aquests forma el que se'n diu una pàgina. Així doncs, hem d'entendre l'estructura de la memòria de programa com un conjunt de pàgines de 64 words cada una.

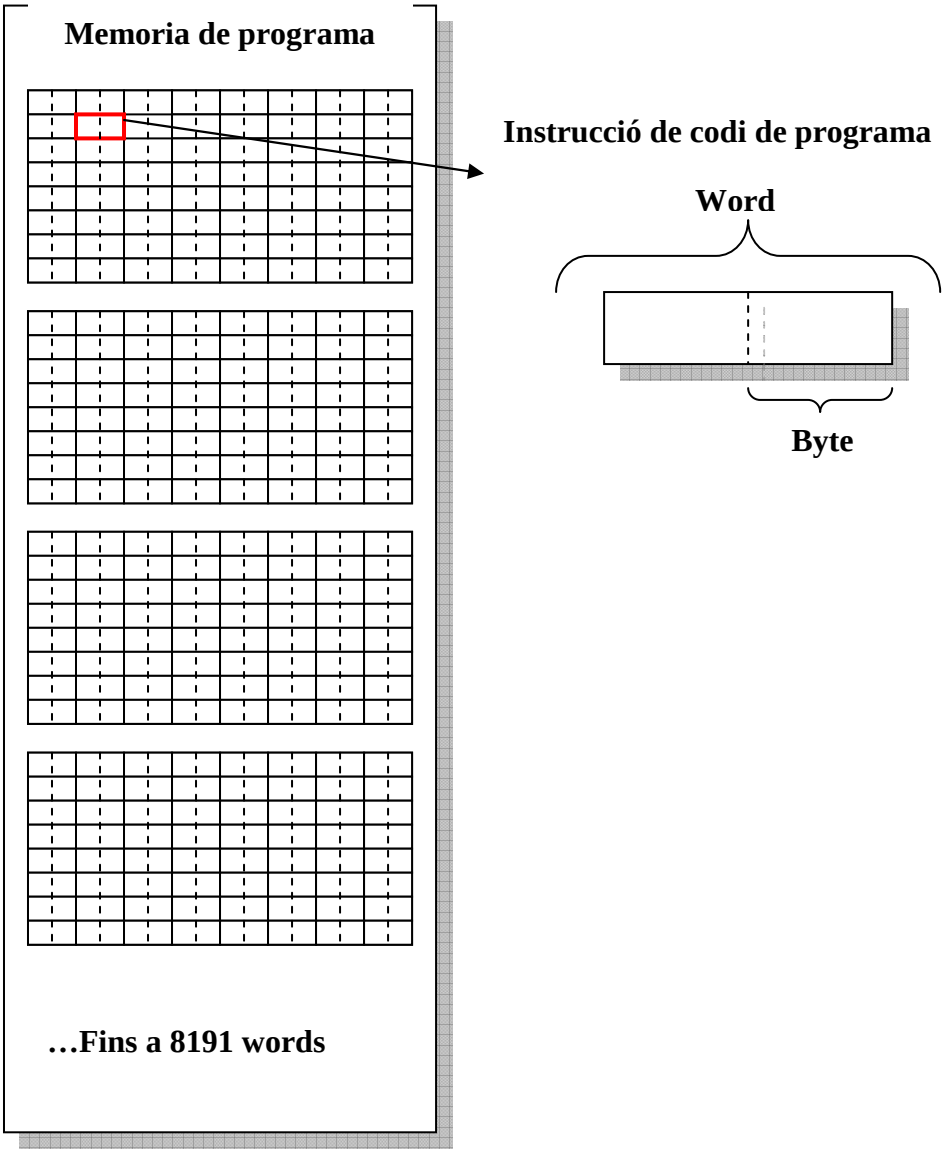


Fig. 5.3.1: Representació gràfica de la composició de la memoria de programa

És per aquest motiu, que la gravació de la memòria de programa s'ha de fer per pàgines, i no és possible fer-la word a word, ni byte a byte.

5.3.2 Modificació de la memòria

Com hem comentat anteriorment, l'ATMEGA16 és un microcontrolador auto-programable, és a dir, mitjançant unes instruccions específiques, l'usuari pot modificar la memòria de programa. Per escriure una pàgina a la memòria de programa s'utilitza un buffer que té la mateixa mida que una pàgina, és a dir, 64 words. Els passos per escriure una pàgina a la memòria són els següents :

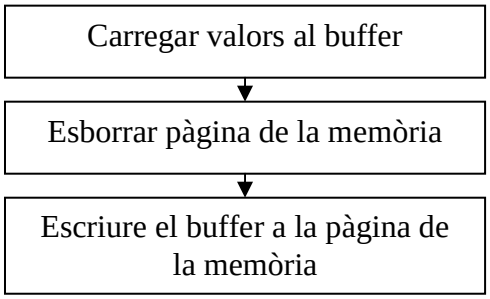


Diagrama 5.3.2: Diagrama de blocs per modificar la memoria de programa

Es carreguen les dades al buffer, s'esborra la pàgina de la memòria flash on si vol gravar la nova i s'hi escriu la nova pàgina. Les funcions per fer aquestes operacions es troben a la llibreria "boot.h" del paquet de llibreries "AVR lib-c".

5.3.3 Divisió de la memòria

El microcontrolador ATMEGA16 està preparat per utilitzar un bootloader, ja que dividint la seva memòria de programa en dues parts o seccions diferents:

- Secció d'aplicació: Hi trobem ubicada l'aplicació principal
- Secció bootloader: Hi trobem ubicat el programa del bootloader

La direcció on acaba la secció aplicació i on comença la secció bootloader pot ser definida per l'usuari mitjançant els bits BOOTSZ1 i BOOTSZ2.Per defecte, aquesta direcció pren el valor de 0x3800 en bytes, equivalent a 0x1C00 en words. Aquesta divisió posa la restricció de que la memòria de programa només pugui ser modificada des de la secció bootloader. Vegeu la figura següent per una representació gràfica d'aquesta divisió.

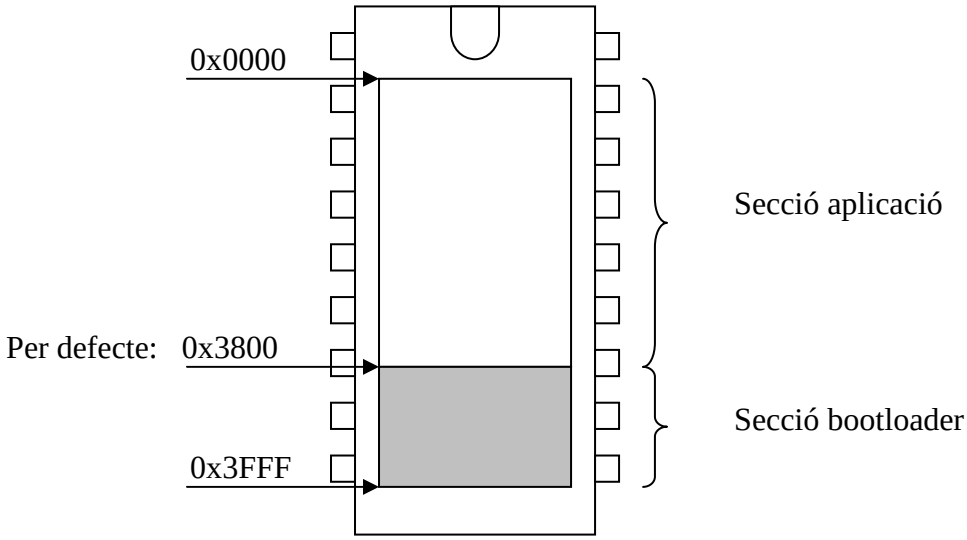


Fig. 5.3.3: Divisió de la memoria de programa del microcontrolador

Els passos que hem seguit per configurar l'adreça inicial del bootloader i la direcció d'arrencada els podeu trobar explicats a l'apartat 5.5.

