Universitat Autònoma de Barcelona

Escola Tècnica Superior d'Enginyeria
Secció d'Enginyeria Electrònica

Sistema de comunicació sense fils utilitzant Zigbee per PDA.

Memòria del projecte Fi de Carrera d'Enginyeria en Electrònica presentada per Joan Campderrós i Canas i dirigida per Jordi Carrabina Bordoll i Màrius Montón Macian.

Bellaterra, 14 de Setembre de 2007

En els moments de crisi, només la imaginació és més important que el coneixement.

Albert Einstein

Els sotasignats Dr. Jordi Carrabina Bordoll i MsC. Marius Montón Macian, professors del Departament de Microelectrònica i Sistemes Electrònics de la Universitat Autònoma de Barcelona

Certifiquen

Que el treball al que correspon la present memòria ha estat realitzada sota la seva direcció per JOAN CAMPDERRÓS i CANAS. I per que així consti signen la present

Marius Montón Macian Jordi Carrabina Bordoll

a Bellaterra, a 14 de Setembre de 2007

AGRAÏMENTS

En primer lloc vull agraïr als Dr. Jordi Carrabina i MsC. Màrius Montón en qualitat de co-directors del projecte pel suport constant, consell i autonomia que m'han donat durant l'elaboració del projecte.

En segon lloc, faig extensiu el meu agraïment tant al Departament de Microelectrònica i Sistemes Electrònics d l'ETSE com a l'equip de Cephis per la seva amabilitat i disponibilitat sempre que els he necessitat.

En tercer lloc, a tota la meva família i amics per haver estat comprensius durant els mesos que he estat ocupat en la elaboració del projecte i molt especialment a la Glòria pel seu suport i cooperació constants.

Vull agrair també a l'equip administratiu de l'ETSE per la seva eficiència i celeritat en totes les gestions dutes a terme.

Per últim, vull agrair a tots els amics que he fet durant tots aquests anys d'estudi a la Universitat Autònoma de Barcelona.

Taula de continguts

Objectius

L'objectiu d'aquest projecte és la implementació d'una solució que permeti la comunicació *end-to-end,* des d'un dispositiu portable (tipus PocketPC) basat en Windows CE i dotat amb port SD, i una tarja SDIO connectada a aquest port, i que incorpora un mòdul de comunicació sense fils Zigbee. El sistema es caracteritza per una baixa tassa de transferència de dades.

El projecte es desglossa en diferents tasques per tal de crear un camí de dades bidireccional entre el sistema PocketPC i el mòdul Zigbee. Per tal de fer això, aquests són les peces que cal desenvolupar:

- Driver SDIO per Windows CE al PocketPC. El driver estarà basat en l'arquitectura per capes anomenada "Stream Interface Driver", i que proveirà d'una interfície estàndard per tal que l'usuari final pugui generar les aplicacions que li permetran interactuar amb el hardware.
- Bridge Bus SD entre el PocketPC i la FPGA, implementant en aquesta la funcionalitat del protocol SDIO esperada pel controlador del bus SD.
- Bridge o enllaç de comunicació entre la FPGA i el microcontrolador, que serà qui controlarà directament el mòdul Zigbee.
- Aplicació de control per Windows CE. Es generarà una aplicació genèrica de control que realitzi la comunicació amb el mòdul Zigbee mitjançant la interfície donada pel driver implementat.

Capítol 1.
Introducció i anàlisi

1.1. Introducció

La proliferació de dispositius mòbils, cada cop més rics quant a les característiques que incorporen, juntament amb les capacitats de processament sempre en augment, ha provocat un desig en tenir més opcions d'expansió que fins fa poc només es veien en sistemes de sobretaula i portàtils. Els consumidors demanen un augment d'opcions d'expansió en els seus dispositius per tal d'incrementar capacitat (emmagatzematge) o bé possibilitats (*features*) per tal d'aturar, encara que només sigui momentàniament, l'amenaça d'obsolescència dels seus dispositius i les necessitats futures (majoritàriament desconegudes). El contingut multimèdia ha estat el que ha forçat la indústria a crear els estàndards necessaris i tecnologies en reproductors d'àudio, càmeres digitals, PDAs i telèfons mòbils. L'expansió comença amb la capacitat de memòria en forma de targetes de memòria tipus flash. A mesura que els dispositius mòbils esdevenen més i més potents, la línia entre un dispositiu electrònic de consum amb una única funció i un dispositiu de propòsit més general es comença a diluir. Una indicació d'aquesta tendència ha estat la estandardització de les expansions d'entrada-sortida (I/O) en dispositius mòbils.

Prèviament, l'expansió I/O dels dispositius mòbils consistia en interfícies hardware/software propietàries, típicament només disponibles al fabricant en concret i en molts casos a un model de dispositiu en concret. Ben aviat, una nova generació de dispositius van començar a adaptar tecnologia pròpia dels ordinadors portàtils en forma de CompactFlash I/O. L'expansió CompactFlash I/O ofereix un mètode relativament poc traumàtic d'incorporar estàndards PCMCIA a mides més mòbils. La demanda de dispositius cada cop més petits i lleugers ha empès el pas a tecnologies d'expansió I/O de format encara més reduït.

D'aquesta manera, a finals de 2001 apareix Secure digital I/O (SDIO) com una evolució de la tecnologia Secure Digital Memory (SD). SDIO està basat en l'electrònica i la mecànica presos de la especificació original de les memòries SD, i ofereix als fabricants de dispositius una manera convenient de suportar memòria i targetes I/O en un mateix slot. SDIO estén l'especificació original de SD per tal d'encabir els requeriments específics d'I/O en termes d'alimentació, *plug and play*, comandes I/O i senyals. L'adició de les capacitats d'expansió I/O a un slot SD es poden implementar a un increment de cost mínim, degut a la disponibilitat de més controladors capaços de suportar-ho.

1.2. Entorn de treball

El projecte a desenvolupar parteix d'una tarja SDIO creada per Cephis (http://cephis.uab.es) i que consta d'un mòdul de comunicació sense fils basat en Zigbee. L'objectiu final és el d'implementar un sistema capaç de comunicar-se mitjançant aquesta tarja a d'altres dispositius Zigbee i comandar-los mitjançant un sistema Pocket PC, prenent com avantatge l'existència d'un port SD en aquest últim on poder inserir la tarja SDIO.

En concret, a nivell de blocs, el sistema complert està composat dels elements que s'indiquen a la Figura 1.1.

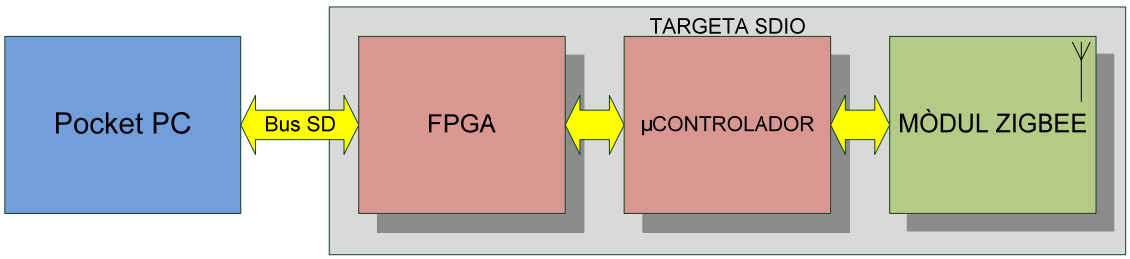


Figura 1.1. Diagrama de blocs corresponents al sistema.

Així, la tarja SDIO està composada d'una FPGA on s'implementa tota la funcionalitat necessària per, d'una banda, la gestió de la comunicació amb el

bus SD i per l'altra, la comunicació amb el microcontrolador que és el que al seu torn, gestiona les comunicacions amb el mòdul Zigbee final.

A la Figura 1.2 es mostra una imatge de la placa SDIO, identificant els components descrits anteriorment.

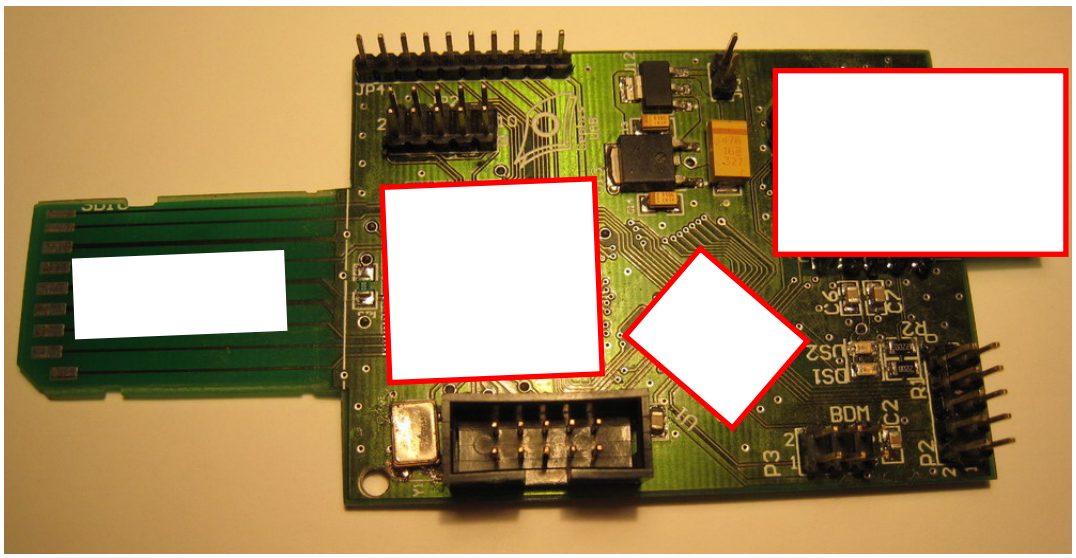


Figura 1.2. Vista general de la tarja SDIO

Com s'ha indicat en l'apartat d'objectius, el projecte que ens ha estat encomanat, és la creació d'un cami de dades bidireccional entre el sistema PocketPC i el mòdul Zigbee.

Per tal de fer això, aquests són les peces que cal desenvolupar:

1. Driver SDIO per Windows CE al PocketPC.
2. Bridge Bus SD entre el PocketPC i la FPGA, implementant en aquesta última tota la funcionalitat SDIO esperada pel controlador del bus SD.
3. Bridge o enllaç de comunicació entre la FPGA i el Microcontrolador.
4. Aplicació de control per Windows CE.

1.3. Descripció general de les memòries SD/MMC

En aquest apartat es farà un repàs a les característiques principals de les memòries SD fins arribar a les característiques pròpies de SDIO. A continuació

es veurà en detall els mecanismes d'inicialització, registres i senyalització per operar amb aquest tipus de targetes.

Secure Digital I/O té les seves arrels en les tecnologies Secure Digital Memory i Multimedia Memory Card. Per tal d'entendre algunes de les característiques i extensions que ofereix l'estàndard SDIO, cal examinar les realitzacions inicials d'aquest format de tarja.

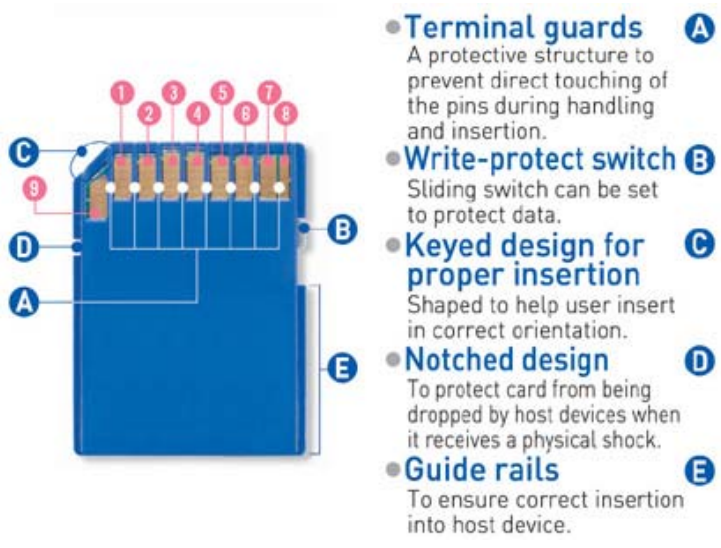
El format Multimedia Memory Card (MMC) va ser dissenyat com un estàndard de tarja de memòria no volàtil amb un format de pocs pins, baix consum i perfil prim. La interfície elèctrica és un bus sèrie simple i directe amb lògica CMOS/TTL basat en dos protocols sèrie síncrons: l'estàndard Serial Peripheral Interface (SPI) i la interfície nadiua Multimedia Card. Totes les targetes han de suportar els dos protocols. El fet de tenir els dos protocols va ésser inicialment el de dissenyar targetes amb controladors existents que suportaven el mode SPI. Malgrat tot, SPI no és una implementació òptima ja que molts controladors es troben limitats en velocitat (~2-8 MHz) i possibilitats (com per exemple hardware de generació de CRC). SPI no conté cap control de flux de lectura/escriptura de dades i requereix software adicional per manegar transferències de dades simples. La interfície nativa MMC sol.luciona molts d'aquests problemes requerint controladors que tinguin generació de CRC integrada i control de flux de dades. La velocitat de transferència de dades es va incrementar per suportar velocitats de rellotge de fins a 20MHz, la qual cosa es pot traduir en una velocitat de transferència de 20Mbs/sec. MMC en mode SPI nadiu fa servir 7 pins, i els senyals SPI són compartits amb els senyals nadius. En mode nadiu es fa servir una línia independent per a comandes i una altra per a dades, mentre que en SPI tot es realitza sobre la mateixa línia.

El format Secure Digital Memory fa una extensió de la especificació MMC per millorar-ne les prestacions i afegir la conformitat SDMI (Secure Digital Music Initiative) per protegir continguts amb drets d'autor. Secure Digital segueix suportant les interfícies elèctriques SPI i MMC 1-bit, però afegeix una interfície de 4-bit amb més prestacions a 25MHz. Això proporciona una tassa de transferència de fins a 100Mb/sec. Les targetes SD comparteixen els 7 pins

de MMC però n'afegeixes dos més per suportar la interfícies 4-bit SD. Les targetes SD i MMC poden fer servir un únic slot per targetes que suporti els dos patillatges. Les targetes de memòria SD poden fer-se servir en sistemes amb controladors SPI i MMC però operaran amb menors prestacions. El software s'encarrega de les diferències d'ús dels dos tipus de targetes.

A continuació es mostra l'arquitectura física i patillatge de les targetes SD Memory:

1.3.1. Vista externa



a)

Figura 1.3. Encapsulat i patillatge SD memory.

Les targetes de memòria SD disposen d'un interruptor mecànic de protecció d'escriptura de tal manera que les dades vitals emmagatzemades no es perdran accidentalment.

Disposen de rails en tots dos cantons de la tarja per tal de prevenir que aquesta sigui inserida a l'inrevés al seu allotjament (*socket*), i també disposen d'una mossa per tal de prevenir que la tarja pugui sortir del seu allotjament si el dispositiu on es troba cau o es colpeja.

Els contactes metàl·lics es troben protegits per costelles les quals minimitzen les probabilitats de dany per electricitat estàtica o dany per contacte com poden ser ratllades a la seva superfície.

1.3.2. Pin out

Pin #	Terminal Name	Function
1	CD/DAT[3]	Card detection/ Data I/O
2	CMD	Command Line
3	VSS1	Ground
4	VDD	Supply Voltage
5	CLK	Clock
6	Vss2	Ground
7	DAT[0]	Data I/O
8	DAT[1]	Data I/O
9	DAT[2]	Data I/O

Figura 1.4. Assignació de pins per SD memory.

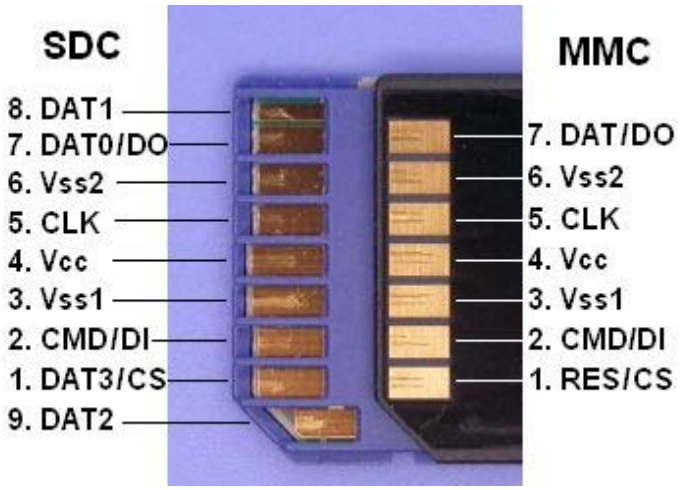


Figura 1.5. Comparació patillatge MMC i SD

1.4. Descripció general de Secure Digital I/O

L'estàndard Secure Digital I/O estén els sistemes capacitats a treballar amb memòries SD per suportar també perifèrics I/O com scanners de codis de barres, càmeres, adaptadors wireless i mòduls GPS entre d'altres. La figura següent en presenta alguns exemples:

Figura 1.6. Exemples de targetes SDIO al mercat.

