EPS
Escola Politècnica Superior
UdG

Projecte/Treball Fi de Carrera

Estudi: Enginyeria Tècn. Ind. Electrònica Ind. Pla 2002

Títol: Disseny de firmware per mòdul USB de microcontroladors PIC

Document: 1. Memòria

Alumne: Xavier Gelada Alfonso

Director/Tutor: Lluís Pacheco Valls
Departament: Arquitectura i Tecnologia de Computadors
Àrea: Arquitectura i Tecnologia de Computadors

Convocatòria (mes/any): Juny/2014

ÍNDEX

1 INTRODUCCIÓ

1.1 Antecedents

Des de l'aparició del port USB a principis dels anys noranta, s'ha anat imposant aquesta interfície de comunicació, primer en els ordinadors personals i després en tot tipus de dispositius electrònics. Tant és així que, en molts casos avui en dia, si es desitja un port sèrie o paral.lel al comprar un ordinador nou, s'ha de demanar de forma expressa.

Si bé és cert que el port USB ha solucionat moltes de les problemàtiques dels ports sèrie i paral.lel, aquest èxit en la seva implementació també s'ha d'entendre per la posició en el mercat mundial de les empreses que, en el seu dia, el van impulsar.

Sigui com sigui, els avantatges d'aquest port són més que evidents i el fet que el seu ús sigui tan generalitzat, en fa molt atractiu el seu coneixement i ús a qualsevol nivell, també a nivell acadèmic.

Durant molts anys ha estat impossible (o molt difícil) pel gran públic en general aproximar-se al port USB a nivell de dissenyador o desenvolupador d'aplicacions. Es tracta d'una interfície amb un grau de complexitat elevat pel que fa al hardware i que necessita d'un firmware molt dens, amb la dificultat afegida que es requereix de desenvolupar també un software dependent del sistema operatiu i, per tant, un "driver".

Per altra banda, la complexitat del hardware ha estat un impediment pels dissenyadors fins que les principals marques de microcontroladors del mercat no han comercialitzat unitats amb mòduls USB com a un perifèric més. Això ha facilitat l'ús del port i el disseny d'aplicacions. En qualsevol cas, cal tenir ben present que el mòdul USB d'un microcontrolador és una pàgina en blanc que cal omplir per crear el nostre dispositiu i per això no podem obviar el funcionament de la interfície USB a cap nivell.

Posar en funcionament una aplicació USB suposa, doncs, una triple feina de disseny de hardware, firmware i software. Encara que l'especificació USB contempla categories de dispositius a les que el dissenyador es pot adaptar (situació que simplifica especialment el desenvolupament del software i la configuració del dispositiu perquè funcioni sense necessitat de "driver"), el nostre dispositiu no sempre encaixarà totalment en alguna de les categories previstes.

L'especificació USB és tan àmplia i pretén cobrir tants supòsits que sovint es fa difícil trobar qüestions sistematitzades i aplicacions genèriques que siguin fàcilment adaptables a les necessitats singulars de cada projecte. Per sort si que existeixen "drivers" més o menys oberts i adaptables, que poden ser aprofitats per persones que no necessàriament tenen perquè tenir coneixements de programació a nivell de sistema operatiu (qüestió imprescindible per poder dissenyar un "driver" per a un dispositiu que ha de funcionar en relació a una computadora).

En conseqüència, amb coneixements d'electrònica i de programació a baix nivell, es pot arribar a dissenyar completament un dispositiu amb un port USB, amb l'ús del corresponent mòdul USB d'un microcontrolador, per poder intercanviar informació amb un ordinador personal i poder d'aquesta manera interactuar com es feia anteriorment amb ports paral.lel i sèrie. S'ha de tenir clar però, que l'ús del port USB requerirà d'un disseny més singular en cada cas, en funció de l'aplicació que es persegueix i que, malauradament, la portabilitat a altres aplicacions no serà tan directa.

1.2 Objecte

L'objectiu del present Projecte Final de Carrera és el disseny o programació d'un codi informàtic (firmware) que controli el mòdul USB d'un microcontrolador PIC, de manera que pugui ser utilitzat com a punt de partida pel disseny d'aplicacions senzilles on la necessitat principal sigui l'intercanvi de dades entre un ordinador personal i un hardware microcontrolat.

Com ja s'ha comentat anteriorment, per les pròpies característiques de les especificacions USB, resulta impossible dissenyar un firmware amb una portabilitat absoluta. Això implica que, en cadascun dels dissenys sobre els que es vulgui aplicar el firmware desenvolupat en aquest projecte, serà necessari un cert grau d'adaptació del codi, per petit que sigui. De totes maneres, gran part del mateix podrà ser directament reutilitzat.

Per portar a terme aquest objectiu, s'ha de partir necessàriament d'una aplicació concreta. Això és així perquè, com s'ha comentat insistentment, cal acotar necessàriament els paràmetres de funcionament de la interfície USB per arribar a algun resultat. L'especificació USB és tan àmplia que si no definim la nostra aplicació no queda definit un punt de partida clar. Però també perquè des d'un punt de vista didàctic facilita molt el desenvolupament de

l'aplicació ja que permet depurar el codi amb més facilitat. Una última raó, no menys important, és que suposa una garantia de funcionament i un resultat concret, demostrable.

En el present Projecte Final de Carrera s'ha escollit, com a aplicació concreta a dissenyar, un lector electrònic de targetes amb xip integrat (com les bancàries o identificatives) amb un microcontrolador PIC amb mòdul USB, de manera que, des d'un ordinador i via el port USB, es puguin enviar comandes a aquest tipus de targetes i rebre les corresponents respostes.

El disseny d'una aplicació concreta planteja una sèrie de reptes addicionals gens despreciables. Implica el disseny conjunt de hardware, firmware i software, amb la problemàtica afegida de la necessitat d'un "driver" relacionat amb el sistema operatiu on el nostre lector electrònic ha de funcionar. En qualsevol cas té l'avantatge de resultar en un sistema funcional, que dona respostes problemàtiques reals i no es queda en un projecte purament teòric on difícilment es podrien demostrar els resultats obtinguts i la perícia del dissenyador.

1.3 Especificacions i abast

A l'hora d'establir l'abast del present projecte, s'ha de començar per deixar clar que no es pot tractar ni d'un manual de microcontroladors, ni d'un tractat de programació, ni tampoc d'un manual sobre les especificacions USB. Ans al contrari. És necessari aproximar-se a aquest document amb coneixements bàsics de l'arquitectura PIC, coneixements de programació en assemblador (o com a mínim nocions, com més endavant es concreta), així com també amb un coneixement bàsic de la interfície USB.

Per tant, queda fora de l'abast d'aquest projecte realitzar un resum o explicació de les especificacions USB, les quals són extensíssimes i ja han estat resumides prèviament amb molt d'èxit en documents que es poden trobar a la xarxa i que figuren al capítol 10 de Bibliografia d'aquest document.

Igualment es poden trobar molts manuals de programació en assemblador per microcontroladors PIC, també a la xarxa i de manera totalment gratuïta, si bé aquest punt no té perquè ser tan problemàtic ja que aquest és un coneixement impartit al llarg de la present carrera universitària cursada.

Al tractar-se d'un projecte que comprèn qüestions tan diferents, també cal definir de manera clara l'abast i especificacions de cadascuna de les seves parts ja que, d'una altra manera, no es podria concretar cap resultat clar.

És necessari doncs establir, a més a més, limitacions específiques a nivell de hardware, firmware i software, així com també realitzar una explicació del tipus de dispositiu USB que es persegueix dissenyar dins de totes les opcions que ofereixen les especificacions d'aquesta interfície.

1.4 Especificacions i abast del hardware

El present projecte està enfocat i utilitza microcontroladors PIC, és a dir, de l'empresa americana "Microchip Technology Inc", la qual ofereix diferents famílies d'integrats segons les prestacions i capacitats que incorporen. Dins dels microcontroladors de 8 bits, només les famílies PIC16 i PIC18 presenten integrats amb mòduls USB. Val a dir que el funcionament d'aquest mòdul es pràcticament igual en ambdues famílies i, per tant, es pot aprendre el funcionament d'aquest mòdul amb independència de la família de microcontroladors amb la que es vulgui treballar. En qualsevol cas per la nostra aplicació s'ha utilitzat un microcontrolador de la família PIC18 (concretament el PIC 18F14K50) per la resta de prestacions i perifèrics que incorpora (apart del mòdul USB) i que ens són molt útils pel nostre disseny del lector electrònic de targetes, especialment:

- Mòdul PWM (modulador de polsos) per generar un senyal de CLOCK per la targeta.
- Mòdul EUART per l'intercanvi asíncron d'informació lector-targeta.
- Diverses interrupcions externes per detectar entrada/sortida de targeta en el lector.
- Possibilitat de programació de l'integrat sense treure'l de la PCB ("In Circuit Serial Programming" o ICSP)

Per programar el microcontrolador s'ha utilitzat un programador específic de la mateixa empresa anomenat PICKit3, compatible amb la programació ICSP, conjuntament amb el software també proporcionat per l'empresa MPLAB. En qualsevol cas això no condiciona de cap manera el projecte i cada dissenyador o desenvolupador pot escollir les eines i sistemes de programació del microcontrolador PIC, essent possible des de fabricar-se el propi hardware programador, seguint esquemes que es poden trobar a la xarxa, fins a utilitzar altres softwares de programació totalment lliures i sobradament provats com pot ser per exemple IC-PROG.

En el present Projecte s'inclou una explicació detallada del funcionament del mòdul USB de l'integrat escollit (PIC18F14K50), ja que l'objectiu central d'aquest document és obtenir un codi que faci funcionar aquest mòdul. El que també queda fora de l'abast d'aquest projecte és explicar altres qüestions generals de microcontroladors PIC com per exemple el funcionament de les portes d'entrada i sortida, l'oscil.lador del sistema, el sistema d'interrupcions, etc. Es tracta de qüestions elementals del funcionament de qualsevol microcontrolador que es donen per sabudes. En qualsevol cas, les explicacions i comentaris que acompanyen cadascuna de les línies del firmware són la millor manera per aprendre el funcionament de qualsevol microcontrolador.

1.5 Especificacions i abast del firmware

Un cop definit el hardware que s'utilitzarà en l'apartat anterior, és fàcil entendre que queden també moltes característiques fixades pel que fa al firmware. Per començar el codi s'haurà de desenvolupar en algun dels tres llenguatges usats per microcontroladors PIC: llenguatge assemblador, llenguatge C o llenguatge BASIC.

En aquest projecte s'ha escollit el llenguatge assemblador per PIC pels següents motius:

- És el llenguatge de programació en el que l'autor del projecte va ser instruït a la Universitat de Girona per la programació a baix nivell.
- És un llenguatge encara acceptat i reconegut per l'empresa fabricant.
- El software compilador que presenta menys conflictes amb el hardware (el proporcionat per la mateixa empresa fabricant) és totalment gratuït.
- Permet un control complert dels temps d'execució i dels registres del microcontrolador i, a nivell acadèmic, exigeix un coneixement més profund del hardware.

També és cert que el codi en assemblador sempre resulta més optimitzat un cop compilat que el que obtenim en llenguatge C, però val a dir que en aquest cas tenim espai en memòria més que suficient ja que, tot i que el firmware acaba essent molt extens, disposem de fins a 16KB de memòria destinada a instruccions.

1.6 Especificacions i abast del software

Pel que fa al software que cal dissenyar perquè funcioni a l'ordinador personal, s'ha de separar el que és pròpiament el programa o aplicació que ha de poder enviar comandes al lector i rebre'n les respostes, del "driver" necessari per permetre que tot això es faci a través del port USB d'un ordinador on hi ha instal.lat un determinat sistema operatiu.

Ambdues qüestions (software i "driver") requereixen d'uns coneixements de programació d'alt nivell molt elevats, fins al punt que molts enginyers informàtics tindrien dificultats per aconseguir programar un "driver" de control del port USB sota entorn Windows, o fins i tot, del software d'aplicació ja que també exigeix un coneixement molt profund del funcionament del sistema operatiu. En conseqüència s'ha optat pel següent:

Pel que fa al software s'ha adaptat un programa existent a la xarxa, amb l'autorització de la seva autora (Jan Axelson) pel qual s'ha hagut de modificar només una part del codi perquè s'adaptés a les necessitats de la nostra aplicació. El codi està present en dos llenguatges de programació: Visual Basic i C#. S'ha escollit el primer per la facilitat que ofereix a no programadors per aconseguir interfícies gràfiques en entorn Windows. Per tant, sense ser necessari tenir grans coneixements de Visual Basic, en calen nocions elementals, així com també de programació general en llenguatges d'alt nivell.

Pel que fa al "driver", queda resolt tenint en compte que el software anterior està optimitzat per funcionar amb un "driver" genèric proveït per Microsoft, anomenat WinUSB. Es tracta d'un "driver" dissenyat per posar en funcionament petites aplicacions com la nostra, i que presenta una sèrie de limitacions d'ús que no ens afecten.

Més endavant, s'explica amb més detall les qüestions relatives al software, però en qualsevol cas és important entendre que tant el programa de windows com el "driver" utilitzats, són adaptacions o modificacions de codi ja existent (dissenyat en part perquè pugui ser reutilitzat per altres dissenys, com ha estat el cas). En conseqüència, en el present Projecte s'explicarà el procediment seguit per a l'hora d'adaptar i modificar aquests softwares.

1.7 Especificacions i abast del dispositiu USB

Resulta impossible abordar cap projecte USB sense acotar els modes i paràmetres de funcionament. L'especificació USB es va desenvolupar en el seu dia pensant a oferir una gran versatilitat a tots els nivells de la interfície, fins al punt que hom es pot perdre molt fàcilment. Per això resulta imprescindible definir per endavant els paràmetres de funcionament del nostre dispositiu. Si bé no forma part de l'objecte d'aquest Projecte explicar o resumir aquestes especificacions, a continuació es detallen les característiques de funcionament del dispositiu dissenyat i una breu explicació perquè el lector tingui una noció més ajustada del que es fa referència:

- Mode de funcionament FS. El dispositiu treballarà a l'anomenada velocitat "Full-Speed" o "FS", segons la qual el bitrate teòric és de 12Mb/s. Existeixen en l'actualitat, segons l'última versió USB (3.0), quatre velocitats compatibles de funcionament: Low-Speed, Full-Speed, High-Speed i Super-Speed. La velocitat FS és la màxima a la que pot treballar el mòdul USB del nostre microcontrolador.

- Transferències tipus Control. Existeixen diferents tipus de transferències definides per l'estàndard USB. La més elemental i que tot dispositiu USB ha de suportar és la de tipus Control. Aquest tipus de transferència és l'usat, com a mínim, per configurar tot dispositiu USB quan és connectat al bus, però també pot ser utilitzat per transferir dades, com en el nostre cas.

- Dispositiu USB versió 1.1 amb un sol Endpoint (número 0). El dispositiu que utilitzarem el definirem com compatible amb la versió 1.1 de l'especificació USB, amb la qual cosa evitem que l'ordinador intenti fer proves per fer-lo funcionar a velocitats més altes. Tot dispositiu USB ha de tenir com a mínim un Endpoint (registre on es reben i envien dades) numerat amb un zero, que és l'usat per la configuració i també, si així ho definim per rebre i enviar dades.

- Dispositiu tipus "Vendor-Specific". L'especificació dona un llistat de dispositius tipus, reconeguts, pels quals no cal "driver" específic perquè el seu funcionament és conegut (per exemple: teclats, ratolins, altaveus, impressores...). Val a dir que també poden existir productes d'aquest tipus que el fabricant no vulgui que segueixin la categoria USB perquè això també condiciona el funcionament i les prestacions del dispositiu. En qualsevol cas, el nostre dispositiu no s'adapta a cap categoria i en conseqüència és definit com a "Vendor-Specific". Per tant requereix d'un "driver" perquè pugui ser configurat i usat en un ordinador personal.

- Dispositiu amb una sola configuració i una sola interfície. L'especificació USB permet definir vàries configuracions possibles per a un mateix dispositiu així com vàries interfícies. El primer cas bàsicament s'utilitza en cas que el dispositiu USB tingui dos modes de funcionament diferents que impliquin consums elèctrics diferents. El segon en cas que tingui dues funcions clarament diferenciades. Nosaltres configurarem el nostre lector com alimentat del bus (màx. 100 mA).

A grans trets, la definició d'aquests paràmetres limita les possibilitats del dispositiu i el concreta dins les nombroses possibilitats que comprèn l'especificació USB.

Cal pensar que si es vol fer funcionar el dispositiu amb altres paràmetres, caldrà modificar o bé el hardware, o bé el firmware o bé el software. Per exemple, per fer funcionar el dispositiu a velocitat LS, caldria només commutar dos pins del hardware. En canvi, per utilitzar un altre tipus de transferències, caldria modificar el firmware i també el software.

En qualsevol cas, per petites aplicacions sense requeriments molt específics, la configuració escollida hardware-firmware-software en aquest Projecte serà vàlida en la immensa majoria dels casos.

2 CONEIXEMENTS PREVIS

2.1 Característiques principals del PIC

Tot i que es pot acudir a la fitxa de característiques tècniques de l'integrat, es considera pràctic incloure a continuació, a mode esquemàtic, les principals característiques del microcontrolador utilitzat per tenir una idea del seu potencial. Qualsevol informació addicional que es pugui necessitar es pot obtenir del manual complert en format pdf que es pot descarregar des de la referència que figura al capítol 10 de Bibliografia.
La següent taula, Taula número 1, mostra les prestacions principals de l'integrat PIC18F14K50:

Memòria Flash de programa:	16 Kbytes
Memòria SRAM de dades:	768 Bytes
Memòria EEPROM de dades:	256 Bytes
Pins destinats a I/O:	15
Canals A/D (10 bit):	11
Mòdul PWM:	1
Bus SPI:	Si
Bus I2C:	Si
EUSART:	1
Comparadors:	2
Timers (8/16 bits):	1/3
Mòdul USB:	Si

Taula número 1

La Figura número 1 mostra un diagrama de pins complert. Aquesta figura ha estat extreta del datasheet del microcontrolador. Aquest datasheet pot obtenir-se a través de la referència inclosa al capítol 10 Bibliografia:

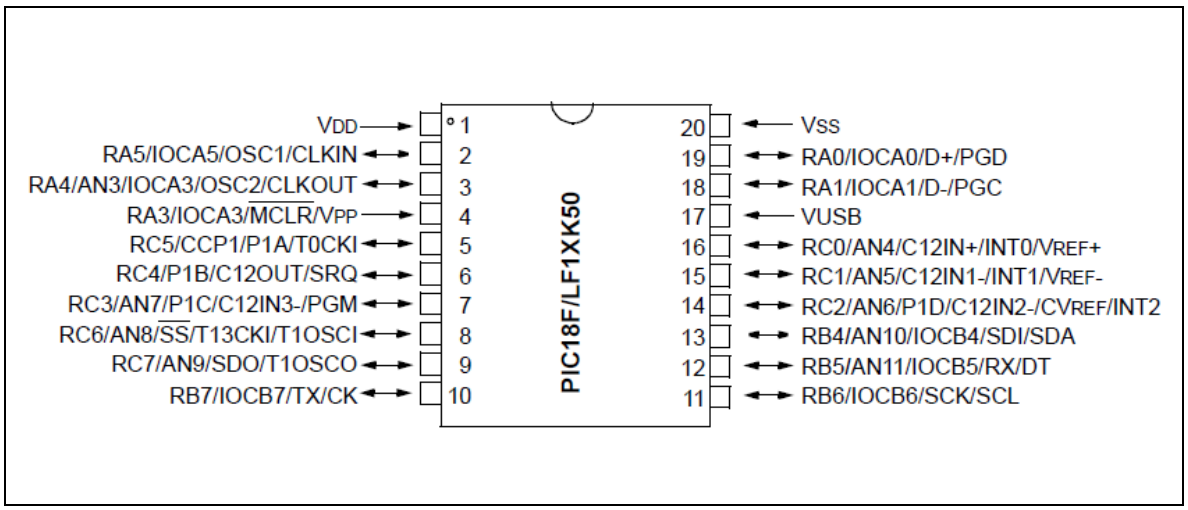


Figura número 1. Diagrama de pins.

En últim lloc, s'inclou un llistat més detallat de les característiques que ens són expressament d'utilitat per a la nostra aplicació:

1) Característiques del mòdul USB:

- Compatible amb la versió 2.0
- Possibilitat de FullSpeed (12Mb/s) i LowSpeed (1.5 Mb/s)
- Suport de tots els tipus de transferència (Control, Bulk, Interrupt i Isochronous)
- Possibilitat de fins a 16 Endpoints (o 8 bidireccionals)
- 256 Bytes de RAM destinats a USB
- Detecció física de connexió al host a través de les línies D+/D-

2) Característiques del mòdul PWM:

- Seleccionables 1, 2, 3 o 4 sortides PWM
- Polaritat commutable
- Inici i parada automàtica

3) Característiques del mòdul EUSART:

- Suport dels estàndards RS232 i RS485
- Funcionament amb oscil.lador intern
- Detecció automàtica de Baudrate

4) Pins destinats a I/O

- 15 pins en total (1 només input)
- Intensitat I/O: 25 mA/25mA
- 7 programables per generar interrupció al canviar d'estat
- 3 programables per generar interrupció externa

2.2 Funcionament del mòdul USB del PIC

Per poder comprendre el funcionament del mòdul USB dels microcontroladors PIC, és necessari conèixer prèviament les principals característiques del protocol USB i del funcionament de la interfície USB. Al capítol 10 de Bibliografia hi ha recollits els enllaços a

diversos documents de la xarxa on s'han realitzat resums de les especificacions USB enfocades al disseny d'aplicacions, els quals es recomana llegir abans d'aquest apartat.

El control del mòdul USB dels microcontroladors PIC es realitza amb 14 registres, recollits a la Taula número 2.

UCON	USB Control Register
UCFG	USB Configuration Register
USTAT	USB Transfer Status Register
UADDR	USB Device Address Register
UFRMH:UFRML	Frame Number Registers
UEP0-UEP7	Endpoint Enable Registers

Taula número 2

Encara hi ha alguns registres addicionals relatius a les interrupcions. En qualsevol cas, per la present aplicació no s'utilitzaran tots.

Com ja s'ha comentat, part de la RAM del microcontrolador està destinada al mòdul USB (en total 256 bytes). Cal dir que no hi ha cap limitació d'escriptura i ha de ser el dissenyador el que tingui present en tot moment amb quines adreces de RAM està treballant per evitar conflictes. Més endavant es tracta el tema de la RAM USB amb més detall.

La Figura número 2 mostra un diagrama esquemàtic del perifèric USB que incorpora l'integrat. Aquesta figura ha estat extreta del datasheet del microcontrolador. Aquest datasheet pot obtenir-se a través de la referència inclosa al capítol 10 Bibliografia.

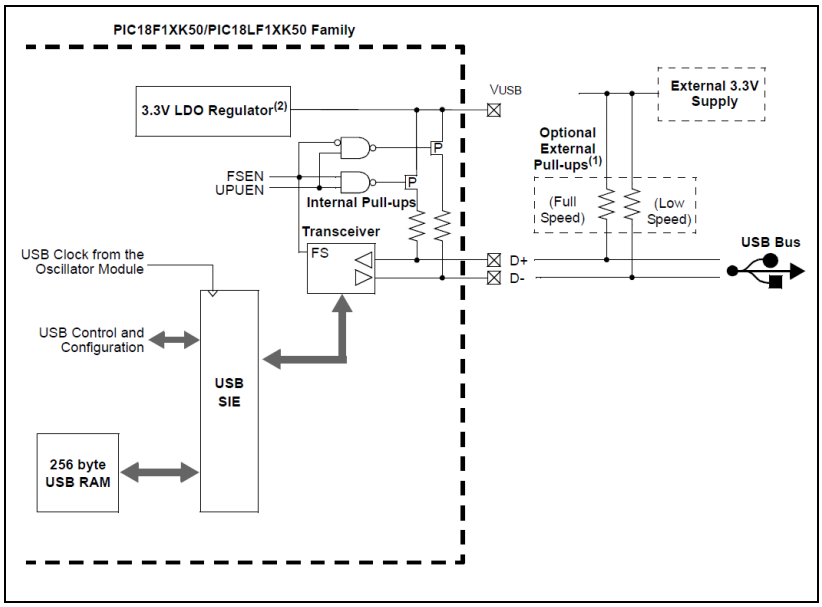


Figura número 2. Diagrama perifèric USB.

Els bits FSEN i UPUEN determinen si es treballa en FS o en LS, segons les resistències de pull-up que s'activin, amb la possibilitat de desactivar-les ambdues per incorporar-les externament.

Es pot optar per alimentar el perifèric externament a través del pin V_{USB}, o bé internament a través del regulador de voltatge necessari per fer funcionar el mòdul als 3.3V que les especificacions fixen.

Portant a massa el pin V_{USB}, usem el regulador intern. En aquest últim cas el fabricant demana interposar un condensador de 470 nF.

En els següents subapartats, es van definint totes les qüestions relatives al funcionament d'aquest perifèric.

2.2.1 Frequència de funcionament del mòdul USB

El PIC18F14K50 incorpora dos oscil.ladors externs (primari i secondari) i un d'intern, per la qual cosa, la freqüència d'oscil.lació que acaba arribant a la CPU pot tenir fonts diverses i es pot aconseguir per camins ben diferents ja que, a més a més, hi ha presents mòduls multiplicadors i divisors al llarg del camí.

Per altre costat, és important dir que el mòdul USB té els seus propis camins a través dels quals li arriba aquesta senyal d'oscil.lador i per tant, pot estar més directament relacionada o no amb la senyal que arriba a la CPU.

Per tenir una noció de la complexitat del mòdul oscil.lador del PIC, s'adjunta a continuació la Figura número 3, on hi ha un diagrama esquemàtic de totes les opcions.

Aquesta figura ha estat extreta del datasheet del microcontrolador. Aquest datasheet pot obtenir-se a través de la referència inclosa al capítol 10 Bibliografia.

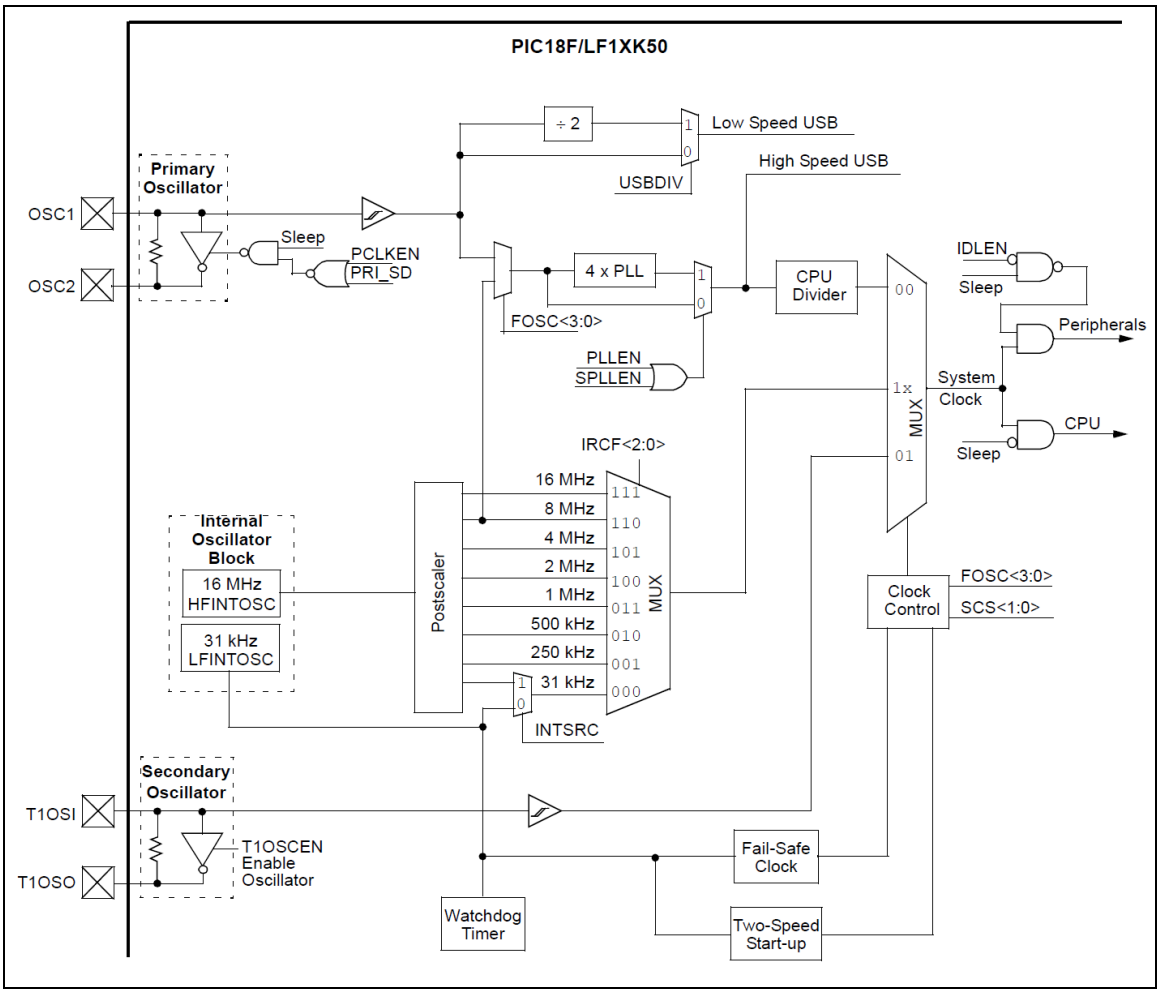


Figura número 3. Diagrama mòdul oscil.lador.

Observem que, addicionalment, les possibilitats canvien en funció de si treballem amb velocitat LS o FS. Si, com és el nostre cas, volem treballar amb FS, les opcions que tenim són les recollides a la Taula número 3:

Mode de Clock	Freqüència	4xPLL Activat	CPUDIV<1:0>	Freqüència del clock de sistema
EC High	48 MHz	NO	00	48 MHz
			01	24 MHz
			10	16 MHz
			11	12 MHz
EC High / HS	12 MHz	SI	00	48 MHz
			01	24 MHz
			10	16 MHz
			11	12 MHz

Taula número 3

Com es veurà a l'apartat següent, on queda reflectit el circuit de la nostra aplicació, nosaltres treballarem amb un cristall de quars de 12 MHz i el multiplicador activat perquè, d'aquesta

manera, el mòdul USB treballi a 48 MHz. La nostra CPU també treballarà a 48 MHz perquè no usarem el divisor intermedi (CPUDIV = 00).

2.2.2 Registre UCON

Aquest registre permet configurar el comportament del mòdul durant les transferències de dades. Entre d'altres coses controla els següents aspectes:

- Activa/Desactiva el mòdul USB
- Configura els buffers ping-pong
- Controla el mode de suspensió
- Activa o desactiva les transferències de "packets"

La Taula número 4 mostra els diferents bits del registre UCON:

b7	Registre: UCON						b0
---	PPBRST	SEO	PKTDIS	USBEN	RESUME	SUSPND	---

Taula número 4

1) PPBRST: bit de reset dels buffers ping-pong. Aquests buffers són d'ús opcional i tenen funció de seguiment de l'intercanvi de dades per evitar errors. No són usats en la present aplicació.
2) SEO: bit senyalitzador de la detecció d'una situació "Single-Ended-Zero" al bus, per la qual ambdues línies diferencials del bus USB estan a nivell baix. Un valor de 1 indica detecció d'aquesta situació al bus. Bit de només lectura. Monitorejant aquest bit podem saber quan sortim de l'estat de reset inicial i el nostre device arriba a l'estat "power-up".
3) PKTDIS: bit per activar/desactivar la transferència de paquets. Un valor de 1 indica que la recepció de paquets està desactivada. Aquest bit automàticament es posa a 1 quan es rep un "token" de tipus "setup". Bit de lectura i únicament resetejable. Aquest bit és activat per la SIE del mòdul quan es rep un "token setup", per poder fer el procés de "setup" del protocol USB. Només pot ser desactivat via software, fet que retorna la possibilitat de transmissió i recepció.
4) USBEN: bit que permet activar i desactivar tot el mòdul USB. Un valor de 1 indica perifèrica activat. Amb aquest bit es pot resetejar el mòdul via software. Bit de lectura i escriptura. Aquest bit activa també les resistències internes de pull-up i requereix de tenir

el mòdul totalment configurat i amb el clock funcionant abans no sigui activat. Si usem el mòdul PLL, cal esperar uns 2 ms entre que l'activem i posem a funcionar el perifèric.

5) RESUME: bit per activar la senyal de "resume". Un valor de 1 indica senyal actiu. Bit de lectura i escriptura. Aquest bit permet fer l'anomenat "remote wake-up" que la nostra aplicació no utilitza.
6) SUSPND: bit que permet posar el perifèric en estat de suspensió. Un valor de 1 posa el perifèric en mode de suspensió. Bit de lectura i escriptura. Aquest estat de suspensió porta el mòdul a un estat de baix consum, de manera que el clock d'entrada de la SIE també es desactiva. Aquest bit hauria d'activar-se com a resposta a una interrupció de tipus IDLEIF. Quan el bit val 1, el dispositiu està connectat al bus però l'estat del "transceiver" és IDLE.

2.2.3 Registre UCFG

Aquest registre permet seleccionar diferents característiques de funcionament del mòdul.
La Taula número 5 mostra els diferents bits del registre UCFG:

b7	Registre: UCFG						b0
UTEYE	---	---	UPUEN	---	FSEN	PPB1	PPB0

Taula número 5

1) UTEYE: bit utilitzat per la calibració i test del perifèric. Sense utilitat.
2) UPUEN: bit per activar o desactivar les resistències internes de pull-up. Un valor de 1 indica que les resistències estan actives (una o altra segons el següent bit). Bit de lectura i escriptura.
3) FSEN: bit per activar o desactivar FullSpeed. Un valor de 1 indica que la velocitat del mòdul USB serà FS (caldrà el clock adequat segons s'ha explicat en el subapartat 2.2.1). Bit de lectura i escriptura.
4) PPB1-PPB0: bits de configuració dels buffers ping-pong. Aquests buffers no són utilitzats en la present aplicació.

2.2.4 Registre USTAT

Aquest registre permet l'estat del procés de comunicació de la SIE. Quan la SIE genera una interrupció per indicar que hi ha una transferència complerta, cal acudir a aquest registre per

saber-ne l'estat. Un cop generada la interrupció, s'ha d'esperar com a mínim dos cicles de clock USB per consultar el registre USTAT.

Bàsicament, trobarem en aquest registre el número d'endpoint al que fa referència la transferència activa, la direcció de la transferència i dades referents als buffers ping-pong en cas de ser utilitzada aquesta característica.

El registre USTAT és una memòria FIFO de quatre posicions, de manera que es poden anar rebent dades alhora que la SIE les despatxa. Aquesta configuració queda reflectida a la Figura número 4. Aquesta figura ha estat extreta del datasheet del microcontrolador. Aquest datasheet pot obtenir-se a través de la referència inclosa al capítol 10 Bibliografia:

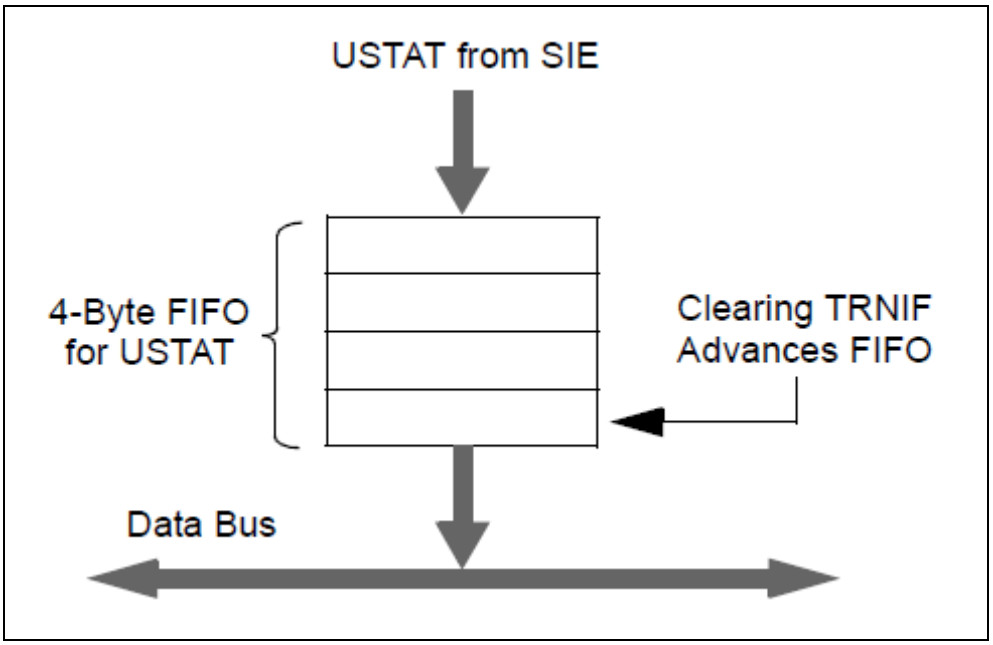


Figura número 4. Memòria FIFO registre USTAT.

La idea és que, quan s'activa el flag d'interrupció USB (TRNIF), s'ha de llegir USTAT i tornar a posar a zero TRNIF. Si hi ha una altra dada esperant, saltarem de posició a la FIFO i es tornarà a activar el flag d'interrupció. Així successivament fins que la FIFO estigui buida.

La Taula número 6 mostra els diferents bits del registre USTAT:

b7	Registre: USTAT						b0
---	---	ENDP2	ENDP1	ENDP0	DIR	PPBI	---

Taula número 6

1) ENDP2-0: bits que indiquen l'endpoint al que fa referència l'última transferència (des de endpoint 0 que correspon als valors <000> fins a l'endpoint 7 que correspon als valors <111>. Bits de lectura únicament.
2) DIR: bit que indica la direcció de la transferència realitzada. Un valor de 1 indica que l'última transacció correspon a un "IN TOKEN". Un valor de 0 indica que correspon a un "OUT TOKEN" o "SETUP TOKEN". Bit de només lectura.
3) PPBI: bit que fa referència als buffers ping-pong i que no aplica en el nostre cas.

2.2.5 Registres UEPn

Aquests registres permeten controlar els diferents endpoints que definim pel nostre dispositiu USB. Disposem de fins a vuit registres (UEP0 a UEP7).

La Taula número 7 mostra els diferents bits d'un registre UEPn:

b7	Registre: UEPn						b0
---	---	---	EPHSHK	EPCONDIS	EPOUTEN	EPINEN	EPSTALL

Taula número 7

1) EPHSHK: bit per activar o desactivar el handshaking (sempre actiu excepte per transferències de tipus "isochronous"). Un valor de 1 activa el handshaking. Bit de lectura i escriptura.
2) EPCONDIS: bit per controlar el tipus de transferències que admet l'endpoint. Un valor de 1 indica que l'endpoint només admet transaccions IN i OUT però no SETUP, per tant no admet transferències de control. Un valor de 0 indica que admet tot tipus de transaccions. Aquest bit és de lectura i escriptura i només aplica si EPOUTEN i EPINEN valen 1 en ambdós casos.
3) EPOUTEN: bit d'activació o desactivació de les transaccions OUT. Bit de lectura i escriptura.
4) EPINEN: bit d'activació o desactivació de les transaccions IN. Bit de lectura i escriptura.
5) EPSTALL: bit per posar l'endpoint en estat STALL. Bit de lectura i escriptura, només funcional si l'endpoint està actiu.

2.2.6 Registre UADDR

Registre que guarda l'adreça que el nostre dispositiu USB rebrà per ser identificat de manera unívoca dins del bus USB. Després del reset, per defecte es posarà a 00h.

Després del procés d'enumeració descrit al protocol USB, l'adreça rebuda pel host USB, serà emmagatzemada en aquest registre.

La Taula número 8 mostra els diferents bits del registre UADDR que bàsicament configuren un valor binari de 7 bits:

b7	Registre: UADDR						b0
---	ADDR6	ADDR5	ADDR4	ADDR3	ADDR2	ADDR1	ADDR0

Taula número 8

2.2.7 Registres UFRMH/L

Aquests dos registres permeten guardar el número de frame durant la comunicació USB. Es tracta de registres que tenen sentit només en transferències de tipus "isochronous".

2.2.8 RAM destinada al mòdul USB

Tota una zona de la memòria RAM té accés dual (hi pot accedir el microcontrolador i el mòdul USB, concretament la SIE).

Aquesta part de RAM es situa al bank2, concretament de la adreça 200h a la 2FFh. De manera més específica, podem usar per definir buffers de control del nostre dispositiu USB des de l'adreça 200h a la 27Fh.

La Figura número 5 mostra la distribució general de memòria RAM i la zona reservada per l'ús USB. Aquesta figura ha estat extreta del datasheet del microcontrolador.

Aquest datasheet pot obtenir-se a través de la referència inclosa al capítol 10 Bibliografia.

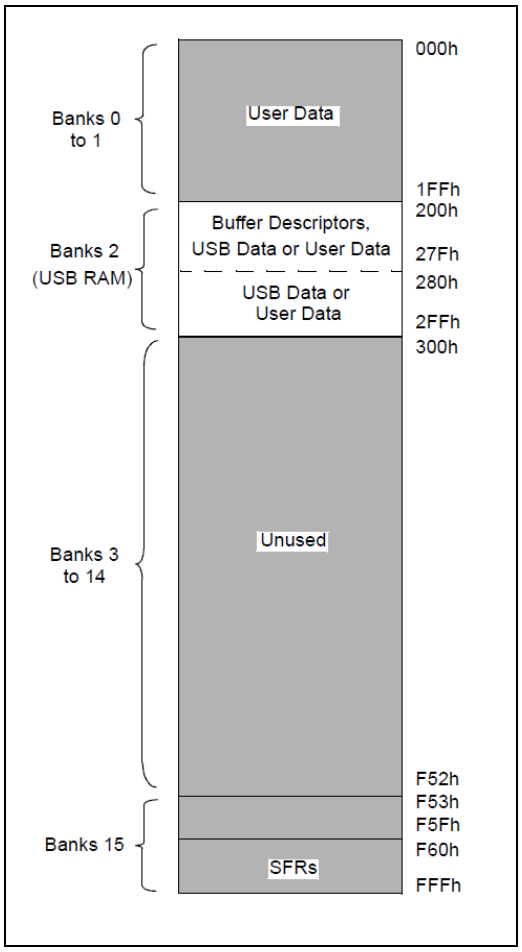


Figura número 5. Distribució memòria RAM.

2.2.9 Buffers descriptors i BDT (Buffer Descriptor Table)

La memòria RAM destinada a USB s'ha de començar ocupant amb una sèrie de buffers descriptors pels diferents endpoints existents en el nostre dispositiu, que conformen l'anomenada "Buffer Descriptor Table".

La idea d'aquesta taula de buffers seria la que es mostra a la Figura número 6. Aquesta figura ha estat extreta del datasheet del microcontrolador.

Aquest datasheet pot obtenir-se a través de la referència inclosa al capítol 10 Bibliografia:

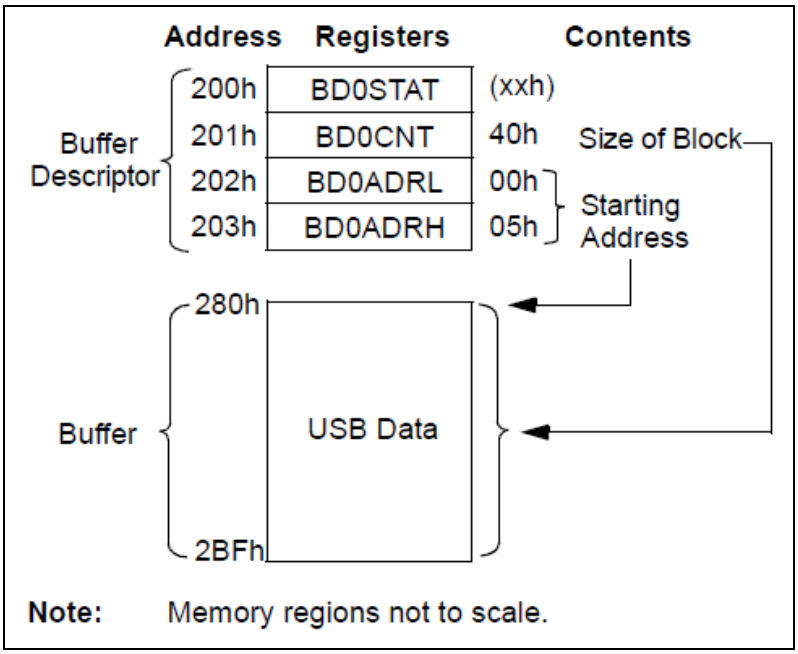


Figura número 6. Taula de buffers descriptors.