5.4 Recol·locació del codi de programa

Per tal de que el bootloader quedi gravat a les direccions inferiors de la memòria de programa del microcontrolador, hem de retocar certs paràmetres del compilador. Modificant aquests paràmetres i un cop haguem compilat el programa del bootloader, es crearà un fitxer hexadecimal amb les direccions modificades. Al descarregar el fitxer hexadecimal del bootloader al microcontrolador via SPI(amb el programa AVRSP) ens quedarà ubicat a les adreces inferiors de la memòria de programa que haguem indicat.

Per especificar la direcció inicial on volem col·locar el bootloader, cal saber abans quan ocuparà el codi de programa. Gràcies a l'optimització que proporciona el compilador de l'AVRStudio podem reduir la mida del bootloader de 1200 bytes a 746 bytes. Ara, que sabem el que ocuparà el programa, cal saber quines direccions inicials hi ha per escollir. Hem de tenir en comte que la quantitat de bytes disponibles entre la direcció seleccionada i la direcció final de la memòria de programa (3FFF) ha de ser suficient per guardar-hi el bootloader. La **Taula 5.4.1** ens mostra les direccions (en bytes) possibles pel bootloader i la quantitat de bytes disponibles per emmagatzemar-lo.

Direcció inicial bootloader	Quantitat de bytes fins a final de memòria de programa
3F00 (1F80)	255
3E00 (1F00)	511
3C00 (1E00)	**1023**
3800 (1C00)	2047

Fig. 5.4.1: Direccions inicials disponibles pel bootloader

Com que necessitem un mínim de 746 bytes per emmagatzemar el bootloader escollirem l'adreça inicial 3C00.

Ara que sabem a on hem de recol·locar el programa del bootloader, modificarem els paràmetres del compilador que ens permetran desplaçar-lo a la direcció seleccionada.

Abans però, cal comentar que el compilador reparteix el codi del programa compilat en diferents segments de memòria predefinits. Aquests segments prenen un nom i comencen en una direcció de la memòria de programa determinada. Aquestes són els segments de programa més bàsics:

Segment	Contingut
.text	Tot el codi referent a les instruccions
.data	Les variables globals inicialitzades
.bss	Les variables globals no inicialitzades

Fig. 5.3.3: Segments de memoria de programa

Primer es va haver de consultar el manual del "lib-c" per saber com moure el codi de les funcions a direccions que no siguin per defecte. Per fer això cal inserir davant de la funció una etiqueta i definir l'etiqueta com un enllaç a una nova secció de codi, de tal manera que el compilador ho pugui reconèixer com un nou segment. Això es fa de la següent manera:

#define ETIQUETA __attribute__ ((section (".seccio")))

Ara cal dirigir-se a les opcions del compilador (Configuration Options –> Custom Option) i afegir un nou paràmetre a l'apartat de "linker options". Aquest paràmetre servirà per definir la direcció inicial on començarà la secció definida.

-Wl,--section-start=.seccio =0x3C00

Acceptem les opcions i un cop s'hagi compilat el programa observarem que el codi d'aquella funció haurà set redireccionat. El problema que hi havia és que el nostre programa de bootloader té més d'una funció i crear una etiqueta per cada funció era pesat i poc pràctic. Per això, es va decidir modificar la direcció inicial del segment

".text", el qual conté el codi de programa i les variables no globals, amb la direcció 0x3C00 de la següent manera.

-Wl,--section-start=.text =0x3C00

D'aquesta manera no era necessari definir cap etiqueta i el codi ja ens quedava recol·locat allà on volíem. Cal remarcar que si utilitzéssim variables globals, aquestes es col·locarien en el segment .data o .bss, per tan, aquests també haurien de ser modificats perquè entressin dins les direccions de les seccions bootloader sense reemplaçar les direccions del segment .text modificat.

El que cal ara, és configurar el microcontrolador per tal de que la direcció d'arrencada sigui la del bootloader, és a dir 0x3C00, i definir-li aquesta direcció i no la que hi ha per defecte, 0x0000. Per configurar aquests paràmetres s'han de modificar els bits BOOTSZ1 i BOOTSZ2, per indiciar la direcció inicial del bootloader, i el bit BOOTRST per especificar que arrenqui des de aquesta direcció. Aquests bits no poden ser configurats per software sinó que s'han de modificar utilitzant un altre mètode. Un dels mètodes, que és el que s'utilitzarà és per SPI, programació en sèrie directa sobre el microcontrolador.

5.5 Preparació del microcontrolador

Perquè el microcontrolador funcioni en el sistema de gravació que volem implementar l'hem de preparar modificant els bits representats en la següent taula :

Descripció	Bits i valor
Especificar l'adreça inicia de la secció bootloader:	BOOTSZ1 = 0 BOOTSZ0 = 1
Indicar que arrenqui per l'adreça del bootloader:	BOOTRST = 1

Fig. 5.5: Bits dels fusibles del microcontrolador que s'han de configurar.

Aquests bits no són modificables per software, en el nostre cas els modificarem per SPI. Per fer això utilitzarem un programa que disposa la Universitat de Vic i seguirem el següents passos per configurar aquests bits:

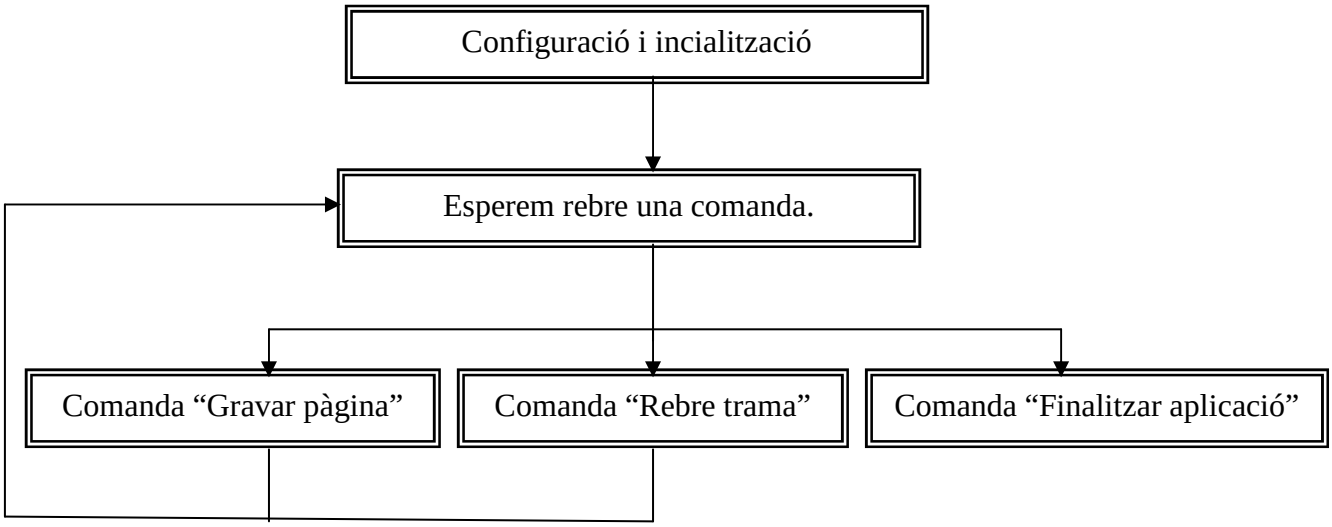
1. Llegim la configuració del microcontrolador i seguim les instruccions d'alimentació i comunicació de la placa per fer la lectura.
2. Modifiquem, en l'apartat corresponent, l'adreça inicial del bootloader i activarem la casella per arrancar des de la secció bootloader.
3. Polsem el botó corresponent per configurar el microcontrolador i seguirem les instruccions que ens indica el programa.