Les targetes SDIO (SD Input/Output) es basen i són compatibles amb les targetes de memòria SD. Aquesta compatibilitat inclou tant la part mecànica (mateix connector de 9 pins), elèctrica, d'alimentació, senyals i software. L'objectiu de les targetes SDIO és la de proveir entrada/sortida de dades a gran velocitat amb poc consum per dispositius electrònics mòbils. Un primer objectiu és que una tarja SDIO inserida en un host no-SDIO no hauria de causar cap dany físic o cap disrupció en aquest host o el seu software. En aquest cas, la tarja SDIO hauria d'esser senzillament ignorada. Un cop la tarja és inserida en un host que reconeix SDIO, la detecció de la tarja es produeix de la manera usual descrita a les especificacions SD amb algunes extensions. En aquest estat, la tarja SDIO està en mode desocupat (*idle*) i rep alimentació elèctrica (15mA de mitja durant 1 segon). Durant el procés normal d'inicialització i interrogació de la tarja per part del host, la tarja s'identifica a sí mateixa com una tarja SDIO. El software situat al host obté, aleshores, la informació de la

tarja en un format de tupla (llista enllaçada) i determina si les funcions d'entrada/sortida de la tarja són acceptables per ser activades. La decisió es basa en alguns paràmetres com els requeriments d'alimentació o la disponibilitat dels *drivers* de software apropiats. Si la tarja és acceptada, aleshores se li permet ser alimentada completament i iniciar les funcions d'entrada/sortida implementades en ella.

L'estàndard fa servir el mateix connector mecànic de 9 pins que SD i defineix nous protocols I/O sobre els pins existents. El factor de forma de la tarja SDIO permet la integració de components de RF, càmeres o altres dispositius. Fonamentalment, SDIO fa servir la mateixa interfície elèctrica de transport, SPI o bé SD nadiua amb l'adició mínima de noves comandes específiques d'I/O i mecanismes de senyalització. Per tal de complir amb els requeriments de senyalització asíncrona, l'estàndard introdueix senyalització I/O mitjançant interrupcions i suspensió de lectura de dades (Read/Wait). El estàndard SDIO conté canvis simples i elementals permetent als controladors SD/MMC existents a ser reaprofitats per esdevenir controladors SDIO amb algunes limitacions "acceptables".

Hi ha dos tipus de targetes SDIO: una versió d'alta velocitat, i una versió de baixa velocitat. La versió d'alta velocitat pot operar en els modes de transferència de 1-bit i 4-bit SD, ambdós amb un rang de rellotge des de 0Hz fins a 25MHz. La versió de 4-bit operant a 25MHz, té una tassa de transferència de fins a 100Mbps. La versió de baixa velocitat fa server el mode de transferència de 1-bit i també pot tenir el mode de transferència de 4-bit SD, però només opera en una tassa de transferència d'entre 0Hz i 400kHz. La versió de baixa velocitat no està prevista per funcions de memòria. La tarja SDIO també suporta la interfície del bus SPI.

A mode de general, es resumeixen a continuació les característiques principals de SDIO:

- ✓ Pensat per aplicacions en dispositius portables i mòbils
- ✓ Cal una mínima o nul·la modificació respecte el bus SD físic.

- ✓ Mínim canvis al software del driver de memòria.
- ✓ Per aplicacions especials, existeix una factor de forma físic ampliat.
- ✓ Suport Plug and play (PnP).
- ✓ Suport multi-funció incloent I/O múltiple i I/O combinada amb memòria.
- ✓ Fins a 7 funcions I/O més una memòria suportats en una mateixa tarja.
- ✓ Existeix un sistema d'interrupcions de la tarja cap el host.
- ✓ Rang de voltatge operacional: 2.7-3.6V (el rang operacional es fa servir per la inicialització)
- ✓ Les especificacions de les aplicacions corresponen a les Funcions SDIO estàndard.
- ✓ Múltiples factor de forma (*form factors*):
    - o SDIO mida completa.
    - o miniSDIO

1.4.1. Vista externa

Les targetes Secure Digital IO fan servir un connector de 9 pins [1 fila de 9 pins]. SDIO es basa en les targetes Secure Digital SD. SDIO és compatible amb les targetes de memòria SD, i es poden allotjar en els mateixos *sockets*. Les dimensions estàndard de les targetes Secure Digital I/O card són: 24mm d'ample x 32mm de llarg x 2,1mm de gruix.

1.4.2. Pin out

Secure Digital I/O Pinout						
Pin #	**SD 4-bit Mode**		**SD 1-bit Mode**		**SPI Mode**	
1	CD/DAT[3]	Data Line 3	N/C	Not Used	CS	Card Select
2	CMD	Command Line	CMD	Command Line	DI	Data Input
3	VSS1	Ground	VSS1	Ground	VSS1	Ground
4	VDD	Supply Voltage	VDD	Supply Voltage	VDD	Supply Voltage
5	CLK	Clock	CLK	Clock	SCLK	Clock
6	Vss2	Ground	Vss2	Ground	Vss2	Ground
7	DAT[0]	Data Line 0	DATA	Data Line	DO	Data Output
8	DAT[1]	Data Line 1 / Interrupt	IRQ	Interrupt	IRQ	Interrupt
9	DAT[2]	Data Line 2 /Read Wait	RW	Read Wait	NC	Not Used

Figura 1.7. Assignació de pins per targetes SDIO.

1.5. Obertura d'estàndards SD/SDIO

Com la majoria dels formats de tarja de memòria, el SD està cobert per nombroses patents i marques registrades, i només es pot llicenciar a través de la Secure Digital Card Association (**Associació de la Tarja Digital Segura**). L'acord de llicència actual d'aquesta organització no permet controladors de codi obert per a lectors de targetes SD, un fet que genera consternació en les comunitats de codi obert i programari lliure. Generalment, es desenvolupa una capa de codi obert per a un controlador SD de codi tancat disponible en una plataforma particular, però això està lluny d'ésser l'ideal. Un altre mètode comú consisteix en utilitzar l'antic mode MMC, on es requereix que totes les targetes SD suportin l'estàndard SD.

Totes les targetes de memòria SD i SDIO necessiten suportar l'antic mode SPI/MMC que suporta la interfície de sèrie de quatre cables lleugerament més lent (**rellotge**, **entrada sèrie**, **sortida sèrie** i **selecció de xip**) que és compatible amb els ports SPI en molts microcontroladors.

Moltes càmeres digitals, reproductors d'àudio digital i d'altres dispositius portàtils, probablement facin servir exclusivament el mode MMC. La

documentació per a aquest mode es pot comprar a MMCA per 500,00$; malgrat això, la documentació parcial per a SDIO és lliure i existeix documentació lliure disponible per a targetes de memòria com part dels fulls d'especificació d'alguns fabricants.

El mode MMC no proporciona accés a les característiques propietàries de xifrat de les targetes SD i la documentació lliure de SD no descriu aquestes característiques. La informació del xifrat és utilitzada primordialment per els productors de mitjans i no és molt utilitzada pels consumidors els quals típicament fan servir targetes SD per emmagatzemar dades no protegides.

Els drets de les llicències per a SD/SDIO són imposats als fabricants i venedors de targetes de memòria i lectors de les mateixes (1000$ per any, més una quota de membre de 1500$ per any). No obstant, les targetes SDIO se poden fabricar sense llicència, així com tampoc és necessària en el cas de la fabricació dels lectors MMC.

1.6. Model de programació SDIO

A continuació es detalla el model de programació de l'estàndard SDIO, fent èmfasi només a aquelles característiques rellevants pel desenvolupament del projecte. Es pot trobar documentació més àmplia i detallada visitant el lloc web de la SD Card Association (http://www.sdcard.org).

Bona part d'aquest model es comú amb el de la interfície SD però tenint en compte que SDIO és en realitat un subconjunt simplificat de SD (comandes i registres) amb algunes funcionalitats modificades. Mentre l'estàndard SD està pensat per manegar dispositius de memòria, SDIO està orientat a la gestió de targetes dedicades a la entrada/sortida.

1.6.1. Registres interns

Cada tarja SDIO disposa d'un conjunt de sis registres d'informació definits a la seva interfície segons la SD Physical Specification Version 1.01. Una tarja que sigui només SDIO, elimina alguns registres i canvia alguns dels bits dels altres. A continuació s'exposa una breu descripció dels mateixos especificant les particularitats per SDIO.

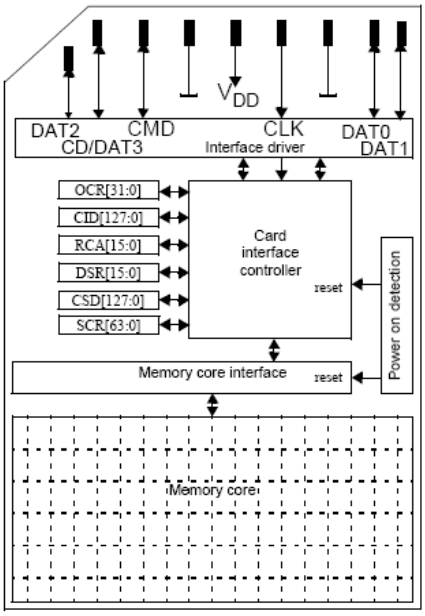


Figura 1.8. Aquitectura SD Memory Card

b) Registre OCR (Operation Condition Register)

Registre de 24 bits que emmagatzema el perfil de voltatge V_{DD} de la tarja. Addicionalment, aquest registre inclou bits d'informació d'estat. Un bit d'estat s'activa si el procediment d'engegada de la tarja ha finalitzat. A continuació es mostren les posicions del registre amb més detall.

I/O OCR bit position	VDD Voltage Window (in Volts)
0-3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	Reserved
8	2.0-2.1
9	2.1-2.2
10	2.2-2.3
11	2.3-2.4
12	2.4-2.5
13	2.5-2.6
14	2.6-2.7
15	2.7-2.8
16	2.8-2.9
17	2.9-3.0
18	3.0-3.1
19	3.1-3.2
20	3.2-3.3
21	3.3-3.4
22	3.4-3.5
23	3.5-3.6

Figura 1.9. Valors OCR per CMD5

c) Registre CID (Card Identification Number)

CID és un registre de 128 bits. Conté informació d'identificació de la tarja SDIO que es connecta al host. Això és determinant per poder carregar al seu torn el *driver* corresponent. A continuació es detallen els camps dels que consta el registre:

Name	Field	Width	CID-slice
Manufacturer ID	MID	8	[127:120]
OEM/Application ID	OID	16	[119:104]
Product name	PNM	40	[103:64]
Product revision	PRV	8	[63:56]
Product serial number	PSN	32	[55:24]
reserved	--	4	[23:20]
Manufacturing date	MDT	12	[19:8]
CRC7 checksum	CRC	7	[7:1]
not used, always 1	-	1	[0:0]

Figura 1.10. Camps del registre CID.

d) Registre CSD (Card-Specific Data)

El registre CSD és de 128 bits i proveeix informació referent a l'accés dels continguts de la tarja. Defineix el format de dades, tipus de correcció d'errors,

màxim temps d'accés a les dades,... A continuació es mostren en detall els camps dels que consta el registre.

Name	Field	Width	Value	Cell Type	CSD-slice
CSD structure	CSD_STRUCTURE	2	00b	R	[127:126]
reserved	-	6	00 0000b	R	[125:120]
data read access-time-1	TAAC	8	xxh	R	[119:112]
data read access-time-2 in CLK cycles (NSAC*100)	NSAC	8	xxh	R	[111:104]
max. data transfer rate	TRAN_SPEED	8	32h or 5Ah	R	[103:96]
card command classes	CCC	12	01x110110101b	R	[95:84]
max. read data block length	READ_BL_LEN	4	xh	R	[83:80]
partial blocks for read allowed	READ_BL_PARTIAL	1	1b	R	[79:79]
write block misalignment	WRITE_BLK_MISALIGN	1	xb	R	[78:78]
read block misalignment	READ_BLK_MISALIGN	1	xb	R	[77:77]
DSR implemented	DSR_IMP	1	xb	R	[76:76]
reserved	-	2	00b	R	[75:74]
device size	C_SIZE	12	xxxh	R	[73:62]
max. read current @VDD min	VDD_R_CURR_MIN	3	xxxb	R	[61:59]
max. read current @VDD max	VDD_R_CURR_MAX	3	xxxb	R	[58:56]
max. write current @VDD min	VDD_W_CURR_MIN	3	xxxb	R	[55:53]
max. write current @VDD max	VDD_W_CURR_MAX	3	xxxb	R	[52:50]
device size multiplier	C_SIZE_MULT	3	xxxb	R	[49:47]
erase single block enable	ERASE_BLK_EN	1	xb	R	[46:46]
erase sector size	SECTOR_SIZE	7	xxxxxxxb	R	[45:39]
write protect group size	WP_GRP_SIZE	7	xxxxxxxb	R	[38:32]
write protect group enable	WP_GRP_ENABLE	1	xb	R	[31:31]
reserved (Do not use)		2	00b	R	[30:29]
write speed factor	R2W_FACTOR	3	xxxb	R	[28:26]
max. write data block length	WRITE_BL_LEN	4	xxxxb	R	[25:22]
partial blocks for write allowed	WRITE_BL_PARTIAL	1	xb	R	[21:21]
reserved	-	5	00000b	R	[20:16]
File format group	FILE_FORMAT_GRP	1	xb	R/W(1)	[15:15]
copy flag (OTP)	COPY	1	xb	R/W(1)	[14:14]
permanent write protection	PERM_WRITE_PROTECT	1	xb	R/W(1)	[13:13]
temporary write protection	TMP_WRITE_PROTECT	1	xb	R/W	[12:12]
File format	FILE_FORMAT	2	xxb	R/W(1)	[11:10]
reserved		2	00b	R/W	[9:8]
CRC	CRC	7	xxxxxxxb	R/W	[7:1]
not used, always'1'	-	1	1b	-	[0:0]

Figura 1.11. Camps del registre CSD.

e) Registre RCA (Relative Card Address)

Registre de 16 bits on es guarda l'adreça de la tarja i que aquesta fa pública durant el procés d'identificació de la tarja amb el host. El valor per defecte és 0x0000.

f) Registre DSR (Driver Stage Register)

Registre de 16 bits usat per millorar les prestacions del bus per condicions d'operació esteses (depenent de paràmetres com la longitud de bus, tassa de transferència o nombre de targetes). La seva implementació és opcional. Les targetes només SDIO (sense memòria) no suporten aquest registre.

g) Registre SCR (SD Configuration Register)

Registre de configuració de 64 bits que reflexa informació de la part de memòria de la tarja. Si la tarja és només I/O, el registre SCR no són suportats.

1.6.2. Topologia de busos

Tal com s'ha comentat anteriorment, el estàndard per les targetes SDIO permet dos modes de funcionament diferent; SPI (de suport obligatori) i SD, en els modes de transferència de dades de 1-bit (de suport també obligat) i el mode 4-bit (obligatori per targetes d'alta velocitat i només opcionals per aplicacions de baixa velocitat). A continuació, descriurem cadascun dels dos modes d'operació en detall.

1.6.3. Bus SD

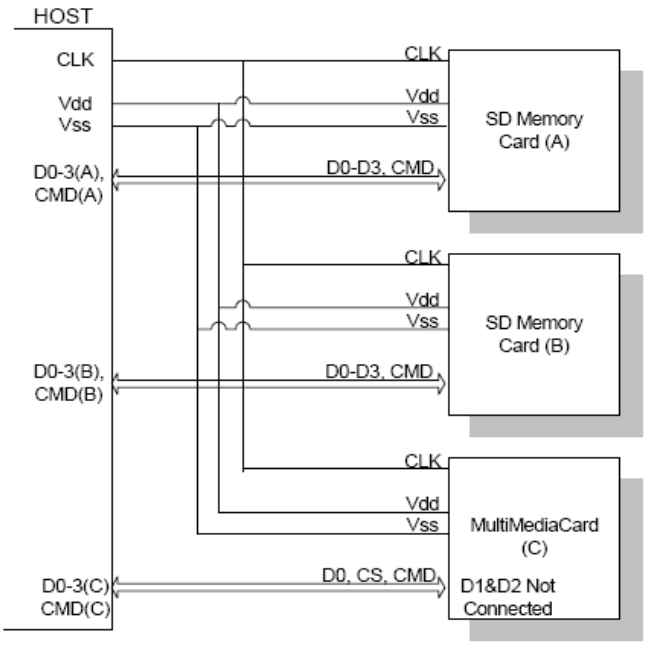


Figura 1.12. Topologia del bus del sistema SD memory.

El bus SD inclou els següents senyals:

CLK : Senyal de rellotge del host a la tarja.
CMD : Senyal bidireccional de Comanda/Resposta.
D_0 - D_3 : 4 senyals de dades bidireccionals.
V_{DD} , V_{SS} : Senyals d'alimentació i terra.

El bus SD Memory Card bus té un únic màster (aplicació), múltiples esclaus (targetes) i topologia en estrella síncrona; tal i com es veu a la Figura 1.12. Els senyals de rellotge, alimentació i terra són comuns a totes les targes. Els senyals de comanda (CMD) i dada (D0 - D3) són dedicades a cada tarja proveint una connexió punt a punt a totes les targetes.

Durant el procés d'inicialització, les comandes són enviades a cada tarja de manera individual, permeten a l'aplicació detectar les targetes i assignar adreces lògiques als slots físics. Les dades són sempre enviades (rebudes) a(des de) cada tarja de manera individual. Malgrat tot, per tal de simplificar la gestió de la pila de tarja, després del procés d'inicialització, totes les comandes

poden enviar-se concurrentment a totes les targetes. La informació d'adreça és continguda en els paquets de comandes.

El bus SD bus permet una configuració dinàmica del nombre de línies de dades. Després de la posta en marxa, per defecte, la tarja SD fa servir només D0 per la transferència de dades. Després de la inicialització, el host pot canviar l'amplada del bus (nombre de línies de dades actives). Aquesta característica permet fer fàcilment un balanceig entre cost del hardware i prestacions del sistema.