El primer descriptor sempre serà present i farà referència a l'endpoint 0, el qual forçosament ha d'existir per a qualsevol dispositiu USB que vulguem implementar. Aquest serà l'únic que usarem en la nostra aplicació. Com es pot veure a l'anterior figura, un buffer descriptor consta de 4 posicions de memòria RAM destinades a definir, respectivament, una configuració i control del buffer, una quantitat de bytes, i una ubicació de les dades d'aquest buffer dins la memòria RAM destinada a USB però començant per l'adreça 280h ja que l'anterior està reservada per successius descriptors que hi pugui haver. En la figura anterior hi ha definit un buffer descriptor per l'endpoint 0 (el primer) que disposa d'una longitud de 64 bytes (40h) i indica una adreça inicial per les dades (els valors correctes per BD0ADRL i BD0ADRH, en coherència amb la resta de la figura, serien respectivament 80h i 02h).

Si un endpoint no està activat, els seus registres no seran usats, però caldrà en tot cas establir primer els buffers descriptors abans de poder activar cap endpoint.

Per tant, aquests descriptors inicials determinen la mida del buffer així com la seva configuració i control. Aquesta configuració i control, com ja s'ha comentat, s'aconsegueix amb la primera de les posicions de memòria dels descriptors dels buffers: BD0STAT. Aquest registre presenta dues configuracions diferents segons que tingui el control de registre en un determinat moment, la CPU o el perifèric USB. En uns determinats moments de la comunicació USB, serà la CPU la que ha d'accedir als buffers descriptors i llavors aquest registre es composa dels bits mostrats a la Taula número 9:

b7	Registre: BD0STAT (Controlat per CPU)						b0
UOWN=0	DTS	---	---	DTSEN	BSTALL	BC9	BC8

Taula número 9

En altres moments, serà la SIE la que haurà de treballar amb aquest registre i llavors la CPU no podrà accedir-hi només que per lectura. Quan és el mòdul USB el que té el control del registre BD0STAT, els bits dels que es composa són els que es poden observar a la Taula número 10:

b7	Registre: BD0STAT (Controlat per SIE)						b0
UOWN=1	---	PID3	PID2	PID1	PID0	BC9	BC8

Taula número 10

Com es pot observar en les dues taules anteriors, atorguem el control d'aquest registre a un o altre part commutant el valor del bit de més pes (UOWN).

El procés de funcionament seria a grans trets el que s'inclou a continuació:

- Al definir els buffers descriptors, abans d'activar els endpoints relacionats, donarem valors als registres BD0STAT amb UOWN=0 perquè serà la CPU la que en aquell moment té el control del registre.
- Un cop omplerts els descriptors com ens convingui i iniciat e procés de comunicació, assignarem UOWN = 1 (passem control a la SIE). Aquesta accedirà als valors dels buffers que hem donat, però sobreescriurà els valors com li convingui i tornarà el control a la CPU, que podrà obtenir els valors proporcionats per la SIE.
- La CPU tornarà a omplir els registres com li convingui i es repetirà el procés.

A continuació s'inclou l'explicació dels diferents bits de BD0STAT que mostren les dues taules anteriors:

1) DTS: bit anomenat "Data Toggle Synchronization bit" pel protocol USB. Permet definir si estem enviant o hem rebut un "Data packet 1" (valor del bit igual a 1) o bé un "Data packet 0" (valor del bit igual 0). Bit de lectura i escriptura que és ignorat si el següent bit no està activat.
2) DTSEN: bit que activa o desactiva el procés de comprovació del "Data Toggle Synchronization bit". Un valor de 1 activa aquest procés que permet monitoritzar el

procés de comunicació amb el dispositiu, mentre que un valor de 0 desactiva aquesta mesura de control de l'integritat de les transaccions. Bit de lectura i escriptura.

3) BSTALL: bit que permet posar el buffer en mode STALL. Un valor de 1 activa aquest estat i un valor de 0 el desactiva. Bit de lectura i escriptura.
4) BC9/8: aquests dos bits venen a completar el valor dels registres BD0CNT, obtenint així un comptador de 10 bits (fins a 1024 bytes) per controlar les dades enviades o rebudes.
5) PID3/2/1/0: bits que configuren un codi identificador del packet d'informació rebut a l'última transferència.

2.2.10 Interrupcions USB

El mòdul USB pot generar moltes interrupcions diferents com a conseqüència del seu funcionament. Totes elles acabaran activant el senyalitzador USBIF que trobem al registre general d'interrupcions PIR2. Quan es dona aquesta senyalització, serà necessari discriminar quina font d'interrupció USB s'ha activat en primer terme.

La Figura número 7 mostra un diagrama amb totes les possibles interrupcions USB que es poden activar i la seva relació per acabar en tots els casos activant USBIF. Aquesta figura ha estat extreta del datasheet del microcontrolador. Aquest datasheet pot obtenir-se a través de la referència inclosa al capítol 10 Bibliografia:

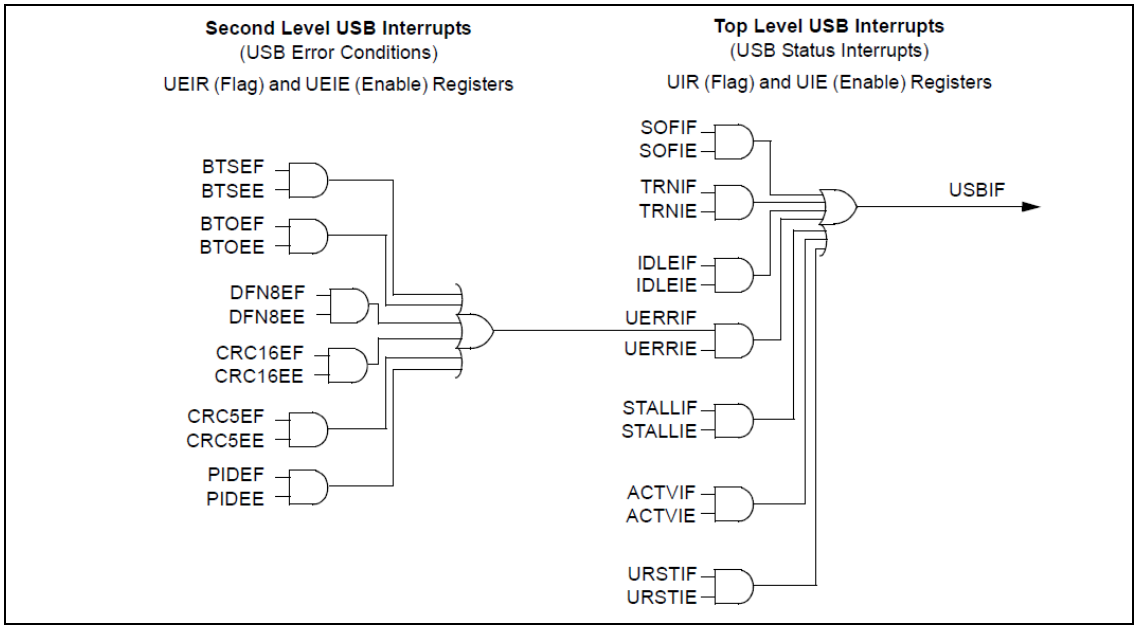


Figura número 7. Diagrama d'interrupcions USB.

Hi ha 4 registres destinats a contenir bits indicadors d'interrupcions USB:

- UIR: interrupcions generals USB
- UIE: activem o desactivem interrupcions generals USB
- UEIR: interrupcions d'errors USB
- UEIE: activem o desactivem interrupcions d'errors USB

No entrarem en detall en cadascuna d'elles perquè el protocol USB en cas d'error, com a principal mesura de notificació a l'altre interlocutor, determina no respondre. Això indica la necessitat de repetir el procés de comunicació i les peticions prèvies. Com es veurà en el capítol en el que s'explica el firmware, això simplifica molt el tractament d'errors de comunicació en la nostra aplicació.

2.2.11 Modes d'alimentació USB

L'especificació USB determina dues possibles modes d'alimentació dels dispositius USB que s'hagin de connectar al bus: "self-powered" (auto-alimentat) o "bus-powered" (alimentació a través del bus).

És important saber que el consum màxim que es permet de manera general i en funcionament normal per a un dispositiu USB connectat al bus són 100 mA. En el cas que no respectem aquesta norma i es demandi més intensitat, el dispositiu serà desconnectat del bus.

Per altra banda, tot dispositiu ha de poder funcionar en un mode de consum reduït, quan de manera continuada no estigui actiu. En aquest cas es demana que el consum no superi els 2.5 mA tot i que aquesta situació, a diferència de l'anterior no és monitorejada i queda sota la responsabilitat del dissenyador.

El consum de la nostra aplicació és inferior als 100 mA en mode de funcionament normal i lleugerament superior als 2.5 mA en mode de baix consum, sense que això impliqui cap conflicte ni problemàtica en el bus.

2.2.12 Instruccions especials utilitzades

S'inclouen en aquest subapartat algunes instruccions del microcontrolador que, no essent d'ús massa habitual, resulten indispensables per la nostra aplicació. Es tracta d'instruccions pensades pel treball en taules dins la memòria de programa.

- TBLPTR: es tracta d'un punter de 21 bits que apunta a una direcció de la memòria de programa on teòricament hi ha d'haver informació formant part d'una taula d'informació (no instruccions, sinó dades). Aquest punter es divideix en les tres parts TBLPTRL, TBLPTRH i TBLPTRU.
- TABLAT: latch de 8 bits on es pot guardar part del contingut d'una posició de memòria de programa.
- TBLRD*: instrucció que copia el contingut al que apunta TBLPTR i el posa a TABLAT.

L'ús d'aquests tres elements, permetrà llegir informació (dades) de la memòria de programa i poder-hi treballar. Aquesta qüestió és fonamental perquè durant el procés de configuració d'un dispositiu USB per part d'un ordinador, se li reclama una sèrie d'informació que l'identifica i explica a l'ordinador el seu funcionament. Serà aquest el moment el que es recuperarà aquesta informació prèviament gravada a la memòria de programa i s'enviarà a l'ordinador.

2.3 Disseny del lector electrònic de targetes

El circuit lector de targetes intel.ligents (també anomenades "smartcards") queda totalment grafiat al document de plànols d'aquest projecte, concretament al plànol número 1.

En qualsevol cas, hi ha alguns aspectes a comentar de forma escrita per tenir una millor comprensió del disseny.

En primer lloc, el dispositiu electrònic està dissenyat per alimentar-se a través del bus USB, com s'ha comentat en l'apartat anterior. De totes maneres es pot observar al circuit que existeix també una part destinada a l'alimentació externa a través d'un connector tipus Jack 1.5 mm. amb l'ús d'un regulador de tensió LM7805. Aquesta alimentació externa s'ha conservat del disseny original, el qual es va projectar així per dos motius. Inicialment no es coneixia amb exactitud el consum en mode d'operació normal i en mode reduït del circuit i es va considerar més prudent fer la previsió d'alimentació externa. Per altra banda, durant el procés de disseny s'han hagut de fer moltes proves de parts parcials del circuit fins que s'ha

aconseguit fer funcionar el conjunt. Aquestes proves requerien d'alimentació i, no estant la part USB acabada, s'ha pogut continuar amb el desenvolupament del disseny de manera independent.

Una altra qüestió a comentar és l'existència dels cinc díodes LEDs (D3 a D7). De manera similar al cas de l'alimentació externa, aquests components s'han utilitzat de recolzament durant el desenvolupament de la placa i per la depuració del codi assemblador. Han permès saber en quins punts el firmware es quedava encallat o quines subrutines presentaven mal funcionament. Un cop acabat i funcionant el disseny, s'han destinat a senyalitzar l'estat del dispositiu dins del bus USB:

- D3: Indica "power-state", el mòdul USB està alimentat correctament.

- D4: Indica "default-state", el dispositiu ha rebut correctament la senyal de reset del bus i ha començat el procés d'enumeració per rebre una adreça única que permetrà identificar unívocament el dispositiu dins el bus USB.

- D5: Indica "adressed-state", el dispositiu ha estat enumerat correctament i se li ha assignat l'adreça única.

- D6: Indica "suspend-state", el dispositiu ha rebut senyal de l'ordinador d'entrar en mode de suspensió.

- D7: Indica "config-state", el dispositiu ha estat interrogat per l'ordinador sobre els seus paràmetres USB per conèixer el mode de funcionament i la informació ha estat enviada amb èxit.

També en referència als díodes indicadors, val a dir que s'ha limitat la intensitat amb resistències de 10K Ohm per evitar un consum elevat i poder complir així el requeriment dels 2.5 mA en estat inactiu. Si bé els díodes no ofereixen tota la seva brillantor, és suficient per detectar quin d'ells està encès.

S'ha disposat també d'un circuit de reset de l'integrat (i per tant del dispositiu sencer) que permet simular una desconnexió i reconnexió del dispositiu al bus USB sense necessitat d'haver-ho de fer de forma física.

Aquest fet ha estat útil també durant el procés de disseny i depuració del codi assemblador i, de manera general es pot pensar que és convenient tenir una opció de resetejat general d'un circuit electrònic.

El connector designat amb la referència CN3 consisteix en una tira de 6 pins rectangulars i està destinat a la connexió del programador de microcontroladors PIC, anomenat PICKit3.

Aquest programador es pot connectar en paral.lel amb el circuit de manera que permet reprogramar l'integrat sense necessitat de treure'l del sòcol. En qualsevol cas, com es pot observar al plànol, només cinc dels pins estan connectats.

El número 6 no és d'aplicació en l'integrat usat, tot i així s'ha disposat d'una tira de 6 pins per millorar la unió mecànica del programador i la placa.

El connector o sòcol destinat a allotjar la targeta intel.ligent no presenta només que una problemàtica. El contacte S1-S2 permet detectar la introducció de la targeta separant dues petites llengüetes de coure, és a dir, obrir un petit interruptor.

Com es pot veure, el contacte S1-S2 està connectat als pins 15 i 16 de l'integrat que permet generar interrupcions externes. Si bé aquestes entrades presenten bona immunitat als rebots, al tractar-se d'un contacte tan rudimentari, s'ha cregut convenient afegir una resistència i un condensador (R9 i C7) a mode de circuit anti-rebots.

Cal tenir present que el contacte del sòcol està normalment tancat, és a dir, sense targeta està tancat i quan s'introdueix la targeta aquesta obre el contacte. La Figura número 8 mostra l'estat del circuit anti-rebots amb i sense targeta intel.ligent introduïda en el sòcol.

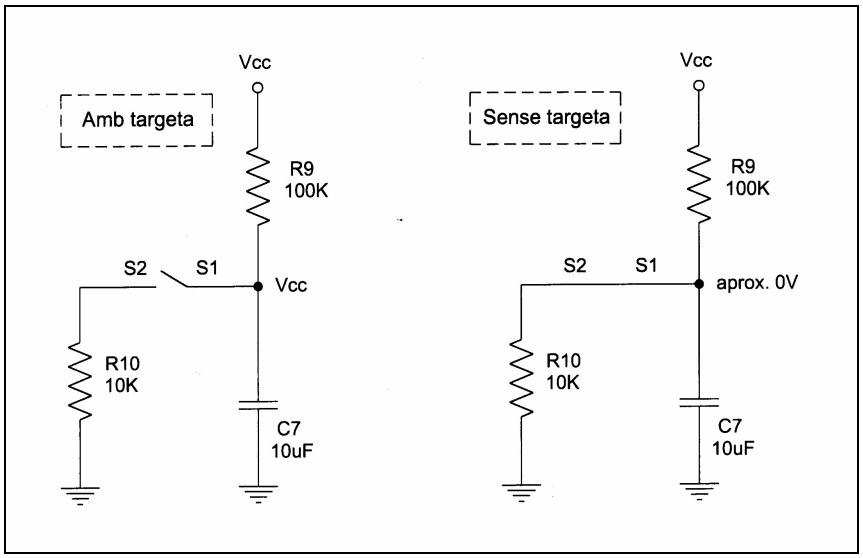


Figura número 8. Circuit anti-rebots.

Una última qüestió interessant té a veure amb la EUSART del PIC, concretament amb els pins 10 i 12 que corresponen al pin de transmissió i al de recepció respectivament. Com es pot observar, ambdós pins es conecten al pin de I/O de la targeta intel.ligent, avantposant una resistència de 4K7 Ohm pel pin TX.

Aquesta és una manera força utilitzada i pràctica de resoldre la problemàtica que genera el fet que la targeta tingui una sola línia destinada tant a entrada com a sortida de dades, mentre que la EUSART en disposa d'una per cada sentit de transmissió. Tal i com es pot observar, amb aquesta configuració, cada cop que transmetem una dada, no només la rebrà la targeta sinó també el nostre propi mòdul receptor.

En conseqüència, la rutina d'enviament de dades ha de tenir present aquesta situació i borrar del registre, de manera consecutiva, de recepció la dada un cop enviada.

3 FIRMWARE USB

Aquest capítol es destina a explicar de manera detallada el codi assemblador que controla el lector de targetes intel.ligents i el procés de comunicació amb l'ordinador a través del mòdul USB del microcontrolador. Resulta imprescindible explicar en primer lloc l'estructura general de tot el codi entenent el firmware com una sola unitat o programa. Evidentment la part de codi destinada al control del lector és secundari en relació a la part de codi que controla el mòdul USB però, no serà fins que no s'entengui tot el firmware, que no podrem entrar a explicar més concretament la part USB.

3.1 Estructura general del codi

La següent Figura número 9 mostra les diferents parts de l'arxiu de firmware de manera esquemàtica:

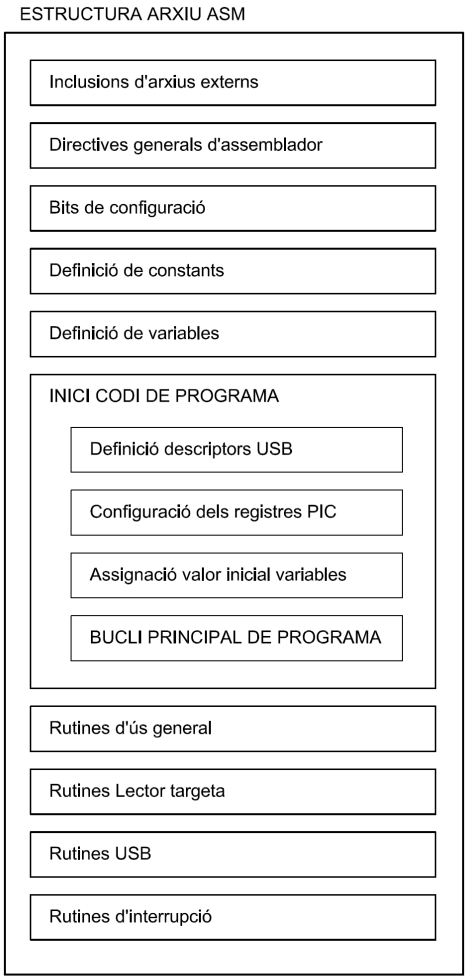


Figura número 9. Estructura arxiu firmware.

Aquestes parts són explicades a continuació només a grans trets i en els següents apartats i subapartats s'expliquen amb més detall.

En primer lloc trobem la inclusió de dos arxius externs en aquest arxiu principal com a mesura per disminuir la complexitat del codi i permetre una actualització i configuració més fàcil d'alguns aspectes que fan referència al funcionament de la comunicació USB i del microcontrolador PIC en general.

Després apareixen dues directives per indicar a l'assemblador utilitzat (MPASM) el model de microcontrolador usat i la base numèrica usada per defecte.

A continuació es defineix el valor de 31 bits de configuració per definir-ne aspectes de funcionament general com per exemple: qüestions relatives al clock de sistema, qüestions de configuració del mòdul USB, protecció contra lectura o escriptura, etc.

Seguidament definim 3 constant relatives al funcionament USB.

Després definim les variables que utilitzarem en el firmware assignant un nom a les diferents posicions de RAM que ocuparan. En total s'utilitzen 66 posicions de RAM.

Finalment comença el codi de programa pròpiament dit. Després de definir les posicions de memòria pels vectors inicial i d'interrupció, començament per definir els descriptors USB. Aquests descriptors seran llegits pel mateix firmware quan sigui necessari enviar-los a l'ordinador i són els que defineixen el nostre dispositiu USB.

Després es dona valor als diferents bits dels registres generals del nostre microcontrolador, ja sigui els que fan referència al mòdul USB com al clock del sistema, timers...

Posteriorment donem valor inicial a aquelles variables que ho requereixin i entrem al bucle principal de programa. Tot i que s'explicarà aquest bucle principal amb més detall en apartats posteriors, bàsicament es dedica a servir les peticions que rep del bus USB de manera continuada. Algunes d'aquestes peticions són estàndard i altres són específiques del nostre software.

Cada cop que arriba una petició de l'ordinador, el processador surt del bucle principal per servir aquesta petició i, per fer-ho, s'invoquen una sèrie de rutines i subrutines que s'inclouen a continuació. Aquestes rutines s'estructuren en 4 grans grups. Un primer grup de rutines generals, un segon amb rutines relatives a la comunicació de lector amb la targeta, un tercer de comunicació USB per donar resposta a les peticions USB i per últim les rutines d'interrupció.

3.2 Parts del codi

Aquest apartat inclou una explicació més detallada de cadascuna de les parts del codi que anteriorment s'han anomenat per definir l'estructura general de l'arxiu de codi assemblador. Val a dir però, que existeixen apartats destinats a veure de manera més concreta tot el codi que apareixi al firmware que faci referència al mòdul USB i el que faci referència al lector de targetes, per intentar aclarir millor el funcionament. Aquests apartats són 3.3 Secció USB i 3.4 Secció lector

En l'Annex A es disposa d'una còpia complerta de l'arxiu de firmware amb els corresponents comentaris per cada línia, per ajudar a la seva comprensió, les quals s'han eliminat en els següents subapartats per motius d'espai i per mantenir el format del document.

3.2.1 Inclusió d'arxius externs

Com s'ha comentat en l'apartat anterior, s'inclouen a l'inici del firmware dos arxius externs. A continuació s'inclouen les dues línies de codi assemblador amb les que es fan aquestes inclusions:

```
;;;;;;;;;;;;;;;;;;;;;;;;
;PFC Xavier Gelada ETIEI;
;;;;;;;;;;;;;;;;;;;;;;;;

#include <p18f14k50.inc>
#include "usb_defs.inc"
```

El primer dels arxius el subministra l'empresa fabricant del microcontrolador i permet fer referència als diferents registres i als seus bits rellevants amb etiquetes predefinides, sense necessitat d'haver de recordar l'adreça de memòria on es situa cadascun d'ells ni el número de bit en cada cas.

El segon dels arxius és un arxiu amb definicions de constants necessàries per la comunicació USB. Per simplificar l'arxiu de firmware s'ha decidit disposar-lo de manera separada per claredat i per permetre modificar paràmetres amb facilitat.

3.2.2 Directives generals assemblador

A continuació s'inclouen les dues línies de codi que inclouen dues directives generals que informen al software compilador (MPASM) del model d'integrat destí del programa i de la base numèrica amb la que es treballa per defecte:

```
LIST P=PIC18F14K50
RADIX HEX
```

Estrictament, ambdues directives són prescindibles ja que, per un costat tenim prèviament la inclusió de l'arxiu del microcontrolador, i per un altre, la base hexadecimal ja és la considerada per defecte per l'assemblador en cas de no definir-ne cap. En qualsevol cas, per compatibilitat amb altres possibles assembladors i per claredat general, s'han afegit aquestes dues línies.

3.2.3 Bits de configuració

A continuació es mostren tots els bits de configuració que s'han definit i els valors que se'ls hi dona:

```
CONFIG CPUDIV = NOCLKDIV
CONFIG USBDIV = OFF
CONFIG FOSC   = HS
CONFIG PLLEN  = ON
CONFIG PCLKEN = ON
CONFIG FCMEN  = OFF
CONFIG IESO   = OFF
CONFIG PWRTEN = OFF
CONFIG BOREN  = OFF
CONFIG BORV   = 30
CONFIG WDTEN  = OFF
CONFIG WDTPS  = 32768
CONFIG HFOFST = OFF
CONFIG MCLRE  = ON
CONFIG STVREN = ON
CONFIG LVP    = OFF
CONFIG BBSIZ  = OFF
CONFIG XINST  = OFF
```

```
CONFIG DEBUG  = OFF
CONFIG CP0    = OFF
CONFIG CP1    = OFF
CONFIG CPB    = OFF
CONFIG CPD    = OFF
CONFIG WRT0   = OFF
CONFIG WRT1   = OFF
CONFIG WRTB   = OFF
CONFIG WRTC   = OFF
CONFIG WRTD   = OFF
CONFIG EBTR0  = OFF
CONFIG EBTR1  = OFF
CONFIG EBTRB  = OFF
```

No es pot incloure en aquest subapartat una explicació detallada de cadascun d'aquests bits ja que són qüestions que queden fora de l'abast d'aquest document, com ja s'ha explicat al capítol introductori. Al capítol 10 Bibliografia es recull un enllaç al datasheet del microcontrolador per més informació.

3.2.4 Definició de constants

Es defineixen les següents constants, segons es pot veure a continuació:

```
#define     SHOW_ENUM_STATUS

#define     SOL_SC_IN_OUT     0x01
#define     SOL_ATR_A_PC      0x02
```

La primera constant ens permetrà invalidar l'ús dels LEDs indicadors de la placa en cas de borrar aquesta línia. Com s'ha explicat anteriorment, l'ús dels díodes senyalitzadors és especialment útil durant el desenvolupament i depuració del hardware i firmware però un cop s'ha finalitzat el projecte, són components prescindibles.

Les altres dues constants ens permeten definir peticions específiques que enviarà el nostre software a través del port USB. Aquest tipus de peticions són anomenades pel protocol USB "vendor-specific requests" i són les definides pel dissenyador. Passen a sumar-se a les pròpies del protocol USB que s'anomenen "standard requests".

La primera constant amb valor 0x01 correspon a la petició enviada pel software perquè el lector informi sobre si hi ha targeta present o no. La segona constant (amb valor 0x02) indicarà al lector que el software està sol.licitant que es llegeixi la informació de la targeta intel.ligent.

3.2.5 Definició de variables

La següent part de l'arxiu de firmware està destinada a definir les variables de RAM que utilitzem amb diferents finalitats. El codi és el que segueix:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Definició variables RAM a utilitzar (sempre bank0);
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

CONTROL_DELAY          equu   0x00
SOLLICITUD_ATR         equ   0x01
ESTAT_TARGETA          equ   0x02
SC_DINS                equ   0x03
NOMBRE_BYTES_ATR       equ   0x05
TIME_OUT               equ   0x2C

cblock 0x06
BYTES_ATR:33
endc

cblock 0x2D
USB_buffer_desc:4
USB_buffer_data:8
USB_error_flags
USB_curr_config
USB_device_status
USB_dev_req
USB_address_pending
USB_desc_ptr
USB_bytes_left
USB_loop_index
USB_packet_length
USB_USTAT
USB_USWSTAT
Registre_prov
Registre_prov2
Compta_Send_Descriptor
Comptador_Generic
endc
```

Tot i que a l'arxiu de firmware hi ha comentaris suficients per comprendre cadascuna d'aquestes variables, les anomenarem breument a continuació:

- CONTROL_DELAY: variable per generar retards amb el TIMER0.
- SOLLICITUD_ATR: variable per indicar si hi ha petició o no d'informació de la targeta.
- ESTAT_TARGETA: variable per indicar si la targeta està activa o no.
- SC_DINS: variable per indicar si hi ha targeta introduïda al lector o no.

- NOMBRE_BYTES_ATR: variable per emmagatzemar el nombre de bytes que ha respòs la targeta.
- TIME_OUT: variabler per indicar si s'ha sobrepassat el temps d'espera de resposta de la targeta.

A continuació tenim dos blocs de variables. El primer (BYTES_ATR) està destinat a contenir la resposta de la targeta intel.ligent, que pot tenir una longitud màxima de 33 bytes.

El segon bloc de variables fa referència a la comunicació USB i s'aniran explicant en el corresponent apartat (3.3 Secció USB).

3.2.6 INICI CODI DE PROGRAMA

En aquesta part de l'arxiu de firmware, mostrada a continuació, comença el que pròpiament és codi de programa, ja siguin instruccions o dades. En primer lloc omplim els vectors claus en qualsevol memòria de programa com són el vector 0000h (inici d'execució de programa) i els d'interrupcions. En aquest cas existeixen dos vectors perquè el microcontrolador utilitzat permet classificar les interrupcions en dos graus de prioritats.

A continuació s'inclouen els descriptors USB, que són els que definiran el nostre dispositiu i que durant el procés de configuració des de l'ordinador personal seran sol.licitats. Es tracta de dades de diferent tipologia que més endavant sera explicada amb detall però que s'inclou aquí per coherència amb el seguiment del codi i perquè el lector en tingui una primera presa de contacte. Finalment tenim la etiqueta "Inici_programa" que és a on es fa salt incondicional des del vector zero.

```
;;;;;;;;;;;;;;;;;;;
;Inici de programa;
;;;;;;;;;;;;;;;;;;;


ORG H'0000'
goto Inici_programa

ORG H'0008'
goto Hard_int

ORG H'0018'
goto Soft_int

ORG H'001A'
Descriptor_begin
```

```
ORG H'001C'
Device

      db    0x12, DEVICE
      db    0x10, 0x01
      db    0x00, 0x00
      db    0x00, MAX_PACKET_SIZE
      db    0xD8, 0x04
      db    0x01, 0xA0
      db    0x01, 0x00
      db    0x01, 0x02
      db    0x00, NUM_CONFIGURATIONS

Configuration1

      db    0x09, CONFIGURATION
      db    0x12, 0x00
      db    NUM_INTERFACES, 0x01
      db    0x00, 0x80
      db    0x32, 0x09
      db    INTERFACE, 0x00
      db    0x00, 0x00
      db    0xFF, 0x00
      db    0xFF, 0x00

String0

      db    String1-String0, STRING
      db    0x09, 0x04

String1

      db    String2-String1, STRING
      db    'M', 0x00
      db    'i', 0x00
      db    'c', 0x00
      db    'r', 0x00
      db    'o', 0x00
      db    'c', 0x00
      db    'h', 0x00
      db    'i', 0x00
      db    'p', 0x00
      db    ' ', 0x00
      db    'T', 0x00
      db    'e', 0x00
      db    'c', 0x00
      db    'h', 0x00
      db    'n', 0x00
      db    'o', 0x00
      db    'l', 0x00
      db    'o', 0x00
      db    'g', 0x00
      db    'y', 0x00
      db    ',', 0x00
      db    ' ', 0x00
      db    'I', 0x00
      db    'n', 0x00
      db    'c', 0x00
      db    '.', 0x00
```

```
String2

      db    Descriptor_end-String2, STRING
      db    'P', 0x00
      db    'F', 0x00
      db    'C', 0x00
      db    ' ', 0x00
      db    'E', 0x00
      db    'T', 0x00
      db    'I', 0x00
      db    'E', 0x00
      db    'I', 0x00
      db    ' ', 0x00
      db    'X', 0x00
      db    'G', 0x00
      db    'e', 0x00
      db    'l', 0x00
      db    'a', 0x00
      db    'd', 0x00
      db    'a', 0x00
      db    ' ', 0x00
      db    'U', 0x00
      db    'S', 0x00
      db    'B', 0x00
      db    ' ', 0x00
      db    'F', 0x00
      db    'i', 0x00
      db    'r', 0x00
      db    'm', 0x00
      db    'w', 0x00
      db    'a', 0x00
      db    'r', 0x00
      db    'e', 0x00

Descriptor_end

Inici_programa
```

3.2.7 Configuració dels registres PIC

A continuació es configuren els anomenats registres SFR del microcontrolador, a través dels quals es pot condicionar el seu funcionament i el dels seus perifèrics. Sobrepassa l'abast d'aquest projecte explicar un per un cadascun d'aquests registres i el significat de cadascun dels seus bits. En qualsevol cas en l'arxiu original del firmware hi ha els corresponents comentaris per cadascuna de les línies que permeten entendre les diferents assignacions de valors pels diferents registres. El contingut complert d'aquest arxiu amb els comentaris pot ésser consultat a l'annex A. També es pot trobar més informació en el datasheet, que es pot obtenir des de la referència continguda al capítol 10 Bibliografia. El codi que segueix mostra la configuració dels registres SFR:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Selecció del bank de RAM de treball (bank 0);
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

movlb 0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuracions relatives al CLOCK del sistema;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

bsf OSCTUNE, SPLLEN
bcf OSCCON, SCS0
bcf OSCCON, SCS1

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuracions relatives a perifèrics no utilitzats;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

clrf ANSEL
clrf ANSELH
clrf SLRCON
clrf CM1CON0
clrf CM2CON0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuració de RC0 com a sortida per alimentar la SmartCard;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

bcf TRISC, TRISC7
bcf LATC, LATC7

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuració de RC6 com a sortida per resetejar la SmartCard;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

bcf TRISC, TRISC6
bcf LATC, LATC6

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuracions del PWM/TMR2 per generar el clock de la SmartCard;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

clrf TMR2
clrf T2CON

movlw 3
movwf PR2

movlw 2
movwf CCPR1L

movlw 0x01
movwf PSTRCON

bcf TRISC, TRISC5

clrf CCP1CON

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuració del TIMER0 que s'utilitza per fer delays;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
movlw 0x07
movwf T0CON

clrf CONTROL_DELAY, BANKED

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuració relativa a la EUSART;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

movlw 0x05
movwf SPBRGH
movlw 0xCF
movwf SPBRG

bcf RCSTA, CREN
bcf RCSTA, SPEN
bcf TXSTA, SYNC
bcf LATB, LATB7
bsf TRISB, TRISB7
bsf TRISB, TRISB5
bsf TXSTA, TX9
bsf RCSTA, RX9
bsf TXSTA, BRGH
bsf BAUDCON, BRG16
bcf BAUDCON, DTRXP
bcf BAUDCON, CKTXP


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuració dels pins de l'integrat amb LEDs connectats;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

bcf TRISC, TRISC2
bcf LATC, LATC2

bcf TRISC, TRISC3
bcf LATC, LATC3

bcf TRISC, TRISC4
bcf LATC, LATC4

bcf TRISB, TRISB4
bcf LATB, LATB4

bcf TRISB, TRISB6
bcf LATB, LATB6


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuració relativa a les interrupcions;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

bcf INTCON3, INT1IF
bcf INTCON, INT0IF
bcf INTCON, TMR0IF

bsf RCON, IPEN
bcf INTCON, GIEH
bcf INTCON, GIEL
```

```
bcf INTCON, TMR0IE
bcf INTCON2, TMR0IP

bcf PIE1, RCIE
bcf IPR1, RCIP

bsf TRISC, TRISC0
bsf INTCON, INT0IE
bsf INTCON2, INTEDG0

bsf TRISC, TRISC1
bsf INTCON3, INT1IE
bsf INTCON3, INT1IP
bcf INTCON2, INTEDG1

clrf UIE
clrf UIR
```

3.2.8 Assignació valor inicial variables

En el següent apartat donem valors inicials a diferents de les variables definides per nosaltres. Com es mostra a continuació, en tots els casos es posen a zero:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;ASSIGNACIO VALOR INICIAL VARIABLES;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

clrf SOLLICITUD_ATR, BANKED
clrf ESTAT_TARGETA, BANKED
clrf SC_DINS, BANKED

clrf NOMBRE_BYTES_ATR, BANKED

clrf TIME_OUT, BANKED

clrf USB_error_flags, BANKED
```

3.2.9 BUCLE PRINCIPAL DE PROGRAMA

Arribem a una de les parts essencials d'un firmware, el bucle principal de programa. És la part de codi que s'inclou després dels següents paràgrafs.

Els passos que seguim són els següents. En primer lloc inicialitzem el mòdul USB del microcontrolador amb la rutina "Init_USB", fet que inclou activar-lo, resetejar els flags, esperar que el dispositiu es consideri connectat al bus, etc.

Un cop fet això, invoquem un primer cop la rutina "Service_USB", la qual ens ha de permetre configurar el nostre dispositiu com un més dins del bus USB de l'ordinador. Mentre no s'assoleixi aquest estat configurat, estarem en aquest bucle intentant-ho.

Un cop tenim el nostre dispositiu connectat i configurat, ja és un dispositiu més per l'ordinador. Activem les fonts d'interrupció i quedem en un segon bucle infinit, on anem invocant la mateixa rutina per anar servint les peticions USB que es puguin anar donant regularment.

Aquestes peticions poden ser pròpies del bus USB o bé pròpies de programes concrets, com el nostre, que sol.licitarà informació a través d'una petició pròpia a la que també ha de donar resposta la rutina "Service_USB".

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;BUCLE PRINCIPAL DE PROGRAMA;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Bucli_Primer_De_Programa

call Delay_2ms

call Init_USB

Configuracio_Periferic

call Service_USB

movlw CONFIG_STATE
cpfseq USB_USWSTAT, BANKED
goto Configuracio_Periferic

bsf INTCON, GIEH
bsf INTCON, GIEL

Bucli_Segon_De_Programa

call Service_USB

goto Bucli_Segon_De_Programa
```

3.2.10 Rutines d'ús general

La part on hi ha les rutines generals disposa només d'una rutina anomenada "Configuració Inicial" que és invocada cada cop que s'extreu una targeta (provocant una interrupció) de manera que s'inicialitzen variables i registres, deixant-ho tot apunt per quan es produeixi una nova inserció. El codi que s'inclou a continuació mostra l'estructura general de la rutina tot i

que s'ha eliminat la part de codi per evitar redundàncies, ja que es tracta de variables i registres que ja han aparegut en apartats anteriors. El contingut complert de la rutina es pot trobar si es consulta el llistat complert de l'arxiu de firmware a l'Annex A.

```
;;;;;;;;;;;;;;;;;;;;;;;;;
;ZONA DE RUTINES GENERALS;
;;;;;;;;;;;;;;;;;;;;;;;;;

;---------------------------------------

Configuracio_Inicial

(...)

      return

;---------------------------------------
```

3.2.11 Rutines lector targeta

La següent part de l'arxiu inclou les rutines que permeten fer funcionar el lector perquè pugui interactuar amb les targetes inserides. Existeix ja un apartat destinat a explicar amb més detall aquestes rutines (3.4 Secció Lector) i per aquest motiu no es llistarà aquí cap d'aquestes rutines.

3.2.12 Rutines USB

A continuació s'inclou la part més llarga de tot l'arxiu de codi firmware. Consisteix en el conjunt de rutines i subrutines que permeten materialitzar la comunicació USB amb l'ordinador, i interactuar amb el software. Les principals rutines que hi trobem són "Init USB" i "Service USB", utilitzades en el bucle principal de programa.

Com en el cas del subapartat anterior, aquest conjunt de rutines serà tractat amb més detall en l'apartat 3.3 Secció USB i per això no es detalla aquí el contingut d'aquestes rutines.

3.2.13 Rutines d'interrupció

Finalment, abans de trobar el final de l'arxiu assemblador, hi ha l'apartat de rutines d'interrupció. Disposem de dues rutines principals, una per les interrupcions d'alta prioritat

(Hard_Int) i una per les interrupcions de baixa prioritat (Soft_Int). Com es pot comprovar, no hi ha interrupcions de prioritat baixa activades i s'ha disposat d'una rutina buida per motius de coherència i per fer el contingut del firmware més fàcilment comprensible.

La rutina d'interrupcions d'alta prioritat invoca una subrutina anomenada "Interrupció" que comprova si s'ha donat una interrupció externa per inserció de la targeta o per retirada de la mateixa. En cas que s'hagi introduït una targeta, modifiquem la variable indicadora. En cas que s'hagi extret, invoquem la rutina general "Configuracio_Inicial" por inicialitzar tot allò que sigui necessari per tornar a tenir el lector apunt de rebre una nova targeta.