És necessari llegir abans la configuració del microcontrolador per modificar-la conservant els paràmetres actuals que tingui i que no volem modificar.

Un cop hem seguit tots aquests passos, ja tenim el microcontrolador preparat per arrancar des de la direcció inicial del bootloader 0X3C00.

5.6 Estructura del bootloader

En aquest apartat intentarem explicar les parts del bootloader. Descriurem la seva estructura i funcionament utilitzant pseudocodi sempre que sigui possible. El **Diagrama 5.6** representa el diagrama de blocs general.



- Configuració i inicialització: En aquest bloc i pertanyen les tasques de configuració del port sèrie, inicialització de variables i la condició per seleccionar el bootloader o l'aplicació.
- Rebre una comanda: En aquest bloc s'esperarà rebre una comanda, i un cop rebuda, s'analitzarà per saber quines operacions hem d'efectuar
- Comanda "Gravar pàgina": Pertanyen, en aquest bloc, les operacions per guardar la pàgina emmagatzemada al buffer de la flash.
- Comanda "Rebre una trama": Em aquest bloc es rebrà tota una trama, fent el càlcul del CRC i el control d'errors.
- Comanda "Finalitzar aplicació": Últims passos per finalitzar l'aplicació

5.6.1 Configuració i inicialització

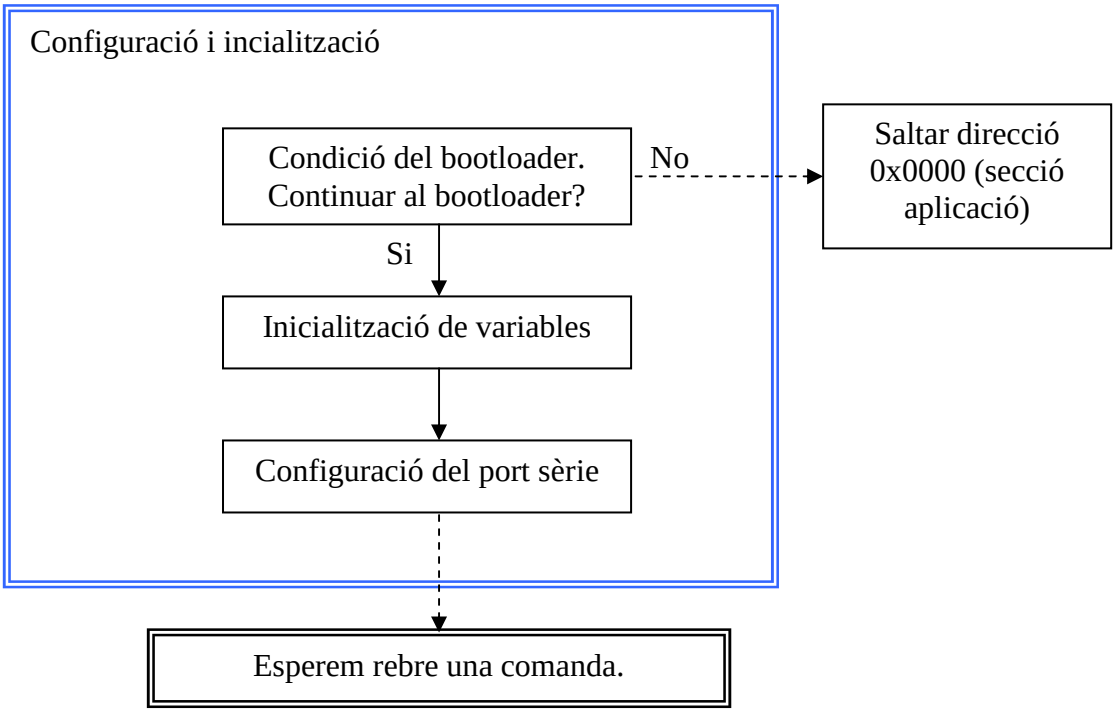


Diagrama 5.6.1: Diagrama de blocs de configuració i inicialització del bootloader

- Condició del bootloader

Aquesta serà la primera tasca que farà el bootloader, i la seva funció serà comprovar si el que es vol és continuar en el bootloader, i procedir a carregar un programa al microcontrolador, o bé anar al programa carregat, situat des de la memòria inicial. La següent figura és una representació gràfica d'aquesta condició:

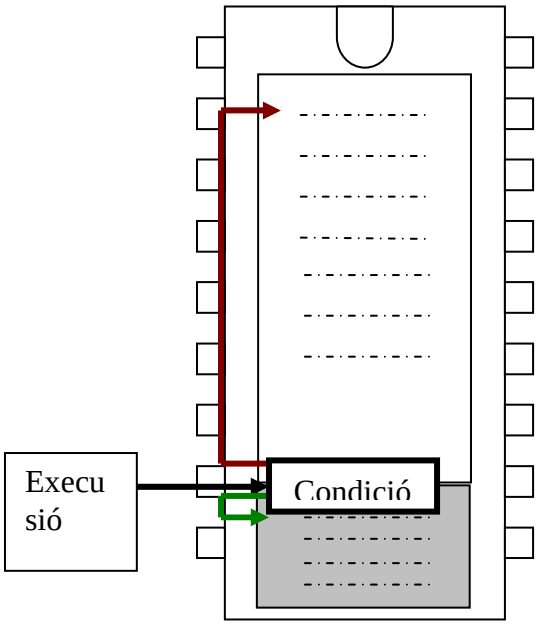


Fig. 5.6.1.1: Condició del bootloader

El microcontrolador comprova si el polsador de l'entrada entrenadora, connectat al pin 2 del port D, està polsat. Si ho està, continuem en el bootloader, sinó, saltem a l'adreça inicial. Per indicar a l'usuari que accedim al bootloader encendrem el led tricolor de la placa entrenadora de color verd.

Configurem el pin 2 del port D com a entrada

- DDRD = 0x78

Comprovem si el polsador està activat

- Si polsador activat (bit_is_set) llavors
- Activar bit 3 de DDRB (pin led verd com sortida)
-Activar bit 4 de DDRD (pin led vermell com sortida);
- Activem bit 3 de PORTB(encenem led verd=
- "Seguir a Inicialització de variables"
- Sinó
- DDRD = 0x00
*- Anar a la secció aplicació (goto *0x0000)*
- Fisi

És important desconfigurar el pin del port D, ja que al saltar a la secció aplicació no hi pot haver cap configuració establerta.

- Inicialització de variables

Les variables que utilitzarem són del tipus sense signe (unsigned). És important especificar-les sense signe, sobretot perquè el CRC s'actualitzi correctament. La variable que pren el valor del crc s'ha de inicialitzar a 0 abans de rebre una nova trama.