1.6.4. Bus SPI

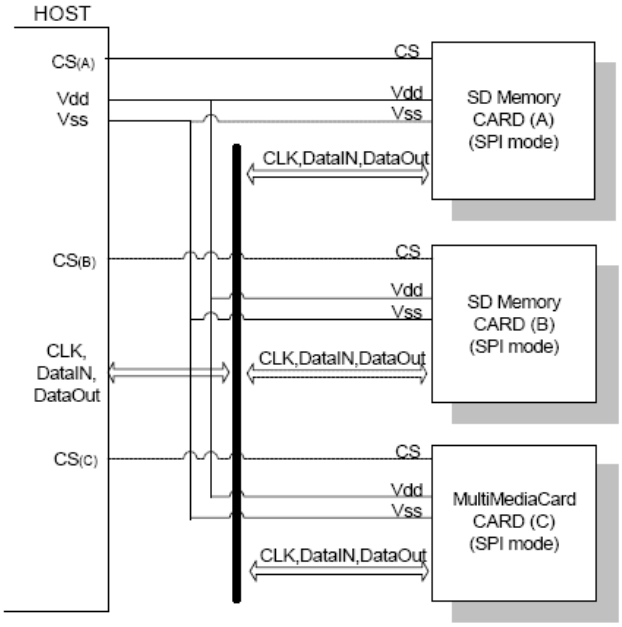


Figura 1.13. Topologia del bus del sistema SD memory (mode SPI).

El mode de comunicació compatible SPI de les targetes de memòria SD està dissenyada per comunicar-se mitjançant el canal SPI, el qual es troba en molts microcontroladors del mercat. La interfície es seleccionada durant la primera comanda de reset, després de la posta en marxa i no es pot canviar mentre el component estigui alimentat. L'estàndard SPI defineix l'enllaç físic només, i no el protocol complert de transferència de dades. La implementació de la tarja SD Memory fa servir el mateix conjunt de comandes que el mode SD. Des del punt

de vista de l'aplicació, l'avantatge del mode SPI és la capacitat de fer servir hosts ja implementats, i d'aquesta manera reduir l'esforç de disseny al mínim. El desavantatge més clar és la pèrdua de prestacions comparat amb el mode SD el qual permet la opció d'utilitzar un bus més ample.

La interfície SPI de les targetes SD memory card és compatible amb els hosts SPI del mercat. De la mateixa manera que d'altres dispositius SPI, el canal SD Memory Card consisteix dels següents quatre senyals:

CS : Senyal Chip Select del host a la tarja.
CLK : Senyal de rellotge del host a la tarja.
DataIN : Senyal de dades del host a la tarja.
DataOut : Senyal de dades de la tarja al host.

Una altra característica comuna de SPI són les transferències per bytes, que també es troba implementat en la tarja. Totes les unitats de dades són múltiples del byte (8 bits) i senyals alineats al byte amb el senyal CS.

La identificació de la tarja i els mètodes d'adreçament es reemplacen pel senyal hardware Chip Select (CS).

No hi ha comandes de *broadcast*. Per cada comanda, es selecciona una tarja (slave) activant (a baixa) el senyal CS. El senyal CS ha d'estar contínuament actiu durant la durada de la transacció SPI (comanda, resposta i dada). La única excepció ocorre durant la programació de la tarja, quan el host pot desactivar el senyal CS sense afectar el procés de programació.

La interfície SPI fa servir 7 dels 9 senyals SD disponibles (D1 i D2 no es fan servir i D3 és el senyal CS) del bus SD vist en l'apartat anterior.

1.6.5. Protocol del Bus

Tal com s'ha comentat anteriorment, el estàndard per les targetes SDIO permet 3 modes de funcionament diferents: SPI (de suport obligatori) i SD, en els modes de transferència de dades de 1-bit (de suport també obligat); i el mode 4-bit (obligatori per targetes d'alta velocitat i només opcionals per

aplicacions de baixa velocitat). A continuació anem a descriure cadascun dels dos modes d'operació en detall.

a) Mode SD

La comunicació sobre el bus SD es basa en corrents de bits (bit strams) de comandes i dades que s'inicien amb un bit d'inici (*start*) i un de parada (*stop*).

Una comanda és una unitat que inicia una operació. La comanda s'envia des del host ja sigui a una única tarja (comanda adreçada) o a totes les que estiguin connectades (comanda difosa). Una comanda es transfereix en forma sèrie per la línia CMD.

Una resposta és una unitat enviada des de una tarja en concret o (síncronament) des de totes les targetes connectades, al host com a resposta d'una comanda rebuda prèviament. Les respostes també són transferides en sèrie per la línia CMD. response is a token which is sent

Les dades es poden transmetre ja sigui des del host a la tarja o viceversa. En aquest cas, les dades utilitzen les línies de dades.

L' adreçament de les targetes s'implementa fent servir una adreça de sessió assignada a la tarja durant la fase d'inicialització.La transacció bàsica al bus SD és una transacció comanda/resposta tal i com es mostra a la Figura 1.14. Aquest tipus de transaccions al bus transfereixen la informació directament dins les estructures de comanda o resposta. Addicionalment, algunes operacions tenen una unitat de dades.

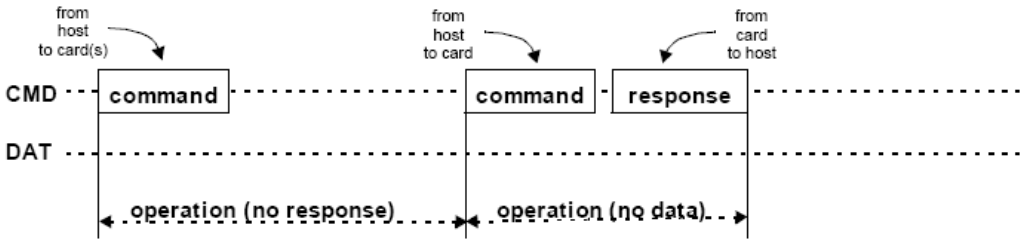


Figura 1.14. Operacions bàsiques al bus SD.

Les transferències de dades a/des de la SD Memory Card es realitzen en blocs. Tots els blocs de dades són seguits de bits CRC. Es defineixen també operacions simples i multibloc; aquestes darreres per augmentar les operacions d'escriptura. Una transmissió en blocs múltiples o multibloc s'acaba quan es

rep una comanda de stop per la línia CSM. Les transferències de dades poden ser configurades pel host per fer servir una única o varies línies de dades.

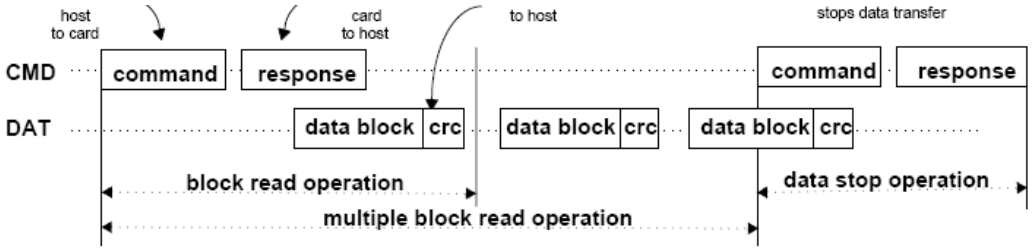


Figure 1.15. Operació de lectura en format multibloc.

La operació d'escriptura d'un bloc fa servir una senyalització simple basada en l'activació de la línia de dades DAT0 (*busy*) mentre dura el procés d'escriptura, i això independentment del nombre de línies de dades que es facin servir per transferir les dades.

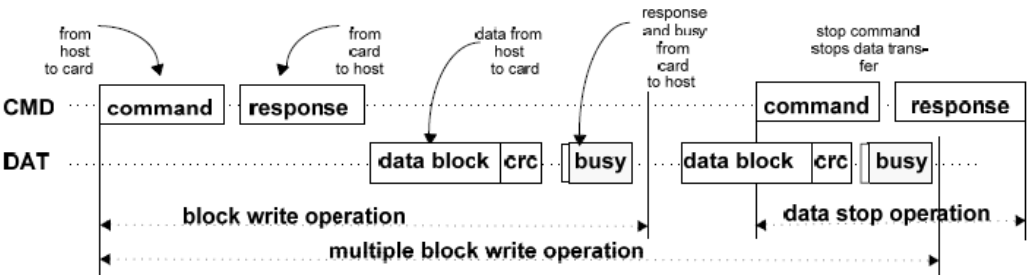


Figura 1.16. Operació d'escriptura.

Les trames de comandes segueixen l'esquema mostrat a continuació:

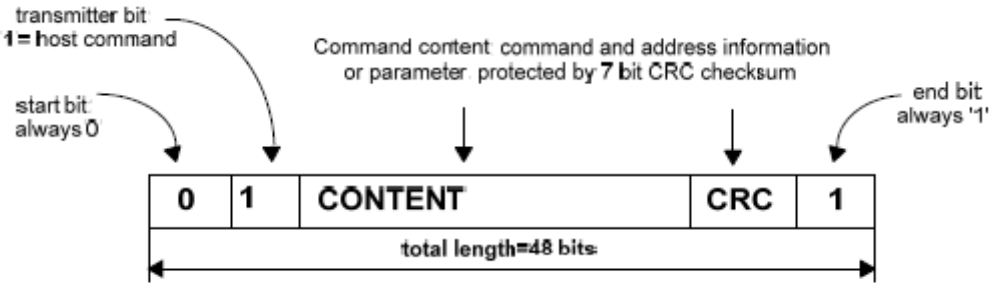


Figura 1.17. Format de comanda.

Cada trama de comanda és precedida per un bit de start (0) i finalitzada amb un bit d'acabament (1). La longitud total és de 48 bits. Cada trama, a més, és protegida per un conjunt de bits CRC, i per tant els errors en la transmissió són detectats, i així les operacions es poden reiniciar.

El camp CONTENT es pot separar en 6 bits corresponents a la comanda sol·licitada i els restants 32 bits corresponents a paràmetres de la comanda sol·licitada. Els 6 bits de la comanda donen possibilitat a 64 comandes diferents que es llisten a l'Apèndix 2.

Les unitats o trames de resposta tenen quatre possibles esquemes de codificació, depenent del seu contingut. Així, la longitud total de la trama pot ser de 48 o 136 bits. L'algorisme de protecció CRC per blocs de dades és un polinomi de 16-bits CCITT.

A la línia CMD, el bit més significant (MSB) es transmet en primer lloc i el menys significant en darrer lloc (LSB). Quan es fa servir la opció de disposar de totes les línies del bus de dades, es transfereixen 4 bits al mateix temps. Els bits d'inici, final i així com també els bits de CRC, es transmeten per cadascuna de les línies de dades DAT. La resposta d'estat de CRC i la indicació d'ocupat (*busy*) serà enviada de la tarja al host a la línia de dades DAT0 només.

b) Mode SPI

Mentre el canal SD es basa en trames de bit de comandes i dades, iniciades amb un bit d'inici i un de final, el canal SPI està orientat a byte. Cada comanda o bloc de dades està constituït per bytes de 8 bits i està alineat al byte al senyal CS (és adir, la longitud total és múltiple de 8 cicles de rellotge).

A l'igual que el protocol SD, els missatges SPI consisteixen d'unitats de comanda, resposta i bloc de dades. Tota la comunicació entre el host i les targetes està controlada pel host (*master*). El host inicia cada transacció del bus abaixant el senyal CS.

El comportament en la resposta del mode SPI difereix del mode SD en tres aspectes fonamentals:

- La tarja seleccionada sempre respon a la comanda.
- Es fa servir dues estructures de resposta noves (8 i 16 bits).
- Quan la tarja troba problemes d'accés a les dades, contestarà amb una resposta d'error (que reemplaça el bloc de dades esperat) en comptes d'una situació de *time-out* com al mode SD.

Addicionalment a la resposta de la comanda, cada bloc de dades enviat a la tarja durant les operacions d'escriptura serà respost amb una unitat especial de resposta de dades.

Lectura de dades

El mode SPI suporta comandes de lectura senzilles i multibloc. Malgrat tot, per tal d'acomplir amb l'estàndard SPI, només es fan servir dos senyals unidireccionals. Quan es rep una comanda de lectura, la tarja contestarà amb una unitat de resposta seguida per una unitat de dades amb la longitud definida en una comanda prèvia SET_BLOCKLEN (CMD16). Una operació de lectura multibloc finalitza, similar al protocol SD, amb una comanda de STOP_TRANSMISSION.

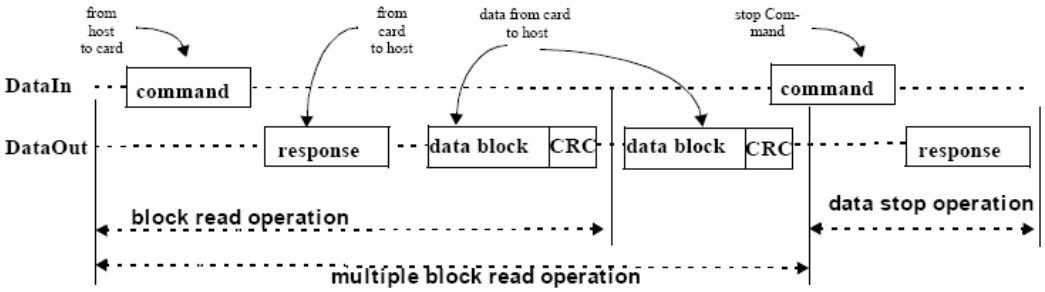


Figura 1.18. Operació de lectura.

A un bloc de dades vàlid se li afegeix el CRC de 16 bits generat pel polinomi CCITT estàndard $x^{16}+x^{12}+x^5+1$.

En cas d'una lectura errònia de dades, la tarja no transmetrà cap dada. En comptes d'això, transmetrà una unitat de error de dades cap el host. A la figura següent es mostra una operació de lectura finalitzada amb una unitat d'error en comptes d'un bloc de dades.

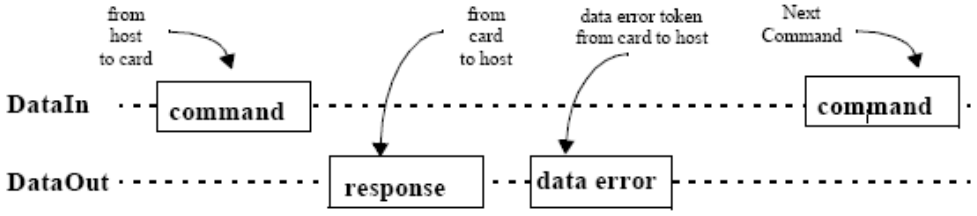


Figura 1.19. Operació de lectura. Error en les dades.

Escriptura de dades

El mode SPI suporta operacions d'escriptura senzilles i múltiples. Després de rebre una comanda d'escriptura, la tarja contestarà amb una unitat de resposta i esperarà que una bloc de dades sigui enviat des del host. En aquest cas també es disposa del control CRC i les restriccions de mida de bloc exposades en el cas de lectura.

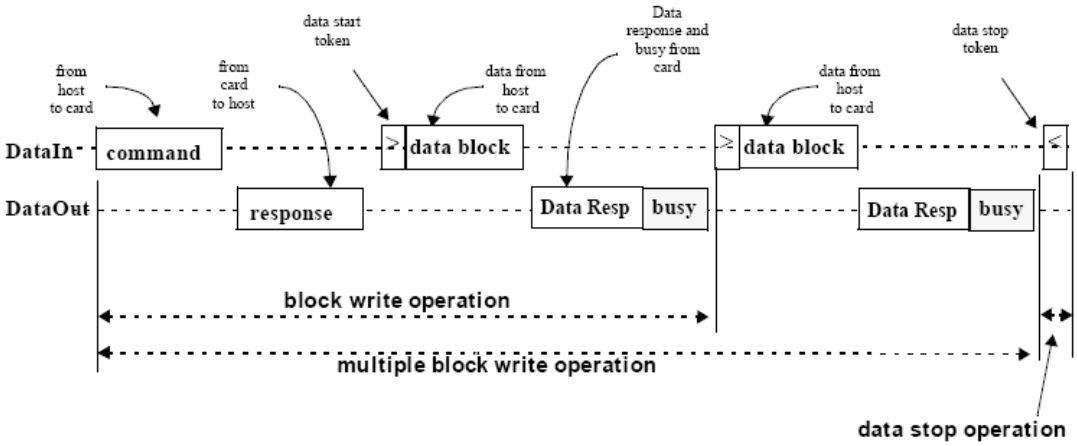


Figura 1.20. Operació d'escriptura.

Després que s'hagi rebut un bloc de dades, la tarja contestarà amb una unitat o trama de resposta. Si aquest bloc no conté errors, serà programat. Mentre la tarja estigui ocupada programant, una trama continua d'unitat indicant ocupat (*busy*) serà enviada cap el host (línia **DataOut** a nivell baix).

1.6.6. Comandes SDIO.

Com a referència es llisten a l'Apèndix 2 les comandes corresponents al subconjunt SDIO tant pels modes SPI com SD. Una informació més detallada de les mateixes es pot trobar a la pàgina web de la SD Card Association (http://www.sdcard.org), en l'especificació SDIO (SDIO Simplified Specification).

1.6.7. Inicialització de la tarja SDIO.

Com hem comentat anteriorment, un dels requeriments de la especificació SDIO es que la tarja SDIO no ha de causar cap mena de dany a hosts no preparats per aquesta especificació. Per fer-ho, s'afegeix una nova comanda

(CMD5) per reemplaçar la comanda ACMD41 per la inicialització per hosts preparats per suportar SDIO.