Aquest contingut s'inclou a continuació:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;ZONA DE RUTINES INTERRUPCIÓ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;----------------------------------------

Hard_int

call Interrupcio, 1

retfie

;----------------------------------------

;----------------------------------------

Soft_int

     retfie

;----------------------------------------

;----------------------------------------

Interrupcio

movlb 0
btfss INTCON, INT0IF
goto Extraccio

bsf SC_DINS, 0, BANKED
clrf SOLLICITUD_ATR, BANKED
clrf NOMBRE_BYTES_ATR, BANKED

bcf INTCON, INT0IF

return 1
```

```
Extraccio

call Configuracio_Inicial

return 1

;---------------------------------------

END
```

3.3 Secció USB

Aquest apartat intenta aglutinar totes aquelles parts de codi dins l'arxiu de firmware que fan referència a la comunicació USB per explicar-les amb més detall.

Les primeres línies de codi que apliquen sobre el mòdul USB del nostre microcontrolador són les que s'inclouen a continuació i es situen a la part dels bits de configuració i a la definició de constants:

```
CONFIG USBDIV = OFF
CONFIG FOSC   = HS
CONFIG PLLEN  = ON

#define    SHOW_ENUM_STATUS
#define    SOL_SC_IN_OUT    0x01
#define    SOL_ATR_A_PC     0x02
```

El bit de configuració "USBDIV" aplicaria únicament en el cas de treballar en "Low Speed USB" (veure l'esquema que apareix a la Figura número 3 a la pàgina 12), situació que no és la de la nostra aplicació, i per tant, en aquest cas, no té cap efecte el valor que se li doni.

Els altres dos bits si que tenen efecte sobre la freqüència de treball del mòdul USB. Amb el primer (FOSC) definim el tipus d'oscil.lador extern que connectarem al microcontrolador i amb el segon (PLLEN), treballant conjuntament amb el bit SPLLEN (que també està activat), la freqüència de treball del mòdul serà quatriplicada i en conseqüència treballarà a 48MHz ja que el cristall de quars extern és de 12 MHz. Aquesta situació també queda clara si s'observa la Figura número 3 anteriorment citada.

A continuació es defineixen 3 constants que tindran efecte en la compilació del programa. En el primer cas, en funció de si existeix aquesta línia o no, s'utilitzaran els díodes LED

presents a la placa de circuit imprès per indicar l'estat USB del dispositiu. Si l'esborrem, el compilador no compilarà el codi que activa i desactiva aquests díodes.
Les altres dues constants són les dues possibles peticions que el nostre software pot fer arribar al dispositiu. Per facilitat, al llarg del codi utilitzarem les etiquetes "SOL_SC_IN_OUT" i "SOL_ATR_A_PC". Es tracta de dues peticions pròpies, anomenades per les especificacions USB "Vendor-requests".

El següent apartat on trobem codi que apliqui al mòdul USB és en la definició de variables pròpies de RAM. Les variables que definim queden recollides a continuació:

```
USB_buffer_desc:4
USB_buffer_data:8
USB_error_flags
USB_curr_config
USB_device_status
USB_dev_req
USB_address_pending
USB_desc_ptr
USB_bytes_left
USB_loop_index
USB_packet_length
USB_USTAT
USB_USWSTAT
Registre_prov
Registre_prov2
Compta_Send_Descriptor
Comptador_Generic
```

A continuació s'explica breument la finalitat de cadascuna d'aquestes variables, tot i que per una correcta comprensió d'alguna d'elles és necessari entendre els conceptes bàsics del protocol USB:

- USB_buffer_desc: variable RAM que ocupa 4 posicions de RAM on es guarden els 4 bytes de la "Buffer Descriptor Table" de l'endpoint que fem anar en cada moment (en el nostre cas únicament el número 0). Per més informació es pot consultar el subapartat "Buffers descriptors i BDT (Buffer Descriptor Table)" a la pàgina X.
- USB_buffer_data: variable que ocupa vuit bytes de la memòria RAM i que es destina a emmagatzemar els vuit bytes que s'envien com a primer paquet de dades en una transferència de control (etapa SETUP).
- USB_error_flags: byte que utilitzem per senyalitzar si s'ha donat o no un error durant l'execució de les rutines i subrutines USB.

- USB_curr_config: variable per emmagatzemar la configuració actual de totes les possibles que tingui el nostre dispositiu.
- USB_device_status: variable on emmagatzemem dos bits indicadors del tipus d'alimentació del nostre dispositiu (externa o bus) i si admet la característica "remote-wakeup").
- USB_dev_req: variable que emmagatzema la petició estàndard en curs que s'està servint (en cas d'haver-n'hi).
- USB_address_pending: variable on es guarda inicialment l'adreça assignada al nostre dispositiu dins el bus USB, fins que s'acaba la transferència de recepció d'aquesta adreça, moment en el que es pot carregar al registre UADDR.
- USB_desc_ptr: variable que utilitzem per emmagatzemar un punter per resseguir els descriptors USB. S'utilitza a mode d'offset des de l'inici dels descriptors.
- USB_bytes_left: variable que ens permet controlar el nombre de bytes que ens falten per enviar d'un descriptor determinat per actuar d'una manera o d'una altra segons si estem per sota del nombre màxim de bytes per packet.
- USB_loop_index: variable prevista inicialment per una tasca que finalment no ha estat necessària.
- USB_packet_length: variable per emmagatzemar la longitud de les dades a enviar en cas de packet curt.
- USB_USTAT: variable on emmagatzemem el valor del registre USTAT
- USB_USWSTAT: variable que ens permet emmagatzemar l'estat USB en el que es troba el nostre dispositiu.
- Registre_prov: variable utilitzada per guardar valors temporalment.
- Registre_prov2: variable utilitzada per guardar valors temporalment.
- Compta_Send_Descriptor: variable utilitzada com un comptador dels bytes a enviar en el cas de descriptors USB.
- Comptador_Generic: variable usada com a comptador per vàries tasques dins les rutines USB.

Seguidament, en el nostre arxiu de firmware i sempre parlant del codi que fa referència a la comunicació USB, hi trobem els descriptors USB del nostre dispositiu. Tot dispositiu USB serà interrogat en el moment de la connexió a l'ordinador per conèixer les característiques i configuracions possibles. L'ordinador sabrà així com ha d'interactuar amb el dispositiu. Aquesta informació està continguda en els descriptors que es mostren a continuació:

```
Descriptor_begin
```

```
Device

db    0x12, DEVICE
db    0x10, 0x01
db    0x00, 0x00
db    0x00, MAX_PACKET_SIZE
db    0xD8, 0x04
db    0x01, 0xA0
db    0x01, 0x00
db    0x01, 0x02
db    0x00, NUM_CONFIGURATIONS

Configuration1

db    0x09, CONFIGURATION
db    0x12, 0x00
db    NUM_INTERFACES, 0x01
db    0x00, 0x80
db    0x32, 0x09
db    INTERFACE, 0x00
db    0x00, 0x00
db    0xFF, 0x00
db    0xFF, 0x00

String0

db    String1-String0, STRING
db    0x09, 0x04

String1

db    String2-String1, STRING
db    'M', 0x00

(...)

db    '.', 0x00

String2

db    Descriptor_end-String2, STRING
db    'P', 0x00

(...)

db    'e', 0x00

Descriptor_end
```

El descriptor del nostre dispositiu es divideix en diversos apartats seguint un esquema imposat per les especificacions USB. El valor d'alguns bytes s'ha substituït pel nom de constants definides a l'arxiu "usb_defs.inc". La primera secció s'anomena "device" i aporta informació general sobre el nostre dispositiu. Bàsicament indiquem el nombre de bytes

d'aquesta secció, la versió USB amb la que és compatible el nostre dispositiu i un conjunt d'identificadors de producte, fabricant i versió. Es defineix també en aquesta secció el màxim nombre de bytes que pot incloure cada packet de dades i el nombre de configuracions que suporta el nostre device.

A continuació hi ha la secció de configuració on s'especifica el nombre de bytes que conté i si el dispositiu és auto-alimentat o s'alimenta a través del bus. També s'indiquen el nombre d'interfícies que inclou aquesta configuració i les dades relatives a aquestes interfícies (en el nostre cas només hi ha una interfície definida).
En aquesta secció trobem el consum màxim que necessita la nostra aplicació i, per últim, el tipus de dispositiu del que es tracta (en el nostre cas "vendor-specific") i el protocol que es segueix.

Per últim hi ha tres seccions relatives a textos descriptius que apareixeran durant el procés de configuració. La primera secció ("String0") indica l'idioma utilitzat (a la pràctica només es reconeix l'anglès). La segona secció ("String1") defineix una sèrie de caràcters de text que conformen la cadena "Microchip Technology Inc." i finalment l'última secció inclou la cadena descriptiva "PFC ETIEI XGelada USB Firmware".

Si es vol trobar més informació sobre el format i possibilitats dels descriptors USB es poden consultar les especificacions USB, a les que es pot accedir a través de la referència al capítol 10 Bibliografia.

A continuació, en l'apartat de configuració de les interrupcions i d'assignació inicial de valor a les variables, trobem tres línies de codi que fan referència al mòdul USB. El codi és el que segueix:

```
clrf UIE
clrf UIR

clrf USB_error_flags, BANKED
```

La primera instrucció desactiva totes les interrupcions USB del microcontrolador i la segona posa a zero tots els flags senyalitzadors USB. Per últim posem a zero una variable que utilitzarem per detectar errors durant l'execució de les nostres rutines USB.

A continuació en el bucle principal de programa invoquem primer la rutina "Init_USB" que inicialitza el mòdul USB del perifèric i espera que es connecti el nostre dispositiu a un bus USB. S'inclou a continuació tot el contingut d'aquesta rutina que es comenta posteriorment:

```
Init_USB

clrf UIE
clrf UIR

movlw 0x14
movwf UCFG

movlw 0x08
movwf UCON

clrf USB_curr_config, BANKED
clrf USB_USWSTAT, BANKED

movlw 0x00
movwf USB_device_status, BANKED

movlw NO_REQUEST
movwf USB_dev_req, BANKED

Esperant_SEO

btfsc UCON,SE0
goto Esperant_SEO

return
```

Tot i que ja s'ha fet anteriorment, com a mesure de precaució es tornen a desactivar les interrupcions i es baixen els flags senyalitzadors pel que fa referència al mòdul USB. Llavors es modifiquen els registres UCON i UCFG de manera que configurem el mòdul per treballar en "Full-Speed" i amb les resistències de pull-up internes. Posteriorment activem el mòdul USB del microcontrolador. Mentre es posa en funcionament, posem a zero determinades variables de RAM que utilitzem durant el procés de configuració del dispositiu.

Un cop fet tot això ens quedem esperant un senyal de "Single-Ended-Zero" o també abreviat "SE0" que indicarà que s'ha connectat el nostre dispositiu a un bus USB. Quan això passi donarem la rutina per acabada i tornarem al bucle principal.

En el bucle principal, invoquem la rutina "Service_USB" un primer cop amb l'objectiu de configurar el dispositiu. Creem un bucle on es va comprovant un indicador que ens

senyalitzarà quan l'ordinador ja ha configurat el nostre dispositiu i podem començar a operar normalment. Fins que això no passi, quedem a l'espera en aquest bucle.

Un cop configurat, ens quedem dins d'un bucle infinit, on anem executant la rutina "Service_USB" de manera infinita. Només la introducció o extracció de la targeta en el lector generarà una interrupció prou important com per sortir temporalment d'aquest bucle per modificar diferents variables.

D'altra banda, aquesta rutina és capaç de servir qualsevol petició USB, ja sigui pròpia del bus ("Standard-request") o del nostre programa específic ("Vendor-request"). Per fer-ho, utilitza gran quantitat de subrutines però, en primer terme, el que fa "Service_USB" és anar comprovant si s'ha donat alguna interrupció USB (ara si que han estat activades des del bucle principal), es busca quina és i s'actua en conseqüència per servir la petició rebuda.

A continuació s'inclou una llista de bits que es van comprovant repetidament per detectar peticions del bus USB:

- Bit UERRIF del registre UIR: bit senyalitzador d'interrupció per error USB.
- Bit SOFIF del registre UIR: bit senyalitzador d'interrupció per "Start of frame".
- Bit ACTVIF del registre UIR: bit senyalitzador d'interrupció al detectar activitat al bus.
- Bit STALLIF del registre UIR: bit senyalitzador d'interrupció per STALL.
- Bit URSTIF del registre UIR: bit senyalitzador de RESET.
- Bit TRNIF del registre UIR: bit senyalitzador de transacció completada.

Per una informació més detallada del que es fa al detectar cadascun d'aquests bits, es poden consultar els comentaris de l'arxiu de firwmare a la rutina "Service USB".

A continuació s'inclou la rutina "Descriptor" que de fet és utilitzada en la majoria dels casos per una rutina més genèrica anomenada "SendDescriptorPacket".

La primera rutina llegeix una determinada posició del descriptor USB a la que hem d'apuntar amb un punter i un offset. La segona rutina envia les dades recuperades amb "Descriptor" com un packet de dades.

Aquestes dues rutines són utilitzades bàsicament per donar resposta a diferents "Standard Request" que tenen a veure amb el procés de configuració del dispositiu.

Les rutines que segueixen són "ProcessSetupToken", "ProcessInToken" i "ProcessOutToken". Són tres rutines invocades quan s'ha detectat que una transacció ha estat completada amb la rutina "Service USB".
Aquesta mateixa rutina mira si es tracta d'una transacció tipus "OUT", "IN" o "SETUP" i invoca la rutina adequada en cada cas.

Seguidament trobem la rutina "StandardRequests" que permet servir totes les peticions d'aquest tipus.

Tenim previst donar resposta a 11 de les possibles peticions d'aquest tipus segons les especificacions USB, de manera que si es rep alguna altra petició senzillament es descarta (es tractarà de peticions molt marginals i específiques que cap dispositiu ha de saber processar de manera obligatòria). Aquestes 11 peticions són:

- GET_STATUS: per conèixer l'estat USB del nostre dispositiu.
- CLEAR_FEATURE: per modificar característiques del nostre dispositiu (no permès).
- SET_FEATURE: per assignar valor a característiques del nostre dispositiu.
- SET_ADDRESS: per assignar adreça única al dispositiu dins del bus USB.
- GET_DESCRIPTOR: per sol.licitar descriptors del nostre dispositiu.
- GET_CONFIGURATION: per sol.licitar l'actual configuració del nostre dispositiu.
- SET_CONFIGURATION: per assignar la configuració al nostre dispositiu.
- GET_INTERFACE: per sol.licitar l'actual interfície del nostre dispositiu.
- SET_INTERFACE: per assignar la interfície al nostre dispositiu.
- SET_DESCRIPTOR: per modificar els descriptors (no permès).
- SYNCH_FRAME: per senyalitzar un frame de sincronització (no utilitzat).

Per donar resposta a totes aquestes peticions s'utilitza un gran conjunt de rutines i subrutines amb fins a tres nivells de funcionament.

La següent Taula número 11 mostra les principals subrutines que pengen de manera directa de "StandardRequests" per donar resposta a les anteriors peticions. Entre parèntesi hi ha el nom de la rutina que es pot trobar a l'arxiu de firmware:

Standard USB Requests (StandardRequests)	GET_STATUS (Rutina_REQ_GET_STATUS)
	CLEAR_FEATURE (Rutina_REQ_CLEAR_FEATURE)
	SET_FEATURE (Rutina_REQ_SET_FEATURE)
	SET_ADDRESS (Rutina_REQ_SET_ADDRESS)
	GET_DESCRIPTOR (Rutina_REQ_GET_DESCRIPTOR)
	GET_CONFIGURATION (Rutina_REQ_GET_CONFIGURATION)
	SET_CONFIGURATION (Rutina_REQ_SET_CONFIGURATION)
	GET_INTERFACE (Rutina_REQ_GET_INTERFACE)
	SET_INTERFACE (Rutina_REQ_SET_INTERFACE)
	SET_DESCRIPTOR (Rutina_REQ_SET_DESCRIPTOR)
	SYNCH_FRAME (Rutina_REQ_SYNCH_FRAME)
	<DEFAULT>

Taula número 11

La següent Taula número 12 mostra un segon nivell de subrutines que presenten algunes de les rutines que s'inclouen a la taula anterior:

Rutina_REQ_GET_STATUS	Rutina_GS_RECIPIENT_DEVICE
	Rutina_GS_RECIPIENT_INTERFACE
	Rutina_GS_RECIPIENT_ENDPOINT
Rutina_REQ_SET_ADDRESS	Rutina_SF_RECIPIENT_DEVICE
	Rutina_SF_RECIPIENT_ENDPOINT
Rutina_REQ_GET_DESCRIPTOR	Rutina_GD_DEVICE
	Rutina_GD_CONFIGURATION
	Rutina_GD_STRING

Taula número 12

Per últim, definim la rutina "VendorRequests" que permet donar resposta a les peticions pròpies de la nostra aplicació, les quals poden ser dues "SOL_SC_IN_OUT" i "SOL_ATR_A_PC". Aquesta rutina està relacionada amb les rutines que treballen directament amb el lector i amb la targeta intel.ligent. Bàsicament el que es fa és discriminar quina de les dues possibles peticions és i retornar al resposta corresponent. En el primer cas, des del software sens està demanant que responguem sobre el si hi ha targeta present o no. En el segons se'ns demana que interroguem la targeta i retornem aquesta resposta. Des de la rutina s'invoquen les altres rutines que fan referència a l'ús del lector (que descrivim en el següent apartat 3.4) i es retorna la informació obtinguda.

3.4 Secció lector

Existeix en l'arxiu de firmware una secció destinada a agrupar les rutines que controlen el lector de la targeta intel.ligent. No s'explicarà en detall el seu contingut perquè no és l'objectiu principal d'aquest projecte el disseny d'un lector de targetes intel.ligents, sinó només la part que fa referència a la comunicació del microcontrolador a través del port USB.

Es pot obtenir més informació a través dels comentaris que acompanyen la majoria de les línies de codi d'aquesta secció, en l'arxiu de firmware, inclòs a l'annex A.

4 DRIVER USB

4.1 Necessitat d'un driver

La interfície de comunicació USB funciona a alt nivell i això implica que està condicionada pel sistema operatiu. En conseqüència, per poder-nos comunicar amb el nostre dispositiu, es requereix d'un driver instal.lat que expliqui al sistema operatiu com treballar-hi.

En realitat, el sistema operatiu "Windows", utilitza les anomenades APIs per comunicar-se amb molts dispositius, i el nostre software podria, a través de l'ús d'aquestes d'aquestes funcions especials, treballar amb el port USB, però això implica tenir un coneixement molt profund de l'arquitectura del sistema operatiu, així com uns coneixements de programació informàtica també elevats.

En comptes d'això, sovint s'utilitzen compenents intermitjos, com llibreries dinàmiques (les anomenades "dll") que ens ofereix funcions invocables des del nostre software les quals, al seu temps, treballen amb les citades APIs. Una altra opció és utilitzar l'anomenat "entorn de programació .NET", que ofereix Windows, el qual incorpora també funcions i rutines que permeten fer de pont entre el nostre software i les APIs.

Existeixen casos en els que no es necessita l'ús d'un driver perquè el mateix sistema operatiu el porta incorporat de sèrie. Això succeeix quan el nostre dispositiu s'ajusta a alguna de les categories establertes per les especificacions USB. Per exemple, existeix la categoria HID ("Human Interface Device") en la que clarament s'hi poden encabir dispositius com teclats o ratolins. Si ens adaptem a les premisses de treball que marquen les especificacions USB, podrem construir un dispositiu que no requereixi de driver, si bé és cert que, haver-nos d'adaptar a les directrius que s'estableixen, ens pot limitar el nostre disseny.

Per aquest motiu, tot i que estrictament hagués estat possible ajustar-se a algun dels tipus prèviament contemplats per, d'aquesta manera no necessitar driver específic, s'ha preferit escollir l'ús d'un driver extern, per la llibertat de disseny que permet.

En el nostre cas, hem utilitzat un driver anomenat "WinUSB" que no deixa de ser una llibreria dinàmica i que a continuació s'explica breument.

4.2 Driver WinUSB

Es tracta d'un driver gratuït de lliure distribució i dissenyat per "Microsoft Corporation", és a dir, la mateixa empresa del sistema operatiu "Windows", on funcionarà el nostre software.

Presenta algunes limitacions que no afecte, ni de bon tros, a la majoria de projectes USB que es puguin desenvolupar a nivell acadèmic o privat. Aquestes limitacions són les següents:

- No es permet que vàries aplicacions alhora treballin amb el mateix dispositiu USB.
- No permet treballar amb tipus de transferències "isochronous".
- Només funciona amb versions de Windows posteriors a la XP

El driver es distribueix com un pack d'arxius d'extensió "dll" i "inf", podent presentar noms diferents segons la versió utilitzada. En el nostre cas disposaríem de 3 arxius:

- WdfCoInstaller01005.dll
- WinUSBCoInstaller.dll
- Gelada.inf

L'últim arxiu és un arxiu de text ASCII que caldrà modificar segons les nostres necessitats i que ajudarà al sistema operatiu a saber com ha d'instal.lar el driver quan sigui el moment. El contingut d'aquest arxiu és mostrat i explicat en l'apartat 4.4.

El procés d'instal.lació d'aquest driver no té cap diferència amb la instal.lació d'altres drivers per dispositius externs al nostre ordinador. Un cop connectem el dispositiu, el sistema operatiu detecta un nou dispositiu desconegut i obre una finestra de diàleg per començar el seu procés de configuració amb la instal.lació del driver adequat. El procés d'instal.lació del driver WinUSB s'explica a l'apartat 4.5.

4.3 Relació driver-dispositiu

Com ja s'ha comentat, quan el sistema operatiu detecta un nou dispositiu en el bus USB, ha de trobar el driver que li correspon. Algunes vegades aquest driver ja està instal.lat (si es

tracta d'un dispositiu estàndard o bé si ja s'ha instal.lat el driver prèviament) i en d'altres ocasions s'ha d'instal.lar amb l'ajuda d'un arxiu INF.

Per conèixer el driver assignat a un dispositiu USB podem anar a l'administrador de dispositius del sistema operatiu.

Un cop instal.lat un nou driver, és recordat per d'altres vegades que es connecti el dispositiu associat, el qual ha de tenir un número identificador únic. La informació corresponent al dispositiu i al seu driver associat es guarda al registre del sistema operatiu.

4.4 Arxius INF

Són arxius de text que contenen informació sobre el dispositiu i el seu driver associat, així com també informació pel registre del sistema operatiu. Per entendre l'estructura d'aquests arxius es mostra a continuació, el contingut del nostre arxiu de text, anomenat "gelada.inf":

```
[Version]
Signature = "$Windows NT$"
Class = Gelada
ClassGuid={d7231438-6472-43f9-9f61-4522d6b00ad7}
Provider = %ProviderName%
DriverVer=10/13/2007,6.00.2064

; ========== Manufacturer/Models sections ===========

[Manufacturer]
%ProviderName% = XGelada,NTx86,NTamd64

[XGelada.NTx86]
%USB\MyDevice.DeviceDesc% =USB_Install, USB\VID_04D8&PID_A001

[XGelada.NTamd64]
%USB\MyDevice.DeviceDesc% =USB_Install, USB\VID_04D8&PID_A001

; ================== Installation ==================

[ClassInstall32]
AddReg=SampleClass_RegistryAdd

[SampleClass_RegistryAdd]
HKR,,,,%ClassName%


;[1]
[USB_Install]
Include=winusb.inf
```

```
Needs=WINUSB.NT

;[2]
[USB_Install.Services]
Include=winusb.inf
AddService=WinUSB,0x00000002,WinUSB_ServiceInstall,

;[3]
[WinUSB_ServiceInstall]
DisplayName     = %WinUSB_SvcDesc%
ServiceType     = 1
StartType       = 3
ErrorControl    = 1
ServiceBinary   = %12%\WinUSB.sys

;[4]
[USB_Install.Wdf]
KmdfService=WINUSB, WinUsb_Install

[WinUSB_Install]
KmdfLibraryVersion=1.5

;[5]
[USB_Install.HW]
AddReg=Dev_AddReg

[Dev_AddReg]
HKR,,DeviceInterfaceGUIDs,0x1000,"{36b9c3fe-5fd0-4dd9-9c2f-
561eb8ddaab4} "

;[6]
[USB_Install.CoInstallers]
AddReg=CoInstallers_AddReg
CopyFiles=CoInstallers_CopyFiles

[CoInstallers_AddReg]
HKR,,CoInstallers32,0x00010000,"WdfCoInstaller01005.dll,
WdfCoInstaller","WinUSBCoInstaller.dll"

[CoInstallers_CopyFiles]
WinUSBCoInstaller.dll
WdfCoInstaller01005.dll

[DestinationDirs]
CoInstallers_CopyFiles=11

; ================ Source Media Section =====================
;[7]

[SourceDisksNames]
1 = %DISK_NAME%,,,\i386
2 = %DISK_NAME%,,,\amd64

[SourceDisksFiles.x86]
WinUSBCoInstaller.dll=1
```

```
WdfCoInstaller01005.dll=1

[SourceDisksFiles.NTamd64]
WinUSBCoInstaller.dll=2
WdfCoInstaller01005.dll=3

; ================== Strings ==================

[Strings]
ProviderName="Gelada"
ClassName="XGelada"
USB\MyDevice.DeviceDesc="LectorSC"
WinUSB_SvcDesc="LectorSC"
DISK_NAME="Drivers"
```

El contingut dels arxius INF no ha estat mai del tot explicat per "Microsoft Corporation" però es pot utilitzar aquesta plantilla per modificar-la segons les necessitats. Aquest arxiu funciona per la instal.lació del nostre dispositiu i, per altres dispositius, es recomana canviar el contingut estrictament essencial.

Aquest tipus d'arxius es divideixen en seccions que s'identifiquen per un nom entre claudàtors. El punt i coma s'utilitza per comentaris el contingut que el precedeix no és processat.

La primera de les seccions es "Version" on només haurem de modificar els camps "Class", "ClassGuid" i "DriverVer". D'aquests el més important és el segon ja que es tracta d'un identificador únic que permet al sistema operatiu identificar dispositius que instal.la i configura de la mateixa manera, i que s'anomena de manera genèrica "Guid". En cas de necessitar un nou "Guid", s'haurà de modificar total o parcialment el que s'ha utilitzat en el nostre arxiu INF.

La següent secció és "Manufacturer" on podem canviar només el camp "XGelada" pel nom que es vulgui, però que condicionarà el nom de les altres dos seccions: "XGelada.NTx86" i "XGelada.NTamd64". En aquestes dues seccions és molt important definir un identificador de fabricant ("VID" o "Vendor ID") i de producte ("PID" o "Product ID"). Cal que els valors aquí inclosos coincideixin amb els inclosos al descriptor USB del nostre arxiu de firmware. El "VID" de "Microchip Technology Inc" és "04D8" i és recomanable usar-lo per evitar conflictes amb altres dispositius. El "PID" el podem escollir al nostre gust.

En aquest punt és important mencionar que per poder comercialitzar un producte sense generar conflictes per coincidència de "VID" i "PID" és necessari pagar un cànon, per obtenir identificadors únics. No fer-ho i intentar vendre productes és delicte.

De totes maneres, per dissenys particulars, acadèmics i no comercials en general, no hi ha cap problema l'ús de "PID" propi, tot i que existeix una mínima possibilitat de generar conflictes per coincidència d'aquests valors amb altres dispositius comercials en el teu ordinador.

En la resta de seccions no cal canviar res fins a arribar a l'última (excepte si utilitzem una versió diferent d'arxius "dll", cas en el que hem de canviar els noms que anem veient que apareixen pels nostres actualitzats).

L'última secció s'anomena "Strings" i és on definim cadenes de text que llavors seran substituïdes al llarg de l'arxiu. En aquest cas també podem escollir els noms que més ens convinguin.

Com es pot comprovar, abans de la secció "Strings" apareix un segon "Guid" que és un identificador per la interfície.

Es recomana també tenir-lo en compte i canviar-lo en cas de canviar el primer (com es pot comprovar ha de ser diferent), tot i que no sembla que el sistema operatiu el tingui en compte durant el procés de configuració del dispositiu.

4.5 Instal.lació del driver

El procés d'instal.lació del driver és exactament el mateix que per altres dispositius de qualsevol tipologia.

En primer lloc guardem els nostres arxius en qualsevol lloc del disc dur del nostre ordinador on els puguem anar a buscar amb facilitat i, preferiblement dins d'una carpeta anomenada "WinUSB".

Quan per primera vegada connectem el nostre dispositiu a l'ordinador, el sistema operatiu desplegarà un quadre de diàleg com el que mostra la Figura número 10 a continuació:

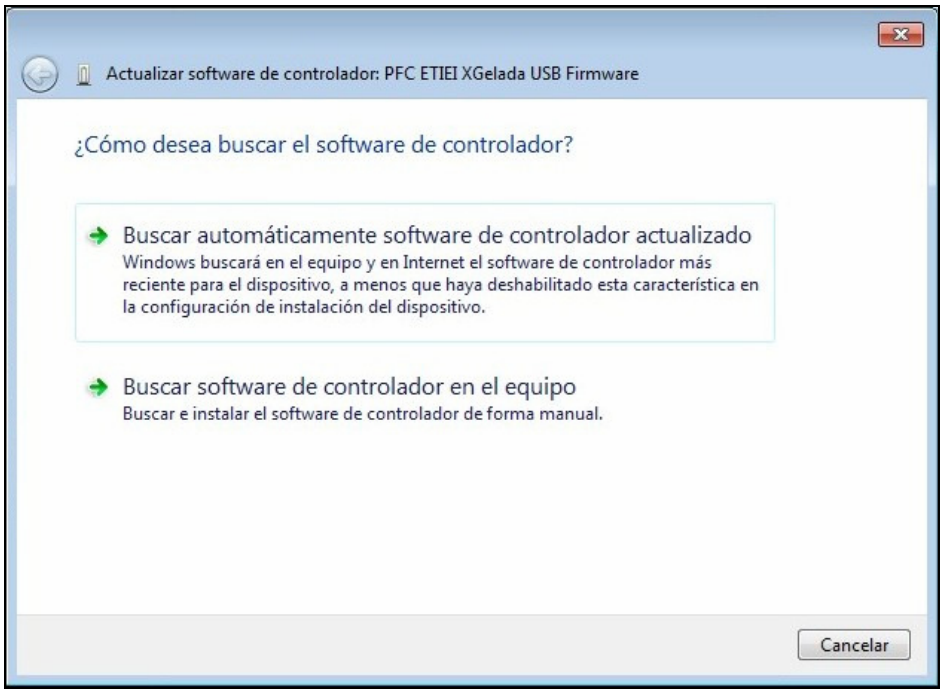


Figura número 10. Quadre diàleg selecció driver.

S'ha de seleccionar la segona opció per designar el controlador o driver de la nostra carpeta. Un cop seleccionem aquesta segona opció apareix un segon quadre de diàleg on podem seleccionar la nostra carpeta, escollint l'opció d'incloure també subcarpetes si és el cas, segons l'estructura de la nostra carpeta. La Figura número 11 mostra aquest segon quadre de selecció de carpetes. Un cop apareixi la nostra carpeta a la casella d'ubicació podem continuar endavant.

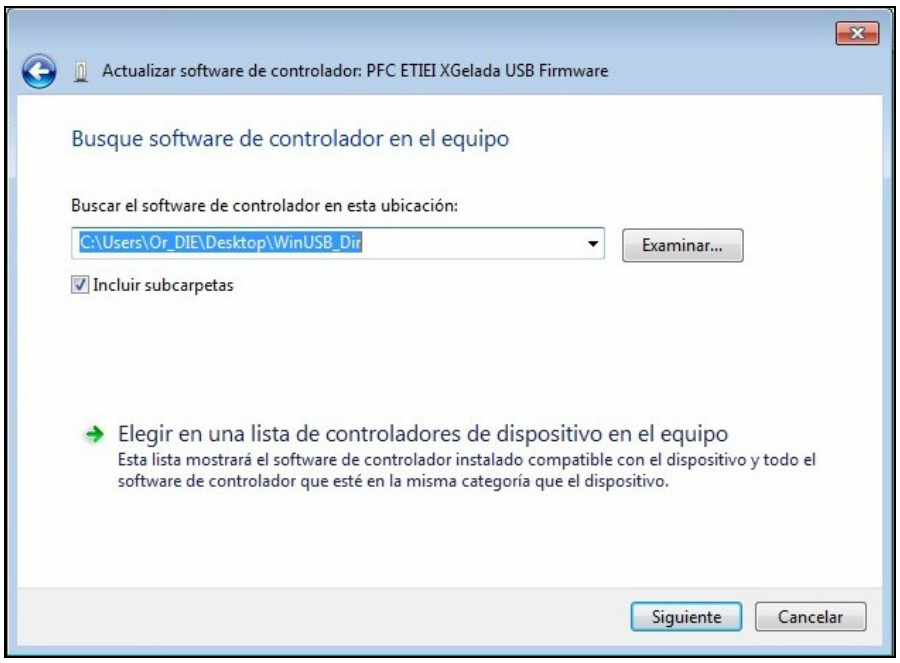


Figura número 11. Quadre de diàleg de selecció de carpetes.

Fet això, el sistema operatiu comença a instal.lar el driver (veure la següent Figura número 12).

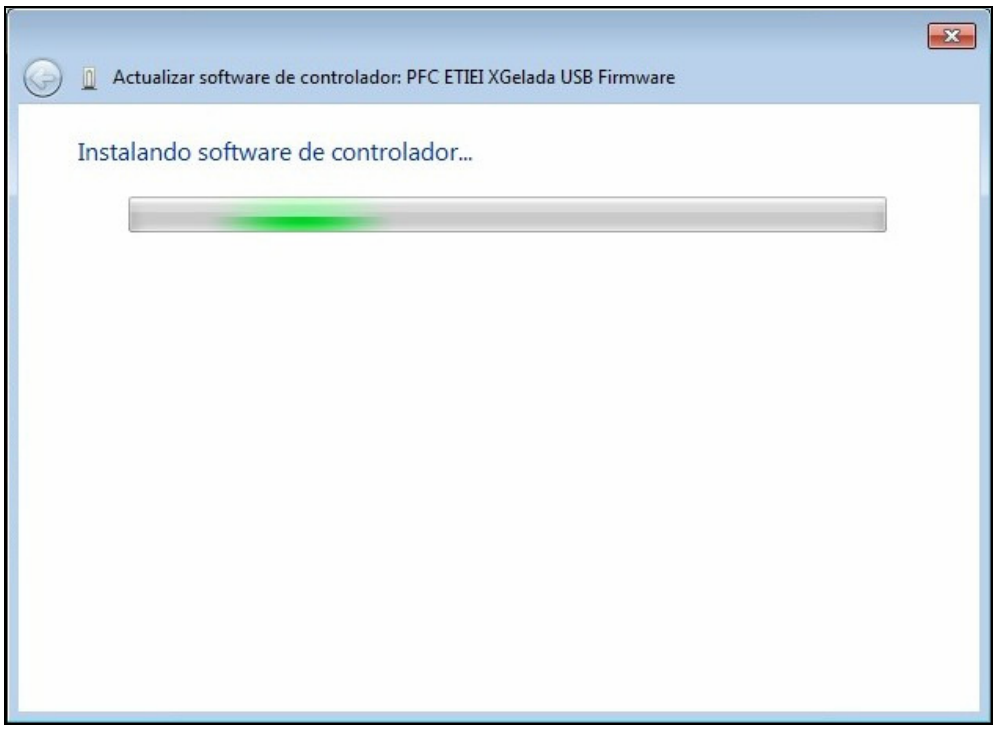


Figura número 12. Quadre de diàleg d'instal.lació.

Durant el procés d'instal.lació, i en funció de la nostra versió del sistema operatiu podem obtenir finestres d'alerta al detectar-se un driver sense signatura digital (veure la següent Figura número 13).

Figura número 13. Quadre de diàleg d'alerta.

Finalment apareix una última finestra de diàleg de confirmació que el nostre controlador està correctament instal.lat, tal i com es mostra a la Figura número 14.

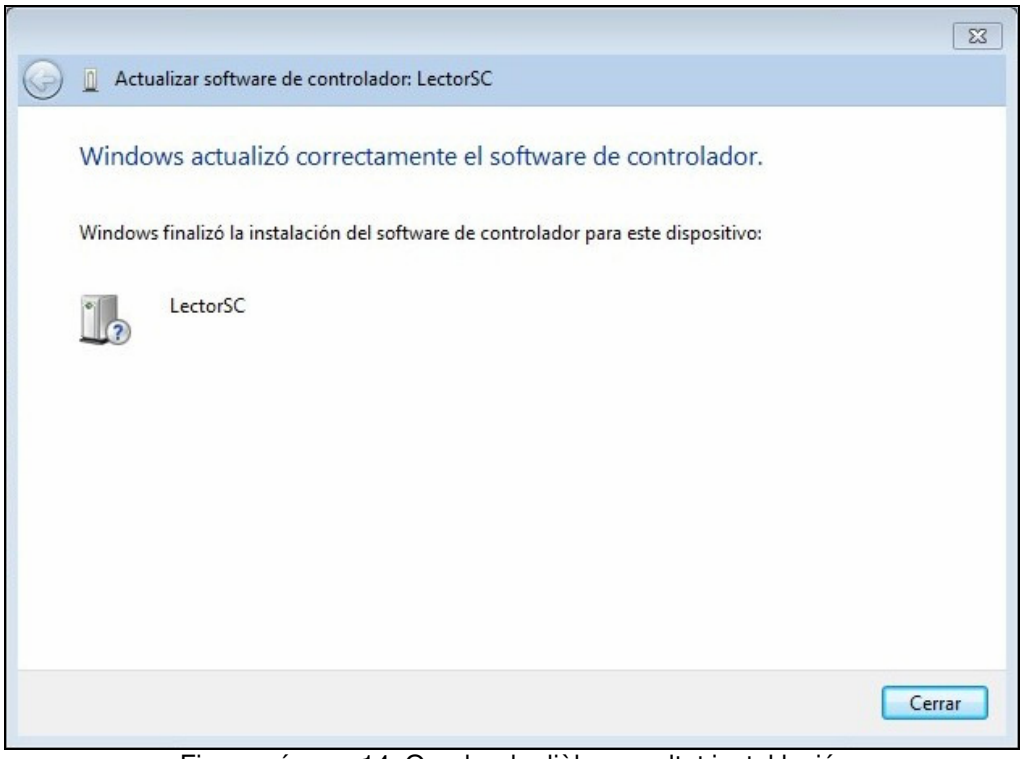


Figura número 14. Quadre de diàleg resultat instal.lació.

Si hem compilat el nostre firmware amb l'opció activada de funcionament de díodes indicadors, veurem com el LED que indica estat configurat està encès, i en conseqüència el nostre dispositiu ha aconseguit ésser configurat correctament pel sistema operatiu a través del driver subministrat.

Una forma de confirmar que els descriptors USB del nostre dispositiu han estat correctament enviats, consisteix en consultar l'administrador de dispositius del nostre sistema operatiu on podrem veure tots els paràmetres del nostre lector USB (sempre i quan estigui connectat).

Hauria d'aparèixer a la llista general de dispositius connectats, com es pot veure a la següent Figura número 15:

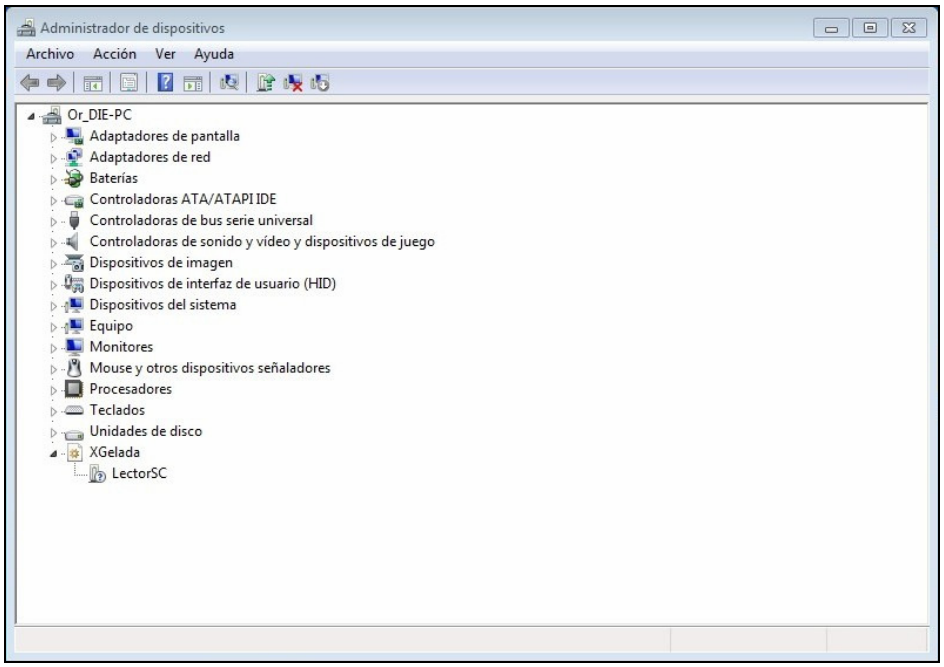


Figura número 15. Llista general de dispositius connectats.

Si fem click sobre el nostre dispositiu a la llista i escollim que el sistema ens mostri les propietats (tal i com es mostra a la Figura número 16), hauria d'aparèixer un quadre de diàleg específic que fa referència al nostre dispositiu i al seu controlador.

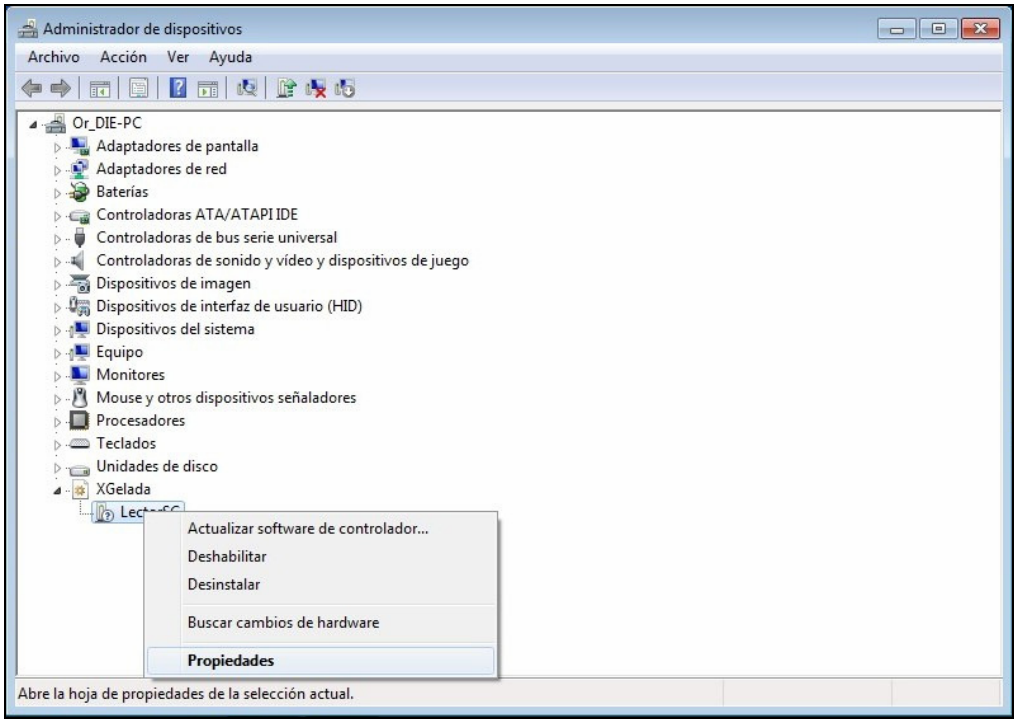


Figura número 16. Dispositiu instal.lat i quadre de propietats.

La següent Figura número 17 mostra el nou quadre de diàleg que apareix on es poden consultar les diferents pestanyes d'informació:

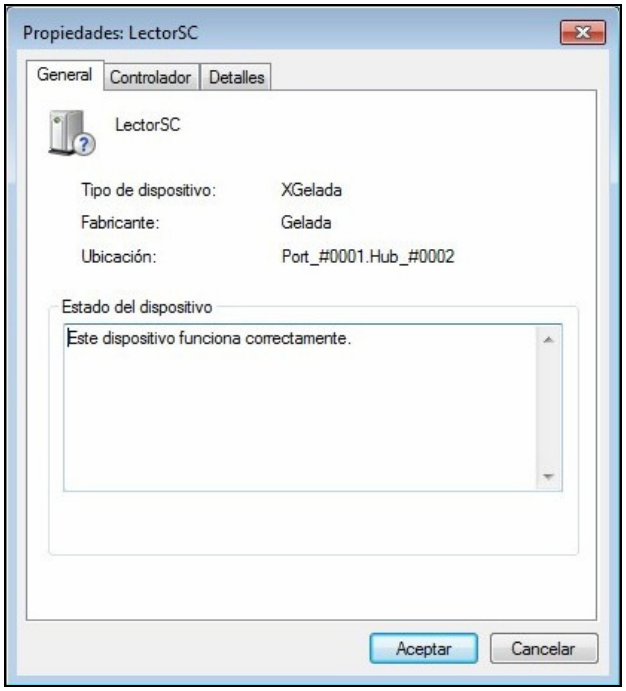


Figura número 17. Quadre d'informació del dispositiu.

A la pestanya de detalls podem consultar tots els paràmetres que defineixen el nostre dispositiu i que haurien de concordar amb els que hem definit als descriptors del firmware i alhora al nostre arxiu INF. A la següent Figura número 18 es pot veure com el "Guid" que hem definit al nostre arxiu INF ha quedat grabat al registre del sistema operatiu.

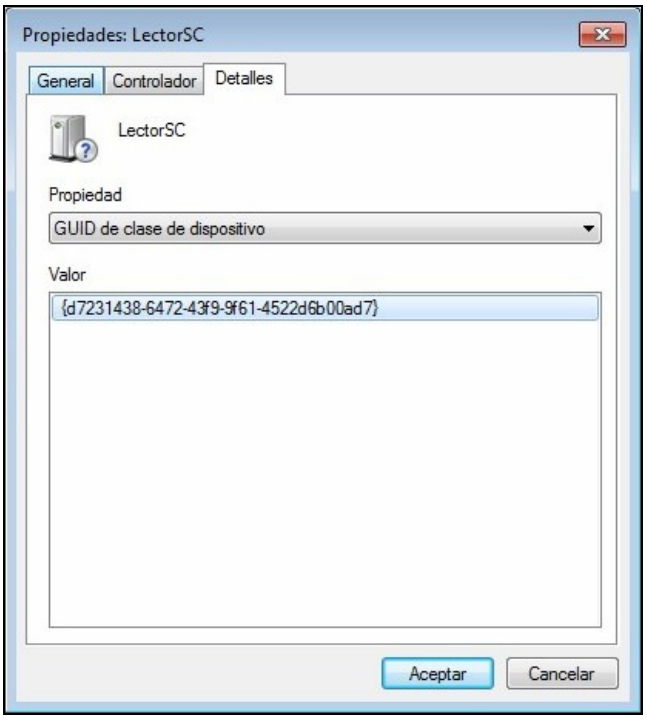


Figura número 18. GUID del dispositiu.

A partir d'aquest moment, cada cop que connectem el nostre dispositiu en el sistema, aquest serà reconegut i configurat usant el corresponent driver (WinUSB), el qual també utilitzarem per comunicar-nos amb el lector des del software, tal i com es veure en el següent capítol.

5 SOFTWARE USB

5.1 Solució adoptada

Com ja s'ha comentat en l'apartat del driver USB, l'elaboració d'un programa informàtic que controli el procés de comunicació USB des del costat de l'ordinador no és una qüestió trivial. Requereix d'un coneixement profund de l'estructura del sistema operatiu i conseqüentment queda reservada a professionals informàtics.

Dins del sistema operatiu "Windows", també pels no programadors, s'hi afegeix el problema d'aconseguir amb facilitat un programa amb una interfície gràfica atractiva i amb la mateixa operativa que la resta de quadres de diàleg dins d'aquest entorn.
Per aquest motiu s'ha optat per buscar un codi de programa ja fet, en un llenguatge de programació que permeti la creació de finestres de diàleg amb facilitat.

El resultat és un codi més o menys obert, elaborat amb el llenguatge de programació "Visual Basic", amb permís per ser utilitzat i modificat per part del seu autor. D'aquesta manera, amb els mínims coneixements de programació informàtica que s'han donat a la carrera, resulta suficient com per modificar únicament allò essencial i adaptar aquest codi a les nostres necessitats.

Aquest programa base del que s'ha partit és de l'autora Jan Axelson i es pot trobar íntegrament a la seva web, de la qual hi ha una referència al capítol 10 Bibliografia. A l'annex B hi ha un llistat complert del programa tal i com ha resultat després de les modificacions. El software de base ha estat elaborat amb l'entorn de programació "Visual Studio 2012 Professional", però nosaltres l'hem modificat i compilat amb la versió "Visual Studio Express 2013", el qual és gratuït i es pot obtenir a la xarxa des de la web de "Microsoft Corporation".

5.2 Estructura d'arxius

Els projectes de Visual Basic 2013 disposen de varis arxius com es pot veure a la següent Figura número 19, la qual mostra una carpeta de projecte amb tot el seu contingut:

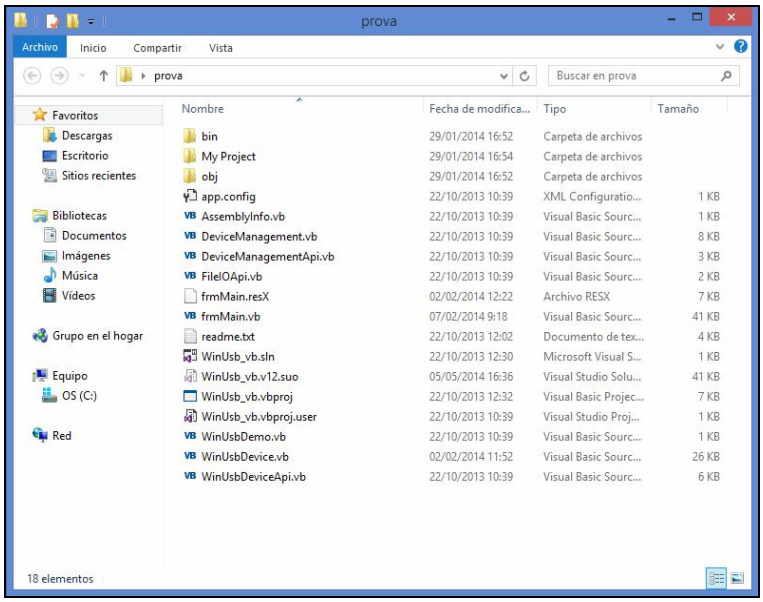


Figura número 19. Carpeta de projecte de Visual Studio.

L'arxiu principal de projecte que cal obrir des de la plataforma "Visual Studio Express 2013" és el que s'anomena "WinUsb_vb.vbproj". Aquest obre una finestra de treball on també es mostra una estructura d'arxius similars al de la carpeta al nostre disc dur. Ho mostra la Figura número 20 a continuació:

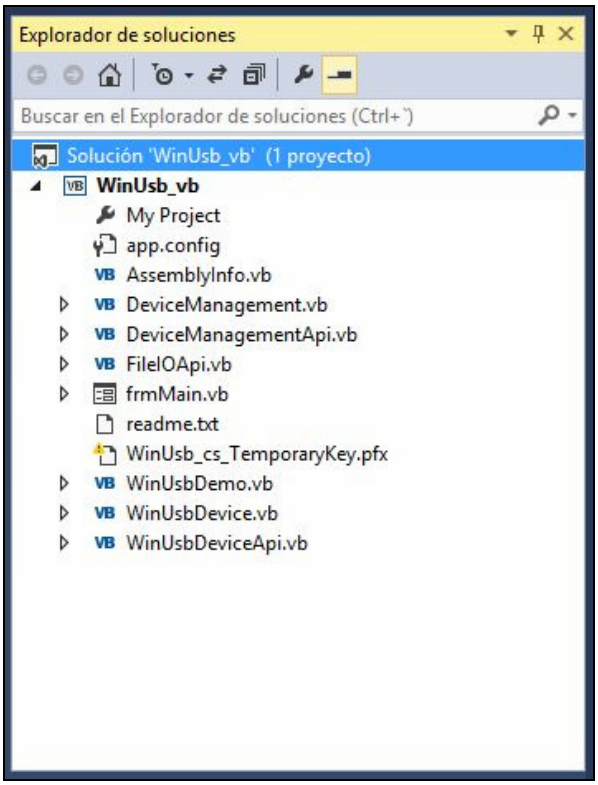


Figura número 20. Estructura d'arxius Visual Studio.

En tots dos casos podem veure que hi ha molts arxius amb extensió "vb" perquè el codi de programa és molt complex i l'autora l'ha estructurat en diferents mòduls de codi.

De tots els arxius que hi ha en el projecte, hem modificat només els següents:

- frmMain.vb: conté la definició de la finestra gràfica amb els diferents elements com botons i caselles de selecció, etc. Aquests elements porten associades les accions a realitzar en cas de ser activats d'alguna forma.
- WinUsbDevice.vb: conté la funció "DoControlReadTransfer" on hem hagut d'especificar el número de request a enviar al lector.

La resta d'arxius no és necessari modificar-los en absolut. La majoria de canvis s'han donat a l'arxiu de codi "frmMain.vb". Val a dir que la modificació d'aquest arxiu s'ha fet bàsicament a través de la interfície gràfica de l'entorn de programació que permet modificar l'aspecte final de la finestra de diàleg de manera gràfica, adaptant el codi de manera automàtica. En qualsevol cas també s'han hagut d'ajustar diverses funcions actuant directament sobre el codi.

En el següent subapartat s'intenta localitzar els punts de codi on s'han realitzat canvis. En qualsevol cas, el codi complert dels diferents arxius està llistat a a l'Annex B.

5.3 Modificacions realitzades

Com s'ha comentat en el subapartat anterior, només són dos els arxius modificats de tots els arxius de codi "Visual Basic" que hi ha en el projecte. Seguidament, es mostra l'única línia modificada de l'arxiu "WinUsbDevice.vb":

```
(...)
'  The request number that identifies the specific request.
setupPacket.Request = 2
(...)
```

Modificant aquest paràmetre podem escollir quin número de "request" enviem al lector, en aquest cas el número 2 que correspon a la lectura de la informació de la targeta.

A continuació es mostra els canvis que s'han realitzat directament al codi de l'arxiu "frmMain.vb" relatius especialment a traduccions de text i a modificació de constants i identificadors del nostre dispositiu USB per aconseguir que el software detecti al lector de manera unívoca. Com ja s'ha comentat, l'aspecte final de la finestra de diàleg del software s'ha modificat gràficament, deixant a l'entorn de programació que re-adaptés el codi escrit. Aquest aspecte es mostra al subapartat següent.