- Configuració del port sèrie

La comunicació sèrie es fa mitjançant el dispositiu UART. Els registres que formen part d'aquest dispositiu són: UDR, UCSRA, UCSRB, UCSRC, UBRRH i UBRRL

Per especificar el baud rate hem de definir un valor al registre UBRR. Per trobar aquest valor consultem la taula del manual del microcontrolador corresponent. El valor que necessitem el trobem representat a la següent Taula:

Freq. oscil·lació	Baud Rate	Mode de velocitat	Valor UBRR
14,7456 MHz	57600 bps	Normal (UCX = 0)	**15**

Taula. 5.6.1.2: Valor de UBRR per un Baud Rate de 57600 bps, velocitat normal i freq. Osc. 14,7456 Mhx

També es pot aplicar la fòrmula vàlida per una comunicació asíncrona i de mode de velocitat normal, per trobar el valor de UBRR

$$UBRR = \frac{f_{osc}}{16BAUD} - 1$$

Fòrmula. 5.6.1.3: Fòrmula per calcular el valor de UBRR, en una comunicació asíncrona i velocitat normal

Activem recepció i transmissió sèrie

- *TXEN = 1*
- *RXEN = 1*

La configuració del bits de dades es fa mitjançant els bits USCZ2 del registre UCSRB i els bits USCZ0 i UCSRC del registre UCSRC. Com podem veure en la **Taula 5.6.1.4**, el valor d'aquests bits especifica el número de bits de dades.

USCZ2	USCZ1	USCZ0	Descripció
0	0	0	5 bits de dades
0	0	1	6 bits de dades
0	1	0	7 bits de dades
0	1	1	8 bits de dades
1	1	1	9 bits de dades

Taula. 5.6.1.4: Configuració dels bits de dades

Els registres UCSRC i UBRRH tenen la mateixa direcció. La manera de modificar UCSRC evitant que se'ns modifiqui UBRRH és activant el bit URSEL al mateix temps que els bits que volem activar del registre UCSRC. Per definir 8 bits de dades, activarem al mateix temps els bits següents:

- *UCSZ1 = 1*
- *UCSZ0 = 1*
- *URSEL = 1*

Hi ha altres bits que no configurem perquè, per defecte, prenen el valor que nosaltres volem (valor 0). Aquests són:

- S'hi s'activa el bit UMSEL del registre UCSR la comunicació serà síncrona. Si es deixa a 0 serà asíncrona
- Els bits UPM1 i UPM0 del registre UCSRC permeten habilitar el control de paritat i d'inserció del bit de paritat en les dades enviades. En el cas que estigués habilitat, el bit UPE del registre UCSRA s'activa si es produeix un error de paritat. Si aquests bits són 0, no s'estableix la paritat.

5.6.2 Rebre una comanda

Si

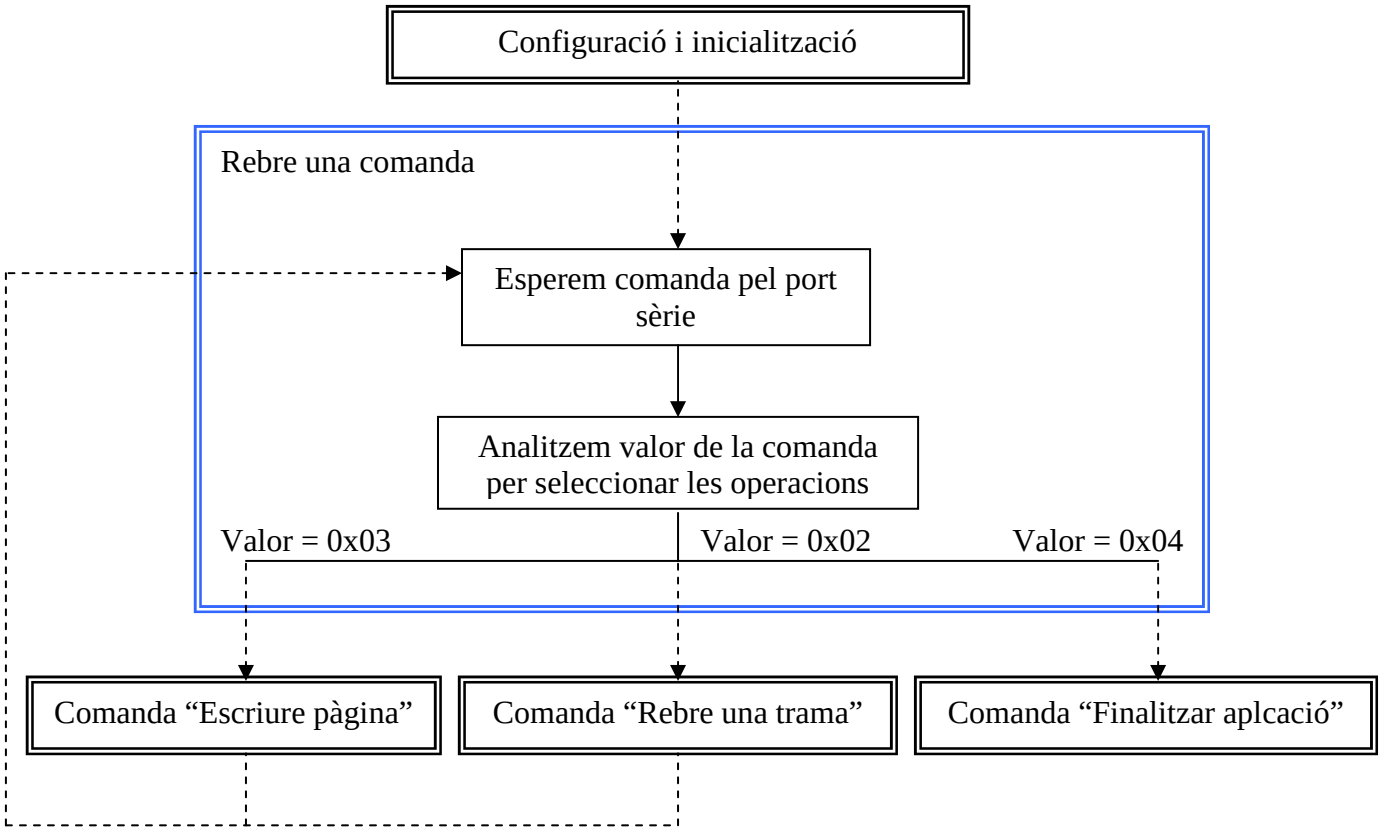


Diagrama 5.6.2: Diagrama de blocs per la recepció d'una comanda del bootloader

- Esperar comanda pel port sèrie

  El registre UDR del microcontrolador és el buffer d'entrada i sortida del port sèrie. Per llegir una dada que es rebi en el port sèrie només cal consultar aquest registre. El bit que ens indica que ha entrat una nova dada al buffer d'entrada és RXC. Haurem d'esperar que RXC estigui activat per llegir el buffer d'entrada.

- Mentre RXC = 0 fer
- No fer res
- FiMentre
- dada = UDR

En cas de que no es rebi cap dada el bootloader quedarà penjat. No hem aplicat cap sistema de "time-out" a la rebuda de dades perquè no ho hem considerat necessari. Com que el programador ja informa dels errors que puguin sorgir, l'usuari ja estarà al corrent si apareix algun problema.

Per indicar que s'ha rebut una comanda encendrem el led vermell del led tricolor de la placa entrenadora. El led quedarà il·luminat de color taronja fins que es finalitzin les operacions de la comanda rebuda.