Després d'un reset o inicialització, totes les funcions I/O de la tarja estan rehabilitades i la porció I/O de la tarja no hauria d'executar cap operació excepte CMD5 o CMD0 amb CS a nivell baix. En el cas que la tarja tingués una part de memòria SD instal·lada (anomenada tarja combo), aquesta memòria hauria de respondre normalment a totes les comandes obligatòries per la memòria.

Una tarja només d'I/O no hauria de respondre a la comanda ACMD41 i d'aquesta manera, aparèixer inicialment, com una tarja MMC. Aquesta tarja tampoc hauria de respondre a la comanda CMD1 usada per inicialitzar les targetes MMC, i apareixen com una tarja sense cap mena de resposta. El host, aleshores deshabilita la tarja i la força a un estat inactiu. La operació d'una tarja I/O connectada a un host no preparat per I/O es mostra a la Figura 1.21. Les línies sòlides corresponen als camins presos mentre les línies puntejades no són executades.

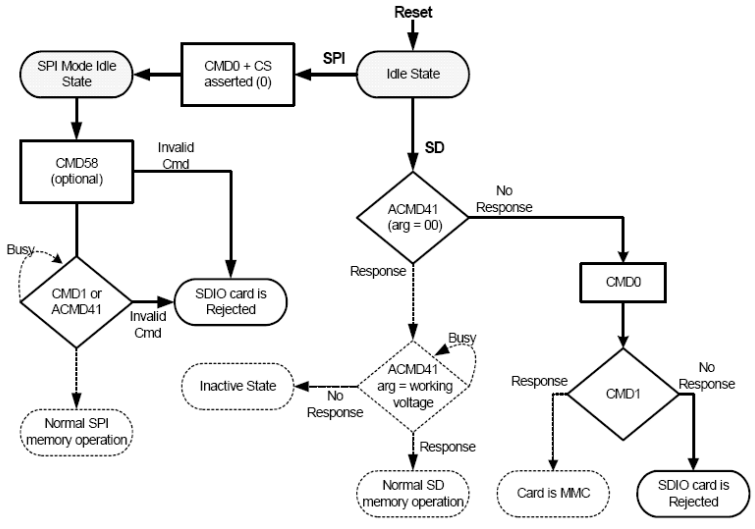
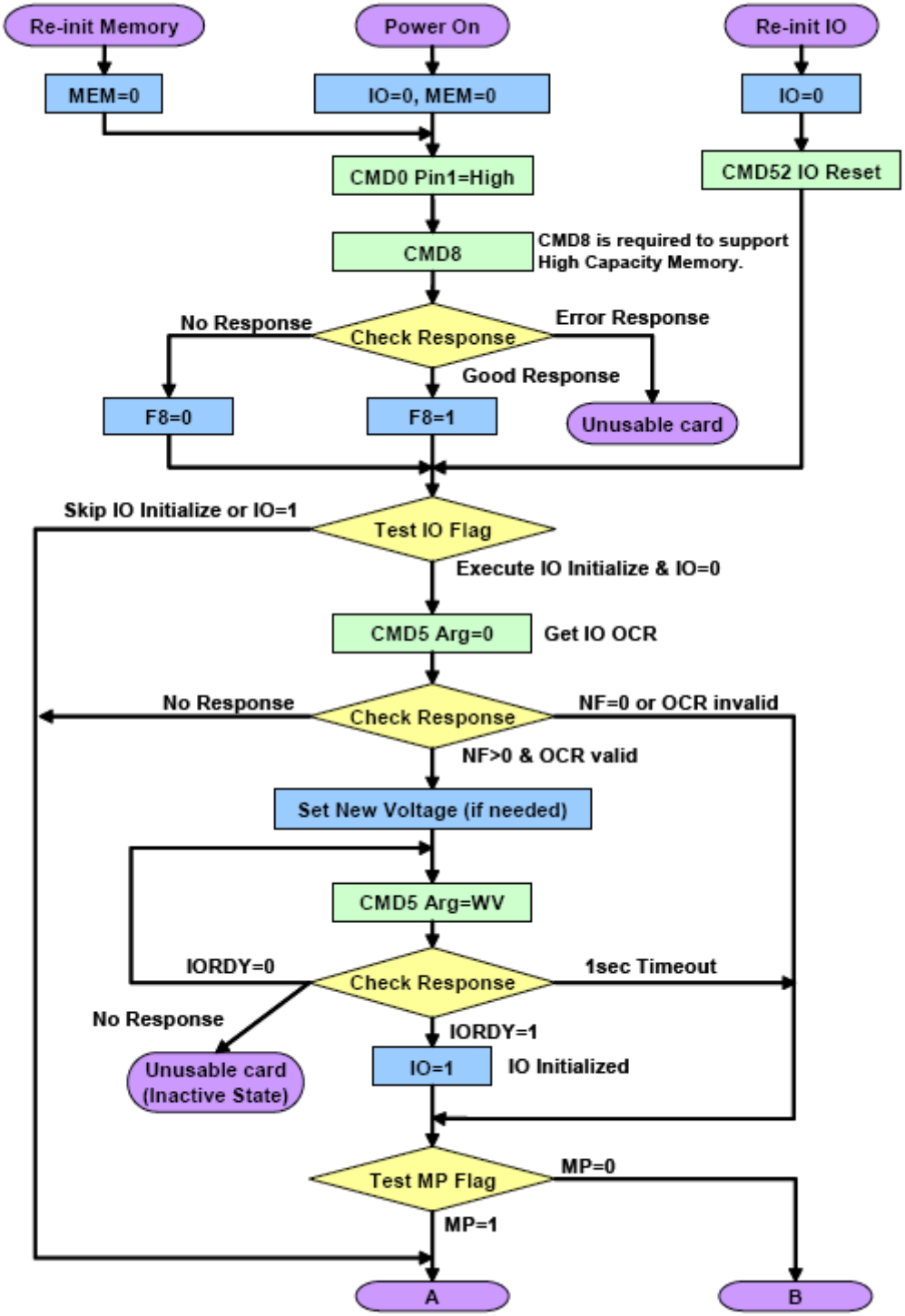
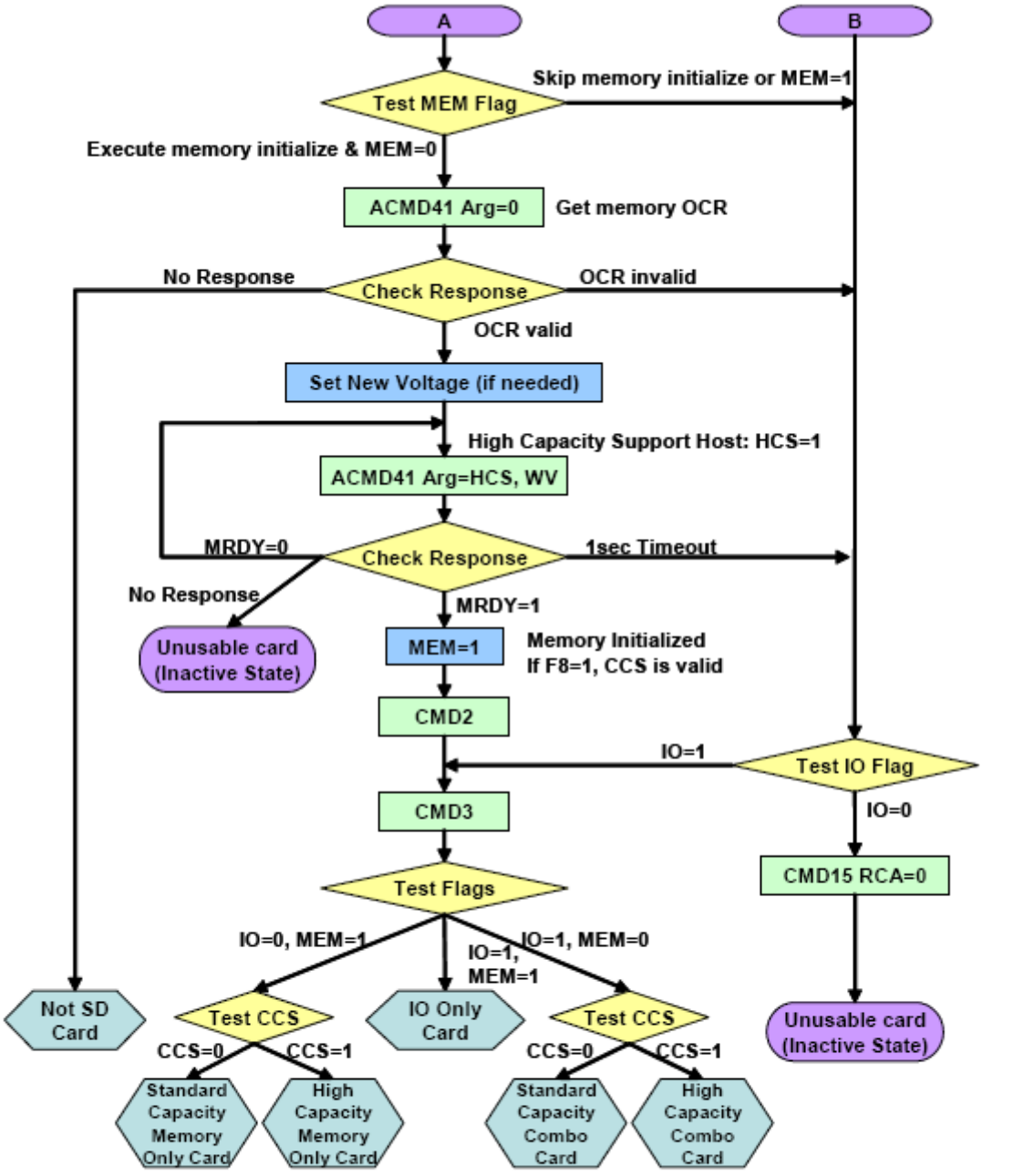


Figura 1.21. Resposta SDIO per sistemes que no el suporten.

Un host preparat per reconèixer targetes SDIO, envia CMD5 abans de la parella CMD55/ACMD41 i d'aquesta manera rebrà un OCR vàlid i continuaria inicialitzant la tarja. La tarja SDIO romandrà inactiva i no respondrà a cap comanda a menys que sigui CMD5. A la Figura 1.22 es mostra el mode

d'operació pel mode SD en hosts preparats per SDIO. Al full d'especificacions SDIO simplificat, es pot consultar també pel cas SPI.





Variables

NF:	Number of I/O Functions (CMD5 Response)	CCS:	Card Capacity Status (ACMD41 Response)
MP:	Memory Present Flag (CMD5 Response)	Flags	
IORDY:	I/O Power-up Status (C bit in the CMD5 response)	IO:	I/O Functions Initialized Flag
MRDY:	Memory Power-up Status (OCR Bit31)	MEM:	Memory Initialized Flag
HCS:	Host Capacity Support (ACMD41 Argument)	F8:	CMD8 Flag

Figura 1.22. Fluxe d'inicialització de targetes en mode SD (per hosts preparats per SDIO)

Una tarja combinada (combo card), amb part I/O i memòria, romandrà en el mode de només memòria en el cas de tenir un host no preparat per I/O. Si és un host que reconeix I/O, enviarà una comanda CMD5 a la tarja i aquesta

respondrà amb un missatge que contindrà el nombre de funcions de I/O disponibles i la existència de memòria SD.

Un cop el host ha inicialitzat la porció I/O de la tarja, llegeix els registres fixes del mapa intern, CIA (Common Information Area), de la mateixa, mitjançant una comanda de lectura. La CIA conté els registres CCR (Card Common Control Registers) i FBR (Function Basic Registers). També a la CIA es troben apuntadors a la CIS (Card Information Structure). Aquesta estructura conté informació d'alimentació, funcions, fabricant i d'altres informacions que el host necessita per determinar si les funcions d'I/O poden activar-se. En cas afirmatiu, un registre a l'àrea CCCR habilita la tarja i cada funció individual.

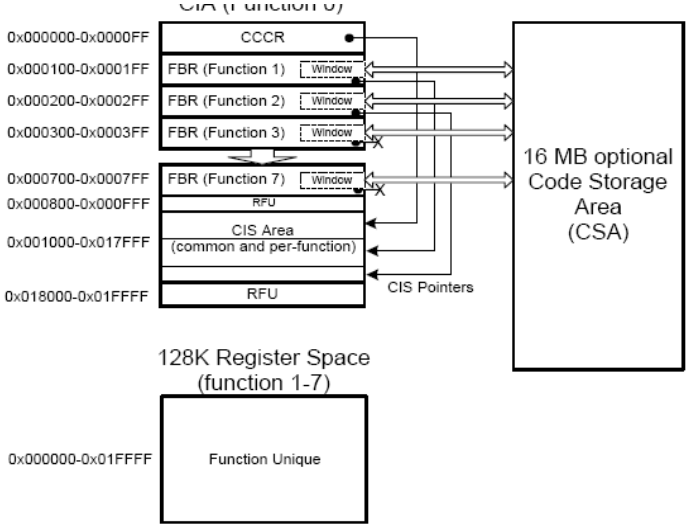


Figura 1.23. Mapa Intern SDIO

1.7. Introducció al protocol Zigbee

Zigbee és un protocol de comunicacions sense fils, similar al bluetooth, basat en l'estàndard IEEE 802.15.4. L'àmbit principal d'ús és troba en aplicacions de baix preu que necessitin una **tassa de dades baixa i consum d'energia baix**. A dia d'avui, el tipus d'aplicacions més habituals d'aquest protocol són, per exemple, en domòtica, sistemes de recollida de dades mèdiques, control industrial, sistemes de seguretat, joguines interactives, xarxes de sensors o xarxes de curt abast per citar-ne d'algunes. En el quadre comparatiu de la Figura 1.24, es mostren algunes de les diferències bàsiques entre Zigbee i

d'altres sistemes de comunicació sense fils. Per una informació més detallada d'aquest protocol, es recomana visitar el lloc web www.zigbee.org

Market Name Standard	ZigBee™ 802.15.4	GPRS/GSM 1xRTT/CDMA	Wi-Fi™ 802.11b	Bluetooth™ 802.15.1
Application Focus	Monitoring & Control	Wide Area Voice & Data	Web, Email, Video	Cable Replacement
System Resources	4KB - 32KB	16MB+	1MB+	250KB+
Battery Life (days)	100 - 1,000+	1-7	.5 - 5	1 - 7
Network Size	Unlimited (2^{64})	1	32	7
Bandwidth (KB/s)	20 - 250	64 - 128+	11,000+	720
Transmission Range (meters)	1 - 100+	1,000+	1 - 100	1 - 10+
Success Metrics	Reliability, Power, Cost	Reach, Quality	Speed, Flexibility	Cost, Convenience

Figura 1.24. Comparació de Zigbee amb altres estàndards sense fils.

1.8. Introducció a Windows CE

Windows CE (sovint abreujat com WinCE) és una variació del sistema operatiu Microsoft Windows per ordinadors mòbils i sistemes encastats (*embedded systems*). El sistema operatiu està optimitzat per dispositius que disposen d'una capacitat d'emmagatzematge mínima i el seu *kernel* pot córrer en sistemes per sota d'un megabyte de memòria. Els dispositius es configuren sovint sense emmagatzematge en disc i poden ser configurats com a sistemes "tancats" que no permetin cap extensió d'usuari (instal·lats en ROM). Associat a un nucli o plataforma de Windows CE, apareixen diferents versions ajustades a diferents tipus de dispositius com telèfons mòbils, ordinadors de butxaca o aplicacions de la indústria de l'automòbil per exemple.

En el gràfic següent, es mostra la evolució temporal i les versions que han aparegut del sistema operatiu des de la seva primera implementació l'any 1996.

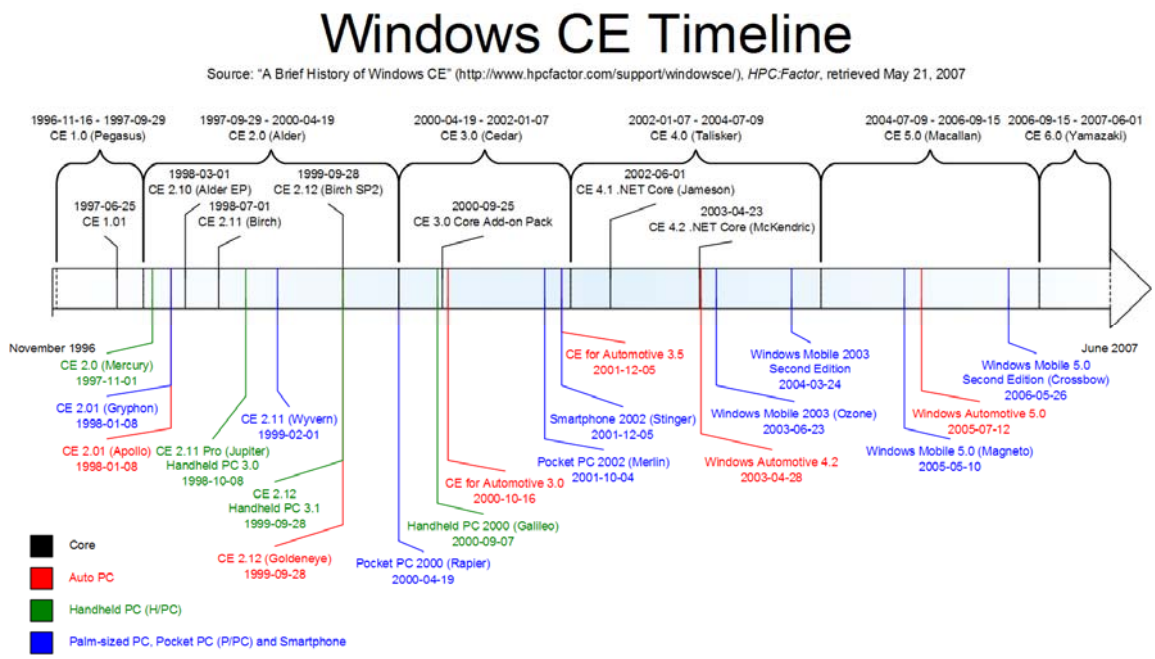


Figure 1.25. Evolució història dels sistemes Windows CE

Fins al nucli CE 5.0 no s'incorpora un *stack* SDIO al sistema per poder utilitzar dispositius que operen amb aquest estàndard. Cada fabricant crea el seu *driver* SDIO propi o en fan servir d'alguns generats per terceres parts.