```
(...)

Private Const DeviceInterfaceGuid As String = "{36b9c3fe-5fd0-4dd9-
9c2f-561eb8ddaab4}"

(...)

Private Const VendorID As String = "04D8"
Private Const ProductID As String = "A001"

(...)

Console.WriteLine("S'ha connectat un dispositiu USB")

(...)

Console.WriteLine("S'ha desconnectat un dispositiu USB")

(...)

speed = "velocitat LS o FS"
Case 3
speed = "velocitat HS o SS"
End Select
LstResults.Items.Add(" ")
LstResults.Items.Add("El dispositiu admet " & speed)
LstResults.Items.Add(" ")

(...)

Const classGuid As String = "d7231438-6472-43f9-9f61-4522d6b00ad7"

(...)

MyMarshalToForm(FormActions.AddItemToListBox.ToString(), "Dispositiu
lector no detectat (WMI)")

(...)

MyMarshalToForm(FormActions.AddItemToListBox.ToString(), "Dispositiu
lector detectat (WMI):")

(...)

LstResults.Items.Add("Dispositiu lector no detectat.")
```

```
(...)

LstResults.Items.Add("Dispositiu lector detectat.")

(...)

LstResults.Items.Add("El sistema operatiu no és Windows XP o
superior.")
LstResults.Items.Add("El driver WinUsb requereix Windows XP o
superior.")

(...)
```

Seguidament s'inclou el contingut més important modificat en l'arxiu "frmMain.vb", en referència a la funció "RequestControlReadTransfer()", que és la que s'executa quan s'activa el botó de lectura de les dades de la targeta intel.ligent insertada al lector. Per les limitacions del format d'aquest document, s'han justificat totes les línies de codi a l'esquerra de la taula, sense tabulacions. Es pot consultar a l'Annex B el mateix codi amb les tabulacions corresponents per facilitar la comprensió del programa resultant.

```
Private Sub RequestControlReadTransfer()
Try
Dim dataStage = New Byte(35) {}

If Not (_deviceReady) Then
FindMyDevice()
End If

If _deviceReady Then
Dim success =
_myWinUsbCommunications.DoControlReadTransfer(_winUsbHandle,
dataStage)

Dim formText As String
Dim Tall_prova As String
Dim Cadena_Origin As String
Dim Longitud_Cadena As Integer

If success Then
formText = "Bytes rebuts a través de transferència CONTROL:"

AccessForm(FormActions.AddItemToListBox.ToString(), " ")

AccessForm(FormActions.AddItemToListBox.ToString(), formText)

formText = ""
For i As Int32 = 0 To dataStage.Length - 1
formText = formText & String.Format("{0:X2} ", dataStage(i)) & " "
Next i
```

```
Cadena_Origin = formText
Longitud_Cadena = Len(Cadena_Origin)

While True
Tall_prova = Microsoft.VisualBasic.Right(Cadena_Origin, 4)
If Tall_prova = "00  " Then
Cadena_Origin = Microsoft.VisualBasic.Left(Cadena_Origin,
Longitud_Cadena - 4)
Longitud_Cadena = Len(Cadena_Origin)
Else
Cadena_Origin = Microsoft.VisualBasic.Left(Cadena_Origin,
Longitud_Cadena - 4)
Exit While
End If
End While

Cadena_Origin = Trim(Cadena_Origin)

If Cadena_Origin = "00" Then
formText = "No es detecta targeta insertada."
ElseIf Cadena_Origin = "EF" Then
formText = "La resposta de la targeta ha acabat prematurament."
ElseIf Cadena_Origin = "FF" Then
formText = "La targeta insertada no respon al RESET."
Else
formText = Cadena_Origin
End If


AccessForm(FormActions.AddItemToListBox.ToString(), formText)

Else
formText = "La transferència CONTROL ha fallat."

AccessForm(FormActions.AddItemToListBox.ToString(), " ")

AccessForm(FormActions.AddItemToListBox.ToString(), formText)

End If
End If
ScrollToBottomOfListBox()
Catch ex As Exception
DisplayException(Name, ex)
Throw
End Try
End Sub
```

Com ja s'ha comentat, aquesta funció ha estat modificada de manera que un cop rebuda la cadena de caràcters enviats per la targeta, es formateja i analitza per ser mostrada per pantalla. Segons la resposta rebuda, es mostren vàries informacions per pantalla. Aquesta part de codi es dependent de l'aplicació concreta.

5.4 Aspecte del software

La Figura número 21 mostra l'aspecte de la finestra de diàleg del software original. Es tracta d'una finestra que intenta incloure els diferents tipus de transferència USB, de manera el més oberta possible, perquè l'usuari, sense modificar res pugui fer proves senzilles de comunicació.

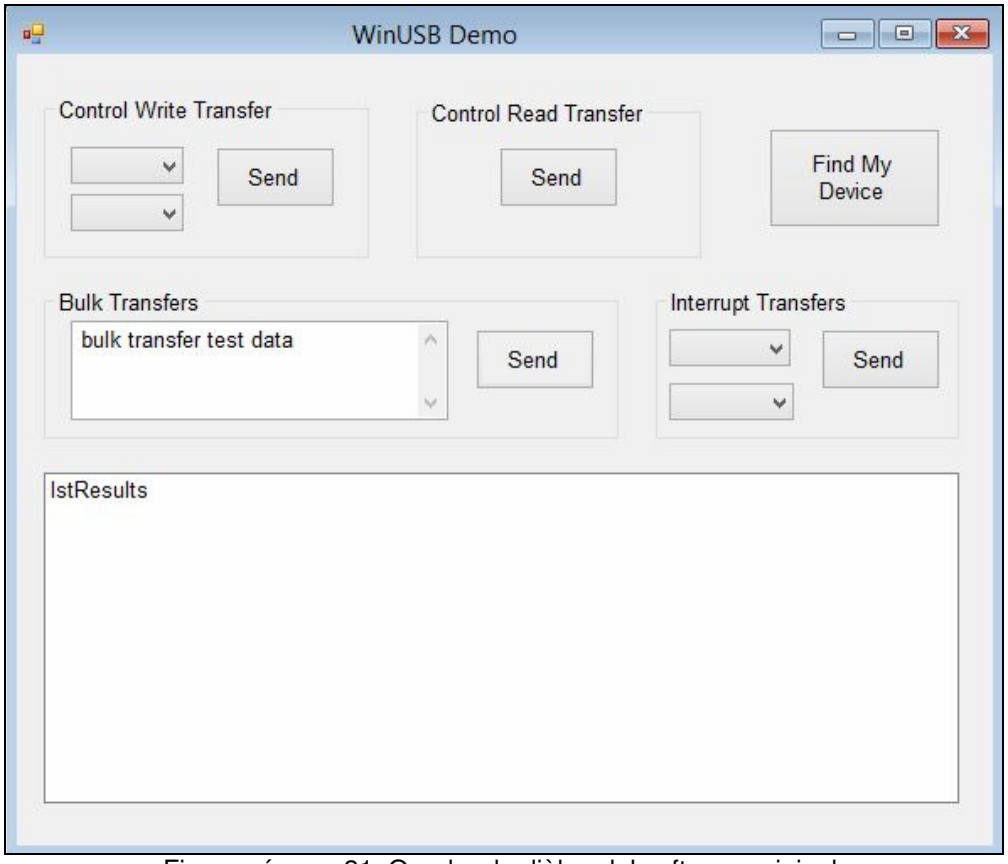


Figura número 21. Quadre de diàleg del software original.

En qualsevol, cas, la majoria de les caselles i botons són prescindibles (en el nostre cas i en la majoria d'aplicacions ja que normalment estaran focalitzades en un tipus de transferència determinada).

Per aquest motiu, amb l'entorn de programació utilitzat ("Visual Studio Express 2013"), s'ha reformatejat la finestra de diàleg, eliminant totes aquelles parts en el nostre cas prescindibles. El fet que sempre enviem la mateixa petició al lector, fa que puguem mantenir només un botó per aquest enviament, i una casella on mostra la resposta de rebuda del lector.

L'aplicació dona l'oportunitat de buscar de manera expressa el lector amb un botó, tot i que utilitza un sistema (anomenat "WMI") pel qual es detecta automàticament la connexió i desconnexió de dispositius USB en el bus.

L'aspecte final de la nostra aplicació es mostra a continuació a la Figura número 22:

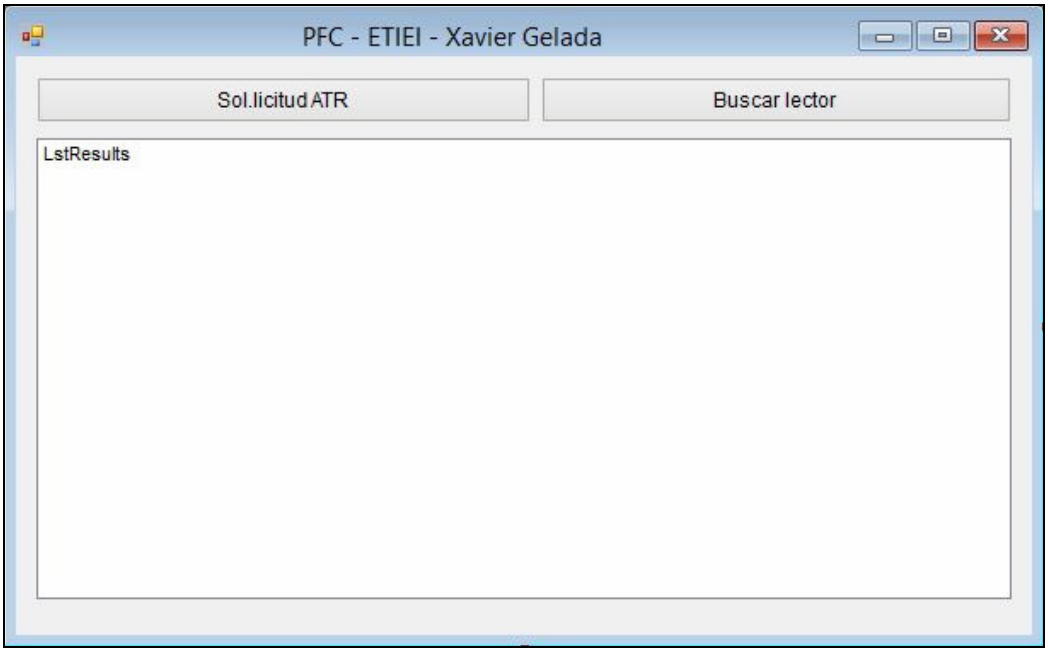


Figura número 22. Quadre de diàleg del software modificat.

6 FUNCIONAMENT DEL CONJUNT

En aquest capítol, es recull el funcionament general del conjunt resultant (hardware, firmware i software). La Fotografia número 1 mostra el conjunt d'elements que necessitem per provar la nostra aplicació:

Fotografia número 1. Conjunt d'elements de l'aplicatiu.

És necessari doncs un ordinador personal amb, com a mínim, una connexió USB lliure, on s'hi hagi instal.lat prèviament el driver WinUSB com s'ha explicat anteriorment, així com també el software adaptat per a la nostra aplicació.

Per altra banda, lògicament, cal el hardware dissenyat, amb un cable de connexió USB, i qualsevol tipus de targeta intel.ligent o "smartcard" compatible amb l'estàndard ISO7816 (la majoria de targetes intel.ligents amb aplicacions generals com són els d'identificació o de sistemes de pagament).

Si s'executa el programa sense el lector connectat, el software així ho indica en la seva finestra de diàleg (veure Figura número 23).

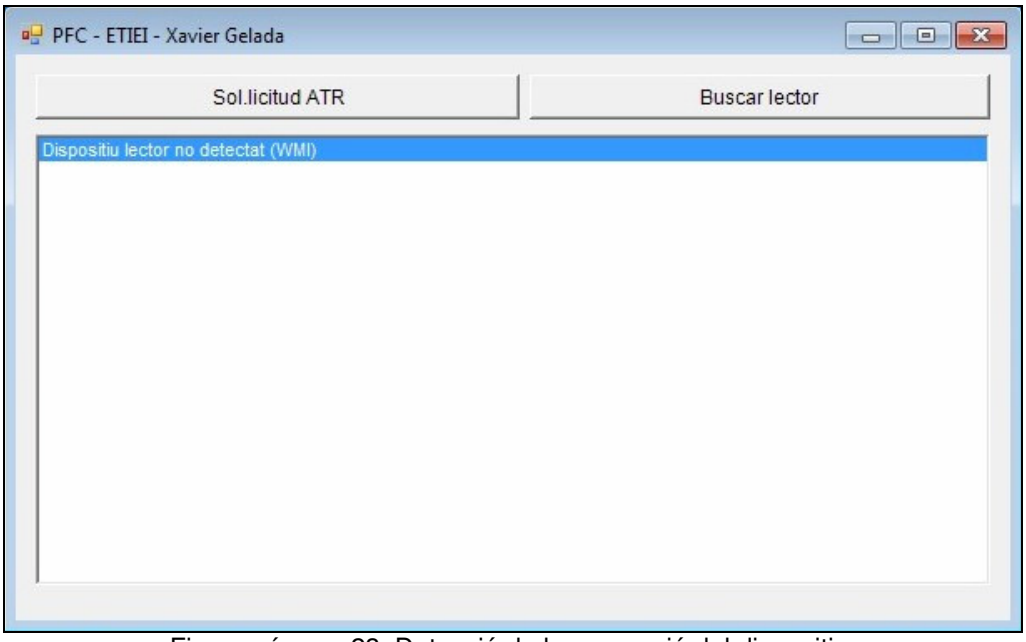
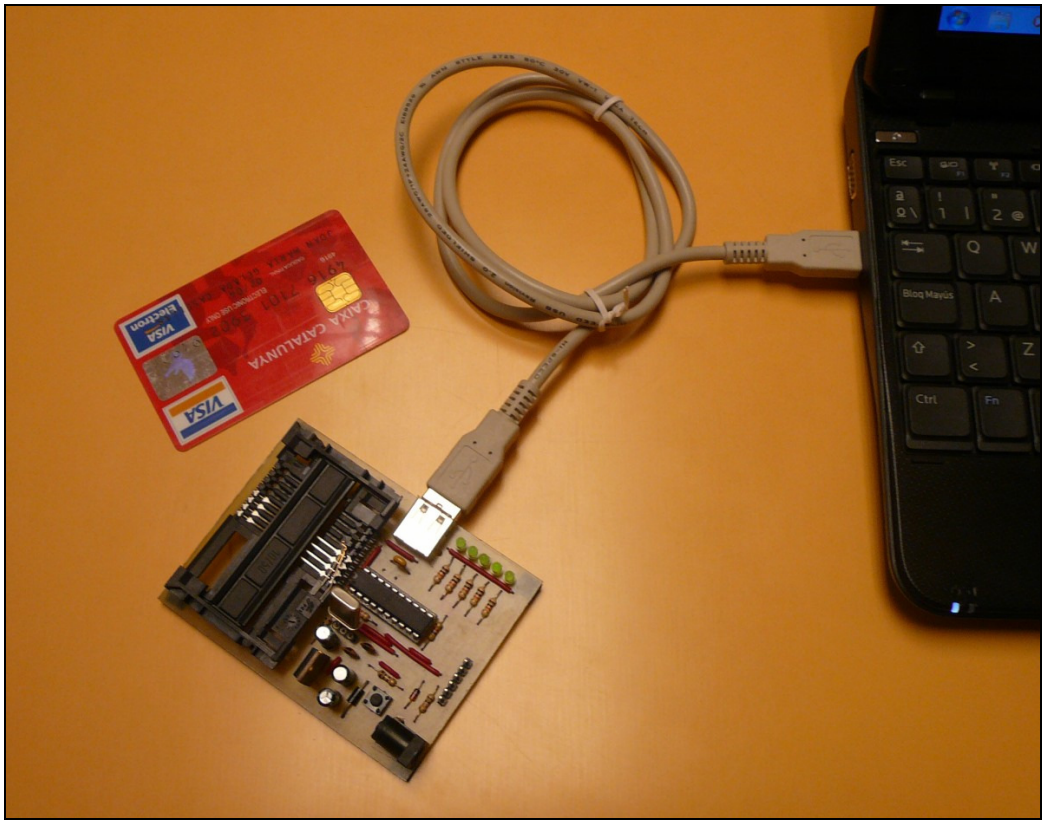


Figura número 23. Detecció de la presenció del dispositiu.

Si es connecta el lector al port USB de l'ordinador (veure Fotografia número 2), podrem observar com els diferents díodes LED es van il.luminant indicant els diferents estats del dispositiu dins del bus.



Fotografia número 2. Connexió del dispositiu a l'ordinador.

La Figura número 24 mostra el que apareix a la finestra de diàleg un cop connectat el lector. El nostre dispositiu és detectat automàticament i es mostren els paràmetres que l'identifiquen dins del sistema operatiu.

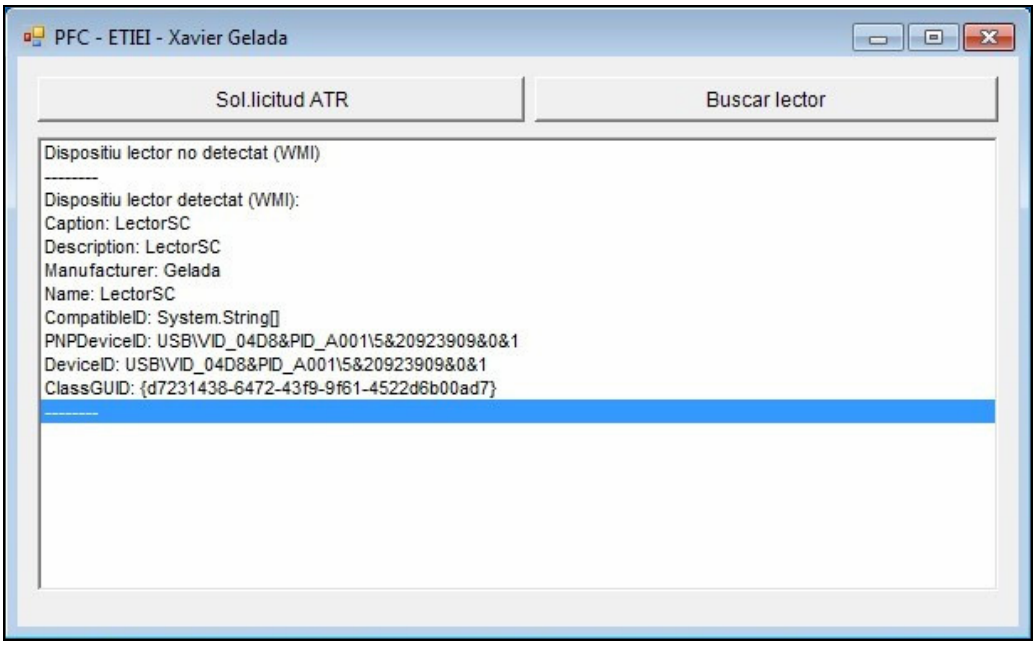


Figura número 24. Detecció del lector.

En aquesta situació, i sense targeta insertada, podem intentar sol.licitar informació de la mateixa, però la resposta serà que no es detecta targeta, com es pot veure reiteradament a la Figura número 25:

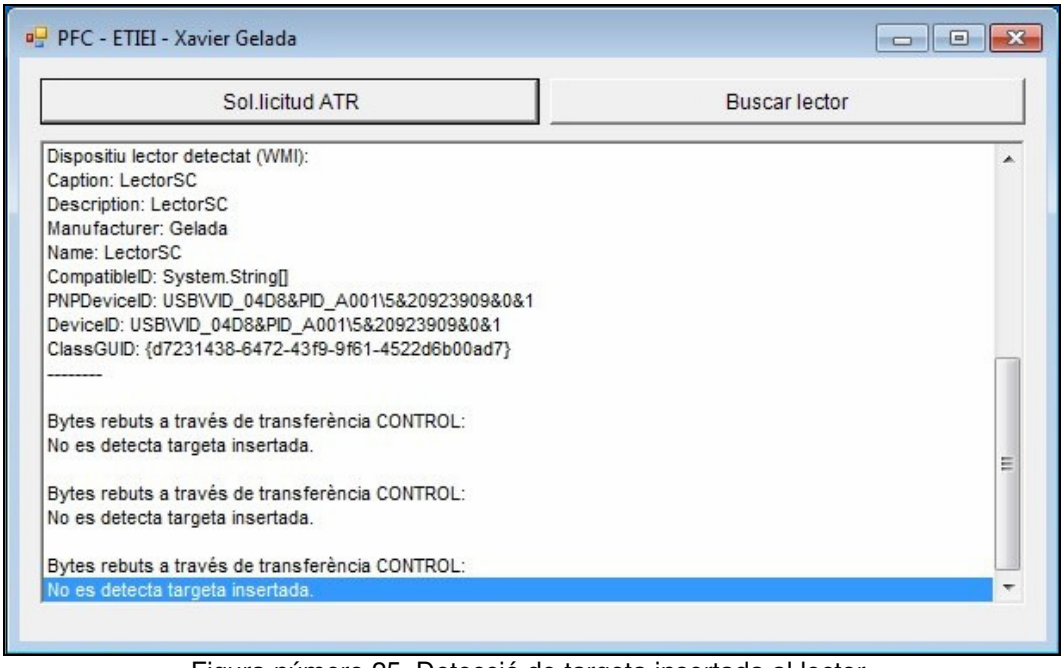
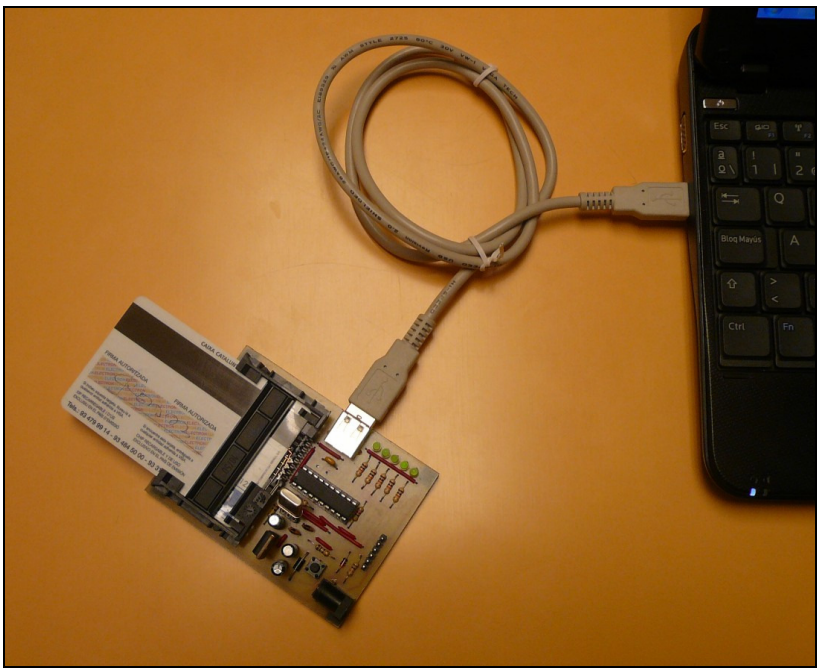


Figura número 25. Detecció de targeta insertada al lector.

Un cop insertada la targeta intel.ligent en el sòcol del dispositiu (veure Fotografia número 3), i si es repeteix el procés, obtenim per pantalla l'anomenada "ATR" o resposta al reset. És una cadena de bytes molt curta però suficient per comprovar que la comunicació cap al lector i des d'ell a través del bus USB.



Fotografia número 3. Inserció de targeta al lector.

La següent Figura número 26 mostra la cadena de bytes enviada per la targeta a través del lector i de la interfície USB:

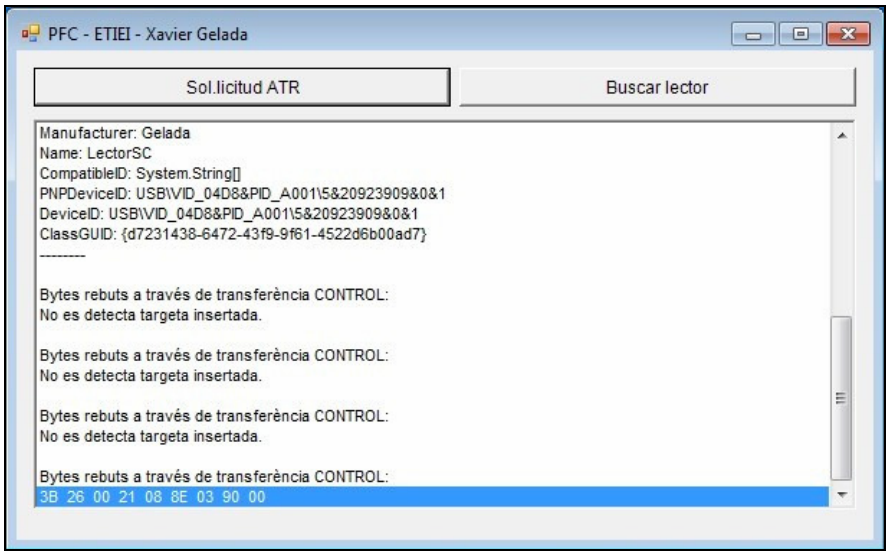


Figura número 26. Recepció de la informació de la targeta.

7 RESUM DEL PRESSUPOST

L'import final per a la realització del present projecte queda recollit de forma detallada al document 5.PRESSUPOST i s'inclou a continuació el quadre resum final on es distingeix bàsicament el cost de la realització del circuit imprès (hardware) i el de programació (firmware i software):

Capítol	Import
1. Circuit imprès	
1.1 Disseny del circuit	500,00€
1.2 Material i components electrònics	33,72€
1.3 Realització de la placa	80,00€
2. Programació firmware i software	4.375,00€
TOTAL PRESSUPOST:	**4.988,72€**

L'import del pressupost resulta doncs en la xifra de QUATRE MIL NOU-CENTS VUITANT VUIT EUROS I SETANTA DOS CÈNTIMS (4.988,72€).

Condicions de validesa:

- El període de validesa d'aquest pressupost és de 6 mesos.
- No s'inclou l'Impost sobre el Valor Afegit (IVA).
- L'import final és funció dels mitjans materials dels que disposi el desenvolupador.

S'inclou també a continuació el cost d'elaboració del present projecte, el qual també s'adjunta en el document 5.PRESSUPOST en forma d'annex:

Concepte	Quantitat	Preu unitari	Import
Material d'oficina	1,00	150,00€	150,00€
Hores d'elaboració	105,00	25,00€	2.625,00€
		TOTAL:	**2.775,00€**

Apliquen les mateixes condicions de validesa que pel pressupost principal.

8 CONCLUSIONS

Un cop planificada, dissenyada i executada l'aplicació descrita en el present document, es constata que funciona correctament, al poder-se comprovar mitjançant el dispositiu resultant, el qual combina el hardware, el firmware que s'han realitzat i el software que s'ha adaptat.

En conseqüència s'ha assolit l'objectiu que inicialment es perseguia amb l'elaboració del present projecte. El principal i més important resultat, el codi firmware en assemblador, s'inclou en el primer dels annexos d'aquesta memòria, per poder ser consultat.

Xavier Gelada Alfonso
Enginyer tècnic industrial en electrònica industrial

Girona, 2 de juny de 2014

9 RELACIÓ DE DOCUMENTS

El present projecte consta dels següents documents:

1. MEMÒRIA
2. PLÀNOLS
3. PLEC DE CONDICIONS
4. ESTAT D'AMIDAMENTS
5. PRESSUPOST

Cadascun d'ells es presenta enquadernat per separat, però recopil.lats dins d'una carpeta de projectes, amb el complement d'un document de resum i d'una còpia en suport informàtic.

10 BIBLIOGRAFIA

ANGULO USATEGUI, J.M., ANGULO MARTINEZ, I. Microcontroladores PIC – Diseño práctico de aplicaciones. Editorial McGraw Hill. Madrid. 2001.

AXELSON, JAN. USB Complete - The developer's guide. Lakeview Research LLC. Madison, Estats Units d'Amèrica. 2009.

BATES, PETER. PC Interfacing using USB. Editorial Bernard Babani Ltd. Londres. 2003.

BEYONDLOGIC.ORG. USB in a NutShell. (http://www.beyondlogic.org/usbnutshell/usb1.shtml, 12 de maig de 2014)

COMPUTER SOLUTIONS LTD. USB – A brief tutorial. (http://www.computer-solutions.co.uk/info/Embedded_tutorials/usb_tutorial.htm, 12 de maig de 2014)

HYDE, JOHN. USB Design by Example. Editorial Intel Press. Estats Units d'Amèrica. 2001

KOON, John. USB Peripheral Design. Editorial Annabooks. San Diego, Estats Units d'Amèrica. 1998.

MALVINO, A.P. Principios de Electrónica. Editorial McGraw Hill. Madrid. 1999.

MCDOWELL, S., SEYER, MARTIN. USB Explained. Editorial Prentice Hall. New Jersey, Estats Units d'Àmerica. 1999.

MICROCHIP TECHNOLOGIES INC., PIC-18F14K50 Datasheet. (http://ww1.microchip.com/downloads/en/devicedoc/41350c.pdf, 12 de maig de 2014).

MQP ELECTRONICS LTD. USB Made Simple. (http://www.usbmadesimple.co.uk/, 12 de maig de 2014)

USB-IF. USB 3.1 Specification. (http://www.usb.org/developers/docs/, 11 de maig de 2014)

11 GLOSSARI

.NET	Plataforma de programació de Microsoft enfocada a treballar amb els llenguatges de programació C++ C# i Visual Basic, amb gran quantitat d'aplicacions i solucions predefinides.
Addressed state	Estat dins del procés d'enumeració USB pel qual un dispositiu ja disposa d'una adreça identificativa única en el bus, però encara no està configurat per poder ser utilitzat.
API	Acrònim de "Application Programming Interface". Fa referència a aquelles funcions proporcionades pel sistema operatiu per realitzar-hi determinades tasques.
Class	En les especificacions USB, conjunt de dispositius USB que poden ser inclosos en una categoria o tipologia determinada i que disposen d'un procediment específic de comunicació, normalment més simple, de manera que no es requereix d'un driver específic i extern al sistema operatiu per fer-los funcionar.
Config state	Estat dins del procés d'enumeració USB pel qual el dispositiu ja ha estat completament enumerat i configurat amb èxit. A partir d'aquest moment ja pot ser utilitzat per qualsevol software.
Datasheet	Fitxa de característiques tècniques d'un circuit integrat.
Default state	Estat dins del procés d'enumeració USB pel qual un dispositiu ja ha estat resetejat i té assignada la adreça zero com a pas previ a rebre la seva adreça definitiva i única dins del bus.
Device	Forma genèrica de fer referència a un dispositiu.

Descriptor	Qualsevol dels fragments d'informació que ha de contenir qualsevol dispositiu USB, els quals són intercanviats amb l'ordinador durant el procés de configuració per obtenir dades i característiques del dispositiu.
Dll	Tipologia d'arxiu de codi executable que es carreguen sota demanda d'un programa per part del sistema operatiu.
Driver	Terme anglès per fer referència al software controlador d'un dispositiu perifèric i que fa d'enllaç entre ell i el sistema operatiu.
FS	Veure el terme "Full Speed".
Full Speed	Una de les possibles velocitats de funcionament dels dispositius USB segons l'especificació 1.0 per la qual el bitrate brut és de 12 Mbits/s.
GUID	Acrònim de "Globally Unique Identifier". Número pseudo-aleatori utilitzats per aplicacions de software per identificar unívocament un element.
HID	Acrònim de "Human Interfice Device". Es tracta d'una classe o categoria USB que engloba tots els dispositius que permeten als humans introduir dades als ordinadors (teclats, ratolins...). Veure el terme "Class".
INF	Tipologia d'arxius relacionats amb els drivers o controladors amb indicacions pel sistema operatiu de quins arxius cal instal.lar i de quina forma per aconseguir que el driver desitjat quedi correctament instal.lat a l'ordinador.
ICSP	Veure "In Circuit Serial Programming".
In Circuit Serial Programming	Sistema pel qual un circuit integrat que requereix

	programació externa, pot ser programat en sèrie sense necessitat de retirar-lo del sòcol o circuit on està instal.lat.
Isochronous	Una de les quatre tipologies de transferència de dades incloses a les especificacions USB. Es caracteritza perquè la transferència de dades es realitza a intervals regulars i no hi ha comprovació d'error. Especialment pensada per la transferència de so o imatge en temps real.
LS	Veure "Low Speed".
Low Speed	Una de les possibles velocitats de funcionament dels dispositius USB segons l'especificació 1.0 per la qual el bitrate brut és de 1.5 Mbits/s.
Packet	Unitat d'informació segons l'especificació USB que configura les transaccions, les quals, alhora, conformem les transferències.
PID	Veure "Product ID".
Power state	Estat dins del procés d'enumeració USB pel qual un dispositiu ha estat connectat al bus i està correctament alimentat, apunt per ser sotmès al procés d'enumeració.
Product ID	Identificador que permet distingir els diferents productes per a un mateix fabricant de productes USB. Un dispositiu USB concret queda definit pel un identificador de fabricant, un identificador de producte, un número de sèrie i una versió.
Remote wake-up	Característica que permet a un dispositiu USB demanar la sortida de l'estat de baix consum a l'ordinador.
Request	Terme que fa referència a una de les moltes peticions que el l'ordinador pot fer al dispositiu a través del bus USB.

	Algunes d'aquestes peticions responen al funcionament normal de la interfície USB i són les anomenades "Standard requests". Altres són pròpies d'un software concret i normalment formen part de l'aplicació específica que fa funcionar el dispositiu. Aquestes últimes són anomenades "Vendor requests".
SOF	Veure "Start Of Frame".
Start Of Frame	Tipologia de packet enviada cada 1 ms per definir les finestres temporals en les que el sistema USB intercanvia informació en forma de transaccions.
SC	Acrònim de "SmartCard". Veure aquest terme.
SE0	Acrònim de "Single Ended Zero". Veure aquest terme.
Single Ended Zero	Situació en la que ambdues línies diferencials del bus USB estan alhora a nivell baix. Això indica situacions com resets, final de packet i desconnexió.
Smartcard	Tipologia de targeta plàstica que porta un circuit integrat incorporat, normalment un microcontrolador, orientada a aplicacions diverses com per exemple sistemes de pagament o d'identificació. Existeix un estàndard mundial que és la ISO7816.
String	Terme anglès per fer referència a una cadena o successió de dades, normalment de text, però de manera més general poden ser de qualsevol tipologia.
Token	Tipus de packet USB que normalment encapçala una transacció i que indica la tipologia de la transacció així com l'origen i el destí.
Transceiver	Circuit electrònic que connecta el dispositiu o l'ordinador

	amb el bus i que interpreta i interactiu amb el mateix per materialitzar la comunicació USB.
USB	Acrònim de "Universal Serial Bus". Veure aquest terme.
Universal Serial Bus	Interfície de comunicació d'alt nivell, que permet connectar perifèrics a l'ordinador i avui en dia també perifèrics entre ells, apareguda als anys 90, impulsada per diverses empreses privades i que combina hardware, firmware i software per aconseguir uns ports de comunicació sense els inconvenients i limitacions dels ports tradicionals.
Vendor-specific	Concepte de les especificacions USB per distingir aquelles qüestions que són definides pel fabricant i que queden fora de les classes o categories USB.
Vendor ID	Identificador que permet distingir els diferents fabricants de productes USB. Un dispositiu USB concret queda definit pel un identificador de fabricant, un identificador de producte, un número de sèrie i una versió.
VID	Veure "Vendor ID".

A CODI FIRMWARE COMPLERT

S'inclou en aquest capítol, a continuació, el contingut complert de l'arxiu de firmware en llenguatge assemblador:

```
;;;;;;;;;;;;;;;;;;;;;;;;
;PFC Xavier Gelada ETIEI;
;;;;;;;;;;;;;;;;;;;;;;;;


      #include <p18f14k50.inc>
      #include "usb_defs.inc"

      LIST P=PIC18F14K50            ;Automàticament és carregat l'arxiu
P18F14K50.INC
      RADIX HEX

      CONFIG CPUDIV = NOCLKDIV      ;No es divideix el clock del sistema
      CONFIG FOSC   = HS            ;Escollim el tipus d'oscil.lador extern
HS (cristall quars 12MHz)
      CONFIG PLLEN  = ON            ;L'oscil.lador es multiplicarà per 4
(acció conjunta amb SPLLEN)
      CONFIG PCLKEN = ON            ;L'oscil.lador no es pot
activar/desactivar per software
      CONFIG FCMEN  = OFF           ;Desactiva el mode "Fail-Safe monitor"
      CONFIG IESO   = OFF           ;Desactiva el mode "Oscillator
switchover"
      CONFIG PWRTEN = OFF           ;Desactiva el PoWeR-up Timer (timer
inicial a l'alimentar l'integrat)
      CONFIG BOREN  = OFF           ;Desactiva el "Brown-out Reset" tant per
hardware com per software
      CONFIG BORV   = 30            ;VBOR val 3 volts (no aplicaria perquè
BOREN=OFF)
      CONFIG WDTEN  = OFF           ;WatchDog desactivat
      CONFIG WDTPS  = 32768         ;Post escala del WatchDog = 1:32768 (no
aplica perquè WDTEN=OFF)
      CONFIG HFOFST = OFF           ;S'espera a que s'estabilitzi el clock
per entrar-lo a la CPU.
      CONFIG MCLRE  = ON            ;El pin 4 es destina a MCLRE (Reset
extern) i no a RA3 (porta e/s)
      CONFIG STVREN = ON            ;El desbordament de la pila causa RESET
      CONFIG LVP    = OFF           ;Desactiva "Low Voltage Programing"
      CONFIG BBSIZ  = OFF           ;La protecció de lectura es fa amb un
codi de 1K word (2K bytes)
      CONFIG XINST  = OFF           ;No s'activa l'adreçament indexat i el
joc d'instruccions ampliat
      CONFIG DEBUG  = OFF           ;No s'està en mode debug i RA0/RA1 són
pins de I/O
      CONFIG CP0    = OFF           ;Block 0 no protegit per codi
      CONFIG CP1    = OFF           ;Block 1 no protegit per codi
      CONFIG CPB    = OFF           ;Block Boot no protegit per codi
      CONFIG CPD    = OFF           ;Dades de la E2PROM no protegides per
codi
      CONFIG WRT0   = OFF           ;Block 0 no protegit contra escriptura
      CONFIG WRT1   = OFF           ;Block 1 no protegit contra escriptura
```