- Activem bit 4 de Port D (encenem led vermell)

Aquest codi anirà situat dins la funció "Rebre dada", que cridarem cada cop que s'hagi de rebre una nova dada. Aquesta funció retornarà el valor rebut.

- Analitzar el valor de la comanda rebuda

- Si comanda = 0x02
- Anar al bloc "Rebre una trama"
- Si comanda = 0x03
- Anar al bloc "Escriure pàgina"
- Si comanda = 0x04
- Anar al bloc "Finalitzar aplicació"

5.6.3 Comanda de rebre una trama

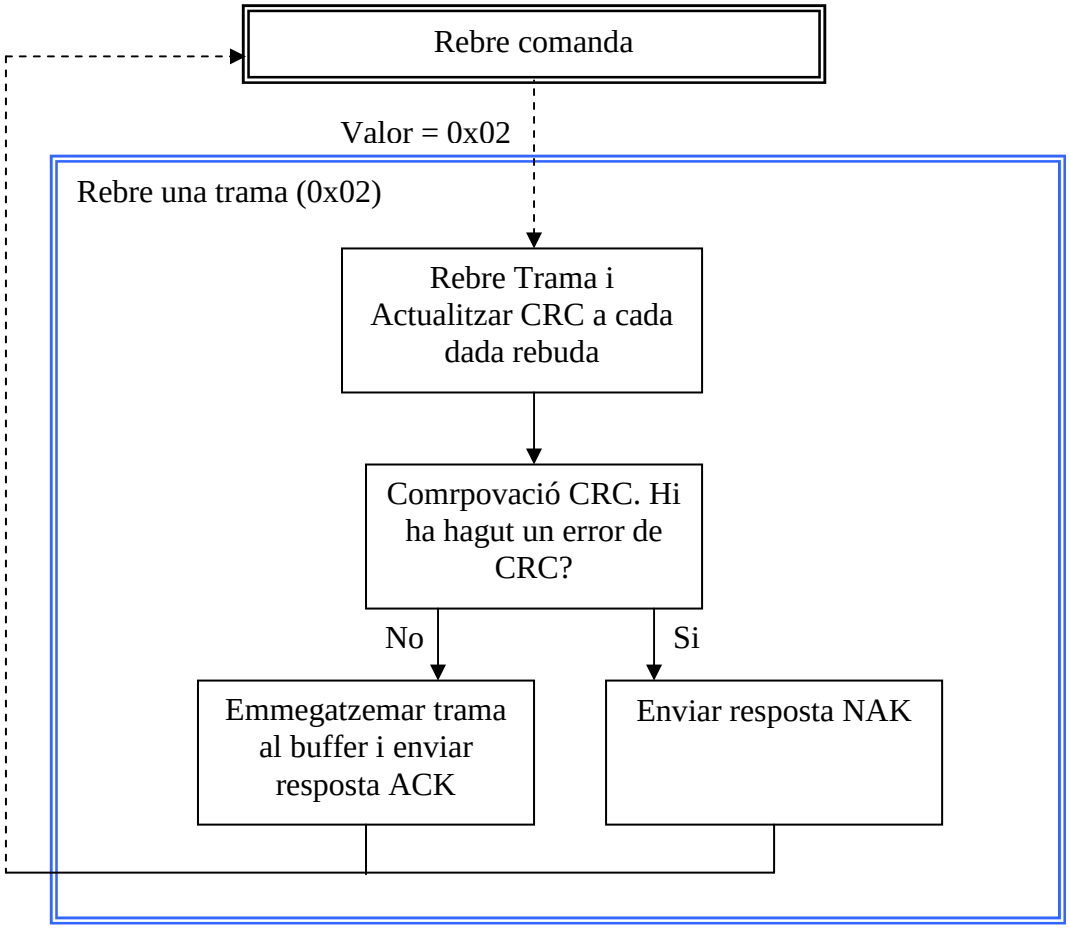


Diagrama 5.6.3: Diagrama de blocs de la recepció d'una trama del bootloader

- Rebre una trama i actualitzar el crc

  Anirem cridant la funció "Rebre dada" per rebre els diferents paràmetres de la trama. Després de cada dada rebuda (incloent el valor de la comanda) actualitzar el crc amb la dada. La funció que actualitza el CRC serà la mateixa que podem trobar en el programador i està descrita en la llibreria "CRC.h" del paquet de llibreries "AVR lib-c".

  Recordem que la trama està organitzada de la següent manera (**Figura 3.3.1)**:

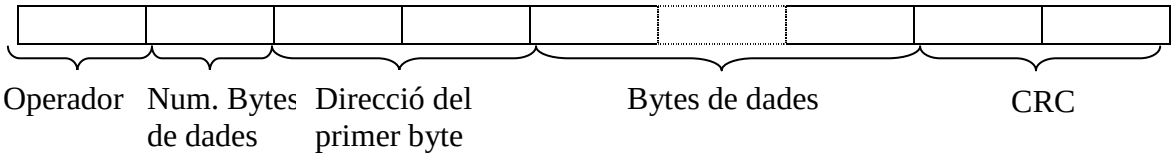


Fig. 3.3.1: Organització d'una trama de dades

Aquest es el pseudocodi de recepció d'una trama:

- Actualitzar crc amb el valor 0x02
- num. de bytes = Rebre dada
- Actualitzar crc amb el valor "num. de bytes"
- direcció alta = Rebre dada
- Actualitzar crc amb el valor "direcció alta"
- direcció baixa = Rebre dada
- Actualitzar crc amb el valor "direcció baixa"
- Definim una trama de longitud "num. de bytes"
- i = 0
- Mentre i<num de bytes fer
    - Trama[i] = Rebre dada
    - Actualitzar crc amb valor de "Trama[i]
    - i = i+1
- FiMentre
- crc rebut alt = Rebre dada
- crc rebut baix = Rebre dada

- Comprovació del crc

- Unificar crc rebut alt i crc rebut baix en la variable crc rebut
- si crc calculat = crc rebut llavors
    - Guardar trama al buffer i enviar resposta ACK
- sinó
    - Enviar resposta NAK
- Inicialitzem crc calculat a 0

- Guardar trama al buffer de la flash

Si les dades s'han rebut correctament, les guardarem al buffer de la flash. Seguidament enviarem el caràcter ACK (0x06) per confirmar al programador que tot a set correcte.

És important comentar un aspecte a l'hora de gravar sobre el buffer de la flash. Aquest ha de ser gravat per words amb la funció "boot_page_fill_safe". A aquesta funció se li passa el word que es vol gravar i la direcció on es vol gravar. La mida d'aquest buffer és de 64 words (com un pàgina) i si posem una direcció major a 0x7E (última posició del buffer) donarem la volta. Per exemple, si escrivim sobre la direccio 0x84, es gravarà realment sobre la direcció 0x04. Aquest fet ens permetrà utilitzar directament la direcció que rebem.

- Unifiquem "direcció alta" i "direcció baixa" a la variable "direcció"
- i=0;
- Mentre i< num. de bytes
- Unifiquem Trama[i],Trama[i+1] a la variable "paraula"
- Guardem la paraula al buffer de la flash dins la "direcció+i"
- i=i+2;
-FiMentre
- Enviar ACK

Per enviar el caràcter ACK utilitzarem la funció "Enviar resposta" i li passarem el valor de la resposta (0x06 o 0x15).