Com el sistema de desenvolupament inicial del projecte, proporcionat pel departament de microelectrònica, està basat en Windows CE 4.2 .NET, no tenim suport SDIO per part del sistema, per la qual cosa haurem de crear un controlador que ens permeti treballar amb aquest estàndard.

1.9. Introducció a les FPGA

Les FPGA (field-programmable gate array) són dispositius semiconductors que contenen components de lògica programable anomenats blocs lògics i connexions programables. Aquests blocs lògics poden programar-se per realitzar des de la funció de portes lògiques fins a funcions combinacionals més complexes com descodificadors o funcions matemàtiques simples. En moltes FPGA, aquests components lògics programables també inclouen elements de memòria que poden ser des de simples flip-flops a bloc de memòria més complexos.

Les FPGA són generalment més lentes que els ASIC equivalents ja que utilitzen mes transistors que aquests (en un factor superior a 10) per

implementar les mateixes funcions. De totes maneres, les FPGA tenen certes avantatges com la reducció de temps de disseny fins a disposar d'un prototip funcional, la possibilitat de ser reprogramats en qualsevol moment amb l'objectiu de corregir possibles errors, i reduir les despeses d'investigació, disseny i test de qualsevol desenvolupament.

Últimament, hi ha una tendència en combinar els blocs lògics i interconnexions de les FPGA tradicionals amb processadors encastats (*embedded*) i perifèrics relacionats per tal de formar un "sistema en un xip programable" (SoC) complert. Hi ha una aproximació alternativa que consisteix en comptes d'usar processadors "hard" en usar nuclis de processadors implementats dintre la lògica de la FPGA ("soft processors"). En el cas del nostre projecte, per les FPGA Altera es poden crear els "soft processors" Nios i Nios II malgrat el codi de la seva estructura HW no sigui obert.

Per tal de definir el comportament de la FPGA, el desenvolupador ha de proveir el seu disseny en forma esquemàtica o en llenguatge de descripció de hardware (HDL). Els llenguatges HDL més comuns són el VHDL i el Verilog. Aleshores, mitjançant alguna eina automàtica de disseny electrònic, es genera la *netlist* corresponent a la tecnologia usada. Aquesta netlist s'adapta a l'arquitectura de la FPGA usada (*place-and-route*) i que generalment es fa de forma automàtica mitjançant un software proveït pel fabricant de la FPGA. A partir d'aquest moment, el desenvolupador validarà els resultats anteriors mitjançant simulació, anàlisi de temps i altres metodologies de verificació. Un cop el disseny i el procés de validació ha acabat, el fitxer binari generat s'usa per (re)configurar la FPGA, de nou mitjançant un software del fabricant.

Sovint el disseny amb HDLs és massa complex. Alguns fabricants promocionen el llenguatge SystemC com una manera de combinar llenguatges d'alt nivell en un intent de reduir el temps de desenvolupament incrementant el nivell d'abstracció del disseny.

Per simplificar el disseny de sistemes complexos amb FPGAs, existeixen llibreries de funcions complexes predefinides i circuits que han estat prèviament testejats i optimitzats per tal d'accelerar el procés de disseny. Aquests circuits

ja predefinits s'anomenen "IP cores" i són proveïts tant pels fabricants de FPGA com per terceres parts (i rarament de manera gratuïta).

Capítol 2.
Planificació i entorn de desenvolupament

2.1. Planificació del projecte.

Tal i com s'ha explicat al punt 1.1, el projecte consta de quatre parts diferenciades encara que interrelacionades i que s'han efectuat bàsicament en l'ordre mostrat a continuació.

1. Driver SDIO per Windows CE al PocketPC.
2. Bridge Bus SD entre el PocketPC i la FPGA, implementant en aquesta última tota la funcionalitat SDIO esperada pel controlador del bus SD.
3. Bridge o enllaç de comunicació entre la FPGA i el microcontrolador.
4. Aplicació de control per Windows CE.

Per abordar els punts 1 i 2 anteriors, primerament va ser necessari fer una cerca d'informació referent a l'estàndard SD i en concret, al subconjunt SDIO a implementar, entenent els modes de funcionament, protocol, senyals, etc. Aquest primer punt no va estar exempt de dificultats degut que, com ja s'ha esmentat, l'estàndard SD no està completament obert i per tant la informació obtinguda és limitada. Per tal de complementar aquesta informació, es va optar per fer una cerca web d'altres projectes que fessin servir el bus i també realitzar experiments diversos a partir d'una càmera SDIO i una tarja de memòria SD.

Es va procedir al desenvolupament del driver per la PDA i posteriorment el bridge per la tarja SDIO i el microcontrolador. Per aquest últim, es va definir una arquitectura del sistema i posteriorment es van desenvolupar separadament els components del mateix amb llenguatge VHDL. Mitjançant eines de simulació es va comprovar el correcte funcionament tant dels components com del sistema sencer abans de procedir a la sintesi del disseny sobre la FPGA.

A continuació, es va desenvolupar una petita aplicació sobre la PDA per testejar funcionalment la comunicació entre el driver i la tarja SDIO mitjançant l'enviament de trames de control cap a la tarja i la comprovació de les trames rebudes a la PDA.

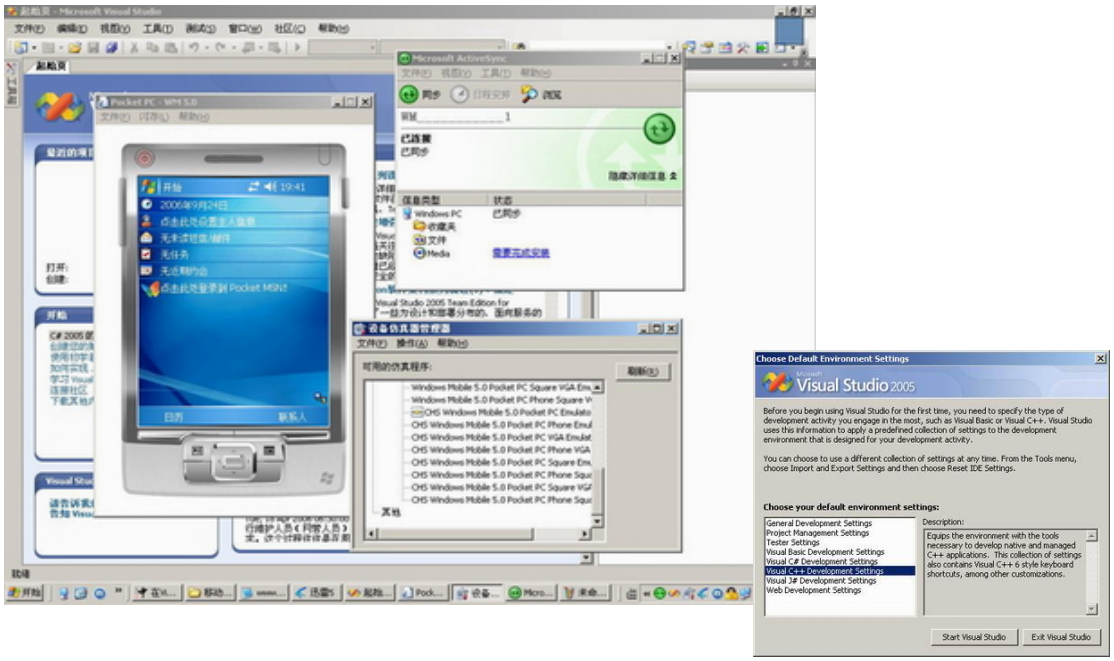
2.2. Eines i entorn de desenvolupament.

En les diverses fases del projecte s'han utilitzat un conjunt d'eines software específiques per dur a terme el desenvolupament. Cal anotar, que la majoria d'eines són comercials, i en molts casos hem hagut de treballar amb versions de demostració o bé amb versions reduïdes de prova. Afortunadament, la complexitat del sistema a desenvolupar no ha estat obstacle per poder utilitzar aquests programes malgrat les seves limitacions. Bàsicament, els programes de desenvolupament utilitzats es poden dividir en dues famílies: aquells destinats al desenvolupament de la part software del projecte (driver, aplicació); i els destinats al desenvolupament de la part hardware (a la FPGA i microcontrolador). A continuació, s'enumeren els programes usats i les seves característiques principals.

- **Visual Studio 2005 Professional Edition.**

Visual Studio és un entorn de desenvolupament integrat per a sistemes Windows. Es suporten varis llenguatges de programació com el Visual C++, Visual C#, Visual J#, ASP.NET i Visual Basic .NET entre d'altres.

Permet als desenvolupadors crear aplicacions en qualsevol entorn que suporti la plataforma .NET. Així entre d'altres, es poden crear aplicacions que s'intercomuniquin amb dispositius mòbils.



- **Windows Mobile 5.0 SDKs for Pocket PC and Smartphone.**
- **Windows Mobile 6.0 SDK.**

Els SDK (Software Development Kits) de Windows Mobile, són un conjunt d'eines software, recursos tècnics i exemples proporcionades per Microsoft pel desenvolupament d'aplicacions mòbils i que mitjançant un model de programació consistent, permet reduir considerablement els temps de desenvolupament. Té una integració total amb Microsoft Visual Studio i així, podem treballar amb els nostres dispositius de manera virtual o simulada.

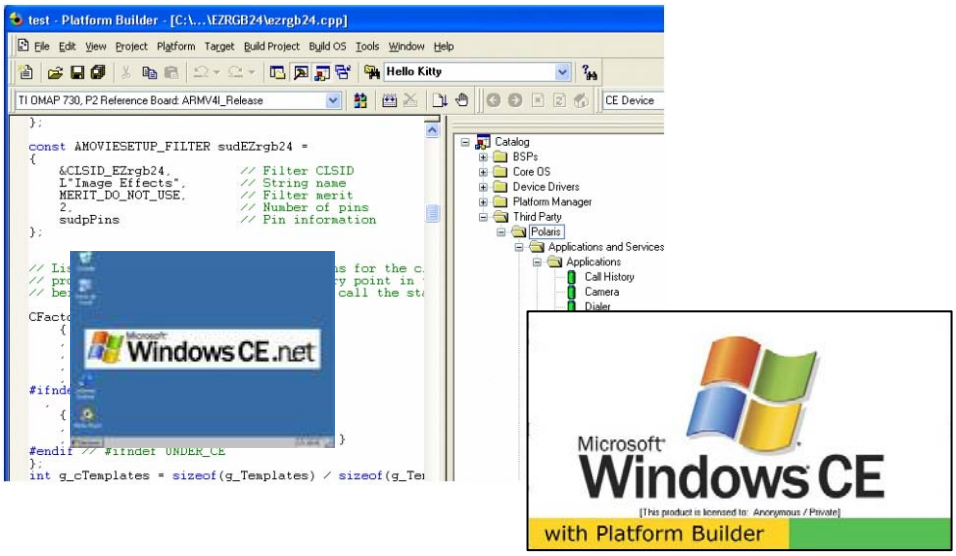
En el projecte s'ha usat bàsicament pel desenvolupament de l'aplicació externa de test del SDIO driver i, inicialment també es va fer servir per desenvolupament d'aquest últim fins que ens vam decantar finalment per Platform Builder comentat posteriorment.

- **.NET Compact Framework 1.0 i 2.0**

La plataforma .NET de Microsoft és un component software que s'afegeix al sistema operatiu Windows. Aporta un extens conjunt de solucions predefinides per les necessitats en la programació general d'aplicacions i administra l'execució dels programes escrits específicament amb la plataforma.

La versió Compact Framework, està dissenyada específicament per córrer en dispositius mòbils com PDAs, telèfons mòbils,...

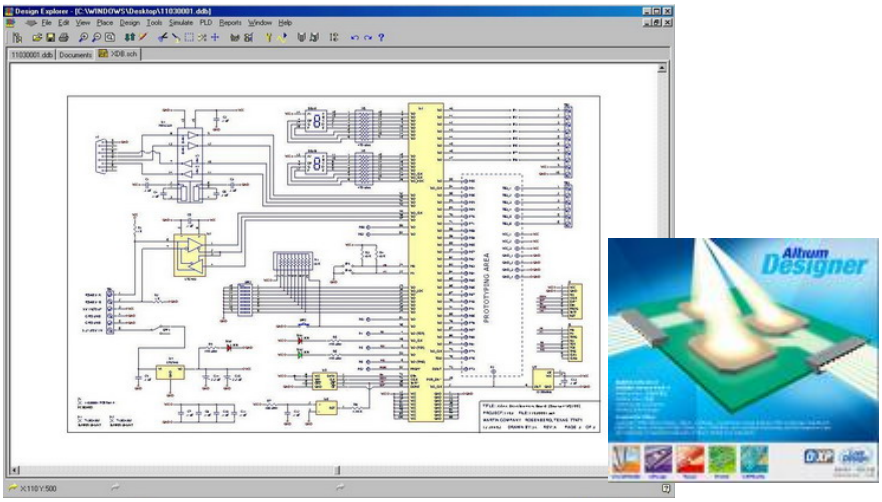
- **Platform Builder 5.0**



Platform Builder per Windows CE és un entorn de desenvolupament integrat per construir sistemes operatius embedded personalitzats basats en Windows CE. Platform Builder conté totes les eines de desenvolupament necessàries pel disseny, creació, construcció, test i depuració d'una imatge del Sistema Operatiu complert basat en Windows CE. És a dir permet dissenyar els propis nuclis del sistema operatiu i després baixar-los al dispositiu real.

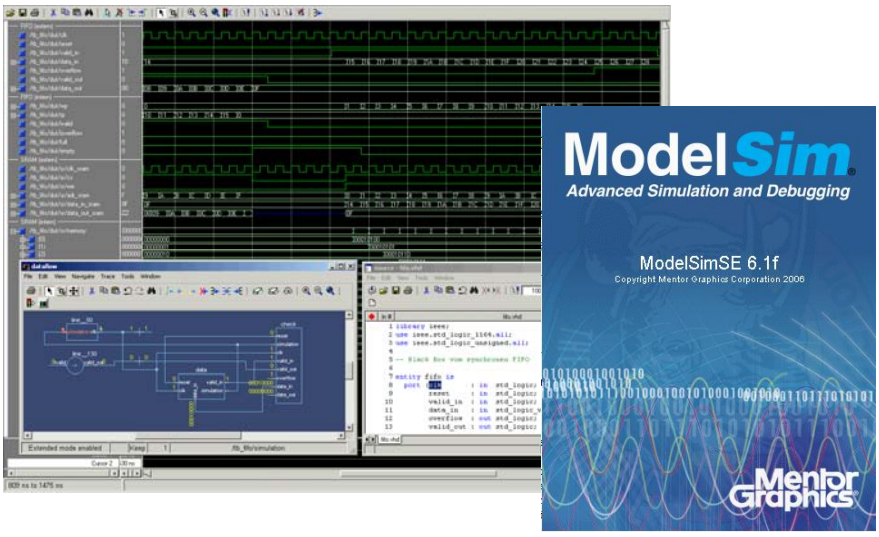
Aquesta eina ha estat amb diferència la més utilitzada ja que permet la creació específica de *drivers*. Així mateix, és la eina recomanada per Microsoft en aquest tipus de desenvolupaments.

- **Altium Designer (Protel 2004)**



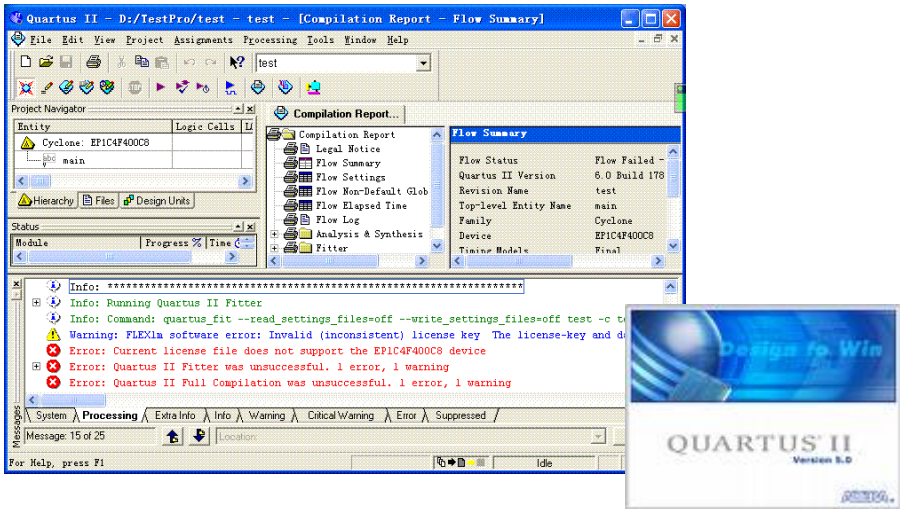
Altium Designer és un entorn CAD de desenvolupament de sistemes electrònics unificat que permet en una única aplicació des del disseny d'esquemàtics, fins a PCBs. El software també permet la programació i simulació de circuits FPGA. En el nostre cas s'ha fet servir bàsicament per gestionar fitxers esquemàtics i com a entorn ràpid de prova d'estructures escrites en VDHL abans d'introduir-la en el disseny final.