```
      CONFIG WRTB   = OFF           ;Block Boot no protegit contra
escriptura
      CONFIG WRTC   = OFF           ;Registre de configuració no protegits
contra escriptura
      CONFIG WRTD   = OFF           ;EEPROM no protegida contra escriptura
      CONFIG EBTR0  = OFF           ;Block 0 no protegit de lectures
executades des d'altres blocks
      CONFIG EBTR1  = OFF           ;Block 1 no protegit de lectures
executades des d'altres blocks
      CONFIG EBTRB  = OFF           ;Block Boot no protegit de les lectures
executades des d'altres blocks


      #define     SHOW_ENUM_STATUS  ;Si està definit encendrem un LED per
indicar diversos estats USB

      #define     SOL_SC_IN_OUT     0x01        ;Vendor-specific request per
saber si hi ha present SC
      #define     SOL_ATR_A_PC      0x02        ;Vendor-specific request per
sol.licitar l'ATR


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Definició variables RAM a utilitzar (sempre bank0);
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


CONTROL_DELAY           equ   0x00  ;Utilitzada per generar retards amb el
TIMER0
SOLLICITUD_ATR          equ   0x01  ;Utilitzada per saber si via software es
demana l'ATR
ESTAT_TARGETA           equ   0x02  ;Si val 1 indica targeta activada, Si
val 0 desactivada
SC_DINS                 equ   0x03  ;Si val 1 la targeta està dins del
lector, si val zero no
NOMBRE_BYTES_ATR  equ   0x05  ;Variable amb el comptatge de bytes d'ATR que
portem
TIME_OUT          equ   0x2C

      cblock 0x06             ;Les posicions de RAM de la 0x06 a la 0x28 (33
posicions)
      BYTES_ATR:33                  ;queden reservades per guardar l'ATR de
la targeta. Per
      endc                    ;tant la següent posició de memoria operativa
és la 0x29


      cblock 0x2D             ;Reservem 28 posicions de RAM per una sèrie de
variables
      USB_buffer_desc:4       ;necessàries pel funcionament del mòdul USB i
que tenen
      USB_buffer_data:8       ;a veure bàsicament amb la inicialització i
funcionament
      USB_error_flags               ;dels buffers dels endpoints. La següent
direcció de RAM
      USB_curr_config               ;que queda lliure és la 0x48
      USB_dev_req
      USB_address_pending
      USB_desc_ptr
      USB_bytes_left
      USB_loop_index
```

```
      USB_packet_length
      USB_USTAT
      USB_USWSTAT
      Registre_prov
      Registre_prov2
      Compta_Send_Descriptor
      Comptador_Generic
      endc

;;;;;;;;;;;;;;;;;;
;Inici de programa;
;;;;;;;;;;;;;;;;;;


      ORG H'0000'
      goto Inici_programa

      ORG H'0008'
      goto Hard_int

      ORG H'0018'
      goto Soft_int

      ORG H'001A'
Descriptor_begin

      ORG H'001C'
Device

      db    0x12, DEVICE                ;bLength, bDescriptorType
      db    0x10, 0x01              ;bcdUSB (low byte), bcdUSB (high byte)
      db    0x00, 0x00              ;bDeviceClass, bDeviceSubClass
      db    0x00, MAX_PACKET_SIZE       ;bDeviceProtocol, bMaxPacketSize
      db    0xD8, 0x04              ;idVendor (low byte), idVendor (high
byte)
      db    0x01, 0xA0              ;idProduct (low byte), idProduct (high
byte)
      db    0x01, 0x00              ;bcdDevice (low byte), bcdDevice (high
byte)
      db    0x01, 0x02              ;iManufacturer, iProduct
      db    0x00, NUM_CONFIGURATIONS    ;iSerialNumber (none),
bNumConfigurations

Configuration1

      db    0x09, CONFIGURATION         ;bLength, bDescriptorType
      db    0x12, 0x00              ;wTotalLength (low byte), wTotalLength
(high byte)
      db    NUM_INTERFACES, 0x01        ;bNumInterfaces,
bConfigurationValue
      db    0x00, 0x80              ;iConfiguration (none), bmAttributes
      db    0x32, 0x09              ;bMaxPower (100 mA), bLength (Interface1
descriptor starts here)
      db    INTERFACE, 0x00             ;bDescriptorType, bInterfaceNumber
      db    0x00, 0x00              ;bAlternateSetting, bNumEndpoints
(excluding EP0)
      db    0xFF, 0x00              ;bInterfaceClass (vendor specific class
code), bInterfaceSubClass
      db    0xFF, 0x00              ;bInterfaceProtocol (vendor specific
protocol), iInterface (none)
```

```
String0

      db    String1-String0, STRING       ;bLength, bDescriptorType
      db    0x09, 0x04              ;wLANGID[0] (low byte), wLANGID[0] (high
byte)

String1

      db    String2-String1, STRING       ;bLength, bDescriptorType
      db    'M', 0x00               ;bString
      db    'i', 0x00
      db    'c', 0x00
      db    'r', 0x00
      db    'o', 0x00
      db    'c', 0x00
      db    'h', 0x00
      db    'i', 0x00
      db    'p', 0x00
      db    ' ', 0x00
      db    'T', 0x00
      db    'e', 0x00
      db    'c', 0x00
      db    'h', 0x00
      db    'n', 0x00
      db    'o', 0x00
      db    'l', 0x00
      db    'o', 0x00
      db    'g', 0x00
      db    'y', 0x00
      db    ',', 0x00
      db    ' ', 0x00
      db    'I', 0x00
      db    'n', 0x00
      db    'c', 0x00
      db    '.', 0x00

String2

      db    Descriptor_end-String2, STRING     ; bLength, bDescriptorType
      db    'P', 0x00               ; bString
      db    'F', 0x00
      db    'C', 0x00
      db    ' ', 0x00
      db    'E', 0x00
      db    'T', 0x00
      db    'I', 0x00
      db    'E', 0x00
      db    'I', 0x00
      db    ' ', 0x00
      db    'X', 0x00
      db    'G', 0x00
      db    'e', 0x00
      db    'l', 0x00
      db    'a', 0x00
      db    'd', 0x00
      db    'a', 0x00
      db    ' ', 0x00
      db    'U', 0x00
      db    'S', 0x00
      db    'B', 0x00
      db    ' ', 0x00
```

```
      db    'F', 0x00
      db    'i', 0x00
      db    'r', 0x00
      db    'm', 0x00
      db    'w', 0x00
      db    'a', 0x00
      db    'r', 0x00
      db    'e', 0x00

Descriptor_end

Inici_programa


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Selecció del bank de RAM de treball (bank 0);
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


      movlb 0           ;Comencem per treballar amb el bank 0 de la RAM.
Els SFR estan
                  ;al bank 15 però el compilador hi afegeix l'indicador
ACCESS
                  ;automàticament (bank virtual) i per aquest motiu ometrem
aquesta
                  ;etiqueta a les instruccions. Només cal preocupar-se pels
SFRs que
                  ;queden fora del bank virtual (bank 15 de F53h a F5Fh),
els GPRs
                  ;que estan al bank 0 i la USB RAM (bank 2). En aquest cas
caldrà
                  ;usar la instrucció movlb per escollir el bank
corresponent i
                  ;afegir BANKED a la corresponent instrucció.


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuracions relatives al CLOCK del sistema;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


      bcf OSCCON, SCS0  ;Selecciono el clock extern primari (primer bit)
      bcf OSCCON, SCS1  ;Selecciono el clock extern primari (segon bit)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuracions relatives a perifèrics no utilitzats;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


      clrf ANSEL  ;Desactivem mode analògic de qualsevol pin (registre 1)
      clrf ANSELH ;Desactivem mode analògic de qualsevol pin (registre 2)

      clrf CM1CON0      ;Desactivem sortides del comparador (registre 1)
      clrf CM2CON0      ;Desactivem sortides del comparador (registre 2)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuració de RC0 com a sortida per alimentar la SmartCard;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
      bcf TRISC, TRISC7 ;Definim RC7 com a sortida per poder alimentar la
SmartCard
      bcf LATC, LATC7         ;De moment no alimentem la targeta


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuració de RC6 com a sortida per resetejar la SmartCard;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


      bcf TRISC, TRISC6 ;Definim RC6 com a sortida per poder resetejar la
SmartCard


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuracions del PWM/TMR2 per generar el clock de la SmartCard;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


      clrf TMR2         ;Carreguem 0 al registre TMR2 (TMR2=0)
      clrf T2CON        ;Configurem el timer 2 amb pre/post escalat = 1 i
el desactivem (T2CON=0)

      movwf PR2         ;segons fòrmula datasheet (PR2=3)

      movlw 2                 ;Carreguem 2 a CCPR1L (definim duty cicle
clock Smartcard = 50%
      movwf CCPR1L            ;segons fòrmula datasheet (CCPR1L=2)

      movlw 0x01        ;Senyal modulada (clock SC) únicament per PIN5
(sortida P1A) (PSTRCON=0x01)
      movwf PSTRCON

      bcf TRISC, TRISC5 ;Per fer P1A operatiu també cal definir RC5 com a
sortida (PIN5) (TRISC5=0)

      clrf CCP1CON            ;Mantenim el mòdul ECCP apagat (SmartCard
sense clock) (CCP1CON=0)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuració del TIMER0 que s'utilitza per fer delays de diferent durada;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


      movlw 0x07              ;Treballem en 16 bits / Treballem com a timer
(no com a comptador)
                                    ;El timer des de zero desborda als
1.398s (cada increment = 21.33us)
      clrf CONTROL_DELAY, BANKED    ;Posem a zero la variable CONTROL_DELAY
que fa anar el TIMER0


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuració relativa a la EUSART;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


      movlw 0x05        ;Carrego a SPBRGH:SPBRG el valor en binari 1487d
que permet que el generador
```

```
      movwf SPBRGH            ;de BaudRate de la EUSART generi el Baudrate
necessari segons el clock que
      movlw 0xCF        ;s'aplica a la SmartCard i les especificacions de
la ISO7816. La forma de
      movwf SPBRG       ;calcular el valor ve donada pel datasheet de
l'integrat.

      bcf RCSTA, CREN         ;Desactivem mòdul receptor
      bcf RCSTA, SPEN         ;Desactivem la EUSART
      bcf TXSTA, SYNC         ;Activem funcionament Asíncron del la UART
      bcf LATB, LATB7         ;Portem TX a 0 actuant directament sobre el
latch
      bsf TRISB, TRISB7 ;Configurem TX com a sortida
      bsf TRISB, TRISB5 ;Configurem RX com a entrada assignant un 1 al
corresponent valor TRIS
      bsf TXSTA, TX9          ;Activem novè bit de transmissió
      bsf RCSTA, RX9          ;Activem novè bit de recepció
      bsf TXSTA, BRGH         ;Activem mode High BaudRate
      bcf BAUDCON, DTRXP      ;Polaritat Rx segons ISO7816
      bcf BAUDCON, CKTXP      ;Polaritat TX segons ISO7816


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuració dels pins de l'integrat amb LEDs connectats;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


      bcf TRISC, TRISC2 ;Definim RC2 com a sortida per poder encendre un
LED
      bcf LATC, LATC2         ;De moment el LED està apagat

      bcf TRISC, TRISC3 ;Definim RC3 com a sortida per poder encendre un
LED
      bcf LATC, LATC3         ;De moment el LED està apagat

      bcf TRISC, TRISC4 ;Definim RC4 com a sortida per poder encendre un
LED
      bcf LATC, LATC4         ;De moment el LED està apagat

      bcf TRISB, TRISB4 ;Definim RB4 com a sortida per poder encendre un
LED
      bcf LATB, LATB4         ;De moment el LED està apagat

      bcf TRISB, TRISB6 ;Definim RB6 com a sortida per poder encendre un
LED
      bcf LATB, LATB6         ;De moment el LED està apagat


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Configuració relativa a les interrupcions;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


      bcf INTCON3, INT1IF     ;Baixem el flag de INT1
      bcf INTCON, INT0IF      ;Baixem el flag de INT0
      bcf INTCON, TMR0IF      ;Baixem el flag de TMR0

      bsf RCON, IPEN          ;Mode de funcionament d'interrupcions amb
prioritats
      bcf INTCON, GIEH  ;De moment, desactivem interrupcions prioritat alta
```

```
      bcf INTCON, GIEL  ;De moment, desactivem interrupcions prioritat
baixa

      bcf INTCON, TMR0IE      ;Desactivem interrupció del TIMER0
      bcf INTCON2, TMR0IP     ;Definim prioritat Low pel TIMER0

      bcf PIE1, RCIE          ;Desactivem interrupció RX (mòdul receptor
EUSART)
      bcf IPR1, RCIP          ;Definim prioritat Low per RX (mòdul receptor
EUSART)

      bsf TRISC, TRISC0 ;Defineixo el pin com entrada
      bsf INTCON, INT0IE      ;Activo interrupció externa INT0 (prioritat
alta per defecte)
      bsf INTCON2, INTEDG0    ;Configuro interrupció en flanc actiu de
pujada (introducció SC)

      bsf TRISC, TRISC1 ;Defineixo el pin com entrada
      bsf INTCON3, INT1IE     ;Activo interrupció externa INT1
      bsf INTCON3, INT1IP     ;Definim prioritat High d'interrupció
      bcf INTCON2, INTEDG1    ;Configuro interrupció en flanc actiu de
baixada (extracció SC)

      clrf UIE          ;Desactivem totes les interrupcions USB
      clrf UIR          ;Posem a zero tots els flags USB


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;ASSIGNACIO VALOR INICIAL VARIABLES;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


      clrf SOLLICITUD_ATR, BANKED   ;Quan val 1 significa que via software
s'ha demanat l'ATR
      clrf ESTAT_TARGETA, BANKED    ;Posem a 0 la variable que indica si
està activada o no
      clrf SC_DINS, BANKED          ;Posem a 0 perquè no hi ha targeta
inicialment i s'hi n'hi ha
                                    ;s'ha de treure i tornar a posar.

      clrf NOMBRE_BYTES_ATR, BANKED ;Poso a 0 el comptador de bytes rebuts
d'ATR

      clrf TIME_OUT, BANKED         ;Assignem zero inicialment a la variable
per indicar time-out

      clrf USB_error_flags, BANKED  ;Posem a zero la variable per indicar
errors de USB


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;BUCLE PRINCIPAL DE PROGRAMA;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Bucli_Primer_De_Programa

      call Delay_2ms

      call Init_USB                 ;Inicialitzem els registres USB i el SIE

Configuracio_Periferic
```

```
      call Service_USB        ;Servim els requests USB fins que el host
configuri el perifèric

      movlw CONFIG_STATE
      cpfseq USB_USWSTAT, BANKED    ;Salta si els valors són iguals (si
USB_USWSTAT == W)
      goto Configuracio_Periferic   ;Diferents (el host encara no ha
configurat el perifèric)

      bsf INTCON, GIEH        ;Activem interrupcions prioritat alta per
detectar la targeta
      bsf INTCON, GIEL        ;Activem interrupcions prioritat baixa per
detectar la targeta

Bucli_Segon_De_Programa

      call Service_USB        ;Servim els requeriments USB infinitament

      goto Bucli_Segon_De_Programa


;;;;;;;;;;;;;;;;;;;;;;;;;
;ZONA DE RUTINES GENERALS;
;;;;;;;;;;;;;;;;;;;;;;;;;


;----------------------------------------


Configuracio_Inicial

      bcf INTCON3, INT1IF     ;Poso a zero el flag de INT1
      bcf LATC, LATC7         ;No alimentem la targeta
      bcf LATC, LATC6         ;No resetegem la targeta
      clrf TMR2         ;Carreguem 0 al registre TMR2 (TMR2=0)
      clrf T2CON        ;Configurem el timer 2 amb pre/post escalat = 1 i
el desactivem (T2CON=0)
      clrf CCP1CON            ;Mantenim el mòdul ECCP apagat (SmartCard
sense clock) (CCP1CON=0)

      movlw 0x07        ;Treballem en 16 bits / Treballem com a timer (no
com a comptador)
      movwf T0CON       ;Activem pre-escalat / Assignem un valor de pre-
escalat de 256
                        ;El timer des de zero desborda als 1.398s (cada
increment = 21.33us)

      bcf RCSTA, CREN         ;Desactivem mòdul receptor
      bcf RCSTA, SPEN         ;Desactivem la EUSART
      bcf TXSTA, SYNC         ;Activem funcionament Asíncron del la UART
      bcf LATB, LATB7         ;Portem TX a 0 actuant directament sobre el
latch
      bsf TRISB, TRISB7 ;Configurem TX com a sortida
      bsf TRISB, TRISB5 ;Configurem RX com a entrada assignant un 1 al
corresponent valor TRIS
      bsf TXSTA, TX9          ;Activem novè bit de transmissió
      bsf RCSTA, RX9          ;Activem novè bit de recepció
      bsf TXSTA, BRGH         ;Activem mode High BaudRate
      bsf BAUDCON, BRG16      ;Activem BaudRate 16 bits
      bcf BAUDCON, DTRXP      ;Polaritat Rx segons ISO7816
```

```
      bcf BAUDCON, CKTXP      ;Polaritat TX segons ISO7816

      bcf INTCON3, INT1IF     ;Baixem el flag de INT1
      bcf INTCON, INT0IF      ;Baixem el flag de INT0
      bcf INTCON, TMR0IF      ;Baixem el flag de TMR0

      bcf INTCON, TMR0IE      ;Desactivem interrupció del TIMER0
      bcf INTCON2, TMR0IP     ;Definim prioritat Low pel TIMER0


      clrf SOLLICITUD_ATR, BANKED   ;Quan val 1 significa que via software
s'ha demanat l'ATR
      clrf ESTAT_TARGETA, BANKED    ;Posem a 0 la variable que indica si
està activada o no
      clrf SC_DINS, BANKED          ;Posem a 0 perquè no hi ha targeta
inicialment i s'hi n'hi ha
                                    ;s'ha de treure i tornar a posar.

      clrf NOMBRE_BYTES_ATR, BANKED ;Poso a 0 el comptador de bytes rebuts
d'ATR

      return


;---------------------------------------


;;;;;;;;;;;;;;;;;;;;;;;;;;
;ZONA DE RUTINES SMARTCARD;
;;;;;;;;;;;;;;;;;;;;;;;;;;


;---------------------------------------


Obtenir_ATR

      btfss SC_DINS, 0, BANKED      ;Salta si SC_DINS val 1 (hi ha targeta)
      goto RETORN_00                ;No hi ha targeta

      btfsc SOLLICITUD_ATR, 0, BANKED    ;Salta si no hi ha un ATR vàlid
demanat
      goto HIHA_ATR_ANTERIOR        ;Ja hi ha ATR demanat

                                  ;No hi ha ATR anterior i comencem activació SC

      call Delay_2ms                ;Esperem 10 mil.lisegons
      call Delay_2ms
      call Delay_2ms
      call Delay_2ms
      call Delay_2ms

      bcf LATC, LATC6               ;Comencem la seqüència d'obtenció de
l'ATR. Línia RST nivell baix

      call Delay_2ms                ;Esperem 2 mil.lisegons

      bsf LATC, LATC7               ;Alimentem la targeta activant RC7

      call Delay_2ms                ;Esperem 2 milisegons
```

```
      call Delay_2ms                ;Esperem 2 milisegons

      movlw 0x0C              ;Configurem el mòdul PWM fer generar el clock
de la SmartCard
      movwf CCP1CON                 ;carregant el valor 0x0C que fa
treballar per separat les P1X

      bsf T2CON, TMR2ON       ;Activem el TMR2 que implica activar el clock
de la targeta

      call Delay_2ms                ;Esperem 2 milisegons

      movlw 0xFC              ;Carreguem a TMR0H:TMR0L el valor FCF0h que
fins arribar a
      movwf TMR0H             ;FFFFh farà 783 comptatges equivalents a 50000
clocks a 3MHz que al
      movlw 0xF0              ;seu temps equivalen a 16,7ms amb el pre-
escalat de 256 passos amb
      movwf TMR0L             ;que tenim configurat el TIMER0. Cada
comptatge equival a 21.33us

      bcf INTCON, TMR0IF            ;Poso a zero el flag d'interrupció del
TIMER0
      bsf T0CON, TMR0ON       ;Activo el TMR0 perquè comenci a comptar

      bsf LATC, LATC6               ;Línia RST nivell alt

Bucli_ATR_Global

      btfsc PIR1, RCIF        ;Salta si RCIF=0
      goto Fora_ATR_Global
      btfsc INTCON, TMR0IF          ;Salta si TMR0IF=0
      goto Fora_ATR_Global
      goto Bucli_ATR_Global

Fora_ATR_Global                     ;Un dels dos flags ha saltat

      bcf T0CON, TMR0ON       ;Apago el TMR0

      btfsc PIR1, RCIF        ;Salta si RCIF=0
      goto Inici_ATR                ;Hi ha caràcter i per tant comencem la
recepció del ATR
      btfsc INTCON, TMR0IF          ;Salta si TMR0IF=0
      goto ATR_TimeOut        ;Si val 1, no hem rebut ATR i el temps ha
passat

Inici_ATR

      clrf NOMBRE_BYTES_ATR, BANKED

      movlw 0x00              ;Carrego la primera posició al punter de bytes
d'ATR. Hi
      movwf FSR1H             ;escriurem usant INDF1
      movlw 0x06
      movwf FSR1L

Bucli_ATR

      movf RCREG, 0                 ;Guardem el byte a la posició actual de
memòria a la que està
```

```
      movwf INDF1             ;apuntant el punter (FSR0L)

      incf FSR1L              ;Incremento el punter
      incf NOMBRE_BYTES_ATR, BANKED ;Incremento el comptador de bytes

      movlw 0x21              ;Comprovo si el comptador de bytes ja ha
arribat a 33
      cpfseq NOMBRE_BYTES_ATR, BANKED    ;Si hi ha arribat, saltem
      goto Continua_ATR       ;Encara no hem arribat al límit de caràcters
d'ATR
      goto Desactivacio_i_Sortida  ;Hem arribat als 33 càracters d'ATR i
sortim del procés

Continua_ATR

      movlw 0xDB              ;Carreguem a TMR0H:TMR0L el valor DB5Fh que
fins arribar a
      movwf TMR0H             ;FFFFh farà 9376 comptatges equivalents a
200ms amb el pre-escalat
      movlw 0x5F              ;de 256 passos amb que tenim configurat el
TIMER0. Cada comptatge
      movwf TMR0L             ;equival a 21.33Us

      bcf INTCON, TMR0IF            ;Poso a zero el flag d'interrupció del
TIMER0
      bsf T0CON, TMR0ON       ;Activo el TMR0 que es posa a comptar

Bucli_ATR_Byte

      btfsc PIR1, RCIF        ;Salta si RCIF=0
      goto Fora_ATR_Byte
      btfsc INTCON, TMR0IF          ;Salta si TMR0IF=0
      goto Fora_ATR_Byte
      goto Bucli_ATR_Byte

Fora_ATR_Byte                       ;Un dels dos flags ha saltat

      bcf T0CON, TMR0ON       ;Apago el TMR0

      btfsc PIR1, RCIF        ;Salta si RCIF=0
      goto Bucli_ATR                ;Hi ha caràcter i per tant tornem a
començar el procés
      btfsc INTCON, TMR0IF          ;Salta si TMR0IF=0
      goto ATR_Byte_TimeOut         ;Si val 1, no hem rebut byte d'ATR i el
temps ha passat


ATR_TimeOut                   ;Time_OUT esperant l'ATR (cap caràcter rebut)

      movlw 0xFF              ;Cal retornar FFh al host
      movwf BYTES_ATR, BANKED
      movlw 0x01
      movwf NOMBRE_BYTES_ATR, BANKED

      goto Desactivacio_i_Sortida


ATR_Byte_TimeOut              ;Time_OUT esperant un byte (potser ATR
completat o erroni)
```

```
      movlw 0x04              ;Comprovem que la longitud del ATR és mínim 4
bytes i sinó
      cpfslt NOMBRE_BYTES_ATR, BANKED     ;retornem error
      goto Desactivacio_i_Sortida   ;No salta i per tant ATR té una longitud
de 4 o més i sortim

      movlw 0xEF              ;Salta (ATR curt, s'ha donat un error!)
      movwf BYTES_ATR, BANKED       ;Cal retornar EFh al host
      movlw 0x01
      movwf NOMBRE_BYTES_ATR, BANKED


Desactivacio_i_Sortida

                        ;Desactivem la targeta
      bcf LATC, LATC6         ;Línia RST de la targeta a nivell baix

      call Delay_1ms          ;Esperem 1 mil.lisegon

      clrf CCP1CON            ;Borrem la configuració del PWM per tancar les
sortides P1X

      bcf T2CON, TMR2ON ;Desactivem el TMR2 que implica activar el clock de
la targeta
      clrf TMR2         ;Posem a 0 el TMR2 per la següent vegada que
l'activem

      bcf RCSTA, CREN         ;Desactivo mòdul receptor
      bcf RCSTA, SPEN         ;Desactivem la EUSART
      bcf LATB, LATB7         ;Portem TX a 0 actuant directament sobre el
latch

      bcf LATC, LATC7         ;Tallem l'alimentació de la targeta a RC7

      return

RETORN_00

      movlw 0x00
      movwf BYTES_ATR, BANKED
      movlw 0x01
      movwf NOMBRE_BYTES_ATR, BANKED

HIHA_ATR_ANTERIOR

      return


;---------------------------------------


;---------------------------------------


Delay_05s


      movlw 0xA4              ;Carreguem a TMR0H:TMR0L el valor FFA2h que
fins arribar a
      movwf TMR0H             ;FFFFh farà 93 comptatges equivalents a 2 ms
amb el pre-escalat
```

```
      movlw 0x6E              ;de 256 passos amb que tenim configurat el
TIMER0. Cada comptatge
      movwf TMR0L             ;equival a 21.33Us

      bcf INTCON, TMR0IF            ;Poso a zero el flag d'interrupció del
TIMER0
      bsf T0CON, TMR0ON       ;Activo el TMR0 que es posa a comptar

Bucli07

      btfss INTCON, TMR0IF          ;Salta si TMR0IF=1
      goto Bucli07

      bcf INTCON, TMR0IF            ;Poso a zero el flag d'interrupció del
TIMER0
      bcf T0CON, TMR0ON       ;Apago el TMR0 perquè no continuï comptant

      return


;---------------------------------------


;---------------------------------------


Delay_2ms


      movlw 0xFF              ;Carreguem a TMR0H:TMR0L el valor FFA2h que
fins arribar a
      movwf TMR0H             ;FFFFh farà 93 comptatges equivalents a 2 ms
amb el pre-escalat
      movlw 0xA2              ;de 256 passos amb que tenim configurat el
TIMER0. Cada comptatge
      movwf TMR0L             ;equival a 21.33Us

      bcf INTCON, TMR0IF            ;Poso a zero el flag d'interrupció del
TIMER0
      bsf T0CON, TMR0ON       ;Activo el TMR0 que es posa a comptar

Bucli00

      btfss INTCON, TMR0IF          ;Salta si TMR0IF=1
      goto Bucli00

      bcf INTCON, TMR0IF            ;Poso a zero el flag d'interrupció del
TIMER0
      bcf T0CON, TMR0ON       ;Apago el TMR0 perquè no continuï comptant

      return


;---------------------------------------


;---------------------------------------


Delay_1ms
```

```
      movlw 0xFF              ;Carreguem a TMR0H:TMR0L el valor FFD0h que
fins arribar a
      movwf TMR0H             ;FFFFh farà 47 comptatges equivalents a 1 ms
amb el pre-escalat
      movlw 0xD0              ;de 256 passos amb que tenim configurat el
TIMER0. Cada comptatge
      movwf TMR0L             ;equival a 21.33Us

      bcf INTCON, TMR0IF            ;Poso a zero el flag d'interrupció del
TIMER0
      bsf T0CON, TMR0ON       ;Activo el TMR0 que es posa a comptar

Bucli04

      btfss INTCON, TMR0IF          ;Salta si TMR0IF=1
      goto Bucli04

      bcf INTCON, TMR0IF            ;Poso a zero el flag d'interrupció del
TIMER0
      bcf T0CON, TMR0ON       ;Apago el TMR0 perquè no continuï comptant

      return


;---------------------------------------


;;;;;;;;;;;;;;;;;;;;
;ZONA DE RUTINES USB;
;;;;;;;;;;;;;;;;;;;;


;---------------------------------------


Init_USB

      clrf UIE                ;Desactiva totes les fonts d'interrupció
originades pel mòdul USB
      clrf UIR                ;Reseteja tots els flags indicadors de
interrupcions USB

      movlw 0x08              ;Activem el mòdul USB de l'integrat.
      movwf UCON

      clrf USB_curr_config, BANKED  ;Posem a zero la variable
USB_curr_config
      clrf USB_USWSTAT, BANKED      ;Posem a zero USB_USWSTAT

      movlw 0x00
      movwf USB_device_status, BANKED     ;Posem a 0 la variable
USB_device_status (desactivem "self powered" i
                        ;desactivem "remote wakeup")

      movlw NO_REQUEST        ;Carreguem el valor 0xFF (veure arxiu
usb_defs.inc) a la variable
      movwf USB_dev_req, BANKED     ;USB_dev_req per indicar que no hi ha
peticions en curs
```

```
Esperant_SEO                  ;Ens fiquem en un bucli esperant que el bit
SEO del registre UCON

      btfsc UCON,SE0                ;deixi de valdre 1, cosa que indicarà
que s'ha deixat d'estar en
      goto Esperant_SEO       ;situació "single-ended-zero" (ja està
"attached" i "powered")

      #ifdef SHOW_ENUM_STATUS       ;Si hi ha definida l'etiqueta
SHOW_ENUM_STATUS...
                              ;Posem RC2 a 1 per encendre el LED de
POWERED_STATUS
      bcf LATB, LATB4               ;Apaguem LED que no toca
      bcf LATB, LATB6               ;Apaguem LED que no toca
      bcf LATC, LATC3               ;Apaguem LED que no toca
      bcf LATC, LATC4               ;Apaguem LED que no toca
      bsf LATC, LATC2               ;Encenem LED que toca

      #endif

      return


;---------------------------------------


;---------------------------------------


Service_USB

      btfss UIR, UERRIF       ;Comprova si s'ha donat una interrupció per
error USB
      goto No_UIR_UERRIF            ;No s'ha donat, per tant saltem
      movlb 0x0F                    ;Seleccionem el bank de memòria 15 (SFR
fora del bank virtual)
      clrf UEIR, BANKED       ;S'ha donat, desactivem les interrupcions per
error USB
      movlb 0
      goto Final_Service_USB        ;Sortim de la comprovació
d'interrupcions USB

No_UIR_UERRIF

      btfss UIR, SOFIF        ;Comprova si s'ha donat una interrupció TOKEN
de Start_of_frame
      goto No_UIR_SOFIF       ;No s'ha donat, per tant saltem
      bcf UIR, SOFIF                ;S'ha donat, posem a zero el flag
d'aquesta interrupció
      goto Final_Service_USB        ;Sortim de la comprovació
d'interrupcions USB

No_UIR_SOFIF

      btfss UIR, IDLEIF       ;Comprova si s'ha donat una interrupció per
detecció d'estat IDLE
      goto No_UIR_IDLEIF            ;No s'ha donat, per tant saltem
      bcf UIR, IDLEIF               ;S'ha donat, posem a zero el flag
d'aquesta interrupció
      bsf UCON, SUSPND        ;Entrem en mode suspensió
```

```
      #ifdef SHOW_ENUM_STATUS       ;Si hi ha definida l'etiqueta
SHOW_ENUM_STATUS...
                              ;Posem RC4 a 1 per encendre el LED de
SUSPEND_STATUS
      bcf LATB, LATB4               ;Apaguem LED que no toca
      bcf LATB, LATB6               ;Apaguem LED que no toca
      bcf LATC, LATC3               ;Apaguem LED que no toca
      bcf LATC, LATC2               ;Apaguem LED que no toca
      bsf LATC, LATC4               ;Encenem LED que toca

      #endif

      goto Final_Service_USB        ;Sortim de la comprovació
d'interrupcions USB

No_UIR_IDLEIF

      btfss UIR, ACTVIF       ;Comprova si s'ha donat interrupció al
detectar activitat al bus
      goto No_UIR_ACTVIF            ;No s'ha donat, per tant saltem
      bcf UIR, ACTVIF               ;S'ha donat, posem a zero el flag
d'aquesta interrupció
      bcf UCON, SUSPND        ;Sortim del mode suspensió

      #ifdef SHOW_ENUM_STATUS       ;Si hi ha definida l'etiqueta
SHOW_ENUM_STATUS...

                              ;Apaguem tots els LEDs
      bcf LATB, LATB4               ;Apaguem LED que no toca
      bcf LATB, LATB6               ;Apaguem LED que no toca
      bcf LATC, LATC3               ;Apaguem LED que no toca
      bcf LATC, LATC4               ;Apaguem LED que no toca
      bcf LATC, LATC2               ;Apaguem LED que no toca

      movlw POWERED_STATE
      movwf Registre_prov, BANKED
      movf USB_USWSTAT, 0, BANKED
      cpfseq Registre_prov, BANKED  ;Comparem els dos valors i saltem si són
iguals
      goto Cas_DEFAULT        ;Són diferents. Anem al següent cas
      bsf LATC, LATC2               ;Encenem LED que toca
      goto Estat_Assignat

Cas_DEFAULT
      movlw DEFAULT_STATE
      movwf Registre_prov, BANKED
      movf USB_USWSTAT, 0, BANKED
      cpfseq Registre_prov, BANKED  ;Comparem els dos valors i saltem si són
iguals
      goto Cas_ADDRESS        ;Són diferents. Anem al següent cas
      bsf LATB, LATB4               ;Encenem LED que toca
      goto Estat_Assignat

Cas_ADDRESS
      movlw ADDRESS_STATE
      movwf Registre_prov, BANKED
      movf USB_USWSTAT, 0, BANKED
      cpfseq Registre_prov, BANKED  ;Comparem els dos valors i saltem si són
iguals
      goto Cas_CONFIG               ;Són diferents. Anem al següent cas
      bsf LATB, LATB6               ;Encenem LED que toca
```

```
      goto Estat_Assignat

Cas_CONFIG              ;Ja només pot ser aquest cas, per tant assigno el
valor corresponent
      bsf LATC, LATC3              ;Encenem LED que toca

Estat_Assignat
      #endif


No_UIR_ACTVIF

      btfss UIR, STALLIF           ;Comprova si s'ha donat una interrupció
STALL en un handshake
      goto No_UIR_STALLIF          ;No s'ha donat, per tant saltem
      bcf UIR, STALLIF        ;S'ha donat, borrem el corresponent flag

      goto Final_Service_USB       ;Sortim de la comprovació
d'interrupcions USB

No_UIR_STALLIF

      btfss UIR, URSTIF       ;Comprova si s'ha donat una interrupció per
RESET USB
      goto No_UIR_URSTIF           ;No s'ha donat, per tant saltem
      clrf USB_curr_config, BANKED  ;S'ha donat i s'haurà carregat 00h al
registre UADDR

      bcf UIR, TRNIF               ;Posem a 0 el flag TRNIF quatre vegades
per buidar la FIFO de USTAT
      bcf UIR, TRNIF
      bcf UIR, TRNIF
      bcf UIR, TRNIF

      movlb 0x0F
      clrf UEP0, BANKED       ;Borrem tots els registres de control per
tancar tots els Endpoints
      clrf UEP1, BANKED
      clrf UEP2, BANKED
      clrf UEP3, BANKED
      clrf UEP4, BANKED
      clrf UEP5, BANKED
      clrf UEP6, BANKED
      clrf UEP7, BANKED

      movlb 2

      movlw MAX_PACKET_SIZE
      movwf BD0OBC, BANKED              ;Carrego al BD byte count la
quantitat de bytes per packet
      movlw low USB_Buffer              ;S'assigna un buffer a EP0 OUT
      movwf BD0OAL, BANKED
      movlw high USB_Buffer
      movwf BD0OAH, BANKED              ;S'assigna adreça al buffer
      movlw 0x88                   ;Bit UOWN a 1 (el mòdul USB pot escriure
al registre)
      movwf BD0OST, BANKED
      movlw low (USB_Buffer+MAX_PACKET_SIZE)    ;S'assigna un buffer a EP0
IN
      movwf BD0IAL, BANKED
      movlw high (USB_Buffer+MAX_PACKET_SIZE)
```

```
      movwf BD0IAH, BANKED              ;S'assigna adreça al buffer
      movlw 0x08                    ;Bit UOWN a 0 (el micro pot escriure al
registre)
      movwf BD0IST, BANKED
      movlb 0x0F
      clrf UADDR, BANKED                ;Posem la Adreça USB a 0
      clrf UIR                      ;Posem a 0 tots els flags d'interrupció
USB
      movlw ENDPT_CONTROL
      movwf UEP0, BANKED                ;Configurem EP0 com a endpoint de
control amb ACK
      movlw 0xFF                    ;Activem totes les interrupcions USB
      movwf UEIE, BANKED
      movlb 0
      movlw DEFAULT_STATE
      movwf USB_USWSTAT, BANKED
      movlw 0x00
      movwf USB_device_status, BANKED          ;Configurem el device com
"self powered" i no permetem
                              ;que es doni "remote wakeup"
      #ifdef SHOW_ENUM_STATUS
                              ;Posem RC2 a 1 per encendre el LED de
POWERED_STATUS
      bsf LATB, LATB4               ;Encenem LED que toca
      bcf LATB, LATB6               ;Apaguem LED que no toca
      bcf LATC, LATC3               ;Apaguem LED que no toca
      bcf LATC, LATC4               ;Apaguem LED que no toca
      bcf LATC, LATC2               ;Apaguem LED que no toca

      #endif

      goto Final_Service_USB            ;Sortim de la comprovació
d'interrupcions USB

No_UIR_URSTIF

      btfss UIR, TRNIF          ;Comprova si s'ha donat una interrupció per
transacció completada
      goto Final_Service_USB        ;No s'ha donat, per tant saltem al final

      movlw high BD0OST
      movwf FSR0H
      movf USTAT, 0
      andlw 0x7C                ;Enmascarem els bits 0, 1 i 7 de USTAT
      movwf FSR0L
      movf POSTINC0, 0
      movwf USB_buffer_desc, BANKED
      movf POSTINC0, 0
      movwf USB_buffer_desc+1, BANKED
      movf POSTINC0, 0
      movwf USB_buffer_desc+2, BANKED
      movf POSTINC0, 0
      movwf USB_buffer_desc+3, BANKED
      movf USTAT, 0
      movwf USB_USTAT, BANKED       ;Guardem el "USB status register"
      bcf UIR, TRNIF                ;Borrem el bit senyalitzador
d'interrupció TRNIF

      #ifdef SHOW_ENUM_STATUS

      andlw 0x18                ;Extraiem els bits de EP
```

```
      movwf Registre_prov2, BANKED

      movlw EP0
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED  ;Comparem els dos valors i saltem si són
iguals
      goto Cas_EP1                  ;Són diferents. Anem al següent cas
      movlw 0x20              ;Són iguals, poso a WREG el valor d'aquest
estat
      goto EPX_Assignat

Cas_EP1

      movlw EP1
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED  ;Comparem els dos valors i saltem si són
iguals
      goto Cas_EP2                  ;Són diferents. Anem al següent cas
      movlw 0x40              ;Són iguals, poso a WREG el valor d'aquest
estat
      goto EPX_Assignat

Cas_EP2

      movlw EP2
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED  ;Comparem els dos valors i saltem si són
iguals
      goto EPX_Assignat       ;Són diferents. Anem al següent cas
      movlw 0x80              ;Són iguals, poso a WREG el valor d'aquest
estat

EPX_Assignat

      #endif

      andlw 0x3C
      movwf Registre_prov2, BANKED  ;Extraiem els bits de PID i ho guardem a
Registre_prov2

      movlw TOKEN_SETUP
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED  ;Comparem els dos valors i saltem si són
iguals
      goto Cas_TOKEN_IN       ;Són diferents. Anem al següent cas
      call ProcessSetupToken        ;Són iguals, invoco rutina per processar
el TOKEN OUT
      goto TOKEN_Assignat

Cas_TOKEN_IN

      movlw TOKEN_IN
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED  ;Comparem els dos valors i saltem si són
iguals
      goto Cas_TOKEN_OUT            ;Són diferents. Anem al següent cas
```

```
      call ProcessInToken           ;Són iguals, invoco rutina per processar
el TOKEN IN
      goto TOKEN_Assignat

Cas_TOKEN_OUT

      movlw TOKEN_OUT
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED  ;Comparem els dos valors i saltem si són
iguals
      goto TOKEN_Assignat           ;Són diferents. Anem al final
      call ProcessOutToken          ;Són iguals, invoco rutina per processar
el TOKEN OUT

TOKEN_Assignat


      btfss USB_error_flags, 0, BANKED    ;Comprovem si el bit 0 de
USB_error_flags val 1
      goto Final_Service_USB              ;Val 0, no s'ha donat error i per
tant saltem per sortir
      movlb 2
      movlw MAX_PACKET_SIZE               ;Val 1 i per tant s'ha donat error
      movwf BD0OBC, BANKED                ;Tornem a posar-ho apunt per rebre
el següent TOKEN SETUP
      movlw 0x84
      movwf BD0IST, BANKED
      movwf BD0OST, BANKED                ;Fem sortir un STALL per EP0
      movlb 0

Final_Service_USB             ;Sortim de la comprovació d'interrupcions USB

      return


;---------------------------------------


;---------------------------------------


Descriptor

      movlw upper Descriptor_begin  ;Obtinc la part més alta de la direcció
de programa de
      movwf TBLPTRU                 ;"Descriptor_begin" i la passo a
TBLPTRU. Faig el mateix amb la
      movlw high Descriptor_begin   ;part alta i la copio a TBLPTRH.
      movwf TBLPTRH
      movlw low Descriptor_begin    ;Poso a W la part baixa de la direcció
de programa de
      addwf USB_desc_ptr, 0, BANKED ;"Descriptor_begin" i li sumo
USB_desc_ptr a mode d'offset.

      btfss STATUS,C                ;Comprovem el carry del registre STATUS
      goto No_C_Tblptr        ;Si val zero no hi ha hagut carry i sortim
      incf TBLPTRH, 1               ;Hi ha carry per tant incremento TBLPTRH
      btfsc STATUS,Z                ;Comprovo el bit Zero del registre
STATUS
```

```
      incf TBLPTRU, 1               ;Si s'ha donat un zero cal incrementar
TBLPTRU

No_C_Tblptr             ;Sortida en tots dos casos

      movwf TBLPTRL                 ;Passo el valor de WREG (part baixa de
la direcció de programa de
      tblrd*                        ;"Descriptor_begin" a TBLPTRL. Agafo el
valor que conté la direcció
      movf TABLAT, 0                ;de la memòria de programa a la que
apunta TBLPTR i la poso a TABLAT
                        ;i de TABLAT la passo a WREG.
      return


;---------------------------------------


;---------------------------------------


ProcessSetupToken

      movf USB_buffer_desc+ADDRESSH, 0, BANKED ;Carreguem per
direccionament indirecte els FSRs amb
      movwf FSR0H                       ;l'adreça de USB_buffer_desc i
anem carregant les 8
      movf USB_buffer_desc+ADDRESSL, 0, BANKED ;posicions de
USB_buffer_data
      movwf FSR0L
      movf POSTINC0, 0
      movwf USB_buffer_data, BANKED
      movf POSTINC0, 0
      movwf USB_buffer_data+1, BANKED
      movf POSTINC0, 0
      movwf USB_buffer_data+2, BANKED
      movf POSTINC0, 0
      movwf USB_buffer_data+3, BANKED
      movf POSTINC0, 0
      movwf USB_buffer_data+4, BANKED
      movf POSTINC0, 0
      movwf USB_buffer_data+5, BANKED
      movf POSTINC0, 0
      movwf USB_buffer_data+6, BANKED
      movf POSTINC0, 0
      movwf USB_buffer_data+7, BANKED
      movlw MAX_PACKET_SIZE
      movwf BD0OBC, BANKED              ;Inicialitzem el byte counter
      movlw 0x08
      movwf BD0IST, BANKED              ;Desfem qualsevol request pedent
      movlb 0

      btfss USB_buffer_data+bmRequestType, 7, BANKED ;Comprovem si el bit 7
de la variable val 1
      goto No_bit7_Zero                 ;Val 0. Saltem al següent
condicional
      movlw 0x88                        ;Val 1. Hi carreguem el valor 0x88
i acabem
      goto Final_Setup_TOKEN                  ;Sortim de la comprovació
d'interrupcions USB
```