- Enviar resposta ACK o NAK

Per enviar un caràcter per sèrie hem d'escriure en el buffer de sortida del port sèrie, un cop haguem comprovat que aquest estigui buit. Aquesta comprovació s'ha de fer per tal d'assegurar-nos que no hi hagi un caràcter al buffer que encara no s'hagi enviat. Un cop està buit el buffer de sortida, hi escrivim. Aquest buffer de sortida s'hi accedeix mitjançant el mateix registre que el buffer d'entrada, UDR.

El bit UDRE s'activa quan el buffer de sortida està buit. Serà qüestió d'esperar que UDRE sigui 1 per enviar la resposta escrivint al registre UDR.

- Mentre UDRE = 0 fer
- No fer res
- Fi mentre
- UDR = resposta (ACK o NAK)

La funció "Enviar resposta" contindrà aquest codi i li passarem el valor que volem enviar. En cas de que la comprovació del control d'errors sigui correcta enviarem el valor 0x06, que equival al caràcter ACK. En cas contrari, enviarem el valor 0x15, que equival al caràcter NAK.

5.6.4 Comanda d'escriure pàgina

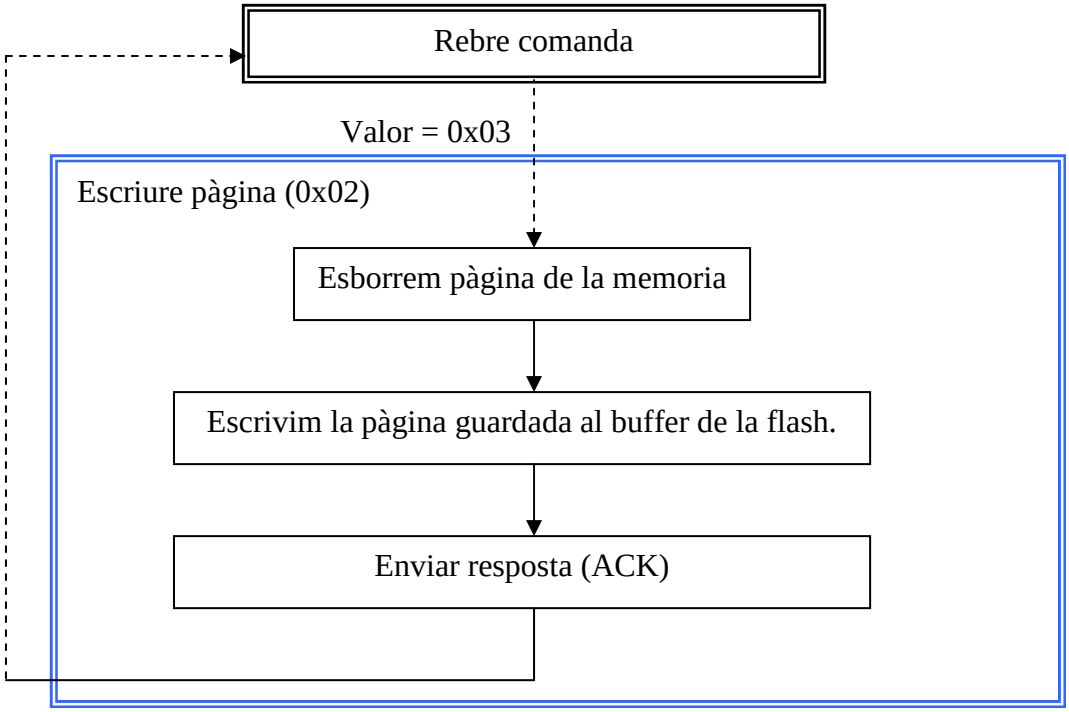


Diagrama 5.6.4: Diagrama de blocs per l'escriptura d'una pàgina del bootloader

- Esborrar la pàgina de la memòria de programa

  Abans d'escriure sobre una pàgina de la memòria l'hem de esborrar. És necessari esborrar-la perquè el sistema de gravació permet canviar els bits de la memòria del valor 1 a 0, però no al revés. Aquest fet explica perquè els valors "buits" de la memòria de programa, on no hi resideix codi, prenen el valor 0xFF i no 0x00.

  La funció per esborrar una pàgina de la memòria flash és "boot_page_erase" i se una direcció que estigui compresa dins la pàgina que volem esborrar com a paràmetre

  Fer que el programador ens enviï aquesta direcció és una cosa redundant, ja que a la variable "direcció", utilitzada en la comanda de rebre trama, hi trobem la última direcció tractada, que pertany a la pàgina que volem escriure. Per tant, podem perfectament passar-li aquesta variable com a paràmetre.

- Escriure una pàgina a la memòria de programa

  La funció per escriure una pàgina és "boot_page_write" i se li passa una direcció que es trobi dins la pàgina que volem gravar com a paràmetre.

L'operació d'escriptura copiarà el buffer de la flash a la pàgina corresponent. Li passarem com a paràmetre la mateixa direcció que li hem passat a "boot_page_erase".

Haurem d'esperar que es completi la gravació de la pàgina mitjançant la funció "boot_spm_busy_wait".

5.6.5 Comanda de finalitzar l'aplicació

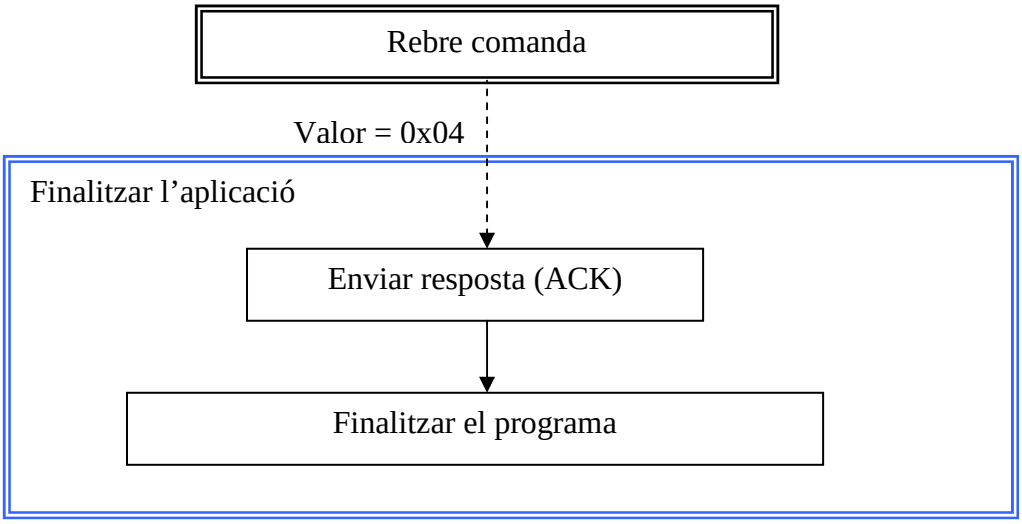


Diagrama 5.6.5: Diagrama de blocs per finalitzar l'aplicació del bootloader

- Finalitzar el programa

Després d'enviar la resposta ACK al programador, per indicar-li que s'ha rebut la comanda, farem les operacions per finalitzar el bootlaoder.

S'ha optat per finalitzar en un bucle infinit i no saltar a la secció aplicació per els següents motius:

- Saltar a la secció aplicació suposava desconfigurar molts registres, i això feia que el bootloader ocupés més

-Potser l'usuari no tenia d'intenció d'entrar en el programa que havia carregat al microcontrolador. Si hi vol accedir, només cal polsar el polsador de "reset"

Per indicar a l'usuari que el bootloader ha arribat a l'estat final encendrem el led tricolor de la placa a color vermell.