- **ModelSimSE**



ModelSim aporta un entorn còmode per la simulació i depuració per dissenys complexos d'ASICs i FPGAs. A més, compta amb el suport de multiples llenguatges com Verilog, SystemVerilog, VHDL i SystemC. Ha estat la eina de referència per efectuar totes les simulacions dels dissenys de la FPGA del projecte. Inicialment s'ha simulat els components VHDL per separat i finalment integrats en el conjunt de disseny. Així mateix per validar el sistema complert, s'han utilitzat uns vectors de test per certificar la validesa dels resultats obtinguts.

- **Altera Quartus II**



Quartus II és l'entorn específic de disseny d'Altera per les seves FPGA i CPLDs. L'entorn de disseny de Quartus II es composa d'un conjunt d'eines pel disseny a nivell de sistema, programació de software embedded, disseny de FPGA i CPLDs, síntesi, place-and-route, verificació i programació de dispositius. En el projecte, s'ha utilitzat aquest entorn de manera combinada amb ModelSim. De fet, hi ha versions de Quartus que l'incorporen.

En aquest punt, tant ModelSim com Quartus II s'han anat utilitzant de manera reiterativa per anar ajustant el disseny hardware fins que a les simulacions s'obté la resposta temporal esperada. És en aquest punt que el disseny es sintetitza finalment i es baixa a la FPGA.

Capítol 3.
Disseny i implementació

3.1. Arquitectura del sistema complert.

A la Figura 3.1 es mostra gràficament l'arquitectura del sistema complert amb els blocs principals de que consta:

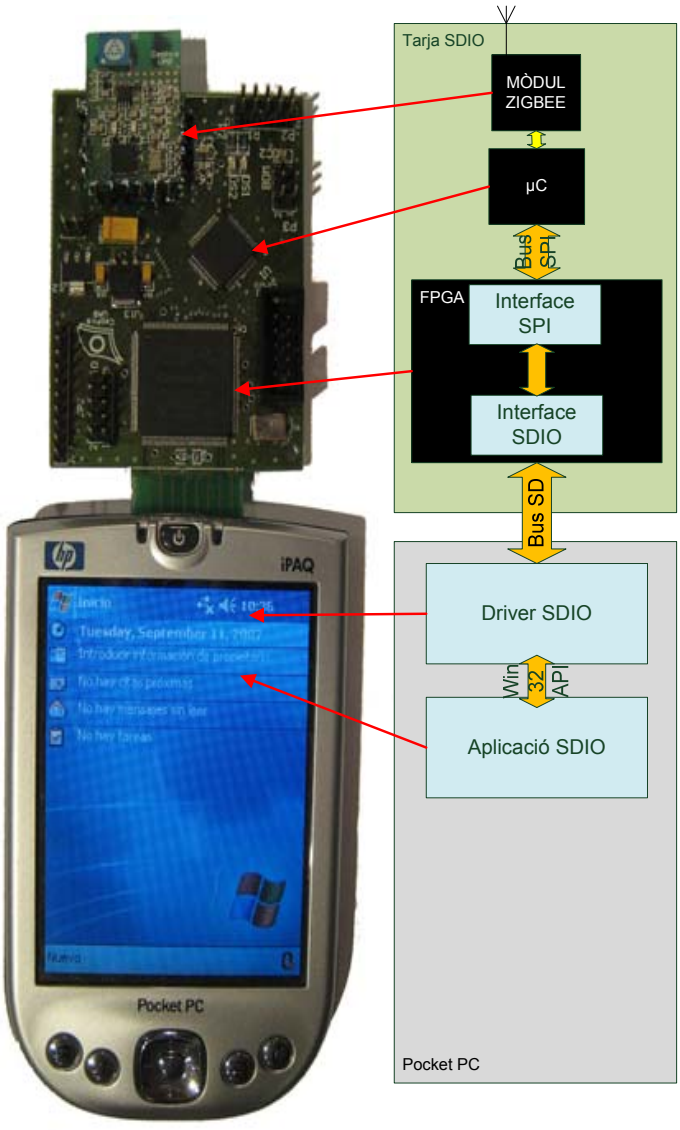


Figura 3.1. Arquitectura del sistema complet.

S'han destacat en color blau els blocs a desenvolupar en el projecte i que ja s'han comentat en els capítols anteriors.

En els apartats següents es detalla el disseny i implementació per cadascún d'ells.

3.2. Desenvolupament del driver SDIO.

Per tal de desenvolupar el stream driver SDIO s'ha seguit els passos següents:

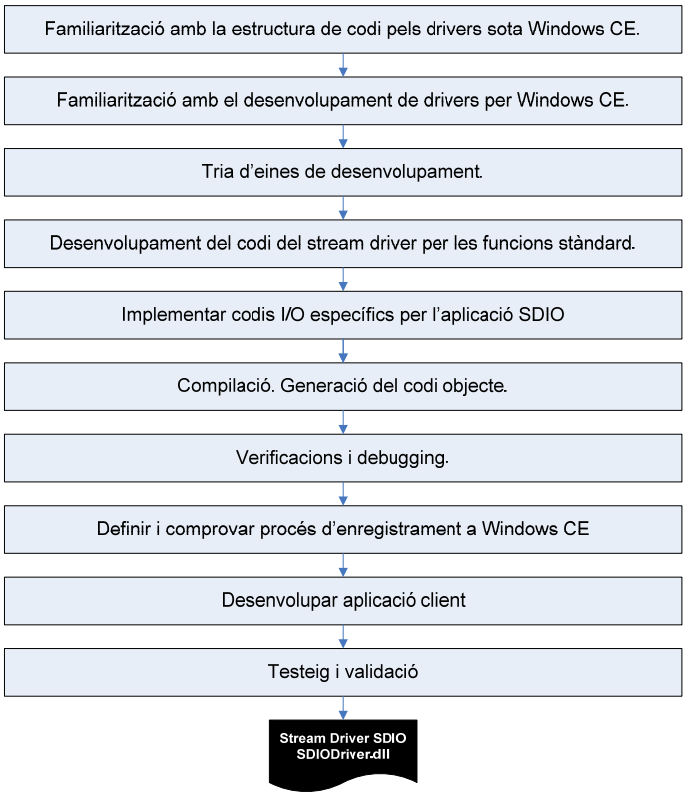


Figura 3.2. Flux de desenvolupament del driver.

3.2.1. Device Drivers (Controladors de dispositius)

a) Propòsit dels controladors de dispositius

Abans d'entrar en el procés d'escriure un driver SDIO, cal entendre el propòsit d'un driver: abstraure el hardware suportat del sistema operatiu i encara millor, del desenvolupador d'aplicacions. Així, a un desenvolupador no li caldria conèixer els detalls específics del hardware d'un port sèrie o un monitor,

per exemple. Windows, exposa als desenvolupadors el que s'anomena interfícies de programació per aplicacions (API) per tal que puguin fer crides al hardware però aquests no necessiten conèixer quin és el hardware físicament. Aquesta és la potència de les APIs.

b) Exemples d'APIs

Per exemple, si ens fixem la escriptura de dades pel port sèrie un desenvolupador només hauria de fer el següent:

1. Cridar a la funció **CreateFile( )** a **COMx.**
2. Cridar a la funció **WriteFile( )** per escriure alguns bytes de dades pel port.
3. Cridar a la funció **CloseHandle( )** per tancar el port sèrie.

Aquesta mateixa seqüència d'APIs funcionarà amb independència de quin sigui el hardware que incorpora el port sèrie anterior (or fins i tot, quin sistema operatiu Windows s'estigui fent servir). Això és pot extrapolar a altres tipus d'APIs com per exemple les de tipus gràfic que ens permeten crear i manipular imatges per pantalla amb independència del tipus de tarja de gràfica que s'estigui utilitzant.

Els desenvolupadors tenen d'aquesta manera, un conjunt consistent i ben conegut d'APIs per realitzar les seves crides, les quals abstrauen les seves aplicacions dels hardware subjacent.

3.2.2. Llibreries dinàmiques (DLLs)

Un driver no és més que una llibreria dinàmica (dynamic-link library o DLL), la qual es carrega en l'espai d'adreçament d'un procés pare. El procés pare, aleshores, pot fer crides a qualsevol de les interfícies o funcions exposades des de la DLL. El driver es carrega típicament pel seu procés pare a través d'una crida a **LoadLibrary( )** o **LoadDriver( )**.

Com pot saber un procés quina API o quines funcions s'exposen des d'una DLL o un driver? Doncs senzillament amb una crida, per part del procés pare, a la funció **GetProcAddress( )**, que pren el nom d'una funció i la **hInstance** de la

DLL carregada. La crida retorna un apuntador a la funció si aquesta existeix, o bé una valor NULL si la funció no és exportada des de la DLL.

3.2.3. Classificació dels drivers

A les plataformes que fan servir Windows CE, bàsicament trobem dos tipus de dispositius: dispositius integrats (built-in) i dispositius instal·lables. D'entre els primers, podem considerar la pantalla, el port sèrie, el teclat, LEDs, bateries i els sockets per targetes. El desenvolupament dels drivers corresponents, té lloc durant el procés de desenvolupament de la plataforma i els drivers s'integren a la imatge final del sistema operatiu Windows CE, el qual es troba emmagatzemat en memòria ROM o flash.

Els dispositius instal·lables són perifèrics fabricats per tercers que poden connectar-se i desconnectar-se de la plataforma en qualsevol moment per l'usuari final. Això pot ser per exemple, un lector de codis de barres que es connecti pel port sèrie o una tarja SDIO amb una càmera que es connecti mitjançant el port SD. Són els mateixos fabricants d'aquests dispositius els qui proveeixen els drivers corresponents. Aquests drivers poden ser instal·lats en qualsevol moment i, per tant, s'instal·len en memòries tipus RAM no volàtils.

Apart d'aquesta classificació inicial, Windows CE suporta diferents tipus de drivers: nadius, stream interface, USB, NDIS, Printer Miniport,....

3.2.4. Arquitectura dels drivers

Bàsicament, trobem dos tipus d'arquitectura de drivers dintre de Windows CE, monolítics i per capes.

El driver per capes es basa en una part de codi que pot ser reutilitzada en diferents plataformes per tal de simplificar i escurçar els temps de desenvolupament. Aquest codi, s'anomena *module device driver* (MDD) i implementa la funcionalitat central del driver. El MDD no accedeix al hardware directament, ja que es basa en una altra peça de codi, que és depenent del hardware i que s'anomena *platform-dependent driver* (PDD). Aquestes dues

peces independents, un cop combinades, constitueixen el driver real. Quan es porta un driver per capes d'una plataforma a una altra, només la part PDD ha d'ésser reescrita, no el driver sencer. Existeix una interfície entre el MDD i el PDD (depenent del driver), que s'anomena *driver service provider interface* (DDSPI). Aquesta interfícies defineix les funcions a la PDD que són cridades per la MDD en temps d'execució.

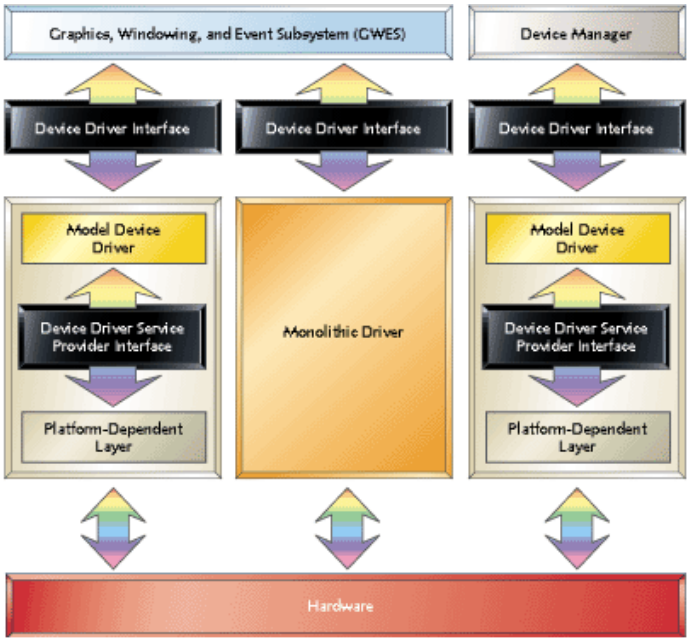


Figura 3.3. Arquitectures de drivers monolítica i per capes a CE.

Si el rendiment del sistema és un requisit primordial, l'aproximació monolítica amb una única peça de codi, pot ser l'apropiada degut a una millor optimització del codi i eliminació d'*overhead* en la gestió de les capes. De totes maneres, el temps de desenvolupament incrementa i la portabilitat queda reduïda

Per l'aplicació del projecte, ens centrarem únicament en els anomenats stream interface drivers per capes i que es detallen en els apartats següents.

3.2.5. Stream Interface Drivers

Tots els stream interface drivers comparteixen una mateixa interfície. Majoritàriament es fan servir per controlar dispositius instal·lables encara que

alguns pocs també ho fan per dispositius integrats com el driver del port sèrie o el del port SD ja que són apropiats per qualsevol dispositiu d'entrada/sortida que es pugui pensar com a font o magatzem de dades. És a dir, qualsevol perifèric que produeixi o consumeixi fluxos de dades (streams) com a funció principal, és un bon candidat a exposar un stream interface. Un bon exemple d'això, podria ser un port sèrie, una tarja SD de memòria o també SDIO. El contraexemple seria algun dispositiu de visualització que no produeix ni consumeix dades en el sentit tradicional.

Els stream drivers que controlen dispositius instal·lables, són típicament accedits per aplicacions. Per exemple, si es connecta una càmera a la plataforma, serà l'usuari el que iniciarà una aplicació que accedirà i farà servir aquest dispositiu.

Microsoft ha escollit reutilitzar un API ja existent (específicament, el API del sistema de fitxers). D'aquesta manera, els stream interface drivers han estat dissenyats per exposar les capacitats dels dispositius a les aplicacions presentant el dispositiu com si fos un fitxer especial, el qual pot ésser obert, llegit, escrit o tancat, per exemple.

Per tal d'ésser fàcilment identificats, els stream drivers segueixen una convenció de nom de fitxer única, la qual està composada d'un prefix de tres lletres (CAM per càmera o BCR per lector de codis de barres – bar code reader), un dígit que identifica un dispositiu en concret quan moltes instàncies d'un mateix driver estan disponibles i dos punts finals. Per exemple, alguns noms vàlids serien COM1:, COM2, BCR1: ... El prefix de tres lletres pot ser qualsevol combinació de lletres en majúscules, però ha d'ésser única en la plataforma. Quan una aplicació obre un fitxer que segueix aquesta convenció, el mòdul del sistema de fitxers reconeix que és accedit un driver i encamina totes les crides posteriors al sistema de fitxers cap el driver específic.

El stream driver rep comandes des del gestor de dispositius (Device Manager) i des de les aplicacions mitjançant crides al sistema de fitxers. El driver encapsula tota la informació que sigui necessària per traduïr les crides a accions apropiades sobre els dispositius que controla. Tots els stream drivers, ja maneguin dispositius incorporats o instal·lables, o ja siguin carregats durant

la inicialització o dinàmicament, tenen interaccions similars amb altres components dels sistema. A la Figura 3.4. es mostra les interaccions entre els components del sistema per un stream driver genèric que manega un dispositiu incorporat i que és carregat pel Device Manager en temps d'inicialització.

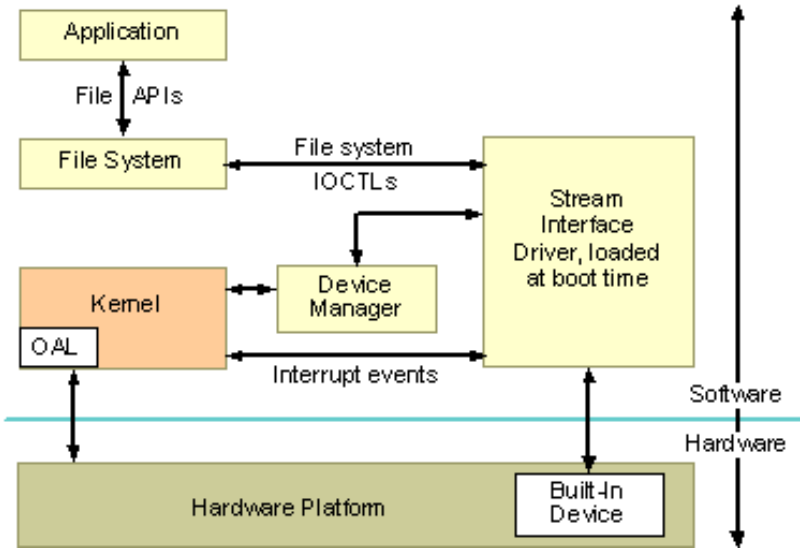


Figura 3.4. Arquitectura d'un stream driver.

En el nostre projecte de crear un bridge entre la PDA i la tarja SDIO, implementarem un stream driver, el qual exportarà una interfície que permetrà a una aplicació instal·lada a la PDA accedir a tota la funcionalitat de la tarja SDIO.