```
No_bit7_Zero

      movlw 0x00
      cpfseq USB_buffer_data+wLength, BANKED         ;Mirem si el registre
és zero i saltem si ho és
      goto Si_UnDelsDos_If                      ;No val zero. Ja podem
carregar-hi 0xC8
      cpfseq USB_buffer_data+wLengthHigh, BANKED     ;Si que val zero,
comparo l'altra variable per si és zero
      goto Si_UnDelsDos_If                      ;No és zero. Ja podem
carregar-hi 0xC8
      movlw 0x88                           ;Si que val zero, per tant
carreguem 0x88 i sortim
      goto Final_Setup_TOKEN

Si_UnDelsDos_If
      movlw 0xC8

Final_Setup_TOKEN

      movlb 2
      movwf BD0OST, BANKED                      ;Retornem el control al
mòdul USB de EP0 OUT UOWN
      movlb 0                                   ;DATA0/DATA1 packet segons
request type

      bcf UCON, PKTDIS                     ;Posem a zero el bit que desactiva
els packets
      movlw NO_REQUEST
      movwf USB_dev_req, BANKED                 ;Borrem el device request
que hi ha en procés
      movf USB_buffer_data+bmRequestType, 0, BANKED
      andlw 0x60                           ;Extrec els bits que indiquen el
request type
      movwf Registre_prov2, BANKED

      movlw STANDARD
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED  ;Comparem els dos valors i saltem si són
iguals
      goto Cas_CLASS                ;Són diferents. Anem al següent cas
      call StandardRequests         ;Són iguals, invoco rutina i marxo
      goto Request_Assignat

Cas_CLASS

      movlw CLASS
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED  ;Comparem els dos valors i saltem si són
iguals
      goto Cas_VENDOR               ;Són diferents. Anem al següent cas
      call ClassRequests            ;Són iguals, invoco rutina i marxo
      goto Request_Assignat

Cas_VENDOR

      movlw VENDOR
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
```

```
      cpfseq Registre_prov, BANKED  ;Comparem els dos valors i saltem si són
iguals
      goto Cas_Process_DEFAULT      ;Són diferents. Anem al següent cas
      call VendorRequests           ;Són iguals, invoco rutina i marxo
      goto Request_Assignat

Cas_Process_DEFAULT

      bsf USB_error_flags, 0, BANKED     ;Si cap dels anteriors casos es
compleix, per defecte assignem
                        ;el valor 1 al bit zero de la variable.
Request_Assignat

      return


;--------------------------------------


;--------------------------------------


ProcessInToken


      movf USB_USTAT, 0, BANKED
      andlw 0x18              ;Extreiem els bits de EP
      movwf Registre_prov2, BANKED


      movlw EP0
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED  ;Comparem els dos valors i saltem si són
iguals
      goto Rutina_PIT_Servida       ;No és EP0, tant si és EP1 com EP2
sortim

      movf USB_dev_req, 0, BANKED
      movwf Registre_prov2, BANKED


      movlw SET_ADDRESS
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED       ;Comparem els dos valors i saltem
si són iguals
      goto Cas_PIT_GET_DESCRIPTOR        ;No són iguals, saltem al següent
cas

      movf USB_address_pending, 0, BANKED
      movlb 0x0F
      movwf UADDR, BANKED

      movlw 0x00
      cpfseq UADDR, BANKED               ;Mirem si UADDR és zero
      goto Cas_PITSA_Default             ;No és zero, anem al cas per
defecte

      movlb 0
      movlw DEFAULT_STATE
```

```
      movwf USB_USWSTAT, BANKED

      #ifdef SHOW_ENUM_STATUS
                              ;Posem RB4 a 1 per encendre el LED de
DEFAULT_STATUS
      bcf LATC, LATC4               ;Apaguem LED que no toca
      bcf LATB, LATB6               ;Apaguem LED que no toca
      bcf LATC, LATC3               ;Apaguem LED que no toca
      bcf LATC, LATC2               ;Apaguem LED que no toca
      bsf LATB, LATB4               ;Encenem LED que toca

      #endif

      goto Rutina_PIT_Servida

Cas_PITSA_Default

      movlb 0
      movlw ADDRESS_STATE
      movwf USB_USWSTAT, BANKED

      #ifdef SHOW_ENUM_STATUS
                              ;Posem RB6 a 1 per encendre el LED de
ADDRESS_STATUS
      bcf LATB, LATB4               ;Apaguem LED que no toca
      bcf LATC, LATC4               ;Apaguem LED que no toca
      bcf LATC, LATC3               ;Apaguem LED que no toca
      bcf LATC, LATC2               ;Apaguem LED que no toca
      bsf LATB, LATB6               ;Encenem LED que toca

      #endif

      goto Rutina_PIT_Servida

Cas_PIT_GET_DESCRIPTOR

      call SendDescriptorPacket

Rutina_PIT_Servida


      return


;---------------------------------------


ProcessOutToken


      movf USB_USTAT, 0, BANKED
      andlw 0x18              ;Extreiem els bits de EP
      movwf Registre_prov2, BANKED

      movlw EP0
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED  ;Comparem els dos valors i saltem si són
iguals
      goto Rutina_POT_Servida       ;No és EP0, tant si és EP1 com EP2
sortim
```

```
      movlb 2
      movlw MAX_PACKET_SIZE
      movwf BD0OBC, BANKED
      movlw 0x88
      movwf BD0OST, BANKED
      clrf BD0IBC, BANKED           ;Posem el byte count a 0
      movlw 0xC8
      movwf BD0IST, BANKED          ;Enviem packet com DATA1 i activem bit
UOWN
      movlb 0

Rutina_POT_Servida


      return


;---------------------------------------


;---------------------------------------


SendDescriptorPacket


      movlw MAX_PACKET_SIZE
      cpfslt USB_bytes_left, BANKED      ;Salta si USB_bytes_left < W
      goto Cas_bytes_mes_8

      movlw NO_REQUEST
      movwf USB_dev_req, BANKED          ;Enviem un packet curt i responem
al device request
      movf USB_bytes_left, 0, BANKED

      goto Cas_bytes_menys_8

Cas_bytes_mes_8

      movlw MAX_PACKET_SIZE

Cas_bytes_menys_8


      subwf USB_bytes_left, 1, BANKED
      movwf USB_packet_length, BANKED
      movlb 2
      movwf BD0IBC, BANKED               ;Posem el packet size al byte
count de EP0 IN
      movf BD0IAH, 0, BANKED             ;Posem el valor corresponent al
EP0 IN buffer pointer...
      movwf FSR0H, ACCESS                ;Posem ACCESS per si de cas el
compilador es pot confondre
      movf BD0IAL, 0, BANKED
      movwf FSR0L, ACCESS                ;...dins FSR0
      movlb 0

      movf USB_packet_length, 0, BANKED
      movwf Compta_Send_Descriptor, BANKED
```

```
Bucli_Send_Descriptor

      call Descriptor                          ;Obtenim el següent byte del
descriptor que enviem
      movwf POSTINC0                           ;El fiquem a EP0 IN buffer i
incrementem FSR0
      incf USB_desc_ptr, 1, BANKED             ;Incrementem el descriptor
pointer
      decfsz Compta_Send_Descriptor, 1, BANKED ;Decrementem i saltem si val
zero
      goto Bucli_Send_Descriptor               ;Encara no val zero i per
tant saltem al principi del bucli

      movlb 2
      movlw 0x40
      xorwf BD0IST, 0, BANKED            ;Extraiem el bit DATA01
      andlw 0x40                    ;Borrem els bits de PIDs
      iorlw 0x88                    ;Activem els bits UOWN i DTS
      movwf BD0IST, BANKED
      movlb 0

      return


;---------------------------------------


;---------------------------------------


StandardRequests

      movf USB_buffer_data+bRequest, 0, BANKED ;Carrego a WREG el tipus de
request i miro de quin tipus
      movwf Registre_prov2, BANKED             ;es tracta


      movlw GET_STATUS
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED       ;Comparem els dos valors i saltem
si són iguals
      goto Cas_CLEAR_FEATURE             ;Són diferents. Anem al següent
cas
      call Rutina_REQ_GET_STATUS         ;Són iguals, invoco rutina i marxo
      goto Strd_Request_Servit

Cas_CLEAR_FEATURE


      movlw CLEAR_FEATURE
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED       ;Comparem els dos valors i saltem
si són iguals
      goto Cas_SET_FEATURE               ;Tant si són diferents com iguals
no es fa res i es salta
      goto Cas_SET_FEATURE               ;al següent cas

Cas_SET_FEATURE
```

```
      movlw SET_FEATURE
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED       ;Comparem els dos valors i saltem
si són iguals
      goto Cas_SET_ADDRESS               ;Són diferents. Anem al següent
cas
      call Rutina_REQ_SET_FEATURE        ;Són iguals, invoco rutina i marxo
      goto Strd_Request_Servit

Cas_SET_ADDRESS


      movlw SET_ADDRESS
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED       ;Comparem els dos valors i saltem
si són iguals
      goto Cas_GET_DESCRIPTOR            ;Són diferents. Anem al següent
cas
      call Rutina_REQ_SET_ADDRESS        ;Són iguals, invoco rutina i marxo
      goto Strd_Request_Servit

Cas_GET_DESCRIPTOR

      movlw GET_DESCRIPTOR
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED       ;Comparem els dos valors i saltem
si són iguals
      goto Cas_GET_CONFIGURATION         ;Són diferents. Anem al següent
cas
      call Rutina_REQ_GET_DESCRIPTOR           ;Són iguals, invoco rutina i
marxo
      goto Strd_Request_Servit

Cas_GET_CONFIGURATION

      movlw GET_CONFIGURATION
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED       ;Comparem els dos valors i saltem
si són iguals
      goto Cas_SET_CONFIGURATION         ;Són diferents. Anem al següent
cas
      call Rutina_REQ_GET_CONFIGURATION  ;Són iguals, invoco rutina i marxo
      goto Strd_Request_Servit

Cas_SET_CONFIGURATION

      movlw SET_CONFIGURATION
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED       ;Comparem els dos valors i saltem
si són iguals
      goto Cas_GET_INTERFACE             ;Són diferents. Anem al següent
cas
      call Rutina_REQ_SET_CONFIGURATION  ;Són iguals, invoco rutina i marxo
      goto Strd_Request_Servit
```

```
Cas_GET_INTERFACE

      movlw GET_INTERFACE
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED             ;Comparem els dos valors i
saltem si són iguals
      goto Cas_SET_INTERFACE              ;Són diferents. Anem al següent
cas
      call Rutina_REQ_GET_INTERFACE       ;Són iguals, invoco rutina i marxo
      goto Strd_Request_Servit

Cas_SET_INTERFACE

      movlw SET_INTERFACE
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED        ;Comparem els dos valors i saltem
si són iguals
      goto Cas_SET_DESCRIPTOR             ;Són diferents. Anem al següent
cas
      call Rutina_REQ_SET_INTERFACE       ;Són iguals, invoco rutina i marxo
      goto Strd_Request_Servit

Cas_SET_DESCRIPTOR

      movlw SET_DESCRIPTOR
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED        ;Comparem els dos valors i saltem
si són iguals
      goto Cas_SYNCH_FRAME                ;Són diferents. Anem al següent
cas
      goto Cas_SYNCH_FRAME                ;Són iguals, també anem al següent
cas perquè aquest cas
                              ;no és contemplat
Cas_SYNCH_FRAME

      movlw SYNCH_FRAME
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED        ;Comparem els dos valors i saltem
si són iguals
      goto Cas_REQ_DEFAULT                ;Són diferents. Anem al següent
cas
      call Rutina_REQ_SYNCH_FRAME         ;Són iguals, invoco rutina i marxo
      goto Strd_Request_Servit

Cas_REQ_DEFAULT

      bsf USB_error_flags, 0, BANKED            ;Activem el flag de Request
Error

Strd_Request_Servit

      return


;---------------------------------------
```

```
;----------------------------------------


ClassRequests


      bsf USB_error_flags, 0, BANKED           ;Activem el flag de Request
Error


      return


;----------------------------------------


;----------------------------------------


VendorRequests

      movf USB_buffer_data+bRequest, 0, BANKED ;Guardo a WREG el Vendor-
request rebut (1 o 2)
      movwf Registre_prov2, BANKED

      movlw SOL_SC_IN_OUT                ;Vendor-request igual a 0x01
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED       ;Comparem els dos valors i saltem
si són iguals
      goto Cas_Vendor_Req_2              ;Són diferents. Anem al següent
cas

      movlb 2                            ;Són iguals. Es tracta de retornar
el valor de SC_DINS
      movf BD0IAH, 0, BANKED             ;Apuntem amb el punter al buffer
de l'endpoint
      movwf FSR0H, ACCESS
      movf BD0IAL, 0, BANKED
      movwf FSR0L, ACCESS

      movlb 0
      movf SC_DINS, 0, BANKED            ;Posem a WREG el valor de la
variable SC_DINS
      movf POSTINC0

      movlb 2
      movlw 0x01
      movwf BD0IBC, BANKED               ;Posem el byte count a 1
      movlw 0xC8
      movwf BD0IST, BANKED               ;Enviem al host la informació com
DATA1 i activem bit UOWN

      movlb 0
      goto Vendor_Req_Servida

Cas_Vendor_Req_2

      movlw SOL_ATR_A_PC                 ;Vendor-request igual a 0x02
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
```

```
      cpfseq Registre_prov, BANKED       ;Comparem els dos valors i saltem
si són iguals
      goto Cas_Vendor_Req_Default              ;Són diferents. Anem al cas
per defecte

      call Obtenir_ATR             ;Són iguals. Invoquem rutina per obtenir
ATR

      movlb 2
      movf BD0IAH, 0, BANKED             ;Apuntem amb el punter al buffer
de l'endpoint
      movwf FSR0H, ACCESS
      movf BD0IAL, 0, BANKED
      movwf FSR0L, ACCESS

      movlb 0
      movf NOMBRE_BYTES_ATR, 0, BANKED   ;Faig una còpia del nombre de
bytes que hi ha d'ATR a la
      movwf Comptador_Generic, BANKED          ;variable NOMBRE_BYTES_ATR a
la variable Comptador_Generic

      movlw 0x00                   ;Carrego un punter a la variable ATR
      movwf FSR1H
      movlw 0x06
      movwf FSR1L

Recorre_Var_ATR

      movf INDF1, 0                        ;Agafo el comptingut de la primera
posició de l'ATR i el poso
      movwf POSTINC0                       ;a WREG i d'aquí l'envio al buffer
de l'endpoint i incremento
      incf FSR1L, 1                        ;tots dos punters.
      decfsz Comptador_Generic, 1, BANKED ;Decremento el comptador i no
continuo si és zero.
      goto Recorre_Var_ATR                 ;No és zero i per tant queden
bytes per enviar

      movf NOMBRE_BYTES_ATR, 0, BANKED
      movwf POSTINC0
      movlw 0x01
      addwf NOMBRE_BYTES_ATR, 0, BANKED

;     movf NOMBRE_BYTES_ATR, 0, BANKED   ;És zero i per tant ja podem
enviar les dades
      movlb 2
      movwf BD0IBC, BANKED                 ;Carreguem al byte count el número
de bytes a enviar
      movlw 0xC8
      movwf BD0IST, BANKED                 ;Enviem al host les dades com
DATA1 i activem bit UOWN

      movlb 0
      goto Vendor_Req_Servida

Cas_Vendor_Req_Default

      bsf USB_error_flags, 0, BANKED           ;Activem el flag de Request
Error

Vendor_Req_Servida
```

```
      return


;---------------------------------------


;---------------------------------------


;Zona de subrutines de la funció "StandardRequests" per servir tots els
casos possibles
;de requeriments de la interfície USB.



Rutina_REQ_GET_STATUS

      movf USB_buffer_data+bmRequestType, 0, BANKED
      andlw 0x1F                          ;Extreiem els "request recipient
bits"
      movwf Registre_prov2, BANKED

      movlw RECIPIENT_DEVICE
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED        ;Comparem els dos valors i saltem
si són iguals
      goto Cas_RECIPIENT_INTERFACE        ;Són diferents. Anem al següent
cas
      call Rutina_GS_RECIPIENT_DEVICE           ;Són iguals, invoco rutina i
marxo
      goto Rutina_RGS_Servida

Cas_RECIPIENT_INTERFACE

      movlw RECIPIENT_INTERFACE
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED        ;Comparem els dos valors i saltem
si són iguals
      goto Cas_RECIPIENT_ENDPOINT         ;Són diferents. Anem al següent
cas
      call Rutina_GS_RECIPIENT_INTERFACE  ;Són iguals, invoco rutina i marxo
      goto Rutina_RGS_Servida

Cas_RECIPIENT_ENDPOINT

      movlw RECIPIENT_ENDPOINT
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED        ;Comparem els dos valors i saltem
si són iguals
      goto Cas_RECIPIENT_DEFAULT          ;Són diferents. Anem al següent
cas
      call Rutina_GS_RECIPIENT_ENDPOINT   ;Són iguals, invoco rutina i marxo
      goto Rutina_RGS_Servida

Cas_RECIPIENT_DEFAULT
```

```
      bsf USB_error_flags, 0, BANKED           ;Activem el flag de "Request
Error"

Rutina_RGS_Servida

      return



;La següent rutina no aplica ja que el request CLEAR_FEATURE no és
processat i es passa a
;processar el següent request

;Rutina_REQ_CLEAR_FEATURE
;return



Rutina_REQ_SET_FEATURE

      movf USB_buffer_data+bmRequestType, 0, BANKED
      andlw 0x1F                          ;Extraiem els recipient bits
      movwf Registre_prov2, BANKED

      movlw RECIPIENT_DEVICE
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED        ;Comparem els dos valors i saltem
si són iguals
      goto Cas_RSF_RECIPIENT_ENDPOINT          ;Són diferents. Anem al
següent cas
      call Rutina_SF_RECIPIENT_DEVICE          ;Són iguals, invoco rutina i
marxo
      goto Rutina_RSF_Servida

Cas_RSF_RECIPIENT_ENDPOINT

      movlw RECIPIENT_ENDPOINT
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED        ;Comparem els dos valors i saltem
si són iguals
      goto Cas_RSF_Default                ;Són diferents. Anem al següent
cas
      call Rutina_SF_RECIPIENT_ENDPOINT   ;Són iguals, invoco rutina i marxo
      goto Rutina_RSF_Servida

Cas_RSF_Default

      bsf USB_error_flags, 0, BANKED           ;Activem el flag de Request
Error

Rutina_RSF_Servida

      return
```

```
Rutina_REQ_SET_ADDRESS

      btfss USB_buffer_data+wValue, 7, BANKED ;Si l'adreça del device és
il.legal enviem Request Error
      goto Adreca_Legal
      bsf USB_error_flags, 0, BANKED           ;Activem flag de Request
Error
      goto Rutina_REQSA_Servida

Adreca_Legal

      movlw SET_ADDRESS
      movwf USB_dev_req, BANKED          ;Processem una petició SET_ADDRESS
      movf USB_buffer_data+wValue, 0, BANKED
      movwf USB_address_pending, BANKED  ;Guardem la nova adreça
      movlb 2
      clrf BD0IBC, BANKED                ;Posem byte count a zero
      movlw 0xC8
      movwf BD0IST, BANKED               ;Enviem packet com DATA1 i activem
el bit UOWN
      movlb 0

Rutina_REQSA_Servida

      return




Rutina_REQ_GET_CONFIGURATION

      movlb 2
      movf BD0IAH, 0, BANKED
      movwf FSR0H, ACCESS
      movf BD0IAL, 0, BANKED
      movwf FSR0L, ACCESS
      movlb 0
      movf USB_curr_config, 0, BANKED
      movwf INDF0
      movlb 2                ;Copiem l'actual configuració del device al
buffer EP0 IN
      movlw 0x01
      movwf BD0IBC, BANKED         ;Posem el EP0 IN byte count a 1
      movlw 0xC8
      movwf BD0IST, BANKED         ;Enviem packet com DATA1, activem el bit
UOWN
      movlb 0

      return
```

```
Rutina_REQ_SET_CONFIGURATION

      movlw NUM_CONFIGURATIONS

      cpfseq USB_buffer_data+wValue, BANKED    ;Salta si són iguals
      cpfsgt USB_buffer_data+wValue, BANKED    ;Són diferents però encara
pot ser USB_buffer_data+wValue < WREG
      goto RSETC_Condicio_Correcte
      goto RSETC_Activem_Error           ;No es compleix cap de les dues
opcions, per tant marxem

RSETC_Condicio_Correcte

      movlb 0x0F
      clrf UEP1, BANKED       ;Borrem tots els registres de control EP menys
el EP0 per evitar
      clrf UEP2, BANKED       ;la configuració primer de EP1-EP15
      clrf UEP3, BANKED
      clrf UEP4, BANKED
      clrf UEP5, BANKED
      clrf UEP6, BANKED
      clrf UEP7, BANKED
      movlb 0

      movf USB_buffer_data+wValue, 0, BANKED
      movwf USB_curr_config, BANKED

      movlw 0x00
      cpfseq USB_curr_config, BANKED     ;Salta si són iguals (si val 0x00)
      goto RSETC_Cas_Default        ;Són diferents

      movlw ADDRESS_STATE
      movwf USB_USWSTAT, BANKED

      #ifdef SHOW_ENUM_STATUS
                              ;Posem RB6 a 1 per encendre el LED de
ADDRESS_STATUS
      bcf LATB, LATB4               ;Apaguem LED que no toca
      bcf LATC, LATC4               ;Apaguem LED que no toca
      bcf LATC, LATC3               ;Apaguem LED que no toca
      bcf LATC, LATC2               ;Apaguem LED que no toca
      bsf LATB, LATB6               ;Encenem LED que toca

      #endif

      goto RSETC_Cas_FinalIF

RSETC_Cas_Default

      movlw CONFIG_STATE
      movwf USB_USWSTAT, BANKED

      #ifdef SHOW_ENUM_STATUS
                              ;Posem RC3 a 1 per encendre el LED de
CONFIG_STATUS
      bcf LATB, LATB4               ;Apaguem LED que no toca
      bcf LATB, LATB6               ;Apaguem LED que no toca
      bcf LATC, LATC4               ;Apaguem LED que no toca
      bcf LATC, LATC2               ;Apaguem LED que no toca
      bsf LATC, LATC3               ;Encenem LED que toca
```

```
      #endif

RSETC_Cas_FinalIF

      movlb 2
      clrf BD0IBC, BANKED          ;Posem el byte count a zero
      movlw 0xC8
      movwf BD0IST, BANKED         ;Enviem packet com DATA1. Activem bit
UOWN
      movlb 0

      goto Rutina_RSETC_Servida

RSETC_Activem_Error

      bsf USB_error_flags, 0, BANKED     ;Activem el flag de Request Error

Rutina_RSETC_Servida

      return









Rutina_REQ_SET_INTERFACE

      movf USB_USWSTAT, 0, BANKED

      movwf Registre_prov2, BANKED
      movlw CONFIG_STATE

      cpfseq Registre_prov2, BANKED ;Salta si són iguals
      goto RSI_Cas_Default          ;No ho són. Anem al cas default.

      movlw NUM_INTERFACES
      cpfslt USB_buffer_data+wIndex, BANKED    ;Si USB_buffer_data+wIndex <
WREG llavors saltem
      goto RSI_Cas_Default               ;Cal activar el flag d'error i
aprofitem el cas per defecte


      movlw 0x00
      cpfseq USB_buffer_data+wValue, BANKED    ;Miro si
USB_buffer_data+wValue val zero. Si és així saltem
      goto RSI_Cas_Default               ;Diferents, cal activar el flag
d'error i aprofitem el cas per defecte

      movlb 2
      clrf BD0IBC, BANKED          ;Posem el byte count a 0
      movlw 0xC8
      movwf BD0IST, BANKED         ;Enviem packet com DATA1, activem el bit
UOWN
      movlb 0
```

```
      goto Rutina_RSI_Servida

RSI_Cas_Default

      bsf USB_error_flags, 0, BANKED     ;Activem el flag de Request Error

Rutina_RSI_Servida

      return



Rutina_REQ_SYNCH_FRAME

      bsf USB_error_flags, 0, BANKED     ;Activem el flag de Request Error

      return



Rutina_GS_RECIPIENT_DEVICE

      movlb 2
      movf BD0IAH, 0, BANKED
      movwf FSR0H, ACCESS
      movf BD0IAL, 0, BANKED       ;Obtenim el buffer pointer
      movwf FSR0L, ACCESS
      movlb 0
      movf USB_device_status, 0, BANKED  ;Copiem el byte amb l'estat del
device al EP0 buffer
      movwf POSTINC0
      clrf INDF0
      movlb 2
      movlw 0x02
      movwf BD0IBC, BANKED               ;Inicialitzem el byte count a 2
      movlw 0xC8
      movwf BD0IST, BANKED               ;Enviem el packet com DATA1. Posem
a 1 el bit UOWN
      movlb 0

      return



Rutina_GS_RECIPIENT_INTERFACE

      movf USB_USWSTAT, 0, BANKED
      movwf Registre_prov2, BANKED

      movlw ADDRESS_STATE
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED       ;Comparem els dos valors i saltem
si són iguals
```

```
      goto Cas_GSRI_CONFIG_STATE          ;Són diferents. Anem al següent
cas
      bsf USB_error_flags, 0, BANKED            ;Són iguals, setejo un bit i
marxo
      goto Rutina_GSRI_Servida

Cas_GSRI_CONFIG_STATE

      movlw CONFIG_STATE
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED        ;Comparem els dos valors i saltem
si són iguals
      goto Rutina_GSRI_Servida            ;Són diferents. S'ha acabat i
marxem
                              ;Són iguals, servim i marxem
      movlw NUM_INTERFACES

      cpfslt USB_buffer_data+wIndex, BANKED           ;Salta si
USB_buffer_data+wIndex < NUM_INTERFACES
      goto GSRI_Es_Major                        ;NUM_INTERFACES és major
      bsf USB_error_flags, 0, BANKED                  ;NUM_INTERFACES és
menor i
                                    ;activem el flag de "Request Error"
      goto Rutina_GSRI_Servida

GSRI_Es_Major

      movlb 2
      movf BD0IAH, 0, BANKED
      movwf FSR0H, ACCESS
      movf BD0IAL, 0, BANKED              ;Obtenim el buffer pointer
      movwf FSR0L, ACCESS
      clrf POSTINC0, ACCESS
      clrf INDF0, ACCESS
      movlw 0x02
      movwf BD0IBC, BANKED                ;Inicialitzem el byte count a 2
      movlw 0xC8
      movwf BD0IST, BANKED
      movlb 0                 ;Enviem el packet com DATA1. Posem a 1 el bit
UOWN

Rutina_GSRI_Servida

      return



Rutina_GSRE_ADDRESS_STATE

      movf USB_buffer_data+wIndex, 0, BANKED   ;Obtinc EP get EP
      andlw 0x0F                    ;Obtinc el bit de direcció

      btfss STATUS, Z                     ;Salta si Z=1. Mirem si EP0
      goto No_Es_EP0                      ;Z=0
      movlb 2
      movf BD0IAH, 0, BANKED              ;Posem EP0 IN buffer pointer
      movwf FSR0H, ACCESS
      movf BD0IAL, 0, BANKED
      movlb 0
```

```
      movwf FSR0L, ACCESS                ;dins de FSR0
      btfss USB_buffer_data+wIndex, 7, BANKED  ;Comprovem si la direcció
especificada és IN
      goto No_Es_Dir_IN
      movlb 2                ;No és IN. Saltem i activem error
      movf BD0IST, 0, BANKED             ;És direcció IN
      movlb 0
      goto Si_Es_Dir_IN

No_Es_Dir_IN

      movlb 2
      movf BD0OST, 0, BANKED
      movlb 0

Si_Es_Dir_IN

      andlw 0x04                    ;Extraiem el bit BSTALL
      movwf INDF0
      rrncf INDF0, 1
      rrncf INDF0, 1                     ;Desplacem el bit BSTALL a la
posició lsb
      clrf PREINC0
      movlb 2
      movlw 0x02
      movwf BD0IBC, BANKED               ;Posem a 2 el byte count
      movlw 0xC8
      movwf BD0IST, BANKED               ;Enviem packet DATA1. Activem el
bit UOWN
      movlb 0

      goto Rutina_GSREAS_Servida

No_Es_EP0

      bsf USB_error_flags, 0, BANKED           ;Activem el flag de Request
Error

Rutina_GSREAS_Servida

      return



Rutina_GSRE_CONFIG_STATE

      movlb 2
      movf BD0IAH, 0, BANKED             ;Posem la EP0 IN buffer pointer...
      movwf FSR0H, ACCESS
      movf BD0IAL, 0, BANKED
      movlb 0
      movwf FSR0L, ACCESS                ;...dins del registre FSR0
      movlw high UEP0                    ;Posem la UEP0 address...
      movwf FSR1H, ACCESS
      movlw low UEP0
      movwf FSR1L, ACCESS                ;...dins del registre FSR1
      movlw high BD0OST             ;Posem la BDndST address...
      movwf FSR2H, ACCESS
      movf USB_buffer_data+wIndex, 0, BANKED
```

```
      andlw 0x8F                    ;Enmascarem tots els bits menys els de
direcció i de
      movwf FSR2L                   ;EP Number
      rlncf FSR2L, 1
      rlncf FSR2L, 1
      rlncf FSR2L, 1                      ;FSR2L ara conté l'offset adequat
de la BD taula per
      movlw low BD0OST              ;l'EP especificat
      addwf FSR2L, 1                      ;...dins del registre FSR2

      btfsc STATUS, C                     ;Mirem el valor del carry
      incf FSR2H, 1                       ;Val 1, per tant cal incrementar
FSR2H
                              ;Val 0, no cal fer res.

      movf USB_buffer_data+wIndex, 0, BANKED   ;Obtenim EP i...
      andlw 0x0F                    ;...extraiem el bit de direcció

      btfss USB_buffer_data+wIndex, 7, BANKED  ;Si la direcció de l'EP és
IN...
      goto Segona_Condicio_IF
      btfsc PLUSW1, EPINEN                ;...i l'EP especificat no està
activat per IN transfers...
      goto Segona_Condicio_IF
      bsf USB_error_flags, 0, BANKED            ;...activem el Request Error
Flag
      goto Doble_Condicional_Acabat

Segona_Condicio_IF

      btfsc USB_buffer_data+wIndex, 7, BANKED        ;Com a segona
condició, si l'EP especificat té direcció OUT...
      goto Alternativa_doble_IF
      btfsc PLUSW1, EPOUTEN               ;...i l'EP especificat no està
activat per OUT transfers...
      goto Alternativa_doble_IF
      bsf USB_error_flags, 0, BANKED            ;...activem el Request Error
Flag
      goto Doble_Condicional_Acabat

Alternativa_doble_IF                ;Cap de les condicions anteriors s'ha
complert

      movf INDF2, 0                       ;Movem el contingut del registre
BDnST a WREG
      andlw 0x04                    ;Extraiem el BSTALL bit
      movwf INDF0
      rrncf INDF0, 1
      rrncf INDF0, 1                      ;shift del bit BSTALL a la posició
lsb
      clrf PREINC0
      movlb 2
      movlw 0x02
      movwf BD0IBC, BANKED                ;Posem el byte count a 2
      movlw 0xC8
      movwf BD0IST, BANKED                ;Enviem packet DATA1 i posem a 1
el bit UOWN
      movlb 0

Doble_Condicional_Acabat
```

```
      return



Rutina_SF_RECIPIENT_ENDPOINT

      movf USB_USWSTAT, 0, BANKED
      movwf Registre_prov2, BANKED


      movlw ADDRESS_STATE
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED       ;Comparem els dos valors i saltem
si són iguals
      goto Cas_SFRE_CONFIG_STATE         ;Són diferents. Anem al següent
cas
      call Rutina_SFRE_ADDRESS_STATE           ;Són iguals, invoco rutina i
marxo

      goto Rutina_SFRE_Servida

Cas_SFRE_CONFIG_STATE

      movlw CONFIG_STATE
      movwf Registre_prov, BANKED
      movf Registre_prov2, 0, BANKED
      cpfseq Registre_prov, BANKED       ;Comparem els dos valors i saltem
si són iguals
      goto Cas_SFRE_Default              ;Són diferents. Anem al següent
cas
      call Rutina_SFRE_CONFIG_STATE      ;Són iguals, invoco rutina i marxo

      goto Rutina_SFRE_Servida

Cas_SFRE_Default

      bsf USB_error_flags, 0, BANKED           ;Activem el flag de Request
Error

Rutina_SFRE_Servida

      return


Rutina_SFRE_ADDRESS_STATE


      movf USB_buffer_data+wIndex, 0, BANKED         ;Obtenim EP
      andlw 0x0F                         ;Extraiem el bit de direcció

      btfsc STATUS, Z                    ;Si és zero saltem (si Z=1 es
tracta de EP0)
      goto SFRE_Es_EP0             ;Es tracta de EP0
      bsf USB_error_flags, 0, BANKED           ;És zero i per tant activem
el flag d'error
      goto SFRE_ADDRESS_STATE_Servit

SFRE_Es_EP0
```

```
      movlb 2
      clrf BD0IBC, BANKED                       ;Posem el byte count a 0
      movlw 0xC8
      movwf BD0IST, BANKED                      ;Enviem packet com DATA1 i
activem el bit UOWN
      movlb 0

SFRE_ADDRESS_STATE_Servit

      return


Rutina_SFRE_CONFIG_STATE

      movlw high UEP0              ;Posem l'adreça de UEP0...
      movwf FSR0H
      movlw low UEP0
      movwf FSR0L             ;...dins de FSR0
      movlw high BD0OST       ;Posem l'adreça de BD0OST...
      movwf FSR1H
      movlw low BD0OST
      movwf FSR1L                   ;...dins de FSR1
      movf USB_buffer_data+wIndex, 0, BANKED   ;Obtenim EP
      andlw 0x0F                    ;Extraiem el bit de direcció

      btfsc STATUS, Z               ;Si no es tracta de EP0...
      goto SFRE_Es_Dir_EP0          ;Es tracta de EP0

      addwf FSR0L, 1                ;Afegim el número de EP a FSR0

      btfsc STATUS, C               ;Saltem si C=0
      incf FSR0H, 1                 ;C=1

      rlncf USB_buffer_data+wIndex, 1, BANKED  ;Rotem el valor fins que a
WREG hi haurà l'offset adequat que
      rlncf USB_buffer_data+wIndex, 1, BANKED  ;apuntarà al corresponent EP
dins de la taula BD
      rlncf USB_buffer_data+wIndex, 0, BANKED
      andlw 0x7C                    ;enmascara tot menys el bit de direcció
i el número de EP després
      addwf FSR1L, 1                ;de rotar 3 cops a l'esquerra. Afegim
l'offset de la taula BD a FSR1

      btfsc STATUS, C               ;Saltem si C=0
      incf FSR1H, 1                 ;C=1

      btfsc USB_buffer_data+wIndex, 1     ;Si el bit val zero saltem
      goto IFSET1             ;El bit val 1
      goto OTHERWISE1               ;El bit val 0

IFSET1
      btfsc INDF0, EPINEN           ;Si el bit val zero saltem
      goto IFSET2
      goto OTHERWISE2

IFSET2
      movlw CLEAR_FEATURE
      cpfseq USB_buffer_data+bRequest, BANKED  ;Salta si són iguals
      goto OTHERWISE3
      clrf INDF1                    ;Borrem STALL en el EP especificat
```

```
      goto SFRE_Es_Dir_EP0

OTHERWISE3
      movlw 0x84
      movwf INDF1             ;STALL del EP especificat
      goto SFRE_Es_Dir_EP0

OTHERWISE2
      bsf USB_error_flags, 0, BANKED     ;Activem el flag Request Error
      goto SFRE_Es_Dir_EP0

OTHERWISE1
      btfsc INDF0, EPOUTEN          ;Si el bit val zero saltem
      goto IFSET3             ;Val 1 (EP activat per transferències OUT)
      goto OTHERWISE4               ;Val 0

IFSET3
      movlw CLEAR_FEATURE
      cpfseq USB_buffer_data+bRequest, BANKED  ;Salta si són iguals
      goto OTHERWISE5

      movlw 0x88
      movwf INDF1             ;Borrem STALL en el EP especificat clear the
stall on the specified EP
      goto SFRE_Es_Dir_EP0

OTHERWISE5
      movlw 0x84
      movwf INDF1             ;STALL del EP especificat
      goto SFRE_Es_Dir_EP0

OTHERWISE4
      bsf USB_error_flags, 0, BANKED     ;Activem el flag Request Error

SFRE_Es_Dir_EP0

      btfsc USB_error_flags, 0, BANKED    ;Mirem si el flag d'error s'ha
activat
      goto Rutina_SFRECS_Servida          ;S'ha activat, per tant ja sortim
      movlb 2
      clrf BD0IBC, BANKED                 ;No s'ha activat. Posem el byte
count a 0
      movlw 0xC8
      movwf BD0IST, BANKED                ;Enviem packet com DATA1 i activem
bit UOWN
      movlb 0

Rutina_SFRECS_Servida

      return



Rutina_GD_DEVICE


      movlw low (Device-Descriptor_begin)
      movwf USB_desc_ptr, BANKED
      call Descriptor                     ;Obtenim la longitud del
descriptor
```

```
      movwf USB_bytes_left, BANKED

      movlw 0x00
      cpfseq USB_buffer_data+(wLength+1), BANKED     ;Comparem si el valor
val 0
      goto Rutina_GD_DEVICE_Servida            ;Són diferents, sortim del
condicional

      movf USB_bytes_left, 0, BANKED
      cpfslt USB_buffer_data+wLength, BANKED   ;Si USB_buffer_data+wLength
menor que USB_bytes_left salta
      goto Rutina_GD_DEVICE_Servida      ;No es compleix, sortim del
condicional

      movf USB_buffer_data+wLength, 0, BANKED  ;Es compleix que
USB_buffer_data+wLength < USB_bytes_left
      movwf USB_bytes_left, BANKED

Rutina_GD_DEVICE_Servida

      call SendDescriptorPacket


      return



Rutina_GD_STRING


      movf USB_buffer_data+wValue, 0, BANKED
      movwf Registre_prov, BANKED

      movlw 0x00
      cpfseq Registre_prov, BANKED       ;Mirem si el valor val zero
      goto Cas_GDS_1                     ;No val zero
      movlw low (String0-Descriptor_begin)     ;Val zero

      goto Rutina_GDS_General

Cas_GDS_1

      movlw 0x01
      cpfseq Registre_prov, BANKED       ;Mirem si el valor val zero
      goto Cas_GDS_2                     ;No val zero
      movlw low (String1-Descriptor_begin)     ;Val zero

      goto Rutina_GDS_General

Cas_GDS_2

      movlw 0x02
      cpfseq Registre_prov, BANKED       ;Mirem si el valor val zero
      goto Cas_GDS_Default               ;No val zero
      movlw low (String2-Descriptor_begin)     ;Val zero

      goto Rutina_GDS_General

Cas_GDS_Default
```

```
      bsf USB_error_flags, 0, BANKED           ;Activem el flag Request
Error

Rutina_GDS_General


      btfsc USB_error_flags, 0, BANKED         ;Si el bit val 0 saltem
      goto Rutina_GDS_Servida             ;Hi ha error i per tant sortim de
la rutina

      movwf USB_desc_ptr, BANKED
      call Descriptor                     ;Obtenim la longitud del
descriptor
      movwf USB_bytes_left, BANKED

      movlw 0x00
      cpfseq USB_buffer_data+(wLength+1), BANKED     ;Miro si la variable
val zero
      goto Rutina_GDS_General2                  ;No val zero i surto

      movf USB_bytes_left, 0, BANKED
      cpfslt USB_buffer_data+wLength, BANKED   ;Si USB_buffer_data+wLength
< USB_bytes_left saltem
      goto Rutina_GDS_General2            ;No es compleix i sortim

      movf USB_buffer_data+wLength, 0, BANKED
      movwf USB_bytes_left, BANKED

Rutina_GDS_General2

      call SendDescriptorPacket

Rutina_GDS_Servida


      return


;---------------------------------------


;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;ZONA DE RUTINES INTERRUPCIÓ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;---------------------------------------


Hard_int                      ;Rutina per discriminar la font d'interrupció
High (INT0 o INT1)

      call Interrupcio, 1

      retfie


;---------------------------------------
```

```
;----------------------------------------


Soft_int                ;Rutina per discriminar la font d'interrupció Low


      retfie


;----------------------------------------


;----------------------------------------


Interrupcio

      movlb 0
      btfss INTCON, INT0IF          ;Si val zero no salto i vaig a extracció
perquè s'ha tret la SC
      goto Extraccio                ;Si val 1 és que s'ha ficat la targeta i
per tant salto el goto

      bsf SC_DINS, 0, BANKED        ;S'ha introduït la targeta i posem a 1
la variable
      clrf SOLLICITUD_ATR, BANKED   ;Posem a zero el recordartori d'ATR
      clrf NOMBRE_BYTES_ATR, BANKED ;Posem a zero el comptador de bytes de
ATR

      bcf INTCON, INT0IF            ;Baixem el flag de INT0

      return 1

Extraccio                     ;S'ha produït una extracció. Recarrego tot el
programa

      call Configuracio_Inicial     ;Configuro de nou registres i variables
com em convé

      return 1


;----------------------------------------

      END
```

B SOFTWARE COMPLERT EN VISUAL BASIC

A continuació es llista el codi complert en el llenguatge "Visual Basic" dels 8 arxius que configuren el software de l'aplicació. Els arxius, per aquest ordre són:

- AssemblyInfo.vb
- DeviceManagement.vb
- DeviceManagementApi.vb
- FileIOapi.vb
- frmMain.vb
- WinUsbDemo.vb
- WinUsbDevice.vb
- WinUsbDeviceApi.vb

AssemblyInfo.vb

```
Imports System.Reflection
'  General Information about an assembly is controlled through the
following
'  set of attributes. Change these attribute values to modify the
information
'  associated with an assembly

<Assembly: AssemblyTitle("WinUSB Demo Application")>
<Assembly: AssemblyDescription("www.Lvr.com")>
<Assembly: AssemblyCompany("Lakeview Research")>
<Assembly: AssemblyProduct("WinUsbDemo")>
<Assembly: AssemblyCopyright("c. 2008 by Jan Axelson")>
<Assembly: AssemblyTrademark("")>
<Assembly: AssemblyCulture("")>

'  Version information for an assembly consists of the following four
values:

'     Major version
'     Minor Version
'     Revision
'     Build Number

'  You can specify all the values or you can default the Revision and Build
Numbers
'  by using the '*' as shown below


<Assembly: AssemblyVersion("1.2.*")>
```

DeviceManagement.vb

```
Imports System
Imports System.Runtime.InteropServices
Imports System.Windows.Forms
Imports System.Diagnostics

Namespace WinUsbDemo
      '''  <summary>
      '''  Routines for detecting devices.
      '''  </summary>

      Friend NotInheritable Partial Class DeviceManagement
            Private Const ModuleName As String = "Device Management"

            '''  <summary>
            '''  Provides a central mechanism for exception handling.
            '''  Displays a message box that describes the exception.
            '''  </summary>
            '''
            '''  <param name="Name"> the module where the exception
occurred. </param>
            '''  <param name="e"> the exception </param>

            Friend Shared Sub DisplayException(ByVal Name As String, ByVal
e As Exception)
                  Try
                        '  Create an error message.