- Posem a 0 el pin 3 del port B (desactivem led verd)
- Mentre (1) (bucle infinit)

6. CONCLUSIONS

En aquest apartat descriurem el resultat que hem obtingut a partir de la creació d'aquest projecte i les conclusions que hem extret, així com els problemes que ens hem anat trobant i la opinió personal.

Com a resultat de tot aquest projecte, obtenim un sistema de programació per a una placa entrenadora amb el microcontrolador ATMEGA16. Aquest sistema ha estat dissenyat amb eines de lliure accés i està a l'abast de tothom de forma gratuïta.

Per una banda, el hardware necessari per utilitzar aquest mètode de programació és:

- Una computadora amb l'entorn Windows.
- La placa entrenadora amb el microcontrolador ATMEGA16
- Un cable "Null-Modem" per fer la connexió d'aquests dos dispositius.

El sistema de Windows el podem trobar a la majoria d'ordinadors, i el cable Null-Modem és econòmic i pot ser adquirit fàcilment. Aquest fet, fa que el sistema de gravació sigui pràctic, ràpid d'aplicar i pugui ser utilitzat tant a les aules de la Universitat com a casa l'alumne. D'aquesta manera, complim amb les necessitats i els objectius inicials del projecte: aconseguir un sistema econòmic, fàcil i ràpid d'utilitzar.

Per altre banda, el software necessari serà:

- L'executable del programador (la interfície gràfica és optativa)
- L'arxiu de configuració amb els paràmetres determinats
- Tenir el bootloader carregat al microcontrolador.
- Tenir configurat el microcontrolador per tal de que arrenqui des de l'adreça 0x3C00 de la direcció de memòria de programa.

Aquest sistema de gravació tindrà certs desavantatges:

- Al ser un sistema adaptat a una placa entrenadora en concret, amb un microcontrolador específic, tindrem l'inconvenient que només serà vàlid per programar aquesta placa.
- Al utilitzar un bootloader, tindrem una secció de memòria més reduïda al microcontrolador per carregar-hi programes.
- El microcontrolador haurà de portar incorporat el bootlaoder i haurà d'estar configurada l'adreça d'arrencada perquè pugui ser compatible amb el sistema de programació. Per fer això es necessitarà un altre sistema de gravació, com el que s'ha fet servir de SPI.

En definitiva, serà un sistema que funcionarà correctament i complirà amb els objectius inicials, però les seves limitacions faran que no sigui un mètode de gravació universal i només serà vàlid per la placa entrenadora.

6.1 Resultats obtinguts

Per comprovar el bon funcionament del sistema de gravació, s'han provat quatre programes que s'han carregat correctament:

- Programa que encén els leds de la placa: 1 pàgina
- Programa que encén els leds de la placa en funció del polsador: 1 pàgina
- Programa que permet rebre i enviar caràcters per sèrie: 2 pàgines
- Programa de testeig de la placa entrenadora: 23 pàgines

Pel que fa el temps aproximat del sistema de gravació s'ha fet l'estudi descrit en l'apartat corresponent de l'annex 1. Els resultats d'aquest estudi són:

Temps de transferència d'una pàgina (1370 bits)	
Baud Rate	**Temps**
57600	23,784 ms

Taula 6.1: Temps de transferència d'una pàgina

Temps de gravació d'una pàgina	
Temps mínim esborrar i gravar una pàgina	**Temps màxim esborrar i gravar una pàgina**
7.4 ms	9.0 ms

Taula 6.2: Temps de gravació d'un pàgina

Temps de transferència i gravació amb un Baud Rate de 57600 bps		
	Temps mínim	**Temps màxim**
1 pàgina	27.48 + 7,4·10-3 ms	(28.28 + 9·10-3) ms
120 pagines (tota la memòria)	2.85 + 0,88 segons	2.85 + 1.08 segons

Temps total estimat màxim per la transferència i gravació d'una pàgina.	(35,68 +- 1,4) ms
Temps total estimat màxim per la transferència i gravació de totes les 120 pàgines	(3,83 +- 1) segons

Taula 6.2: Temps de transferència i gravació del sistema

Podem observar que:

Temps gravació = 30.8% - 37.9% Temps de transferència.

6.2 Possibles millores

En aquest apartat estan descrites un conjunt de possibles millores:

- Sistema de verificació que permetés assegurar la correcta gravació d'un programa a la memòria de programa del microcontrolador. Aquest sistema s'efectuaria després de la gravació a la memòria. El bootloader llegiria la memòria gravada i l'enviaria el programador. Aquest, compararia el programa enviat amb el rebut per saber si s'ha gravat correctament.
- Solucionar el conflicte del codi ASCII. La codificació ASCII del compilador MingW és diferent a la interpretada en la consola. Per la qual cosa no s'han pogut posar accents ni dièresis en la informació.
- Reduir el codi del bootloader per tal de:
    - Ocupi menys espai i es pugui desplaçar l'adreça inicial del bootloader a una adreça inferior a 0x3C00.
    - Les operacions ubicades entre rebre una dada i una altra no limitin tan el Baud Rate. D'aquesta manera aconseguiríem un programa més ràpid.

6.3 Procés de desenvolupament

Tot i que la complexitat d'aquest projecte no és molt alta, la manca d'informació ha suposat gastar un temps considerable en recerca de dades, codi i mètodes per arribar a complir certes tasques. L'eina més utilitzada per fer aquesta cerca a set Internet.

A mesura que he començant a buscar informació per Internet, he anat agafant agilitat en la cerca, trobant pàgines que poden contenir informació útil en varies situacions. El buscador de "Google"[15] ha set de gran ajuda. També han set de gran utilitat diversos fòrums que s'han trobat a la xarxa, ja que podia trobar persones que tenien un problema semblant al meu i, a vegades, trobar possibles solucions o bé saber com encaminar-me per arreglar problemes que tenia. Tot i així, la informació era escassa i quan les cerques duraven estona i acabaven sense tenir fruits desmoralitzaven.

Pel que fa el programador, al principi no s'havia per on començar. Manipular un dispositiu de l'ordinador com el port sèrie em semblava una tasca difícil, bàsicament perquè mai havia programat prou com per arribar a fer una cosa semblant. Un cop vaig anar trobant com utilitzar les API's de Windows i vaig saber quines funcions havia d'implementar em va resultar molt més fàcil. Aquest fet em va donar una "empenta" per seguir treballant-hi, encara que el temps de cerca havia set molt major al temps d'implementació del codi.

Pel que fa el software del microcontrolador, no només s'ha utilitzant Internet per la cerca d'informació sinó també el manual de l'ATMEGA16. Tot i que no ha suposat gastar tan de temps com en la construcció del programador, ha set necessari iniciar-se en la programació del microcontrolador. Utilitzar el llenguatge assemblador per iniciar-

[15] Direcció 32 de la bibliografia, capítol 7

se, resulta poc pràctic, però t'acaba ensenyant com funciona el micro i aprens part de la seva estructura.

Vull comentar també que la confecció de la memòria ha set molt costosa. S'ha renovat un parell de vegades i hi ha hagut un número considerable de modificacions. M'ha costat intentar explicar el funcionament del programa perquè pugui ser entès pel lector.