Per targetes SDIO, s'haurà de construir un driver SDIO compatible amb el driver SD bus de Microsoft. Les targetes de memòria SD són encara suportades de manera nativa. Els SD host controllers són suportats mentre segueixin l'estàndard PCI-based SDA que acompleix totalment amb la especificació 1.0 per SD Host. Controladors host SD propietaris o amb extensions no són suportats. Amb tot això, el que farem en aquesta part del projecte serà la d'utilitzar la interfície que exporta el bus SD per construir la última capa, que consistirà en un driver client SDIO que s'utilitzarà d'interfície per les aplicacions.

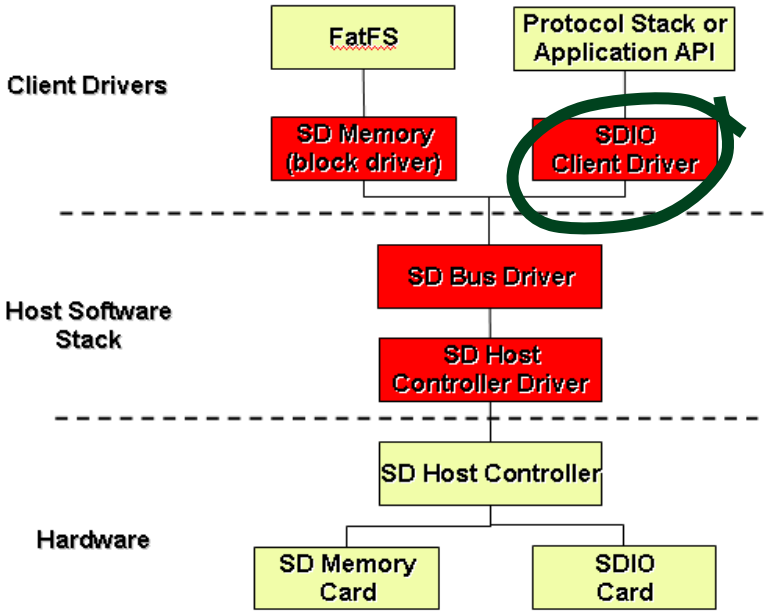


Figura 3.5. Model de capes dels drivers SD i SDIO

3.2.6. Funcions dels Stream Interface Drivers

Els stream drivers exposen un conjunt ben conegut de funcions. Ja que volem construir un stream driver SDIO, volem tenir la possibilitat de escriure una trama de bytes al dispositiu SDIO o llegir una trama de bytes des del dispositiu. Aleshores esperaríem tenir un conjunt de funcions disponibles al driver com: Open, Close, Read, and Write. Els stream drivers també exposen algunes funcions adicionals com: PowerUp, PowerDown, IO Control, Init, and DeInit.

A continuació s'exposa la llista complerta, per ordre alfabètic, de funcions que un stream driver hauria d'exposar, on *XXX_* denota el nom del driver amb que el sistema operatiu el gestionarà. En el nostre disseny utilitzarem el prefix SIO, per designar el driver SDIO.

Funció	Descripció
XXX_Close	Es crida quan una aplicació tanca un "handle" a una instància oberta del dispositiu.
XXX_Deinit	Funció cridada pel OS(Device Manager) per desinicialitzar un dispositiu.
XXX_Init	Funció cridada pel OS (Device Manager) per inicialitzar una instància del driver.
XXX_IOControl	Funció cridada per l'aplicació per demanar al driver realitzar una funció de control d'entrada/sortida.
XXX_Open	Obre un dispositiu per lectura, escriptura, o totes dues coses. Una aplicació evoca indirectament a aquesta funció quan crida **CreateFile** per obrir un nom de dispositiu.
XXX_PowerDown	Apaga elèctricament un dispositiu. És útil només per dispositius que puguin ésser apagats sota control per software.
XXX_PowerUp	Restableix l'alimentació elèctrica d'un dispositiu.
XXX_Read	Crida des de l'aplicació per sol·licitar una lectura des del dispositiu identificat per la instància oberta del mateix.
XXX_Seek	Mou l'apuntador de dades al dispositiu.
XXX_Write	Escriu dades al dispositiu. És cridada per l'aplicació.

Hi ha dues noves funcions afegides a partir de Windows CE 5.0 i que són XXX_PreClose i XXX_PreDeInit que s'utilitza prèviament i acompanyades de XXX_Close i XXX_DeInit respectivament. En els nostre desenvolupament basat en Windows CE 4.2, per tant, no les considerarem.

3.2.7. Implementació del driver

La implementació d'un stream driver es pot descompondre en quatre etapes:

1. Seleccionar un prefix pel driver. Aquest es el nom amb el que es referenciarà el dispositiu.
2. Implementar els punts d'entrada requerits.
3. Crear el fitxer *.DEF
4. Crear els valors del registre de Windows pel nostre dispositiu.

El driver desenvolupat serà exposat al sistema operatiu com SIO1: i per tant inclourà les funcions: SIO_Close, SIO_Deinit, SIO_Init, SIO_IOControl, SIO_Open, SIO_PowerDown, SIO_PowerUp, SIO_Read, SIO_Seek i SIO_Write. Aquestes funcions o punts d'entrada son les funcions estàndard sobre fitxers d'entrada-sortida que utilitza el kernel del sistema operatiu.

En l'arquitectura genèrica del driver, existeix una funció que s'utilitza com punt d'entrada principal en l'accés al driver.

```
// SDIODriver.cpp : Defineix el punt d'entrada per la DLL.
// #include "stdafx.h"
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD ul_rao_de_crida,
                       LPVOID lpReserved
                      )
{
         return TRUE;
}
```

La DLL exposa una funció, DllMain, la qual és el punt d'entrada a la nostra DLL. El punt d'entrada de la DLL es pot cridar en quatre situacions diferents: **Process Attach, Process Detach, Thread Attach i Thread Detach**. Nosaltres només considerarem els dos primers casos en el nostre desenvolupament. La raó per cridar el punt d'entrada a la DLL es passa a DllMain a través del paràmetre DWORD ul_rao_de_crida. A partir d'aquest paràmetre es determina la raó per la qual el stream driver ha estat cridat.

```
// SDIODriver.cpp : Defineix el punt d'entrada per la DLL.
// #include "stdafx.h"
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD ul_rao_de_crida,
                       LPVOID lpReserved
                      )
{
         switch ( ul_rao_de_crida )
          {
             Case  DLL_PROCESS_ATTACH:
                    OutputDebugString(L"SDIODriver DLL_PROCESS_ATTACH\n");
            break;
             case DLL_PROCESS_DETACH:
                    OutputDebugString(L"SDIODriver - DLL_PROCESS_DETACH\n");
            break; }
           return TRUE;
}
```

A continuació es mostren alguns detalls de la implementació de la funcionalitat del driver a través d'algunes de les funcions desenvolupades.

DWORD SIO_Init(DWORD dwContext);

La funció SIO_Init aquesta funció agafa com a primer paràmetre un apuntador a un string que conté el camí dintre del registre de Windows CE a la clau activa (active key) pel stream interface driver. En el moment que el sistema operatiu carrega el driver, s'obté el string del registre.

L'element "Key" del registre apunta a la posició dintre del registre on es pot trobar informació per aquest driver: HKLM\Drivers\BuiltIn\SDIO. Podem fer servir aquesta informació per carregar qualsevol informació específica del driver al registre. A l'apartat següent es detallen els camps del registre.

La funció SIO_Init ha de retornar un handle pel nostre dispositiu en el cas que el driver inicialitzi correctament. En el cas que el driver no s'inicialitzi correctament el valor retornat es un NULL. Aquesta funció es crida durant el procés d'enregistrament del driver i crearà una nova instància del servei per cada dispositiu. De fet la funció es crida quan el Device Manager de Windows CE carrega el driver per inicialitzar-lo Aquesta funció no és cridada per tant, per les aplicacions. Un cop la funció retorna, el Device Manager comprova el registre per tal de trobar una clau anomenada IOCTL per aquest driver. En el cas que aquesta funció estigui present, el Device Manager crida la funció SIO_IOControl, passant-li el valor especificat per IOCTL com el paràmetre dwCode. El driver pot utilitzar aquesta funció per acabar la seva inicialització després d'haver-se instal·lat.

BOOL SIO_Deinit(DWORD hDeviceContext).

La funció SIO_DeInit és cridada pel DeviceManager com a resultat d'una crida a la funció DeactivateDevice. El stream driver haurà de lliurar qualsevol recurs usat fins aleshores i acabar.

DWORD SIO_Open(DWORD hDeviceContext, DWORD AccessCode, DWORD ShareMode);

La funció SIO_Open, obre el dispositiu per llegir i/o escriure. A la funció se li passa el handle obtingut anteriorment. Les aplicacions invoquen indirectament

a aquesta funció quan fan una crida a CreateFile a l'API de Win32 per obrir un fitxer especial que correspon al dispositiu.

BOOL SIO_Close(DWORD hOpenContext);

La funció SIO_Close únicament fa servir el handle. En el nostre cas, retornem un valor TRUE, en el cas que el driver s'hagi tancat correctament. La crida a aquesta funció des d'una aplicació es fa mitjançant CloseHandle i és el sistema operatiu qui, al seu torn, invoca la funció SIO_Close. El handle utilitzat fins aquell moment es perd tant aviat la funció acaba i per tant, si de nou l'aplicació intenta realitzar alguna operació d'entrada/sortida usant aquest handle, l'operació fallarà.

La part més interessant del desenvolupament del driver es troba a les funcions Read, Write, Seek i en el nostre desenvolupament, especialment en IOControl.

DWORD SIO_Read(DWORD hOpenContext, LPVOID pBuffer, DWORD Count); SIO_Read s'invoca quan l'aplicació crida la funció ReadFile.

DWORD SIO_Write(DWORD hOpenContext, LPCVOID pBuffer, DWORD Count); SIO_Write s'invoca quan l'aplicació crida la funció WriteFile.

DWORD SIO_Seek(DWORD hOpenContext, long Amount, WORD Type);

SIO_Seek permet moure l'apuntador d'entrada-sortida actual.

BOOL SIO_IOControl(DWORD hOpenContext, DWORD dwCode, PBYTE pBufIn, DWORD dwLenIn, PBYTE pBufOut, DWORD dwLenOut, PDWORD pdwActualOut);

La funció SIO_IOControl serveix per gestionar peticions variades. Aquestes operacions personalitzades no necessàriament apliquen a fitxers. És a dir, mitjançant la funció IOCTL, es poden afegir funcionalitats que no estan previstes. Mitjançant un codi de control I/O s'identifica la operació a realitzar i aquest és específic al dispositiu. Des d'una aplicació s'invoca la funció DeviceIOControl per especificar la operació a realitzar. El sistema operatiu, al

seu torn, invoca aquesta funció SIO_IOControl. El paràmetre dwCode conté la funció d'entrada o sortida a realitzar; aquests codis son usualment específics a cada dispositiu i s'exposen al programador d'aplicacions mitjançant un fitxer de capçaleres que el programador del dispositiu entrega juntament amb el driver. Apart d'aquest paràmetre, també es poden definir buffers d'entrada i sortida de dades.

Si al registre, la clau HKLM\Drivers\BuiltIn\SDIO\Ioctl està definida pel nostre driver, el Device Manager invocarà la funció SIO_IOCOntrol en el moment que es carrega i inicialitza el driver.

En el nostre driver hem implementat una versió que gestiona les comandes d'inicialització del bus SDIO: CMD0,CMD3,CMD5, CMD52 i CMD53. Per aquestes comandes, s'ha respectat el nombre de comanda en el codi IOCTL.

En un escenari habitual, l'ordre de crides a que es realitzaria a aquestes funcions seria en el driver seria SIO_Init, SIO_Open, SIO_IOControl i SIO_Close. La crida SIO_Open és necessària pel DeviceManager per tenir un handle vàlid, i la crida SIO_Close és necessària per deixar el driver lliure per d'altres aplicacions.

Les funcions SIO_PowerUp i SIO_PowerDown s'utilitzen en el cas de dispositius que puguin ser engegats o aturats controlats per software.

Per tal de confirmar que les funcions desenvolupades per la nostra DLL son correctament exportades, utilitzarem l'aplicació Dumpbin que està inclosa a Platform Builder. A la Figura 3.6 es mostren els resultats després de córrer l'aplicació *Dumpbin* sobre el nostre driver desenvolupat:

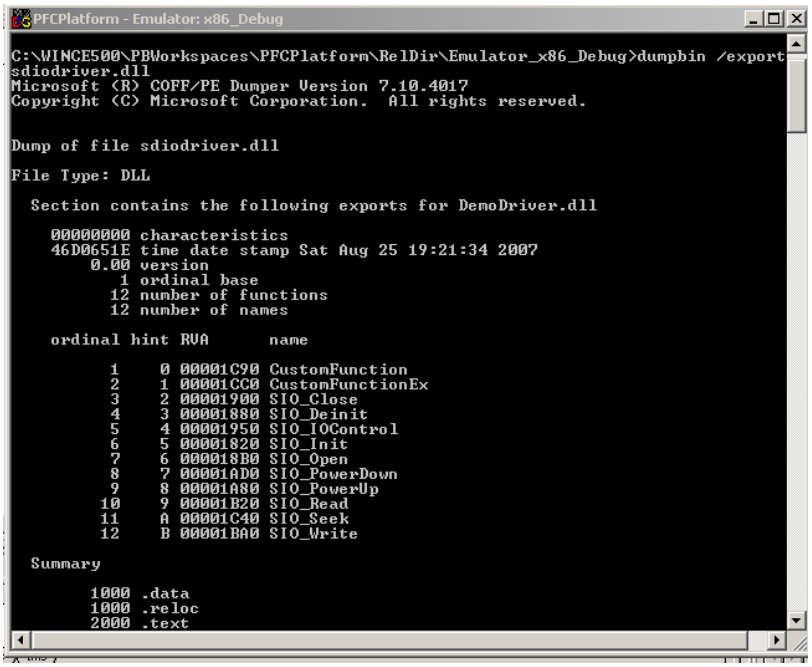


Figura 3.6. Comprovació de les funcions exportades pel driver.

Així doncs es pot veure com el driver SDIODriver.dll és justament una DLL que exporta algunes funcions. Apart d'utilitzar el sistema de fitxers, també carregar la DLL des de una aplicació cridant LoadLibrary, i obtenir l'adreça de qualsevol de les funcions cridant al seu torn GetProcAddress(). Aleshores podríem cridar les funcions individualment.

Finalment, es mostra a continuació els continguts del fitxer SDIODriver.def. Aquest fitxer de definicions es passa al linkador i conté els noms de les funcions exportades.

```
NAME SDIODriver.DLL

EXPORTS
SIO_Close
SIO_Deinit
SIO_Init
SIO_IOControl
SIO_Open
SIO_PowerDownn
SIO_PowerUp
SIO_Read
SIO_Seek
SIO_Write
CustomFunction
CustomFunctionEx
```

3.2.8. Càrrega i enregistrament del driver.

Per tal que el driver pugui ser carregat pel sistema operatiu i estigui llest per ser utilitzat, prèviament l'haurem d'haver instal·lat manualment i crear les entrades necessàries al registre de windows per tal que posteriorment pugui ser carregat pel procés Device Manager del sistema operatiu.

El Device Manager és el procés a nivell d'usuari que és executat contínuament pel kernel (nucli) del sistema operatiu i carregat durant el procés d'arrencada (boot process). S'implementa com a Device.exe. El Device Manager realitza les següents tasques:

- Detecta que un usuari a connectat un perifèric o tarja a la plataforma Windows CE, i intenta carregar un driver per aquest perifèric.
- Per exemple, quan inserim la tarja SDIO, el Device Manager intenta localitzar i carregar un driver específic per aquesta tarja.
- Registra noms de fitxers especials amb el kernel i que mapegen les funcions I/O del stream driver que des de les aplicacions s'utilitzaran a les implementacions d'aquestes funcions dintre del stream interface driver.
- Busca el driver apropiat pel dispositiu obtenint un identificador Plug'n'Play del perifèric o invocant una rutina de detecció per tal de trobar un driver que pugui gestionar el dispositiu.
- Carrega i controla els drivers llegint i escrivint valors al registre.
- Descarrega els drivers quan els dispositius ja no son utilitzats més. Per exemple, el Device Manager descarrega el driver SDIO quan l'usuari extrau la tarja.

En resum, és l'aplicació que interactua amb el kernel, el registre i les DLLs que implementen els stream interface drivers.

A la Figura 3.7 s'exposen les entrades corresponents al registre de Windows que s'han creat per tal d'instal·lar el SDIO driver en el nostre sistema Pocket PC.

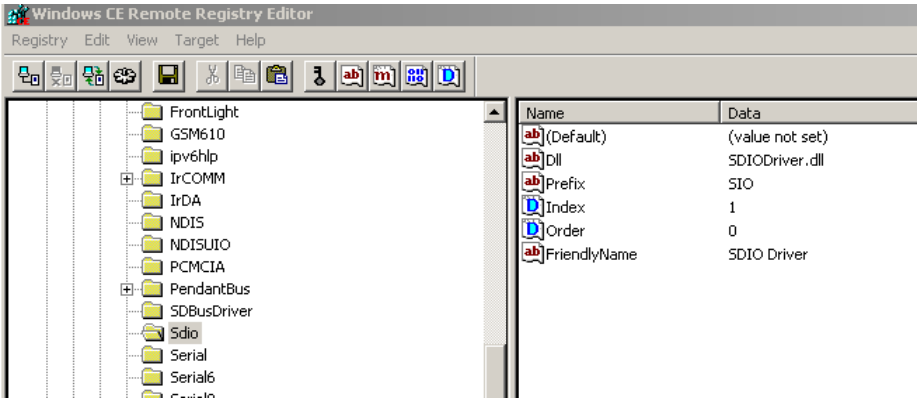


Figura 3.7. Entrades al registre de Windows.