                        Dim message As String = "Exception: " & e.Message &
Environment.NewLine & "Module: " & Name & Environment.NewLine & "Method: "
& e.TargetSite.Name

                        Const caption As String = "Unexpected Exception"

                        MessageBox.Show(message, caption,
MessageBoxButtons.OK)
                        Debug.Write(message)
                  Catch ex As Exception
                        DisplayException(ModuleName, ex)
                        Throw
                  End Try
            End Sub

            '''  <summary>
            '''  Use SetupDi API functions to retrieve the device path name
of an
            '''  attached device that belongs to a device interface class.
            '''  </summary>
            '''
            '''  <param name="myGuid"> an interface class GUID. </param>
            '''  <param name="devicePathName"> a pointer to the device path
name
            '''  of an attached device. </param>
            '''
            '''  <returns>
            '''   True if a device is found, False if not.
            '''  </returns>

            Friend Function FindDeviceFromGuid(ByVal myGuid As Guid, ByRef
devicePathName As String) As Boolean
                  Dim bufferSize As Int32 = 0
                  Dim detailDataBuffer As IntPtr = IntPtr.Zero
```

```
                Dim deviceInfoSet = New IntPtr()
                Dim lastDevice As Boolean = False
                Dim myDeviceInterfaceData = New
NativeMethods.SP_DEVICE_INTERFACE_DATA()

                Try
                        ' ***
                        '  API function

                        '  summary
                        '  Retrieves a device information set for a
specified group of devices.
                        '  SetupDiEnumDeviceInterfaces uses the device
information set.

                        '  parameters
                        '  Interface class GUID.
                        '  Null to retrieve information for all device
instances.
                        '  Optional handle to a top-level window (unused
here).
                        '  Flags to limit the returned information to
currently present devices
                        '  and devices that expose interfaces in the class
specified by the GUID.

                        '  Returns
                        '  Handle to a device information set for the
devices.
                        ' ***

                        deviceInfoSet =
NativeMethods.SetupDiGetClassDevs(myGuid, IntPtr.Zero, IntPtr.Zero,
NativeMethods.DIGCF_PRESENT Or NativeMethods.DIGCF_DEVICEINTERFACE)

                        Dim deviceFound As Boolean = False
                        Dim memberIndex As Int32 = 0

                        ' The cbSize element of the MyDeviceInterfaceData
structure must be set to
                        ' the structure's size in bytes.
                        ' The size is 28 bytes for 32-bit code and 32 bits
for 64-bit code.

                        myDeviceInterfaceData.cbSize =
Marshal.SizeOf(myDeviceInterfaceData)

                        Do
                                ' Begin with 0 and increment through the
device information set until
                                ' no more devices are available.


                                ' ***
                                '  API function

                                '  summary
                                '  Retrieves a handle to a
SP_DEVICE_INTERFACE_DATA structure for a device.
                                '  On return, MyDeviceInterfaceData contains
the handle to a
```

```
                              '  SP_DEVICE_INTERFACE_DATA structure for a
detected device.

                              '  parameters
                              '  DeviceInfoSet returned by
SetupDiGetClassDevs.
                              '  Optional SP_DEVINFO_DATA structure that
defines a device instance
                              '  that is a member of a device information
set.
                              '  Device interface GUID.
                              '  Index to specify a device in a device
information set.
                              '  Pointer to a handle to a
SP_DEVICE_INTERFACE_DATA structure for a device.

                              '  Returns
                              '  True on success.
                              ' ***

                              Dim success As Boolean =
NativeMethods.SetupDiEnumDeviceInterfaces(deviceInfoSet, IntPtr.Zero,
myGuid, memberIndex, myDeviceInterfaceData)

                              ' Find out if a device information set was
retrieved.

                              If Not success Then
                                    lastDevice = True

                              Else
                                    ' A device is present.

                                    ' ***
                                    '  API function:

                                    '  summary:
                                    '  Retrieves an
SP_DEVICE_INTERFACE_DETAIL_DATA structure
                                    '  containing information about a
device.
                                    '  To retrieve the information, call
this function twice.
                                    '  The first time returns the size of
the structure.
                                    '  The second time returns a pointer to
the data.

                                    '  parameters
                                    '  DeviceInfoSet returned by
SetupDiGetClassDevs
                                    '  SP_DEVICE_INTERFACE_DATA structure
returned by SetupDiEnumDeviceInterfaces
                                    '  A returned pointer to an
SP_DEVICE_INTERFACE_DETAIL_DATA
                                    '  Structure to receive information
about the specified interface.
                                    '  The size of the
SP_DEVICE_INTERFACE_DETAIL_DATA structure.
                                    '  Pointer to a variable that will
receive the returned required size of the
```

```
                                    '  SP_DEVICE_INTERFACE_DETAIL_DATA
structure.
                                    '  Returned pointer to an
SP_DEVINFO_DATA structure to receive information about the device.

                                    '  Returns
                                    '  True on success.
                                    ' ***


      NativeMethods.SetupDiGetDeviceInterfaceDetail(deviceInfoSet,
myDeviceInterfaceData, IntPtr.Zero, 0, bufferSize, IntPtr.Zero)

                                    ' Allocate memory for the
SP_DEVICE_INTERFACE_DETAIL_DATA structure using the returned buffer size.

                                   detailDataBuffer =
Marshal.AllocHGlobal(bufferSize)

                                    ' Store cbSize in the first bytes of the
array. The number of bytes varies with 32- and 64-bit systems.


      Marshal.WriteInt32(detailDataBuffer,If(IntPtr.Size = 4, (4 +
Marshal.SystemDefaultCharSize), 8))

                                    ' Call SetupDiGetDeviceInterfaceDetail
again.
                                    ' This time, pass a pointer to
DetailDataBuffer
                                    ' and the returned required buffer size.


      NativeMethods.SetupDiGetDeviceInterfaceDetail(deviceInfoSet,
myDeviceInterfaceData, detailDataBuffer, bufferSize, bufferSize,
IntPtr.Zero)

                                    ' Skip over cbsize (4 bytes) to get the
address of the devicePathName.

                                   Dim pDevicePathName = New
IntPtr(detailDataBuffer.ToInt64() + 4)

                                    ' Get the String containing the
devicePathName.

                                   devicePathName =
Marshal.PtrToStringAuto(pDevicePathName)

                                         deviceFound = True
                                  End If
                                  memberIndex = memberIndex + 1
                            Loop While lastDevice <> True

                            Return deviceFound


                        Catch ex As Exception
                              DisplayException(ModuleName, ex)
                              Throw
```

```
                  Finally
                        If detailDataBuffer <> IntPtr.Zero Then
                              ' Free the memory allocated previously by
AllocHGlobal.

                              Marshal.FreeHGlobal(detailDataBuffer)
                        End If
                        If deviceInfoSet <> IntPtr.Zero Then
                              ' ***
                              '  API function

                              '  summary
                              '  Frees the memory reserved for the
DeviceInfoSet returned by SetupDiGetClassDevs.

                              '  parameters
                              '  DeviceInfoSet returned by
SetupDiGetClassDevs.

                              '  returns
                              '  True on success.
                              ' ***


      NativeMethods.SetupDiDestroyDeviceInfoList(deviceInfoSet)
                        End If
                  End Try
            End Function
      End Class
End Namespace



DeviceManagementApi.vb

Imports System
Imports System.Runtime.InteropServices


Namespace WinUsbDemo
      '''<summary >
      ' API declarations relating to device management (SetupDixxx and
      ' RegisterDeviceNotification functions).
      ''' </summary>

      Friend NotInheritable Partial Class DeviceManagement
            Friend NotInheritable Class NativeMethods

                  Private Sub New()
                  End Sub
                  ' from setupapi.h

                  Friend Const DIGCF_PRESENT As Int32 = 2
                  Friend Const DIGCF_DEVICEINTERFACE As Int32 = &H10

                  Friend Structure SP_DEVICE_INTERFACE_DATA
                        Friend cbSize As Int32
                        Friend InterfaceClassGuid As Guid
                        Friend Flags As Int32
                        Friend Reserved As IntPtr
```

```
                End Structure

                Friend Structure SP_DEVICE_INTERFACE_DETAIL_DATA
                      Friend cbSize As Int32
                      Friend DevicePath As String
                End Structure

                Friend Structure SP_DEVINFO_DATA
                      Friend cbSize As Int32
                      Friend ClassGuid As Guid
                      Friend DevInst As Int32
                      Friend Reserved As Int32
                End Structure

                <DllImport("setupapi.dll", SetLastError := True)> _
                Friend Shared Function SetupDiCreateDeviceInfoList(ByRef
ClassGuid As Guid, ByVal hwndParent As IntPtr) As IntPtr
                End Function

                <DllImport("setupapi.dll", SetLastError := True)> _
                Friend Shared Function SetupDiDestroyDeviceInfoList(ByVal
DeviceInfoSet As IntPtr) As Int32
                End Function

                <DllImport("setupapi.dll", SetLastError := True)> _
                Friend Shared Function SetupDiEnumDeviceInterfaces(ByVal
DeviceInfoSet As IntPtr, ByVal DeviceInfoData As IntPtr, ByRef
InterfaceClassGuid As Guid, ByVal MemberIndex As Int32, ByRef
DeviceInterfaceData As SP_DEVICE_INTERFACE_DATA) As Boolean
                End Function

                <DllImport("setupapi.dll", SetLastError := True, CharSet
:= CharSet.Auto)> _
                Friend Shared Function SetupDiGetClassDevs(ByRef
ClassGuid As Guid, ByVal Enumerator As IntPtr, ByVal hwndParent As IntPtr,
ByVal Flags As Int32) As IntPtr
                End Function

                <DllImport("setupapi.dll", SetLastError := True, CharSet
:= CharSet.Auto)> _
                Friend Shared Function
SetupDiGetDeviceInterfaceDetail(ByVal DeviceInfoSet As IntPtr, ByRef
DeviceInterfaceData As SP_DEVICE_INTERFACE_DATA, ByVal
DeviceInterfaceDetailData As IntPtr, ByVal DeviceInterfaceDetailDataSize As
Int32, ByRef RequiredSize As Int32, ByVal DeviceInfoData As IntPtr) As
Boolean
                End Function
            End Class
      End Class
End Namespace
```

FileIOapi.vb

```
Friend Function DoControlReadTransfer(ByVal winUsbHandle As IntPtr, ByRef
dataStage() As Byte) As Boolean
                Dim bytesReturned As UInt32 = 0
```

```
                Try
                        '  Vendor-specific request to an interface with
device-to-host Data stage.

                        Dim setupPacket As
NativeMethods.WINUSB_SETUP_PACKET
                        setupPacket.RequestType = &HC1

                        '  The request number that identifies the specific
request.

                        setupPacket.Request = 2

                        '  Command-specific value to send to the device.
                        ' For a request directed to an interface, the
WinUSB driver uses this field to specify the interface number
                        ' and will ignore a value set here.
                        ' For a request directed to the device, you can use
this field for vendor-defined purposes.

                        setupPacket.Index = 0

                        '  Number of bytes in the request's Data stage.

                        setupPacket.Length =
Convert.ToUInt16(dataStage.Length)

                        '  Command-specific value to send to the device.

                        setupPacket.Value = 0

                        ' ***
                        '  winusb function

                        '  summary
                        '  Initiates a control transfer.

                        '  paramaters
                        '  Device handle returned by WinUsb_Initialize.
                        '  WINUSB_SETUP_PACKET structure
                        '  Buffer to hold the returned Data-stage data.
                        '  Number of data bytes to read in the Data stage.
                        '  Number of bytes read in the Data stage.
                        '  Null pointer for non-overlapped.

                        '  returns
                        '  True on success.
                        '  ***

                        Dim success =
NativeMethods.WinUsb_ControlTransfer(winUsbHandle, setupPacket, dataStage,
Convert.ToUInt16(dataStage.Length), bytesReturned, IntPtr.Zero)
                        Return success

                Catch ex As AccessViolationException
                        ' The handle might be closed.

                        Return False
                Catch ex As Exception
                        DisplayException(ModuleName, ex)
                        Throw
```

```
                End Try
            End Function



frmMain.vb

Imports System
Imports System.Diagnostics
Imports System.Windows.Forms
Imports Microsoft.Win32.SafeHandles
Imports System.Management
Imports System.Text

Namespace WinUsbDemo


      ''' <summary>
      ''' Project: WinUsb_vb
      '''
      '''
********************************************************************
      ''' Software License Agreement
      '''
      ''' Licensor grants any person obtaining a copy of this software
("You")
      ''' a worldwide, royalty-free, non-exclusive license, for the
duration of
      ''' the copyright, free of charge, to store and execute the Software
in a
      ''' computer system and to incorporate the Software or any portion of
it
      ''' in computer programs You write.
      '''
      ''' THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR
      ''' IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
      ''' FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE
      ''' AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER
      ''' LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,
      ''' OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN
      ''' THE SOFTWARE.
      '''
********************************************************************
      '''
      ''' Author
      ''' Jan Axelson
      '''
      ''' This software was created using Visual Studio 2012 Professional
Edition with .NET Framework 4.0.
      ''' I wrote the application in Visual C# and used Instant VB
(http://www.tangiblesoftwaresolutions.com) to convert to Visual Basic.
      '''
      ''' Purpose:
```

```
      ''' Demonstrates USB communications using Microsoft's WinUSB driver.
      '''
      ''' Requirements:
      ''' Windows XP or later and an attached USB device that uses the
WinUSB driver.
      '''
      ''' Edit VendorID and ProductID to match the values in your device's
device descriptor.
      ''' If using the Windows-provided winusb.inf, edit
DeviceInterfaceGuid to match the value in your device's extended properties
OS feature descriptor .
      ''' If using a vendor-provided INF file, edit DeviceInterfaceGuid to
match the value in your INF file.
      '''
      ''' Description:
      ''' Finds an attached device whose device firmware or host INF file
contains a specific device interface GUID.
      ''' Enables sending and receiving data via bulk, interrupt, and
control transfers.
      '''
      ''' Uses WMI to detect when a device is attached or removed.
      '''
      ''' For bulk transfers, the application uses a Delegate and the
BeginInvoke
      ''' and EndInvoke methods to read and write data asynchronously so
the application's main thread
      ''' doesn't have to wait for the device to return data. A callback
routine uses
      ''' marshaling to send data to the form, whose code runs in a
different thread.
      '''
      ''' Interrupt and control transfers read and write data
synchronously, blocking the thread
      ''' until the operation completes or a timeout. (Normally, the wait
for completion is minimal.)
      '''
      ''' This software and companion device firmware are available from
http://www.Lvr.com/winusb.htm
      '''
      ''' Report any problems to jan@Lvr.com or post to my Ports forum at
http://www.Lvr.com/forum
      ''' This application has been tested under Windows 8 (64 bit).
      '''
      ''' 10/15/13
      ''' V2.1
      ''' For detecting device arrival and removal, replaced the Win32
RegisterDeviceNotification code with .NET WMI functions.
      ''' Added try...catch blocks.
      ''' Other minor edits for clarity.
      '''
      ''' 5/3/13
      '''   V2.0
      '''   To illustrate asynchronous operations, bulk read and write are
asynchronous.
      '''
      '''   To illustrate synchronous operations, interrupt read and write
and control transfers are synchronous.
      '''
      ''' For compatibility, replaced ToInt32 with ToInt64 here:
      ''' IntPtr pDevicePathName = new IntPtr(detailDataBuffer.ToInt64() +
4);
```

```
      '''
      ''' Uncommented QueryDeviceSpeed because Microsoft corrected the
documentation for it.
      '''
      ''' Renamed some routines and variables.
      '''
      ''' Moved myDevInfo and other variables from WinUsbDevice.vb to
frmMain.vb.
      '''
      ''' Moved API declarations to classes named NativeMethods as
recommended by Microsoft.
      '''
      ''' Minor changes to API declarations as advised by Visual Studio's
Code Analysis.
      '''
      ''' Naming changes and other minor coding improvements as recommended
by the Resharper tool.
      ''' </summary>



      ''' <summary>
      ''' The application's form. Buttons enable detecting a device
      ''' and initiating transfers.
      ''' </summary>
      '''
      Friend Class FrmMain
            Inherits Form
#Region "Windows Form Designer generated code "
            '#region '"Windows Form Designer generated code "'
            Public Sub New()
                  MyBase.New()


                  ' This call is required by the Windows Form Designer.
                  InitializeComponent()
            End Sub
            ' Form overrides dispose to clean up the component list.
            Protected Overrides Sub Dispose(ByVal Disposing As Boolean)
                  If Disposing Then
                        If components IsNot Nothing Then
                              components.Dispose()
                        End If
                  End If
                  MyBase.Dispose(Disposing)
            End Sub

            ' Required by the Windows Form Designer
            Private components As System.ComponentModel.IContainer
            Public ToolTip1 As System.Windows.Forms.ToolTip
        Public LstResults As System.Windows.Forms.ListBox
        ' NOTE: The following procedure is required by the Windows Form
Designer
        ' It can be modified using the Windows Form Designer.
        ' Do not modify it using the code editor.
        Friend WithEvents CmdFindDevice As System.Windows.Forms.Button
        Friend WithEvents CmdControlReadTransfer As
System.Windows.Forms.Button

        <System.Diagnostics.DebuggerStepThrough()> _
        Private Sub InitializeComponent()
```

```
            Me.components = New System.ComponentModel.Container()
            Me.ToolTip1 = New System.Windows.Forms.ToolTip(Me.components)
            Me.CmdControlReadTransfer = New System.Windows.Forms.Button()
            Me.LstResults = New System.Windows.Forms.ListBox()
            Me.CmdFindDevice = New System.Windows.Forms.Button()
            Me.SuspendLayout()
            '
            'CmdControlReadTransfer
            '
            Me.CmdControlReadTransfer.Font = New
System.Drawing.Font("Arial", 9.0!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
            Me.CmdControlReadTransfer.Location = New
System.Drawing.Point(12, 12)
            Me.CmdControlReadTransfer.Name = "CmdControlReadTransfer"
            Me.CmdControlReadTransfer.Size = New System.Drawing.Size(290,
27)
            Me.CmdControlReadTransfer.TabIndex = 12
            Me.CmdControlReadTransfer.Text = "Sol.licitud ATR"
            Me.CmdControlReadTransfer.UseVisualStyleBackColor = True
            '
            'LstResults
            '
            Me.LstResults.BackColor = System.Drawing.SystemColors.Window
            Me.LstResults.Cursor = System.Windows.Forms.Cursors.Default
            Me.LstResults.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
CType(0, Byte))
            Me.LstResults.ForeColor =
System.Drawing.SystemColors.WindowText
            Me.LstResults.HorizontalScrollbar = True
            Me.LstResults.ItemHeight = 14
            Me.LstResults.Location = New System.Drawing.Point(12, 48)
            Me.LstResults.Name = "LstResults"
            Me.LstResults.RightToLeft = System.Windows.Forms.RightToLeft.No
            Me.LstResults.Size = New System.Drawing.Size(572, 270)
            Me.LstResults.TabIndex = 0
            '
            'CmdFindDevice
            '
            Me.CmdFindDevice.Font = New System.Drawing.Font("Arial", 9.0!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
CType(0, Byte))
            Me.CmdFindDevice.Location = New System.Drawing.Point(308, 12)
            Me.CmdFindDevice.Name = "CmdFindDevice"
            Me.CmdFindDevice.Size = New System.Drawing.Size(276, 27)
            Me.CmdFindDevice.TabIndex = 11
            Me.CmdFindDevice.Text = "Buscar lector"
            '
            'FrmMain
            '
            Me.ClientSize = New System.Drawing.Size(596, 339)
            Me.Controls.Add(Me.CmdControlReadTransfer)
            Me.Controls.Add(Me.CmdFindDevice)
            Me.Controls.Add(Me.LstResults)
            Me.Font = New System.Drawing.Font("Arial", 8.0!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
CType(0, Byte))
            Me.Location = New System.Drawing.Point(21, 28)
            Me.Name = "FrmMain"
```

```
            Me.StartPosition =
System.Windows.Forms.FormStartPosition.Manual
            Me.Text = "PFC - ETIEI - Xavier Gelada"
            Me.ResumeLayout(False)

        End Sub

#End Region

        '
**********************************************************************
**********
        ' Device-specific values. Edit to match your device.

        ' If using the sytem-provided winusb.inf, this GUID must match the
GUID in the device's
        ' extended properties OS feature descriptor.
        ' If using a vendor-provided INF file, this GUID must match the
GUID in the INF file.
        ' The GUID should be unique to your device.
        ' To create a GUID in Visual Studio, click Tools > Create GUID.

        Private Const DeviceInterfaceGuid As String = "{36b9c3fe-5fd0-4dd9-
9c2f-561eb8ddaab4}"

        ' WMI functions use these values to detect device arrival and
removal and get device properties:

        Private Const VendorID As String = "04D8"
        Private Const ProductID As String = "A001"

        '
**********************************************************************
**********

        Private _deviceArrivedWatcher As ManagementEventWatcher
        Private _deviceDetected As Boolean
        Private _deviceHandle As SafeFileHandle
        Private _deviceNotificationHandle As IntPtr
        Private _deviceReady As Boolean
        Private _deviceRemovedWatcher As ManagementEventWatcher
        Private _myDeviceInfo As New WinUsbCommunications.DeviceInfo()
        Private ReadOnly _myDeviceManagement As New DeviceManagement()
        Private ReadOnly _myWinUsbCommunications As New
WinUsbCommunications()
        Private _winUsbHandle As IntPtr

        Private Enum FormActions
            AddItemToListBox
            EnableCmdSendandReceiveViaBulkTransfers
            EnableCmdSendandReceiveViaInterruptTransfers
            ScrollToBottomOfListBox
        End Enum

        Private Enum WmiDeviceProperties
            Caption
            Description
            Manufacturer
            Name
            CompatibleID ' Returns System.String[]
            PNPDeviceID
```

```
            DeviceID
            ClassGUID
            'Availability ' Always returns zero.
        End Enum

        Friend FrmMy As FrmMain

        '''  <summary>
        '''  Define a delegate with the same parameters as AccessForm.
        '''  Used in accessing the application's form from a different
thread.
        '''  </summary>

        Private Delegate Sub MarshalToForm(ByVal action As String, ByVal
textToAdd As String)

        '''  <summary>
        '''  Define a class of delegates with the same parameters as
        '''  WinUsbDevice.ReadViaBulkTransfer.
        '''  Used for asynchronous reads from the device.
        '''  </summary>

        Private Delegate Sub ReceiveFromDeviceDelegate(ByVal winUsbHandle
As IntPtr, ByVal myDeviceInfo As WinUsbCommunications.DeviceInfo, ByVal
bytesToRead As UInt32, ByRef dataBuffer() As Byte, ByRef bytesRead As
UInt32, ByRef success As Boolean)

        '''  <summary>
        '''  Define a class of delegates with the same parameters as
        '''  WinUsbDevice.SendViaBulkTransfer.
        '''  Used for asynchronous writes to the device.
        '''  </summary>

        Private Delegate Sub SendToDeviceDelegate(ByVal winUsbHandle As
IntPtr, ByVal myDevInfo As WinUsbCommunications.DeviceInfo, ByVal
bufferLength As UInt32, ByVal buffer() As Byte, ByRef lengthTransferred As
UInt32, ByRef success As Boolean)

        '''  <summary>
        '''  Performs various application-specific functions that
        '''  involve accessing the application's form.
        '''  </summary>
        '''
        '''  <param name="action"> A String that names the action to
perform on the form. </param>
        '''  <param name="formText"> Text to display on the form or an
empty String. </param>
        '''
        ''' <remarks>
        '''  In asynchronous calls to WinUsb_ReadPipe, the callback
function
        '''  uses this routine to access the application's form, which runs
in
        '''  a different thread.
        ''' </remarks>
        '''
        Private Sub AccessForm(ByVal action As String, ByVal formText As
String)
            Try
                '  Select an action to perform on the form:
```

```
                Select Case action
                    Case "AddItemToListBox"

                        LstResults.Items.Add(formText)

                    Case "ScrollToBottomOfListBox"

                        LstResults.SelectedIndex = LstResults.Items.Count -
1
                End Select
            Catch ex As Exception
                DisplayException(Name, ex)
                Throw
            End Try
        End Sub

        '''  <summary>
        '''  Add a handler to detect arrival of devices.
        '''  </summary>

        Private Sub AddDeviceArrivedHandler()
            Const pollingIntervalSeconds As Int32 = 3
            Dim scope = New ManagementScope("root\CIMV2")
            scope.Options.EnablePrivileges = True

            Try
                Dim q = New WqlEventQuery()
                q.EventClassName = "__InstanceCreationEvent"
                q.WithinInterval = New TimeSpan(0, 0,
pollingIntervalSeconds)
                q.Condition = "TargetInstance ISA
'Win32_USBControllerdevice'"
                _deviceArrivedWatcher = New ManagementEventWatcher(scope,
q)
                AddHandler _deviceArrivedWatcher.EventArrived, AddressOf
DeviceAdded

                _deviceArrivedWatcher.Start()
            Catch e As Exception
                Console.WriteLine(e.Message)
                If _deviceArrivedWatcher IsNot Nothing Then
                    _deviceArrivedWatcher.Stop()
                End If
            End Try
        End Sub

        '''  <summary>
        '''  Add a handler to detect removal of devices.
        '''  </summary>

        Private Sub AddDeviceRemovedHandler()
            Const pollingIntervalSeconds As Int32 = 3
            Dim scope = New ManagementScope("root\CIMV2")
            scope.Options.EnablePrivileges = True

            Try
                Dim q = New WqlEventQuery()
                q.EventClassName = "__InstanceDeletionEvent"
                q.WithinInterval = New TimeSpan(0, 0,
pollingIntervalSeconds)
```

```
                q.Condition = "TargetInstance ISA
'Win32_USBControllerdevice'"
                _deviceRemovedWatcher = New ManagementEventWatcher(scope,
q)
                AddHandler _deviceRemovedWatcher.EventArrived, AddressOf
DeviceRemoved
                _deviceRemovedWatcher.Start()
            Catch e As Exception
                Console.WriteLine(e.Message)
                If _deviceRemovedWatcher IsNot Nothing Then
                    _deviceRemovedWatcher.Stop()
                End If
            End Try
        End Sub

        ''' <summary>
        ''' Calls a routine to initiate a Control Read transfer. (Data
stage is device to host.)
        ''' </summary>

        Private Sub cmdControlReadTransfer_Click(ByVal sender As Object,
ByVal e As EventArgs) Handles CmdControlReadTransfer.Click
            RequestControlReadTransfer()
        End Sub

        ''' <summary>
        ''' Calls a routine to initiate a Control Write transfer. (Data
stage is host to device.)
        ''' </summary>
        '''


        ''' <summary>
        ''' Search for a specific device.
        ''' </summary>

        Private Sub cmdFindDevice_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles CmdFindDevice.Click
            FindMyDevice()
        End Sub

        ''' <summary>
        ''' Calls a routine to exchange data via bulk transfers.
        ''' </summary>



        ''' <summary>
        ''' Calls a routine to exchange data via interrupt transfers.
        ''' </summary>
        '''


            ''' <summary>
            ''' Called on arrival of any device.
            ''' Calls a routine that searches to see if the desired device
is present.
            ''' </summary>

            Private Sub DeviceAdded(ByVal sender As Object, ByVal e As
EventArrivedEventArgs)
```

```
                Try
               Console.WriteLine("S'ha connectat un dispositiu USB")

                      _deviceDetected = FindDeviceUsingWmi()
                Catch ex As Exception
                      DisplayException(Name, ex)
                      Throw
                End Try
          End Sub

          '''  <summary>
          '''  Add handlers to detect device arrival and removal.
          '''  </summary>

          Private Sub DeviceNotificationsStart()
                AddDeviceArrivedHandler()
                AddDeviceRemovedHandler()
          End Sub

          '''  <summary>
          '''  Stop receiving notifications about device arrival and
removal
          '''  </summary>

          Private Sub DeviceNotificationsStop()
                Try
                      If _deviceArrivedWatcher IsNot Nothing Then
                            _deviceArrivedWatcher.Stop()
                      End If
                      If _deviceRemovedWatcher IsNot Nothing Then
                            _deviceRemovedWatcher.Stop()
                      End If
                Catch ex As Exception
                      DisplayException(Name, ex)
                      Throw
                End Try
          End Sub

          '''  <summary>
          '''  Called on removal of any device.
          '''  Calls a routine that searches to see if the desired device
is still present.
          '''  </summary>
          '''
          Private Sub DeviceRemoved(ByVal sender As Object, ByVal e As
EventArgs)
                Try
               Console.WriteLine("S'ha desconnectat un dispositiu USB")

                      _deviceDetected = FindDeviceUsingWmi()
                Catch ex As Exception
                      DisplayException(Name, ex)
                      Throw
                End Try

          End Sub

          '''  <summary>
          '''  Get and display the device's speed in the list box.
          '''  </summary>
```

```
            Private Sub DisplayDeviceSpeed()
                  Try
                        Dim speed = ""


      WinUsbCommunications.QueryDeviceSpeed(_winUsbHandle, _myDeviceInfo)

                        Select Case _myDeviceInfo.DeviceSpeed
                              Case 1
                         speed = "velocitat LS o FS"
                              Case 3
                         speed = "velocitat HS o SS"
                        End Select
                    LstResults.Items.Add(" ")
                    LstResults.Items.Add("El dispositiu admet " & speed)
                    LstResults.Items.Add(" ")
                      Catch ex As Exception
                        DisplayException(Name, ex)
                        Throw
                      End Try
            End Sub

            '''  <summary>
            '''  Provides a central mechanism for exception handling.
            '''  Displays a message box that describes the exception.
            '''  </summary>
            '''
            '''  <param name="Name"> the module where the exception
occurred. </param>
            '''  <param name="e"> the exception </param>

            Friend Shared Sub DisplayException(ByVal Name As String, ByVal
e As Exception)
                  Try
                        '  Create an error message.

                        Dim message As String = "Exception: " & e.Message &
Environment.NewLine & "Module: " & Name & Environment.NewLine & "Method: "
& e.TargetSite.Name

                        Const caption As String = "Unexpected Exception"

                        MessageBox.Show(message, caption,
MessageBoxButtons.OK)
                        Debug.Write(message)
                  Catch ex As Exception
                        DisplayException(Name, ex)
                        Throw
                  End Try
            End Sub

            '''  <summary>
            '''  Use WMI to find a device by Vendor ID and Product ID. If
found, display device properties.
            '''  Can also search by ClassGUID, description, and other
properties.
            '''  The WMI functions don't detect the device interface GUID
so you need to use another property or properties.
            '''  </summary>
            '''
            Private Function FindDeviceUsingWmi() As Boolean
```

```
                Try
                        ' ClassGUID value from Windows-provided winusb.inf
for use if you want to search by ClassGUID.

                Const classGuid As String = "d7231438-6472-43f9-9f61-
4522d6b00ad7"
                        Const deviceIdString As String = "USB\VID_" &
VendorID & "&PID_" & ProductID

                        _deviceDetected = False
                        Dim searcher = New
ManagementObjectSearcher("root\CIMV2", "SELECT * FROM Win32_PnPEntity")

                        For Each queryObj As ManagementObject In
searcher.Get()
                                ' More query exampless for the device search
below:

                                ' Search for any WinUSB device:
                                ' if
(queryObj["ClassGUID"].ToString().Contains(classGuid))

                                ' Search by description:
                                ' if
(queryObj["Description"].ToString().ToLower().Contains("winusb"))

                                ' Prepend "@" in string below to treat
backslash as a normal character (not escape character):

                                'const String deviceIdString = @"USB\VID_" +
VendorID + "&PID_" + ProductID;

                                If
queryObj("PNPDeviceID").ToString().Contains(deviceIdString) Then
                                        _deviceDetected = True

      MyMarshalToForm(FormActions.AddItemToListBox.ToString(), "--------")

MyMarshalToForm(FormActions.AddItemToListBox.ToString(), "Dispositiu lector
detectat (WMI):")

                                        For Each wmiDeviceProperty As
WmiDeviceProperties In System.Enum.GetValues(GetType(WmiDeviceProperties))

      'MyMarshalToForm(FormActions.AddItemToListBox.ToString(),
CType((wmiDeviceProperty.ToString() & ": " &
queryObj(wmiDeviceProperty.ToString()).ToString), String))

      MyMarshalToForm(FormActions.AddItemToListBox.ToString(),
wmiDeviceProperty.ToString() & ": " &
queryObj(wmiDeviceProperty.ToString()).ToString)

      Console.WriteLine(wmiDeviceProperty.ToString() & ": {0}",
queryObj(wmiDeviceProperty.ToString()))
                                        Next wmiDeviceProperty

      MyMarshalToForm(FormActions.AddItemToListBox.ToString(), "--------")

      MyMarshalToForm(FormActions.ScrollToBottomOfListBox.ToString(), "")
                                End If
                        Next queryObj
```

```
                        If Not _deviceDetected Then
                              _deviceReady = False

MyMarshalToForm(FormActions.AddItemToListBox.ToString(), "Dispositiu lector
no detectat (WMI)")

      MyMarshalToForm(FormActions.ScrollToBottomOfListBox.ToString(), "")
                        End If
                        Return _deviceDetected
                  Catch ex As Exception
                        DisplayException(Name, ex)
                        Throw
                  End Try
            End Function

            '''  <summary>
            '''  If a device with the specified device interface GUID
hasn't been previously detected,
            '''  look for it. If found, open a handle to the device.
            '''  </summary>
            '''
            '''  <returns>
            '''  True if the device is detected, False if not detected.
            '''  </returns>

            Private Sub FindMyDevice()
                  Try
                        Dim devicePathName = ""
                        Dim lastDevice As Boolean
                    Const pipeTimeout As UInteger = 9000

                        If Not (_deviceReady) Then
                              ' Convert the device interface GUID String to
a GUID object:

                              Dim winUsbDemoGuid = New
Guid(DeviceInterfaceGuid)

                              ' Fill an array with the device path names of
all attached devices with matching GUIDs.

                              Dim deviceFound =
_myDeviceManagement.FindDeviceFromGuid(winUsbDemoGuid, devicePathName)

                              If deviceFound Then
                                    _deviceHandle =
_myWinUsbCommunications.GetDeviceHandle(devicePathName)

                                    If Not _deviceHandle.IsInvalid Then
                                          _deviceReady = True


      _myWinUsbCommunications.InitializeDevice(_deviceHandle,
_winUsbHandle, _myDeviceInfo, pipeTimeout)

                                          DisplayDeviceSpeed()
                                    Else
                                          ' There was a problem in
retrieving the information.

                                          _deviceReady = False
```

```
_myWinUsbCommunications.CloseDeviceHandle(_deviceHandle, _winUsbHandle)
                            LstResults.Items.Add(" ")
                            LstResults.Items.Add("Dispositiu lector no
detectat.")
                            LstResults.Items.Add(" ")
                                   End If
                              Else

                              End If
                Else
                    LstResults.Items.Add(" ")
                    LstResults.Items.Add("Dispositiu lector detectat.")
                    LstResults.Items.Add(" ")
                    DisplayDeviceSpeed()
                        End If

                        ' Display device information.

                        FindDeviceUsingWmi()
                        ScrollToBottomOfListBox()
                  Catch ex As Exception
                        DisplayException(Name, ex)
                        Throw
                  End Try
            End Sub

            '''  <summary>
            '''  Perform shutdown operations.
            '''  </summary>

            Private Sub frmMain_Closed(ByVal eventSender As Object, ByVal
eventArgs As EventArgs) Handles MyBase.Closed
                  Shutdown()
            End Sub

            '''  <summary>
            '''  Perform startup operations.
            '''  </summary>

            Private Sub frmMain_Load(ByVal eventSender As Object, ByVal
eventArgs As EventArgs) Handles MyBase.Load
                  FrmMy = Me
                  Startup()
            End Sub

            '''  <summary>
            '''  Retrieves received data from a bulk endpoint.
            '''  This routine is called automatically when
myWinUsbDevice.ReadViaBulkTransfer
            '''  returns. The routine calls several marshaling routines to
access the main form.
            '''  </summary>
            '''
            '''  <param name="ar"> An object containing status information
about the
            '''  asynchronous operation.</param>
            '''
            Private Sub GetBulkDataSent(ByVal ar As IAsyncResult)
                  Try
                        Dim bytesSent As UInt32 = 0
```

```
                        Dim myEncoder = New ASCIIEncoding()
                        Dim sentDataBuffer() As Byte
                        Dim receivedtext As String = ""
                        Dim success As Boolean = False

                        ' Define a delegate using the IAsyncResult object.

                        Dim deleg = (DirectCast(ar.AsyncState,
SendToDeviceDelegate))

                        ' Get the IAsyncResult object and the values of
other paramaters that the
                        ' BeginInvoke method passed by reference.

                        deleg.EndInvoke(bytesSent, success, ar)

                        ' Display the received data in the form's list box.

                        If (ar.IsCompleted AndAlso success) Then

      MyMarshalToForm(FormActions.AddItemToListBox.ToString(), "Data sent
via bulk transfer:")
                        Else

      MyMarshalToForm(FormActions.AddItemToListBox.ToString(), "The attempt
to send bulk data has failed.")
                        End If


      MyMarshalToForm(FormActions.ScrollToBottomOfListBox.ToString(), "")

                        ' Enable requesting another transfer.


      MyMarshalToForm(FormActions.EnableCmdSendandReceiveViaBulkTransfers.T
oString(), "")
                  Catch ex As Exception
                        DisplayException(Name, ex)
                        Throw
                  End Try
            End Sub

            ''' <summary>
            ''' Retrieves received data from a bulk endpoint.
            ''' This routine is called automatically when
myWinUsbDevice.ReceiveViaBulkTransfer
            ''' returns. The routine calls several marshaling routines to
access the main form.
            ''' </summary>
            '''
            ''' <param name="ar"> An object containing status information
about the
            ''' asynchronous operation.</param>
            '''
            Private Sub GetBulkDataReceived(ByVal ar As IAsyncResult)
                  Try
                        Dim bytesRead As UInt32 = 0
                        Dim myEncoder = New ASCIIEncoding()
                        Dim success = False

                        Dim receivedDataBuffer() As Byte = Nothing
```

```
                        ' Define a delegate using the IAsyncResult object.

                        Dim deleg = (DirectCast(ar.AsyncState,
ReceiveFromDeviceDelegate))

                        ' Get the IAsyncResult object and the values of
other paramaters that the
                        ' BeginInvoke method passed ByRef.

                        deleg.EndInvoke(receivedDataBuffer, bytesRead,
success, ar)

                        ' Display the received data in the form's list box.

                        If (ar.IsCompleted AndAlso success) Then

      MyMarshalToForm(FormActions.AddItemToListBox.ToString(), "Data
received via bulk transfer:")

                              '  Convert the received bytes to a String for
display.

                              Dim receivedtext As String =
myEncoder.GetString(receivedDataBuffer)


      MyMarshalToForm(FormActions.AddItemToListBox.ToString(),
receivedtext)
                        Else

      MyMarshalToForm(FormActions.AddItemToListBox.ToString(), "The attempt
to read bulk data has failed.")
                        End If


      MyMarshalToForm(FormActions.ScrollToBottomOfListBox.ToString(), "")

                        ' Enable requesting another transfer.


      MyMarshalToForm(FormActions.EnableCmdSendandReceiveViaBulkTransfers.T
oString(), "")
                  Catch ex As Exception
                        DisplayException(Name, ex)
                        Throw
                  End Try
            End Sub

            ''' <summary>
            ''' Initializes elements on the form.
            ''' </summary>

            Private Sub InitializeDisplay()
                  Try
                        ' Create dropdown list boxes.

                If Not (_myWinUsbCommunications.IsWindowsXpOrLater()) Then
                    LstResults.Items.Add(" ")
                    LstResults.Items.Add("El sistema operatiu no és Windows
XP o superior.")
```

```
                    LstResults.Items.Add("El driver WinUsb requereix
Windows XP o superior.")
                    LstResults.Items.Add(" ")
                End If
                  Catch ex As Exception
                        DisplayException(Name, ex)
                        Throw
                  End Try
            End Sub

            '''  <summary>
            '''  Enables accessing a form from another thread
            '''  </summary>
            '''
            '''  <param name="action"> A String that names the action to
perform on the form. </param>
            '''  <param name="textToDisplay"> Text that the form displays
or uses for
            '''  another purpose. Actions that don't use text ignore this
parameter. </param>

            Private Sub MyMarshalToForm(ByVal action As String, ByVal
textToDisplay As String)
                  Try
                        Dim args() As Object = {action, textToDisplay}

                        '  The AccessForm routine contains the code that
accesses the form.

                        Dim marshalToFormDelegate As MarshalToForm =
AddressOf AccessForm

                        '  Execute AccessForm, passing the parameters in
args.

                        Invoke(marshalToFormDelegate, args)
                  Catch ex As Exception
                        DisplayException(Name, ex)
                        Throw
                  End Try
            End Sub

            '''  <summary>
            '''  Initiates a Control Read transfer. (Data stage is device
to host.)
            '''  </summary>
            '''
            Private Sub RequestControlReadTransfer()
                  Try
                Dim dataStage = New Byte(35) {}

                        ' If the device hasn't been detected or was
removed, look for the device.

                        If Not (_deviceReady) Then
                              FindMyDevice()
                        End If

                        If _deviceReady Then
                              Dim success =
_myWinUsbCommunications.DoControlReadTransfer(_winUsbHandle, dataStage)
```

```
                    Dim formText As String
                    Dim Tall_prova As String
                    Dim Cadena_Origin As String
                    Dim Longitud_Cadena As Integer

                         If success Then
                        formText = "Bytes rebuts a través de transferència
CONTROL:"
                        AccessForm(FormActions.AddItemToListBox.ToString(),
" ")
                        AccessForm(FormActions.AddItemToListBox.ToString(),
formText)

                        formText = ""
                                   For i As Int32 = 0 To dataStage.Length -
1
                            formText = formText & String.Format("{0:X2} ",
dataStage(i)) & " "
                        Next i

                        Cadena_Origin = formText
                        Longitud_Cadena = Len(Cadena_Origin)

                        While True
                            Tall_prova =
Microsoft.VisualBasic.Right(Cadena_Origin, 4)
                            If Tall_prova = "00  " Then
                                Cadena_Origin =
Microsoft.VisualBasic.Left(Cadena_Origin, Longitud_Cadena - 4)
                                Longitud_Cadena = Len(Cadena_Origin)
                            Else
                                Cadena_Origin =
Microsoft.VisualBasic.Left(Cadena_Origin, Longitud_Cadena - 4)
                                Exit While
                            End If
                        End While

                        Cadena_Origin = Trim(Cadena_Origin)

                        If Cadena_Origin = "00" Then
                            formText = "No es detecta targeta insertada."
                        ElseIf Cadena_Origin = "EF" Then
                            formText = "La resposta de la targeta ha acabat
prematurament."
                        ElseIf Cadena_Origin = "FF" Then
                            formText = "La targeta insertada no respon al
RESET."
                        Else
                            formText = Cadena_Origin
                        End If

                        AccessForm(FormActions.AddItemToListBox.ToString(),
formText)

                    Else
                        formText = "La transferència CONTROL ha fallat."
                        AccessForm(FormActions.AddItemToListBox.ToString(),
" ")
                        AccessForm(FormActions.AddItemToListBox.ToString(),
formText)
```

```
                End If
            End If
            ScrollToBottomOfListBox()
        Catch ex As Exception
            DisplayException(Name, ex)
            Throw
        End Try
        End Sub

        '''  <summary>
        '''  Initiates a Control Write transfer. (Data stage is host to
device.)
        '''  Converts hex strings from combo boxes to byte values for
sending.
        '''  </summary>
        '''


        '''  <summary>
        '''  Initiates a read operation from a bulk IN endpoint.
        '''  To enable reading without blocking the main thread, uses
an asynchronous delegate.
        '''  </summary>
        '''
        '''  <param name="bytesToRead"> The number of bytes to read
</param>
        '''  <param name="dataBuffer"> Buffer to hold the bytes read
</param>
        '''  <param name="bytesRead"> The number of bytes read </param>
        '''  <param name="success"> True on success </param>

        Private Sub RequestToReceiveDataViaBulkTransfer(ByVal
bytesToRead As UInt32, ByVal dataBuffer() As Byte, ByRef bytesRead As
UInt32, ByRef success As Boolean)
            Try
                If dataBuffer Is Nothing Then
                    Throw New ArgumentNullException("dataBuffer")
                End If
                Dim ar As IAsyncResult = Nothing

                '  Define a delegate for the ReadViaBulkTransfer
method of WinUsbDevice.