És difícil especificar el temps de treball en aquest projecte. Intentarem fer una estimació:

- Temps en la confecció de la memòria: 240 hores
- Temps de cerca: 120 hores
- Temps en l'aprenentatge i construcció del bootloader: 100 hores
- Temps en l'aprenentatge i construcció del programador: 180 hores
- Temps en la construcció de la interfície gràfica: 60 hores

- Temps total en el projecte: 700 hores

6.4 Opinió personal

La meva valoració sobre la creació d'aquest projecte és positiva. Crec que si a un li agrada programar és un projecte agradable. L'únic que m'ha fet passar mals tràngols és el fet de buscar informació sense trobar-ne, que pot arribar a portar-te a una situació desesperançadora. La inexperiència crec que també juga un gran important i a vegades potser tens la solució de com implementar una tasca a dos pams del nas i no la veus fins l'últim moment

Per altre banda, tot i ser un projecte bàsicament centrat en la programació, toca diferents aspectes. He aprés a controlar diferents dispositius d'un microcontrolador, m'ha ajudat a comprendre aspectes del llenguatge C que no tenia del tot clars i aprofundir en la programació, com crear una interfície gràfica i establir una comunicació en sèrie. Després de fer aquest projecte veig més fàcil fer altres aplicacions relacionades amb lo que s'ha tractat, i que abans no hagués cregut que fossin possibles per mi.

Crec que acabes aprenent que qualsevol aplicació pot ser portada a terme si un s'ho proposa i hi dedica el seu temps.

7. BIBLIOGRAFIA

API'S DE MICROSOFT WINDOWS

1. MSDN (MICROSOFT DEVELOPER NETWORK)
< http://msdn.microsoft.com> [Consulta: Desembre de 2007]

2. PROGRAMANDO CON TOASTY
Diccionario de API para Windows
<http://www.tinet.org/~drt/programacion/api/index.html>
[Consulta: Desembre de 2007]

3. WINAPI CON CLASE
http://winapi.conclase.net/ [Consulta: Novembre de 2007]

4. ZATOR SYSTEMS
Api de MS Windows
<http://www.zator.com/Cpp/E1_7_2.htm> [Consulta: Novembre de 2007]

COMUNICACIÓ SÈRIE

5. FRANCISCO JAVIER CEBALLOS SIERRA
Comunicaciones
<http://www.fjceballos.es/libros/84-7897-302-8/vc03.html>
[Consulta: Gener de 2008]

6. GRUPO ALVOR
Comunicación Series en Windows (2ª parte)
<http://www.grupoalbor.com/delphi/?opc=articulos&id=3&parte=2>
[Consulta: Gener de 2008]

7. NATIONAL INSTRUMENTS
Comunicación serial: conceptos generales
<http://digital.ni.com/public.nsf/allkb/039001258CEF8FB686256E0F005888D1#232> [Consulta: Novembre de 2007}

8. UNIVERSIDAD DE OVIEDO
La capa de enlace
<http://www.isa.uniovi.es/docencia/redes/Apuntes/tema3.pdf>
[Consulta: Desembre de 2007]

9. WIKIPEDIA
SPI
>http://es.wikipedia.org/wiki/SPI> [Consulta: Desembre de 2007]

10. ZSOLUCIONES
Biblioteca para el puerto serie
http://zsoluciones.com/datos/?page_id=11 [Consulta: Desembre de 2007]

CONTROL D'ERRORS

11. UNIVERSITAT AUTONOMA DE MADRID
Control d'errores
http://arantxa.ii.uam.es/~ig/teoria/temas/tema-5-2007-08.pdf
[Consulta: Febrer de 2008]

12. WARIANOS
Fòrum: *Que és el famoso "error de redundancia cíclica"*
<http://www.warianos.cl/foros/showthread.php?t=4586>
[Consulta: Febrer de 2008]

13. WIKIPEDIA
Corrección y detección de errores
<http://es.wikipedia.org/wiki/Detecci%C3%B3n_y_correcci%C3%B3n_de_errores> [Consulta: Gener de 2008]

FITXER HEXADECIMAL

14. KEIL
General: Intel Hex File Format
<http://www.keil.com/support/docs/1584.htm> [Consulta: Febrer de 2008]

15. THE 8052 ONLINE RESOURCE
Intel Hex File format
<http://www.8052.com/tutintel.phtm> [Consulta: Febrer de 2008]

PROGRAMACIÓ EN C

16. CHUIDIANG
Ficheros de texto en C
<http://www.chuidiang.com/clinux/ficheros/fichero-texto.php>
[Consulta: Novembre de 2007]

17. PABLIN
Recibir parámetros al ejecutar el .EXE en C
http://www.pablin.com.ar/computer/programa/c/param.htm
[Consulta: Maig de 2008]

18. SUPERTUTORIALES
Instrucciones y comandos en C++
<http://www.publispain.com/supertutoriales/programacion/c_y_cplus/cursos/3/index.htm> [Consulta: Novembre de 2008]

19. UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Algorítmica
<http://www.ulpgc.es/otros/tutoriales/mtutor/indice.html>
[Consulta: Gener de 2008]

PROGRAMACIÓ EN MICROCONTROLADOR

20. ATMEL CORPORATION
<http://www.atmel.com/>[Consulta: Novembre de 2007]

21. AVR BEGINNERS
<http://www.avrbeginners.net/> [Consulta: Febrer de 2008]

22. AVR FREAKS
<http://www.avrfreaks.net/> [Consulta: Gener de 2008]

23. AVR ASSEMBLER TUTORIAL
Beginners programming in AVR assembler
<http://www.avr-asm-tutorial.net/avr_en/beginner/index.html>
[Constula: Febrer de 2008]

24. DEPARTAMENTO DE ARQUITECTURA Y TECNOLOGIA DE COMPUTADORAS
El ensamblador y el lenguaje C
<http://atc.ugr.es/docencia/udigital/13.html> [Consulta: Març de 2008]

25. ITOOBOT
NanoC, Placa de control para Microbots
<http://www.itoosoft.com/motorolos/php/nanoc15.php#3>
[Consulta: Desembre de 2007]

26. JOSE ALANDRO
Programador de micros AT89C2051 de Atmel
<http://perso.wanadoo.es/josealadro/circuitos/at_programador/atprog6.html>
[Consulta: Febrer de 2008]

27. LA WEB DEL PROGRAMADOR
Fòrum: *Desplazamiento de bits*
<http://www.lawebdelprogramador.com/news/mostrar_new.php?id=13&texto=C/Visual+C&n1=476691&n2=1&n3=0&n4=0&n5=0&n6=0&n7=0&n8=0&n9=0&n0=0> [Consulta: Març de 2008]

28. MCI ELECTRONICS: OLIMEX CHILE
Tutorial AVR-MT
<http://www.olimex.cl/present.php?page=tut_avr_mt_intro&c=1>
[Consulta: Desembre de 2007]

29. MIG SANTIAGO
Curso básico del AVR AT90S1200
<http://mx.geocities.com/migsantiagov/avr/index.htm>
[Consulta: Març de 2008]

30. RWTH
Using interrupts (ATMEGA16)
<http://www-i11.informatik.rwth-aachen.de/fileadmin/user_upload/Redakteure/Vorlesungen/06winter/ies/Using_interrupts.pdf> Consulta: Febrer de 2008]

31. UNIVERSIDAD NACIONAL DE QUILME
Diseño construcción de un madurómetro
<http://iaci.unq.edu.ar/materias/sistemas_digitales/proyectos/Madurometro.pdf>

ALTRES

32. GOOGLE
<http://www.google.es>