                Dim myReceiveFromDeviceDelegate As New
ReceiveFromDeviceDelegate(AddressOf
_myWinUsbCommunications.ReceiveDataViaBulkTransfer)
                'Dim myReceiveFromDeviceDelegate As
ReceiveFromDeviceDelegate =
_myWinUsbCommunications.ReceiveDataViaBulkTransfer

                '  The BeginInvoke method calls
MyWinUsbDevice.ReceiveViaBulkTransfer to attempt
                '  to read data. The method has the same parameters
as ReceiveViaBulkTransfer,
                '  plus two additional parameters:
                '  GetBulkDataReceived is the callback routine that
executes when
                '  ReceiveViaBulkTransfer returns.
                '  MyReceiveFromDeviceDelegate is the asynchronous
delegate object.
```

```
      myReceiveFromDeviceDelegate.BeginInvoke(_winUsbHandle, _myDeviceInfo,
bytesToRead, dataBuffer, bytesRead, success, AddressOf GetBulkDataReceived,
myReceiveFromDeviceDelegate)
                  Catch ex As Exception
                        DisplayException(Name, ex)
                        Throw
                  End Try
            End Sub

            ''' <summary>
            ''' Initiates a read operation from an interrupt IN endpoint.
            ''' Synchronous, blocks until data received or timeout.
            ''' </summary>
            '''
            ''' <param name="bytesToRead"> The number of bytes to read
</param>
            ''' <param name="dataBuffer"> Buffer to hold the bytes read
</param>
            ''' <param name="bytesRead"> The number of bytes read </param>
            ''' <param name="success"> True on success </param>


            Private Sub RequestToReceiveDataViaInterruptTransfer(ByVal
bytesToRead As UInt32, ByVal dataBuffer() As Byte, ByRef bytesRead As
UInt32, ByRef success As Boolean)
                  Try
                        If dataBuffer Is Nothing Then
                              Throw New ArgumentNullException("dataBuffer")
                        End If
                        success =
_myWinUsbCommunications.ReceiveDataViaInterruptTransfer(_winUsbHandle,
_myDeviceInfo, bytesToRead, dataBuffer, bytesRead)

                        If success Then

      MyMarshalToForm(FormActions.AddItemToListBox.ToString(), "Data
received via interrupt transfer:")


      MyMarshalToForm(FormActions.AddItemToListBox.ToString(), " Received
Data:")

                              For i As Int32 = 0 To
dataBuffer.GetUpperBound(0)
                                    ' Convert the byte value to a 2-
character hex String.

                                    Dim byteValue = String.Format("{0:X2} ",
dataBuffer(i))


      MyMarshalToForm(FormActions.AddItemToListBox.ToString(), " " &
byteValue)
                              Next i
                        Else

      MyMarshalToForm(FormActions.AddItemToListBox.ToString(), "The attempt
to read interrupt data has failed.")
                        End If
```

```
      MyMarshalToForm(FormActions.ScrollToBottomOfListBox.ToString(), "")

                        ' Enable requesting another transfer.


      MyMarshalToForm(FormActions.EnableCmdSendandReceiveViaInterruptTransf
ers.ToString(), "")
                  Catch ex As Exception
                        DisplayException(Name, ex)
                        Throw
                  End Try
            End Sub

            '''  <summary>
            '''  Initiates a write operation to a bulk OUT endpoint.
            '''  To enable writing without blocking the main thread, uses
an asynchronous delegate.
            '''  </summary>
            '''
            '''  <param name="bytesToSend"> The number of bytes to send
</param>
            '''  <param name="dataBuffer"> Buffer to hold the bytes to send
</param>
            '''  <param name="bytesSent"> The number of bytes send </param>
            '''  <param name="success"> True on success </param>

            Private Sub RequestToSendDataViaBulkTransfer(ByVal bytesToSend
As UInt32, ByVal dataBuffer() As Byte, ByRef bytesSent As UInt32, ByRef
success As Boolean)
                  Try
                        If dataBuffer Is Nothing Then
                              Throw New ArgumentNullException("dataBuffer")
                        End If
                        Dim ar As IAsyncResult = Nothing

                        '  Define a delegate for the ReadViaBulkTransfer
method of WinUsbDevice.
                        Dim mySendToDeviceDelegate As _
                        New SendToDeviceDelegate(AddressOf
_myWinUsbCommunications.SendDataViaBulkTransfer)


                        'Dim mySendToDeviceDelegate As SendToDeviceDelegate
= _myWinUsbCommunications.SendDataViaBulkTransfer

                        '  The BeginInvoke method calls
MyWinUsbDevice.SendViaBulkTransfer to attempt
                        '  to read data. The method has the same parameters
as SendViaBulkTransfer,
                        '  plus two additional parameters:
                        '  GetBulkDataSent is the callback routine that
executes when
                        '  SendViaBulkTransfer returns.
                        '  MySendToDeviceDelegate is the asynchronous
delegate object.

                        mySendToDeviceDelegate.BeginInvoke(_winUsbHandle,
_myDeviceInfo, bytesToSend, dataBuffer, bytesSent, success, AddressOf
GetBulkDataSent, mySendToDeviceDelegate)
```

```
                Catch ex As Exception
                      DisplayException(Name, ex)
                      Throw
                End Try
          End Sub

          ''' <summary>
          ''' Initiates a write operation to an interrupt OUT endpoint.
          ''' Synchronous, blocks until data sent or timeout.
          ''' </summary>
          '''
          ''' <param name="bytesToSend"> The number of bytes to send
</param>
          ''' <param name="dataBuffer"> Buffer to hold the bytes to send
</param>
          ''' <param name="bytesSent"> The number of bytes send </param>
          ''' <param name="success"> True on success </param>

          Private Sub RequestToSendDataViaInterruptTransfer(ByVal
bytesToSend As UInt32, ByVal dataBuffer() As Byte, ByRef bytesSent As
UInt32, ByRef success As Boolean)
                Try
                      If dataBuffer Is Nothing Then
                            Throw New ArgumentNullException("dataBuffer")
                      End If
                      success =
_myWinUsbCommunications.SendDataViaInterruptTransfer(_winUsbHandle,
_myDeviceInfo, bytesToSend, dataBuffer, bytesSent)

                      If success Then

      MyMarshalToForm(FormActions.AddItemToListBox.ToString(), "Data sent
via interrupt transfer:")
                      Else

      MyMarshalToForm(FormActions.AddItemToListBox.ToString(), "The attempt
to send interrupt data has failed.")
                      End If


      MyMarshalToForm(FormActions.ScrollToBottomOfListBox.ToString(), "")

                      ' Enable requesting another transfer.


      MyMarshalToForm(FormActions.EnableCmdSendandReceiveViaInterruptTransf
ers.ToString(), "")
                Catch ex As Exception
                      DisplayException(Name, ex)
                      Throw
                End Try
          End Sub

          ''' <summary>
          ''' Scroll to the bottom of the list box and trim if needed.
          ''' </summary>

          Private Sub ScrollToBottomOfListBox()
                Try
                      LstResults.SelectedIndex = LstResults.Items.Count -
1
```

```
                        ' If the list box is getting too large, trim its
contents.

                        If LstResults.Items.Count > 1000 Then
                              For i As Int32 = 1 To 800
                                    LstResults.Items.RemoveAt(i)
                              Next i
                        End If
                  Catch ex As Exception
                        DisplayException(Name, ex)
                        Throw
                  End Try
            End Sub

            '''  <summary>
            '''  Initiates sending data via a bulk transfer, then receiving
data via a bulk transfer.
            '''  </summary>



            '''  <summary>
            '''  Initiates sending data via an interrupt transfer, then
retrieving data
            '''  via an interrupt transfer.
            '''  Converts hex strings from combo boxes to byte values for
sending.
            '''  </summary>



            '''  <summary>
            '''  Perform actions that must execute when the program ends.
            '''  </summary>

            Private Sub Shutdown()
                  Try

      _myWinUsbCommunications.CloseDeviceHandle(_deviceHandle,
_winUsbHandle)


                        DeviceNotificationsStop()
                  Catch ex As Exception
                        DisplayException(Name, ex)
                        Throw
                  End Try
            End Sub

            '''  <summary>
            '''  Perform actions that must execute when the program starts.
            '''  </summary>

            Private Sub Startup()
                  Try
                        Dim myWinUsbCommunications = New
WinUsbCommunications()
                        InitializeDisplay()

                        DeviceNotificationsStart()
```

```
                        FindMyDevice()
                    Catch ex As Exception
                        DisplayException(Name, ex)
                        Throw
                    End Try
                End Sub
        End Class
End Namespace
```

WinUsbDemo.vb

```
Imports System
Imports System.Windows.Forms

Namespace WinUsbDemo
      '''<summary>
      ''' Runs the application.
      '''</summary>
      '''
      Public Class WinUsbDemo

            'internal static FrmMain FrmMy;

            ''' <summary>
            '''  Displays the application's main form.
            ''' </summary>

             <STAThread> _
             Public Shared Sub Main()
                  Dim frmMain1 = New FrmMain()
                  Application.Run(frmMain1)
             End Sub
      End Class
End Namespace
```

WinUsbDevice.vb

```
Imports System
Imports Microsoft.Win32.SafeHandles
Imports System.Diagnostics
Imports System.Windows.Forms

Namespace WinUsbDemo
      ''' <summary>
      '''  Routines for the WinUsb driver.
      '''  </summary>
      '''  <remarks>
      '''  I could find no way to test for a valid winUsbHandle. If the
handle is
      '''  already closed, the application gets an
AccessViolationException, which the
```

```
      '''  routines capture and ignore.
      ''' </remarks>

      Friend NotInheritable Partial Class WinUsbCommunications
            ''' <summary>
            ''' Stores information about the WinUSB device.
            ''' </summary>

        Private Const ModuleName As String = "WinUSB Device"

            Public Class DeviceInfo
                  Friend BulkInPipe As Byte
                  Friend BulkOutPipe As Byte
                  Friend InterruptInPipe As Byte
                  Friend InterruptOutPipe As Byte
                  Friend DeviceSpeed As UInt32
            End Class

            '''  <summary>
            '''  Closes the device handle obtained with CreateFile and
frees resources.
            '''  </summary>
            '''  <param name="deviceHandle"> Device handle obtained with
CreateFile </param>
            '''  <param name="winUsbHandle"> Handle for accessing the
WinUSB device </param>

            Friend Sub CloseDeviceHandle(ByVal deviceHandle As
SafeFileHandle, ByVal winUsbHandle As IntPtr)
                  Try
                        NativeMethods.WinUsb_Free(winUsbHandle)

                        If deviceHandle IsNot Nothing Then
                              If Not(deviceHandle.IsInvalid) Then
                                    deviceHandle.Close()
                              End If
                        End If
                  Catch ex As AccessViolationException
                        ' The handle might be closed.

                  Catch ex As Exception
                        DisplayException(ModuleName, ex)
                        Throw
                  End Try
            End Sub

            '''  <summary>
            '''  Provides a central mechanism for exception handling.
            '''  Displays a message box that describes the exception.
            '''  </summary>
            '''
            '''  <param name="Name"> the module where the exception
occurred. </param>
            '''  <param name="e"> the exception </param>

            Friend Shared Sub DisplayException(ByVal Name As String, ByVal
e As Exception)
                  Try
                        '  Create an error message.
```

```
                        Dim message As String = "Exception: " & e.Message &
Environment.NewLine & "Module: " & Name & Environment.NewLine & "Method: "
& e.TargetSite.Name

                        Const caption As String = "Unexpected Exception"

                        MessageBox.Show(message, caption,
MessageBoxButtons.OK)
                        Debug.Write(message)
                  Catch ex As Exception
                        DisplayException(ModuleName, ex)
                        Throw
                  End Try
            End Sub

            '''  <summary>
            '''  Initiates a Control Read transfer. Data stage is device to
host.
            '''  </summary>
            '''
            '''  <param name="winUsbHandle"> Handle for accessing the
WinUSB device </param>
            '''  <param name="dataStage"> The received data </param>
            '''
            '''  <returns>
            '''  True on success, False on failure.
            '''  </returns>

            Friend Function DoControlReadTransfer(ByVal winUsbHandle As
IntPtr, ByRef dataStage() As Byte) As Boolean
                  Dim bytesReturned As UInt32 = 0

                  Try
                        '  Vendor-specific request to an interface with
device-to-host Data stage.

                        Dim setupPacket As
NativeMethods.WINUSB_SETUP_PACKET
                        setupPacket.RequestType = &HC1

                        '  The request number that identifies the specific
request.

                        setupPacket.Request = 2

                        '  Command-specific value to send to the device.
                        ' For a request directed to an interface, the
WinUSB driver uses this field to specify the interface number
                        ' and will ignore a value set here.
                        ' For a request directed to the device, you can use
this field for vendor-defined purposes.

                        setupPacket.Index = 0

                        '  Number of bytes in the request's Data stage.

                        setupPacket.Length =
Convert.ToUInt16(dataStage.Length)

                        '  Command-specific value to send to the device.
```

```
                        setupPacket.Value = 0

                        ' ***
                        '  winusb function

                        '  summary
                        '  Initiates a control transfer.

                        '  paramaters
                        '  Device handle returned by WinUsb_Initialize.
                        '  WINUSB_SETUP_PACKET structure
                        '  Buffer to hold the returned Data-stage data.
                        '  Number of data bytes to read in the Data stage.
                        '  Number of bytes read in the Data stage.
                        '  Null pointer for non-overlapped.

                        '  returns
                        '  True on success.
                        '  ***

                        Dim success =
NativeMethods.WinUsb_ControlTransfer(winUsbHandle, setupPacket, dataStage,
Convert.ToUInt16(dataStage.Length), bytesReturned, IntPtr.Zero)
                        Return success

                  Catch ex As AccessViolationException
                        ' The handle might be closed.

                        Return False
                  Catch ex As Exception
                        DisplayException(ModuleName, ex)
                        Throw
                  End Try
            End Function

            '''  <summary>
            '''  Initiates a Control Write transfer. Data stage is host to
device.
            '''  </summary>
            '''
            '''  <param name="winUsbHandle"> Handle for accessing the
WinUSB device </param>
            '''  <param name="dataStage"> The data to send. </param>
            '''
            '''  <returns>
            '''  True on success, False on failure.
            '''  </returns>

            Friend Function DoControlWriteTransfer(ByVal winUsbHandle As
IntPtr, ByVal dataStage() As Byte) As Boolean
                  Dim bytesReturned As UInteger = 0
                  Dim value = Convert.ToUInt16(0)

                  Try
                        '  Vendor-specific request to an interface with
host-to-device Data stage.

                        Dim setupPacket As
NativeMethods.WINUSB_SETUP_PACKET
                        setupPacket.RequestType = &H41
```

```
                        '  The request number that identifies the specific
request.

                        setupPacket.Request = 1

                        '  Command-specific value to send to the device.
                        ' For a request directed to an interface, the
WinUSB driver uses this field to specify the interface number.
                        ' and will ignore a value set here.
                        ' For a request directed to the device, you can use
this field for vendor-defined purposes.

                        setupPacket.Index = 0

                        '  Number of bytes in the request's Data stage.

                        setupPacket.Length =
Convert.ToUInt16(dataStage.Length)

                        '  Command-specific value to send to the device.

                        setupPacket.Value = value

                        ' ***
                        '  winusb function

                        '  summary
                        '  Initiates a control transfer.

                        '  parameters
                        '  Device handle returned by WinUsb_Initialize.
                        '  WINUSB_SETUP_PACKET structure
                        '  Buffer containing the Data-stage data.
                        '  Number of data bytes to send in the Data stage.
                        '  Number of bytes sent in the Data stage.
                        '  Null pointer for non-overlapped.

                        '  Returns
                        '  True on success.
                        '  ***

                        Dim success =
NativeMethods.WinUsb_ControlTransfer(winUsbHandle, setupPacket, dataStage,
Convert.ToUInt16(dataStage.Length), bytesReturned, IntPtr.Zero)
                        Return success
                  Catch ex As AccessViolationException
                        ' The handle might be closed.
                        Return False
                  Catch ex As Exception
                        DisplayException(ModuleName, ex)
                        Throw
                  End Try
            End Function

            '''  <summary>
            '''  Requests a handle with CreateFile.
            '''  </summary>
            '''
            '''  <param name="devicePathName"> Returned by
SetupDiGetDeviceInterfaceDetail
            '''  in an SP_DEVICE_INTERFACE_DETAIL_DATA structure. </param>
```

```
            '''
            '''  <returns>
            '''  The handle.
            '''  </returns>

            Friend Function GetDeviceHandle(ByVal devicePathName As String)
As SafeFileHandle
                  Try
                        ' ***
                        ' API function

                        '  summary
                        '  Retrieves a handle to a device.

                        '  parameters
                        '  Device path name returned by
SetupDiGetDeviceInterfaceDetail
                        '  Type of access requested (read/write).
                        '  FILE_SHARE attributes to allow other processes
to access the device while this handle is open.
                        '  Security structure. Using Null for this may
cause problems under Windows XP.
                        '  Creation disposition value. Use OPEN_EXISTING
for devices.
                        '  Flags and attributes for files. The winsub
driver requires FILE_FLAG_OVERLAPPED.
                        '  Handle to a template file. Not used.

                        '  Returns
                        '  A handle or INVALID_HANDLE_VALUE.
                        ' ***

                        Dim deviceHandle As SafeFileHandle =
FileIo.NativeMethods.CreateFile(devicePathName,
(FileIo.NativeMethods.GENERIC_WRITE Or FileIo.NativeMethods.GENERIC_READ),
FileIo.NativeMethods.FILE_SHARE_READ Or
FileIo.NativeMethods.FILE_SHARE_WRITE, IntPtr.Zero,
FileIo.NativeMethods.OPEN_EXISTING,
FileIo.NativeMethods.FILE_ATTRIBUTE_NORMAL Or
FileIo.NativeMethods.FILE_FLAG_OVERLAPPED, IntPtr.Zero)
                        Return deviceHandle
                  Catch ex As Exception
                        DisplayException(ModuleName, ex)
                        Throw
                  End Try
            End Function

            '''  <summary>
            '''  Initializes a device interface and obtains information
about it.
            '''  Calls these winusb API functions:
            '''    WinUsb_Initialize
            '''    WinUsb_QueryInterfaceSettings
            '''    WinUsb_QueryPipe
            '''  </summary>
            '''
            '''  <param name="deviceHandle"> Device handle obtained with
CreateFile </param>
            '''  <param name="winUsbHandle"> Handle for accessing the
WinUSB device </param>
```

```
            '''  <param name="myDeviceInfo"> devInfo structure for the
device </param>
            '''  <param name="pipeTimeout"> desired timeout in milliseconds
for transfers </param>
            '''
            '''  <returns>
            '''  True on success, False on failure.
            '''  </returns>

            Friend Function InitializeDevice(ByVal deviceHandle As
SafeFileHandle, ByRef winUsbHandle As IntPtr, ByRef myDeviceInfo As
DeviceInfo, ByVal pipeTimeout As UInt32) As Boolean
                  Try
                        Dim ifaceDescriptor As
NativeMethods.USB_INTERFACE_DESCRIPTOR
                        ifaceDescriptor.bLength = 0
                        ifaceDescriptor.bDescriptorType = 0
                        ifaceDescriptor.bInterfaceNumber = 0
                        ifaceDescriptor.bAlternateSetting = 0
                        ifaceDescriptor.bNumEndpoints = 0
                        ifaceDescriptor.bInterfaceClass = 0
                        ifaceDescriptor.bInterfaceSubClass = 0
                        ifaceDescriptor.bInterfaceProtocol = 0
                        ifaceDescriptor.iInterface = 0

                        Dim pipeInfo As
NativeMethods.WINUSB_PIPE_INFORMATION
                        pipeInfo.PipeType = 0
                        pipeInfo.PipeId = 0
                        pipeInfo.MaximumPacketSize = 0
                        pipeInfo.Interval = 0

                        ' ***
                        '  winusb function

                        '  summary
                        '  get a handle for communications with a winusb
device        '

                        '  parameters
                        '  Handle returned by CreateFile.
                        '  Device handle to be returned.

                        '  returns
                        '  True on success.
                        '  ***

                        Dim success =
NativeMethods.WinUsb_Initialize(deviceHandle, winUsbHandle)

                        If success Then
                              ' ***
                              '  winusb function

                              '  summary
                              '  Get a structure with information about the
device interface.

                              '  parameters
                              '  handle returned by WinUsb_Initialize
                              '  alternate interface setting number
```

```
                                ' USB_INTERFACE_DESCRIPTOR structure to be
returned.

                                ' returns
                                ' True on success.

                                success =
NativeMethods.WinUsb_QueryInterfaceSettings(winUsbHandle, 0,
ifaceDescriptor)

                                If success Then
                                      ' Get the transfer type, endpoint
number, and direction for the interface's
                                      ' bulk and interrupt endpoints. Set
pipe policies.

                                      ' ***
                                      ' winusb function

                                      ' summary
                                      ' returns information about a USB pipe
(endpoint address)

                                      ' parameters
                                      ' Handle returned by WinUsb_Initialize
                                      ' Alternate interface setting number
                                      ' Number of an endpoint address
associated with the interface.
                                      ' (The values count up from zero and
are NOT the same as the endpoint address
                                      ' in the endpoint descriptor.)
                                      ' WINUSB_PIPE_INFORMATION structure to
be returned

                                      ' returns
                                      ' True on success
                                      ' ***

                                     For i = 0 To
ifaceDescriptor.bNumEndpoints - 1

      NativeMethods.WinUsb_QueryPipe(winUsbHandle, 0, Convert.ToByte(i),
pipeInfo)

                                          If ((pipeInfo.PipeType =
NativeMethods.USBD_PIPE_TYPE.UsbdPipeTypeBulk) And
UsbEndpointDirectionIn(pipeInfo.PipeId)) Then
                                              myDeviceInfo.BulkInPipe =
pipeInfo.PipeId

                                              SetPipePolicy(winUsbHandle,
myDeviceInfo.BulkInPipe,
Convert.ToUInt32(NativeMethods.POLICY_TYPE.IGNORE_SHORT_PACKETS),
Convert.ToByte(False))

                                              SetPipePolicy(winUsbHandle,
myDeviceInfo.BulkInPipe,
Convert.ToUInt32(NativeMethods.POLICY_TYPE.PIPE_TRANSFER_TIMEOUT),
pipeTimeout)
```

```
                                        ElseIf ((pipeInfo.PipeType =
NativeMethods.USBD_PIPE_TYPE.UsbdPipeTypeBulk) And
UsbEndpointDirectionOut(pipeInfo.PipeId)) Then
                                            myDeviceInfo.BulkOutPipe =
pipeInfo.PipeId

                                            SetPipePolicy(winUsbHandle,
myDeviceInfo.BulkOutPipe,
Convert.ToUInt32(NativeMethods.POLICY_TYPE.IGNORE_SHORT_PACKETS),
Convert.ToByte(False))

                                            SetPipePolicy(winUsbHandle,
myDeviceInfo.BulkOutPipe,
Convert.ToUInt32(NativeMethods.POLICY_TYPE.PIPE_TRANSFER_TIMEOUT),
pipeTimeout)
                                        ElseIf (pipeInfo.PipeType =
NativeMethods.USBD_PIPE_TYPE.UsbdPipeTypeInterrupt) And
UsbEndpointDirectionIn(pipeInfo.PipeId) Then
                                            myDeviceInfo.InterruptInPipe
= pipeInfo.PipeId

                                            SetPipePolicy(winUsbHandle,
myDeviceInfo.InterruptInPipe,
Convert.ToUInt32(NativeMethods.POLICY_TYPE.IGNORE_SHORT_PACKETS),
Convert.ToByte(False))

                                            SetPipePolicy(winUsbHandle,
myDeviceInfo.InterruptInPipe,
Convert.ToUInt32(NativeMethods.POLICY_TYPE.PIPE_TRANSFER_TIMEOUT),
pipeTimeout)
                                        ElseIf (pipeInfo.PipeType =
NativeMethods.USBD_PIPE_TYPE.UsbdPipeTypeInterrupt) And
UsbEndpointDirectionOut(pipeInfo.PipeId) Then

      myDeviceInfo.InterruptOutPipe = pipeInfo.PipeId

                                            SetPipePolicy(winUsbHandle,
myDeviceInfo.InterruptOutPipe,
Convert.ToUInt32(NativeMethods.POLICY_TYPE.IGNORE_SHORT_PACKETS),
Convert.ToByte(False))

                                            SetPipePolicy(winUsbHandle,
myDeviceInfo.InterruptOutPipe,
Convert.ToUInt32(NativeMethods.POLICY_TYPE.PIPE_TRANSFER_TIMEOUT),
pipeTimeout)
                                        End If
                                    Next i
                                End If
                            End If
                            Return success

                        Catch ex As Exception
                            DisplayException(ModuleName, ex)
                            Throw
                        End Try
                End Function

                ''' <summary>
                ''' Is the current operating system Windows XP or later?
                ''' The WinUSB driver requires Windows XP or later.
                ''' </summary>
```

```
            '''
            '''  <returns>
            '''  True if Windows XP or later, False if not.
            '''  </returns>

            Friend Function IsWindowsXpOrLater() As Boolean
                  Try
                        Dim myEnvironment = Environment.OSVersion

                        '  Windows XP is version 5.1.

                        Dim versionXp = New Version(5, 1)

                        If myEnvironment.Version >= versionXp Then
                              Return True
                        End If
                        Return False

                  Catch ex As Exception
                        DisplayException(ModuleName, ex)
                        Throw
                  End Try
            End Function

            '''  <summary>
            '''  Gets a value that corresponds to a USB_DEVICE_SPEED.
            '''  </summary>
            '''
            '''  <param name="winUsbHandle"> Handle for accessing the
WinUSB device </param>
            '''  <param name="myDevInfo"> devInfo structure for the device
</param>

            Friend Shared Function QueryDeviceSpeed(ByVal winUsbHandle As
IntPtr, ByRef myDevInfo As DeviceInfo) As Boolean
                  Dim length As UInt32 = 1
                  Dim speed = New Byte(0){}

                  Try
                        ' ***
                        '  winusb function

                        '  summary
                        '  Get the device speed.
                        '  (Normally not required but can be nice to know.)

                        '  parameters
                        '  Handle returned by WinUsb_Initialize
                        '  Requested information type.
                        '  Number of bytes to read.
                        '  Information to be returned.

                        '  returns
                        '  True on success.
                        ' ***

                        Dim success =
NativeMethods.WinUsb_QueryDeviceInformation(winUsbHandle,
NativeMethods.DEVICE_SPEED, length, speed(0))

                        If success Then
```

```
                              myDevInfo.DeviceSpeed =
Convert.ToUInt32(speed(0))
                        End If

                        Return success
                  Catch ex As AccessViolationException
                        ' The handle might be closed.
                        Return False

                  Catch ex As Exception
                        DisplayException(ModuleName, ex)
                        Throw
                  End Try
            End Function

            '''  <summary>
            '''  Attempts to read data from a bulk IN endpoint.
            '''  </summary>
            '''
            '''  <param name="winUsbHandle"> Handle for accessing the
WinUSB device </param>
            '''  <param name="myDeviceInfo"> devInfo structure for the
device </param>
            '''  <param name="bytesToRead"> Number of bytes to read.
</param>
            '''  <param name="dataBuffer"> Buffer for storing the bytes
read. </param>
            '''  <param name="bytesRead"> Number of bytes read. </param>
            '''  <param name="success"> Success or failure status. </param>
            '''
            Friend Sub ReceiveDataViaBulkTransfer(ByVal winUsbHandle As
IntPtr, ByVal myDeviceInfo As DeviceInfo, ByVal bytesToRead As UInt32,
ByRef dataBuffer() As Byte, ByRef bytesRead As UInt32, ByRef success As
Boolean)
                  Try
                        ' ***
                        '  winusb function

                        '  summary
                        '  Attempts to read data from a device interface.

                        '  parameters
                        '  Device handle returned by WinUsb_Initialize.
                        '  Endpoint address.
                        '  Buffer to store the data.
                        '  Maximum number of bytes to return.
                        '  Number of bytes read.
                        '  Null pointer for non-overlapped.

                        '  Returns
                        '  True on success.
                        ' ***

                              success =
NativeMethods.WinUsb_ReadPipe(winUsbHandle, myDeviceInfo.BulkInPipe,
dataBuffer, bytesToRead, bytesRead, IntPtr.Zero)
                  Catch ex As AccessViolationException
                        ' The handle might be closed.

                  Catch ex As Exception
                        DisplayException(ModuleName, ex)
```

```
                    Throw
                End Try
            End Sub

            ''' <summary>
            ''' Attempts to read data from an interrupt IN endpoint.
            ''' </summary>
            '''
            ''' <param name="winUsbHandle"> Handle for accessing the
WinUSB device </param>
            ''' <param name="myDeviceInfo"> devInfo structure for the
device </param>
            ''' <param name="bytesToRead"> Number of bytes to read.
</param>
            ''' <param name="dataBuffer"> Buffer for storing the bytes
read. </param>
            ''' <param name="bytesRead"> Number of bytes read. </param>
            '''
            ''' <returns> true on success, false on failure </returns>
            '''
            Friend Function ReceiveDataViaInterruptTransfer(ByVal
winUsbHandle As IntPtr, ByVal myDeviceInfo As DeviceInfo, ByVal bytesToRead
As UInt32, ByRef dataBuffer() As Byte, ByRef bytesRead As UInt32) As
Boolean
                Try
                    ' ***
                    '  winusb function

                    '  summary
                    '  Attempts to read data from a device interface.

                    '  parameters
                    '  Device handle returned by WinUsb_Initialize.
                    '  Endpoint address.
                    '  Buffer to store the data.
                    '  Maximum number of bytes to return.
                    '  Number of bytes read.
                    '  Null pointer for non-overlapped.

                    '  Returns
                    '  True on success.
                    ' ***

                    Dim success As Boolean =
NativeMethods.WinUsb_ReadPipe(winUsbHandle, myDeviceInfo.InterruptInPipe,
dataBuffer, bytesToRead, bytesRead, IntPtr.Zero)

                    Return success
                Catch ex As AccessViolationException
                    ' The handle might be closed.
                    Return False


                Catch ex As Exception
                    DisplayException(ModuleName, ex)
                    Throw
                End Try
            End Function

            ''' <summary>
            ''' Attempts to send data via a bulk OUT endpoint.
```

```
            '''  </summary>
            '''
            '''  <param name="winUsbHandle"> Handle for accessing the
WinUSB device </param>
            '''  <param name="myDeviceInfo"> devInfo structure for the
device </param>
            '''  <param name="bytesToWrite"> Number of bytes to write.
</param>
            '''  <param name="dataBuffer"> Buffer containing the bytes to
write. </param>
            '''  <param name="bytesWritten"> The number of bytes written
</param>
            '''  <param name="success"> True on success </param>

            Friend Sub SendDataViaBulkTransfer(ByVal winUsbHandle As
IntPtr, ByVal myDeviceInfo As DeviceInfo, ByVal bytesToWrite As UInt32,
ByVal dataBuffer() As Byte, ByRef bytesWritten As UInt32, ByRef success As
Boolean)
                  Try
                        ' ***
                        '  winusb function

                        '  summary
                        '  Attempts to write data to a device interface.

                        '  parameters
                        '  Device handle returned by WinUsb_Initialize.
                        '  Endpoint address.
                        '  Buffer with data to write.
                        '  Number of bytes to write.
                        '  Number of bytes written.
                        '  IntPtr.Zero for non-overlapped I/O.

                        '  Returns
                        '  True on success.
                        '  ***

                              success =
NativeMethods.WinUsb_WritePipe(winUsbHandle, myDeviceInfo.BulkOutPipe,
dataBuffer, bytesToWrite, bytesWritten, IntPtr.Zero)
                  Catch ex As AccessViolationException
                        ' The handle might be closed.


                  Catch ex As Exception
                        DisplayException(ModuleName, ex)
                        Throw
                  End Try
            End Sub

            '''  <summary>
            '''  Attempts to send data via an interrupt OUT endpoint.
            '''  </summary>
            '''
            '''  <param name="winUsbHandle"> Handle for accessing the
WinUSB device </param>
            '''  <param name="myDeviceInfo"> devInfo structure for the
device </param>
            '''  <param name="bytesToWrite"> Number of bytes to write.
</param>
```

```
            '''  <param name="dataBuffer"> Buffer containing the bytes to
write. </param>
            '''  <param name="bytesWritten"> The number of bytes written
</param>
            '''
            '''  <returns>
            '''  True on success, False on failure.
            '''  </returns>

            Friend Function SendDataViaInterruptTransfer(ByVal winUsbHandle
As IntPtr, ByVal myDeviceInfo As DeviceInfo, ByVal bytesToWrite As UInt32,
ByVal dataBuffer() As Byte, ByRef bytesWritten As UInt32) As Boolean
                  Try
                        ' ***
                        '  winusb function

                        '  summary
                        '  Attempts to write data to a device interface.

                        '  parameters
                        '  Device handle returned by WinUsb_Initialize.
                        '  Endpoint address.
                        '  Buffer with data to write.
                        '  Number of bytes to write.
                        '  Number of bytes written.
                        '  IntPtr.Zero for non-overlapped I/O.

                        '  Returns
                        '  True on success.
                        '  ***

                        Dim success =
NativeMethods.WinUsb_WritePipe(winUsbHandle, myDeviceInfo.InterruptOutPipe,
dataBuffer, bytesToWrite, bytesWritten, IntPtr.Zero)

                        Return success
                  Catch ex As AccessViolationException
                        ' The handle might be closed.
                        Return False


                  Catch ex As Exception
                        DisplayException(ModuleName, ex)
                        Throw
                  End Try
            End Function

            '''  <summary>
            '''  Sets pipe policy.
            '''  Used when the value parameter is a Byte (all except
PIPE_TRANSFER_TIMEOUT).
            '''  </summary>
            '''
            '''  <param name="winUsbHandle"> Handle for accessing the
WinUSB device </param>
            ''' <param name="pipeId"> Pipe to set a policy for. </param>
            '''  <param name="policyType"> POLICY_TYPE member. </param>
            '''  <param name="value"> Policy value. </param>
            '''
            '''  <returns>
            '''  True on success, False on failure.
```

```
            '''  </returns>
            '''
            Private Function SetPipePolicy(ByVal winUsbHandle As IntPtr,
ByVal pipeId As Byte, ByVal policyType As UInt32, ByVal value As Byte) As
Boolean
                  Try
                        ' ***
                        '  winusb function

                        '  summary
                        '  sets a pipe policy

                        '  parameters
                        '  handle returned by WinUsb_Initialize
                        '  identifies the pipe
                        '  POLICY_TYPE member.
                        '  length of value in bytes
                        '  value to set for the policy.

                        '  returns
                        '  True on success
                        ' ***

                        Dim success =
NativeMethods.WinUsb_SetPipePolicy(winUsbHandle, pipeId, policyType, 1,
value)

                        Return success
                  Catch ex As AccessViolationException
                        ' The handle might be closed.

                        Return False


                  Catch ex As Exception
                        DisplayException(ModuleName, ex)
                        Throw
                  End Try
            End Function

            '''  <summary>
            '''  Sets pipe policy.
            '''  Used when the value parameter is a UInt32
(PIPE_TRANSFER_TIMEOUT only).
            '''  </summary>
            '''
            ''' <param name="winUsbHandle"> Handle for accessing the WinUSB
device </param>
            ''' <param name="pipeId"> Pipe to set a policy for. </param>
            '''  <param name="policyType"> POLICY_TYPE member. </param>
            '''  <param name="value"> Policy value. </param>
            '''
            '''  <returns>
            '''  True on success, False on failure.
            '''  </returns>
            '''
            Private Function SetPipePolicy(ByVal winUsbHandle As IntPtr,
ByVal pipeId As Byte, ByVal policyType As UInt32, ByVal value As UInt32) As
Boolean
                  Dim success = False
                  Try
```

```
                        ' ***
                        '  winusb function

                        '  summary
                        '  sets a pipe policy

                        '  parameters
                        '  handle returned by WinUsb_Initialize
                        '  identifies the pipe
                        '  POLICY_TYPE member.
                        '  length of value in bytes
                        '  value to set for the policy.

                        '  returns
                        '  True on success
                        ' ***

                        success =
NativeMethods.WinUsb_SetPipePolicy1(winUsbHandle, pipeId, policyType, 4,
value)

                              Return success
                        Catch ex As AccessViolationException
                              ' Handle might be closed.

                              Return success

                        Catch ex As Exception
                              DisplayException(ModuleName, ex)
                              Throw
                        End Try
                  End Function

                  '''  <summary>
                  '''  Is the endpoint's direction IN (device to host)?
                  '''  </summary>
                  '''
                  '''  <param name="endpointAddress"> The endpoint address.
</param>
                  '''
                  '''  <returns>
                  '''  True if IN (device to host), False if OUT (host to device)
                  '''  </returns>

                  Private Function UsbEndpointDirectionIn(ByVal endpointAddress
As Int32) As Boolean
                        Try

                              Dim directionIn = False

                              If ((endpointAddress And &H80) = &H80) Then
                                    directionIn = True
                              End If
                              Return directionIn

                        Catch ex As Exception
                              DisplayException(ModuleName, ex)
                              Throw
                        End Try
                  End Function
```

```
            '''  <summary>
            '''  Is the endpoint's direction OUT (host to device)?
            '''  </summary>
            '''
            '''  <param name="addr"> The endpoint address. </param>
            '''
            '''  <returns>
            '''  True if OUT (host to device, False if IN (device to host)
            '''  </returns>

            Private Function UsbEndpointDirectionOut(ByVal addr As Int32)
As Boolean
                  Try
                        Dim directionOut = False

                        If ((addr And &H80) = 0) Then
                              directionOut = True
                        End If
                        Return directionOut


                  Catch ex As Exception
                        DisplayException(ModuleName, ex)
                        Throw
                  End Try
            End Function


      End Class
End Namespace
```

WinUsbDeviceApi.vb

```
Imports System
Imports Microsoft.Win32.SafeHandles
Imports System.Runtime.InteropServices

Namespace WinUsbDemo
      ''' <summary>
      '''  These declarations are translated from the C declarations in
various files
      '''  in the WDK:
      '''
      '''  usb.h
      '''  usb100.h
      '''  winusbio.h
      ''' </summary>

      Friend NotInheritable Partial Class WinUsbCommunications
            Friend NotInheritable Class NativeMethods

                  Private Sub New()
                  End Sub

                  Friend Const DEVICE_SPEED As UInt32 = 1
```

```
Friend Const USB_ENDPOINT_DIRECTION_MASK As Byte = &H80

Friend Enum POLICY_TYPE
      SHORT_PACKET_TERMINATE = 1
      AUTO_CLEAR_STALL
      PIPE_TRANSFER_TIMEOUT
      IGNORE_SHORT_PACKETS
      ALLOW_PARTIAL_READS
      AUTO_FLUSH
      RAW_IO
End Enum

Friend Enum USBD_PIPE_TYPE
      UsbdPipeTypeControl
      UsbdPipeTypeIsochronous
      UsbdPipeTypeBulk
      UsbdPipeTypeInterrupt
End Enum

Friend Enum USB_DEVICE_SPEED
      UsbLowSpeed = 1
      UsbFullSpeed
      UsbHighSpeed
End Enum

<StructLayout(LayoutKind.Sequential)> _
Friend Structure USB_CONFIGURATION_DESCRIPTOR
      Friend bLength As Byte
      Friend bDescriptorType As Byte
      Friend wTotalLength As UShort
      Friend bNumInterfaces As Byte
      Friend bConfigurationValue As Byte
      Friend iConfiguration As Byte
      Friend bmAttributes As Byte
      Friend MaxPower As Byte
End Structure

<StructLayout(LayoutKind.Sequential)> _
Friend Structure USB_INTERFACE_DESCRIPTOR
      Friend bLength As Byte
      Friend bDescriptorType As Byte
      Friend bInterfaceNumber As Byte
      Friend bAlternateSetting As Byte
      Friend bNumEndpoints As Byte
      Friend bInterfaceClass As Byte
      Friend bInterfaceSubClass As Byte
      Friend bInterfaceProtocol As Byte
      Friend iInterface As Byte
End Structure

<StructLayout(LayoutKind.Sequential)> _
Friend Structure WINUSB_PIPE_INFORMATION
      Friend PipeType As USBD_PIPE_TYPE
      Friend PipeId As Byte
      Friend MaximumPacketSize As UShort
      Friend Interval As Byte
End Structure

<StructLayout(LayoutKind.Sequential, Pack := 1)> _
Friend Structure WINUSB_SETUP_PACKET
      Friend RequestType As Byte
```

```
                    Friend Request As Byte
                    Friend Value As UShort
                    Friend Index As UShort
                    Friend Length As UShort
                End Structure

                <DllImport("winusb.dll", SetLastError := True)> _
                Friend Shared Function WinUsb_ControlTransfer(ByVal
InterfaceHandle As IntPtr, ByVal SetupPacket As WINUSB_SETUP_PACKET, ByVal
Buffer() As Byte, ByVal BufferLength As UInt32, ByRef LengthTransferred As
UInt32, ByVal Overlapped As IntPtr) As Boolean
                End Function

                <DllImport("winusb.dll", SetLastError := True)> _
                Friend Shared Function WinUsb_Free(ByVal InterfaceHandle
As IntPtr) As Boolean
                End Function

                <DllImport("winusb.dll", SetLastError := True)> _
                Friend Shared Function WinUsb_Initialize(ByVal
DeviceHandle As SafeFileHandle, ByRef InterfaceHandle As IntPtr) As Boolean
                End Function

                '  Use this declaration to retrieve DEVICE_SPEED (the
only currently defined InformationType).

                <DllImport("winusb.dll", SetLastError := True)> _
                Friend Shared Function
WinUsb_QueryDeviceInformation(ByVal InterfaceHandle As IntPtr, ByVal
InformationType As UInt32, ByRef BufferLength As UInt32, ByRef Buffer As
Byte) As Boolean
                End Function

                <DllImport("winusb.dll", SetLastError := True)> _
                Friend Shared Function
WinUsb_QueryInterfaceSettings(ByVal InterfaceHandle As IntPtr, ByVal
AlternateInterfaceNumber As Byte, ByRef UsbAltInterfaceDescriptor As
USB_INTERFACE_DESCRIPTOR) As Boolean
                End Function

                <DllImport("winusb.dll", SetLastError := True)> _
                Friend Shared Function WinUsb_QueryPipe(ByVal
InterfaceHandle As IntPtr, ByVal AlternateInterfaceNumber As Byte, ByVal
PipeIndex As Byte, ByRef PipeInformation As WINUSB_PIPE_INFORMATION) As
Boolean
                End Function

                <DllImport("winusb.dll", SetLastError := True)> _
                Friend Shared Function WinUsb_ReadPipe(ByVal
InterfaceHandle As IntPtr, ByVal PipeID As Byte, ByVal Buffer() As Byte,
ByVal BufferLength As UInt32, ByRef LengthTransferred As UInt32, ByVal
Overlapped As IntPtr) As Boolean
                End Function

                '  Two declarations for WinUsb_SetPipePolicy.
                '  Use this one when the returned Value is a Byte (all
except PIPE_TRANSFER_TIMEOUT):

                <DllImport("winusb.dll", SetLastError := True)> _
```

```
                Friend Shared Function WinUsb_SetPipePolicy(ByVal
InterfaceHandle As IntPtr, ByVal PipeID As Byte, ByVal PolicyType As
UInt32, ByVal ValueLength As UInt32, ByRef Value As Byte) As Boolean
                End Function

                '  Use this alias when the returned Value is a UInt32
(PIPE_TRANSFER_TIMEOUT only):

                <DllImport("winusb.dll", SetLastError := True, EntryPoint
:= "WinUsb_SetPipePolicy")> _
                Friend Shared Function WinUsb_SetPipePolicy1(ByVal
InterfaceHandle As IntPtr, ByVal PipeID As Byte, ByVal PolicyType As
UInt32, ByVal ValueLength As UInt32, ByRef Value As UInt32) As Boolean
                End Function

                <DllImport("winusb.dll", SetLastError := True)> _
                Friend Shared Function WinUsb_WritePipe(ByVal
InterfaceHandle As IntPtr, ByVal PipeID As Byte, ByVal Buffer() As Byte,
ByVal BufferLength As UInt32, ByRef LengthTransferred As UInt32, ByVal
Overlapped As IntPtr) As Boolean
                End Function
            End Class
      End Class
End Namespace
```