El codi del nostre driver desenvolupat es troba a SDIODriver.dll tal i com es mostra a l'entrada *Dll* del registre. En aquest cas, el prefix utilitzat és *SIO*. Aquest *SIO* correspon a les mateixes tres lletres en cada punt d'entrada al driver, que en el cas d'haver-hi mes instàncies del driver, s'anirien numerant correlativament començant per "1", tal i com estableix la clau *Index* al registre. *Order* fa referència a l'ordre de càrrega i prova de drivers en el cas que n'hi hagi més d'un instal·lat per controlar el nostre dispositiu.

A la Figura 3.8 es mostra el driver SDIO creat actiu entre la llista de processos carregats i executant-se a l'entorn de Windows CE.

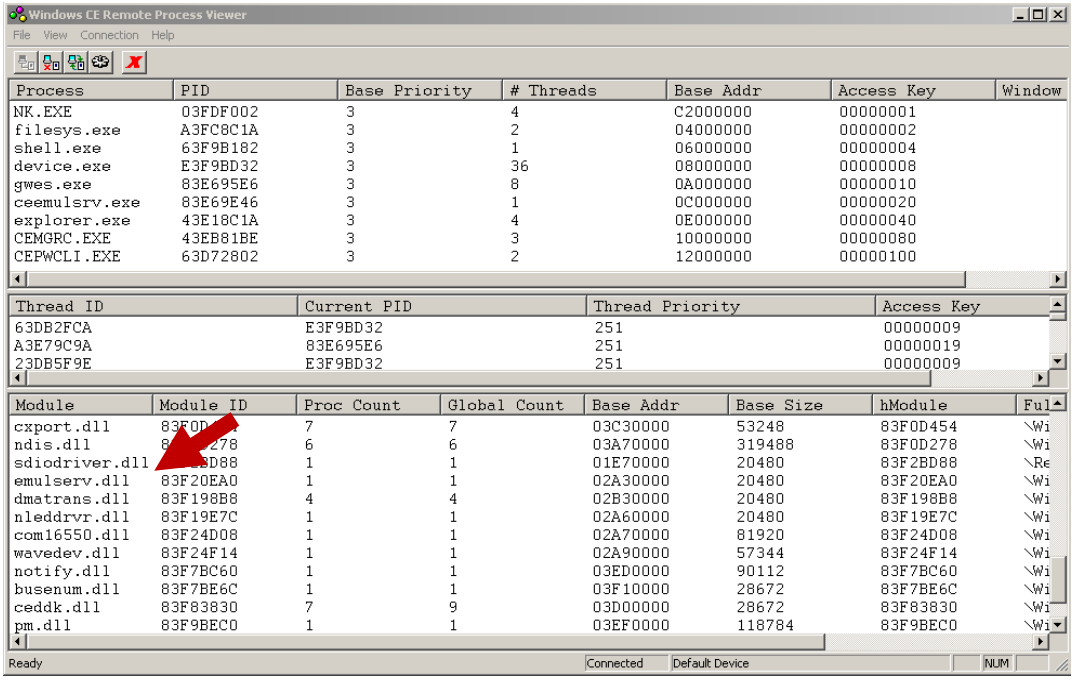


Figura 3.8. Llista de mòduls carregats.

3.3. Implementació de l'aplicació sobre PocketPC

Per tal de comprovar la funcionalitat del stream driver implementat a l'apartat anterior i com a model per desenvolupar una aplicació de control del sistema global amb la tarja SDIO, s'ha creat una aplicació sobre l'entorn PocketPC.

L'aplicació s'ha desenvolupat en l'entorn de Visual Studio 2005 i en concret, utilitzant el llenguatge de programació Visual Basic. La tria s'ha decidit en funció de la facilitat en desenvolupar i depurar tant el codi del programa com la interfície de control. A més, es disposa d'un emulador que permet provar el codi de manera immediata carregant-lo sobre una màquina virtual abans no es passi definitivament al dispositiu físic.

L'aplicació, anomenada SDIOTest.exe, s'instal·la i executa des dels directoris de programes del PocketPC la qual s'ha transferit al mateix fent servir la utilitat ActiveSync 4.5 de Microsoft.

El programa consta bàsicament de tres botons de control des dels quals es poden fer les crides d'inicialització pel dispositiu SDIO, enviar dades i rebre dades. L'aplicació conté rutines bàsiques de test i que en un futur es poden

complementar amb la funcionalitat desitjada pel control de la tarja. A la Figura 3.9 es mostra una imatge del programa executant-se en l'entorn virtual de Visual Studio 2005.

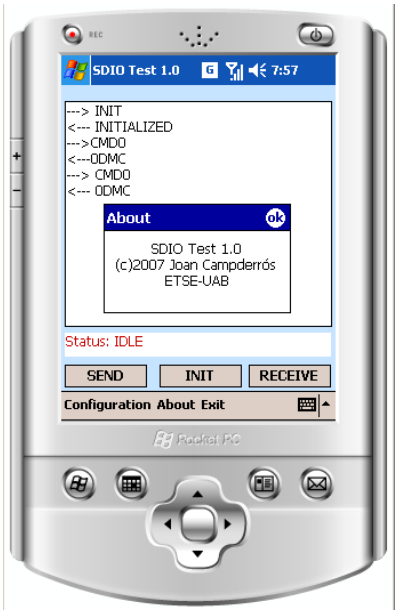


Figura 3.9. Aplicació sobre PocketPC en entorn virtual

Com a test general del driver i l'aplicació funcionant conjuntament, es va dissenyar l'aplicació de tal manera que s'enviava un string mitjançant la funció write i utilitzant el botó Send. Posteriorment obteníem el mateix string quan premiem el botó Receive, el qual activava la funció Read del driver retornant el string emmagatzemat. Per tal d'avaluar la funcionalitat dels codis IOCTL, vam definir un codi IOCTL que cridat per l'aplicació, implementava una funció que retornava el string enviat a l'inrevés en el driver. Els resultats d'aquest test es poden veure a la Figura 3.9 anterior.

3.4. Disseny del subsistema hardware

Com ja s'ha mostrat a l'apartat 3.1, el subsistema hardware es composa de dos mòduls diferenciats. D'una banda el bridge de comunicació SDIO entre el PocketPC i la tarja SDIO i per altra banda, el bridge de comunicació entre aquest anterior i el microcontrolador.

La implementació d'aquest dos "bridge" es realitzarà a la mateixa FPGA de que disposa la tarja SDIO. En concret, es tracta del model Altera Cyclone EP1C6T144. Aquest model de FPGA de tecnologia de 0,13µm i s'alimenta amb 1.5V. Disposa de 5.980 elements lògics i un total de 92.160 bits de RAM. La versió que disposem a la tarja SDIO és amb l'encapsulat de 144 pins dels quals 98 son per funcions d'entrada/sortida d'usuari. La implementació es realitzarà escrivint el codi VHDL necessari pels blocs que a continuació descriurem per tal de garantir la funcionalitat demanada.

Pel que fa al bridge SDIO, la seva funció principal serà la de rebre i enviar dades i comandes mitjançant el bus SD, adequant tant la velocitat de transferència (100KHz fins a 25MHz) com la tensió negociats amb el host (1.5V en el nostre disseny). Així mateix haurà d'interpretar adequadament les trames de comandes i dades rebudes i respondre adequadament. En aquest cas, la implementació SDIO realitzada a la tarja correspon a la d'un esclau del sistema que respondrà a les peticions del host (PDA).

La implementació realitzada, correspon al cas SD 1-bit, on les dades i les comandes-respostes van per línies separades. Veure capítol 1 per més informació sobre el funcionament SD 1-bit.

Així mateix, la tarja SDIO haurà de mantenir l'espai de registres interns que contenen informació sobre la tarja i d'altres registres amb informació obligatòria i d'altres opcionals que es troben en posicions fixes. Aquestes posicions fixes permeten que qualsevol host pugui obtenir la informació sobre la tarja i realitzar operacions amb els registres d'una manera senzilla.

Pel que fa al bridge SPI amb el microcontrolador, l'esquema de comunicació, malgrat ésser més senzill, haurà de disposar d'un registre de desplaçament on poder rebre/enviar dades.

Totes aquestes operacions hauran d'ésser coordinades mitjançant una màquina d'estats i així controlar i seqüenciar els senyals oportuns per respondre a l'estàndard SDIO d'una banda i al SPI de l'altra.

Per la comunicació entre els dos bridges, s'ha disposat d'una FIFO per les dades que s'envien per anar en un sentit o l'altre i així adequar les velocitats de transmissió a cada banda.

Gràficament, podem veure l'arquitectura general del subsistema hardware a la figura següent:

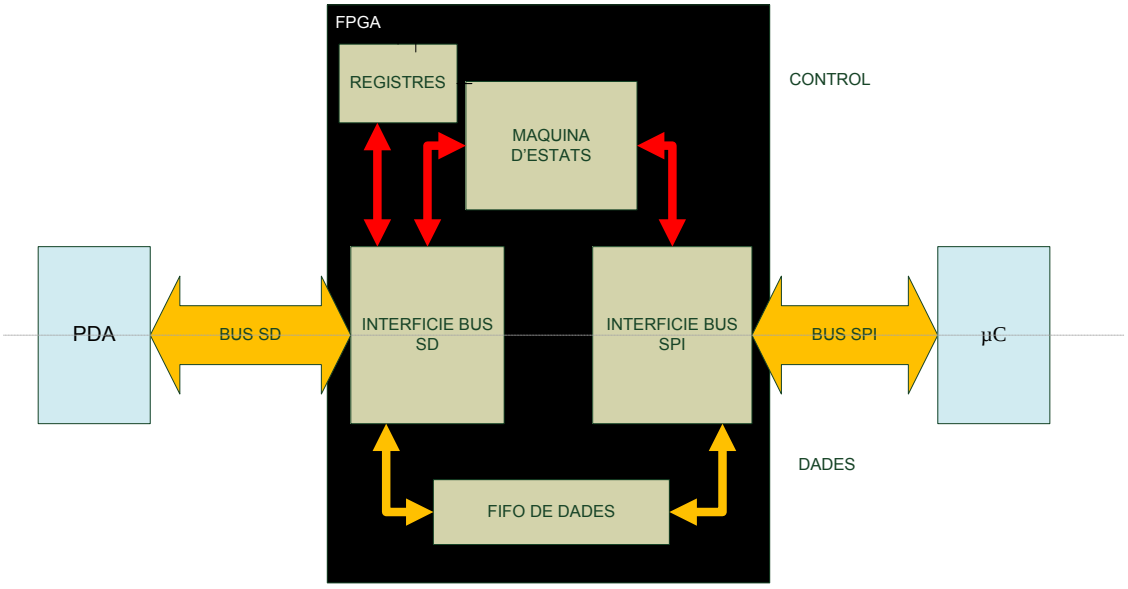


Figura 3.10. Arquitectura general del subsistema.

Conceptualment, tal i com es mostra a la Figura 3.10, el nostre model separa la part de control (fletxes en color vermell), de la part de dades o data-path (fletxes de color taronja), de tal manera que depenent dels volums de trànsit entre el host i el microcontrolador, l'estructura per gestionar les dades es pot dimensionar de manera independent. En el cas exposat, es mostra de forma genèrica, una FIFO de dades com a emmagatzematge temporal de dades.

A cada extrem del datapath, disposem d'una interfície amb el bus SD per una banda i una interfície amb el bus SPI de l'altre. Pel cas de la interfície amb el bus SD, aquesta ha de rebre, per les línies assignades, les comandes i les dades generades pel host de manera seqüencial. Aquestes trames d'informació rebudes en forma sèrie, s'han de verificar d'acord al seu format i la validesa del

seu contingut mitjançant el càlcul i comprovació del seu CRC. Així mateix, s'ha d'adreçar adequadament la informació rebuda pel path de dades o control, depenent de si el que s'ha rebut son dades o comandes. A l'hora de transmetre informació, ja sigui dades o respostes a comandes, s'ha d'empaquetar la informació adequadament d'acord als formats previstos en l'estàndard SD-SDIO. L'enviament de les trames pel bus SD es fa bit a bit de manera sèrie. Per la banda SPI, el model és més senzill en el sentit que disposem d'un única línia de sortida (enviament) i una única d'entrada (rebuda de dades).

En el model proposat, una màquina d'estats s'encarrega de la gestió dels mòduls del subsistema mitjançant senyals de control. Aquesta gestió consisteix en assegurar els fluxos de dades, generar a partir de les comandes rebudes i descodificades, les respostes adients i accedir al grup de registres SDIO on es guarda tota la informació relativa a la tarja i la comunicació amb el host.

A la Figura 3.11 es mostra el model real implementat amb més detall. Bàsicament, es pot observar com la funció de la màquina d'estats comentada anteriorment, es troba de fet dividida en vàries màquines d'estats que gestionen cada mòdul independentment. Així mateix, per simplicitat en aquesta primera implementació, hem substituït la FIFO de dades per registres individuals situats en cadascuna de les interfícies SD i SPI.

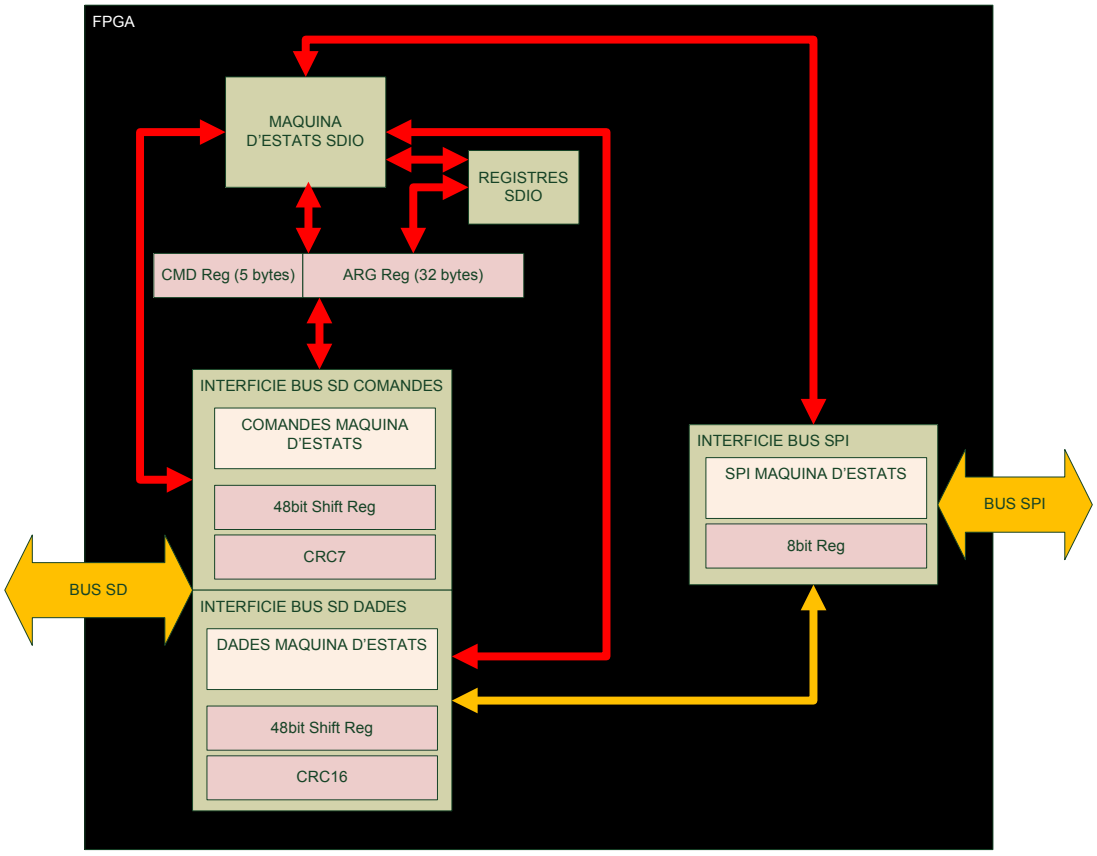


Figura 3.11. Model detallat del subsistema HW implementat

A continuació es detallen les tasques principals efectuades en cadascun dels mòduls.

- **Interfície BUS SD Comandes**. Aquest mòdul s'encarrega de rebre les trames de comandes des del host i enviar les trames de resposta. Per les trames rebudes s'espera a rebre un bit de start i es procedeix a la lectura dels 48 bits de la trama de forma sèrie i es guarda en un registre de desplaçament de 48 bits. Paral·lelament s'efectua el càlcul del CRC7 de la trama rebuda i que es compara a l'últim instant amb el CRC7 que la pròpia trama incorpora en un camp reservat. L'objectiu, segons marca l'estàndard SD, és el de protegir les trames transmeses d'errors de transmissió. La trama es rebutjada en el cas que els CRC7 no siguin

coincidents. Si la trama és correcta, es separen el codi de comanda i l'argument en dos registres CMD i ARG, de 5 i 32 bytes respectivament.

En el cas d'enviament de respostes, es codifiquen els continguts de CMD i ARG en un format de trama, afegint-hi el CRC calculat. La trama final es composa al registre de desplaçament de 48 bits i s'envien les dades de manera sèrie per la línia de comandes del port SD. Una màquina d'estats controla el mòdul de manera adequada per efectuar les operacions descrites. A la Figura 3.12 es mostra gràficament la màquina d'estats implementada per aquest mòdul.

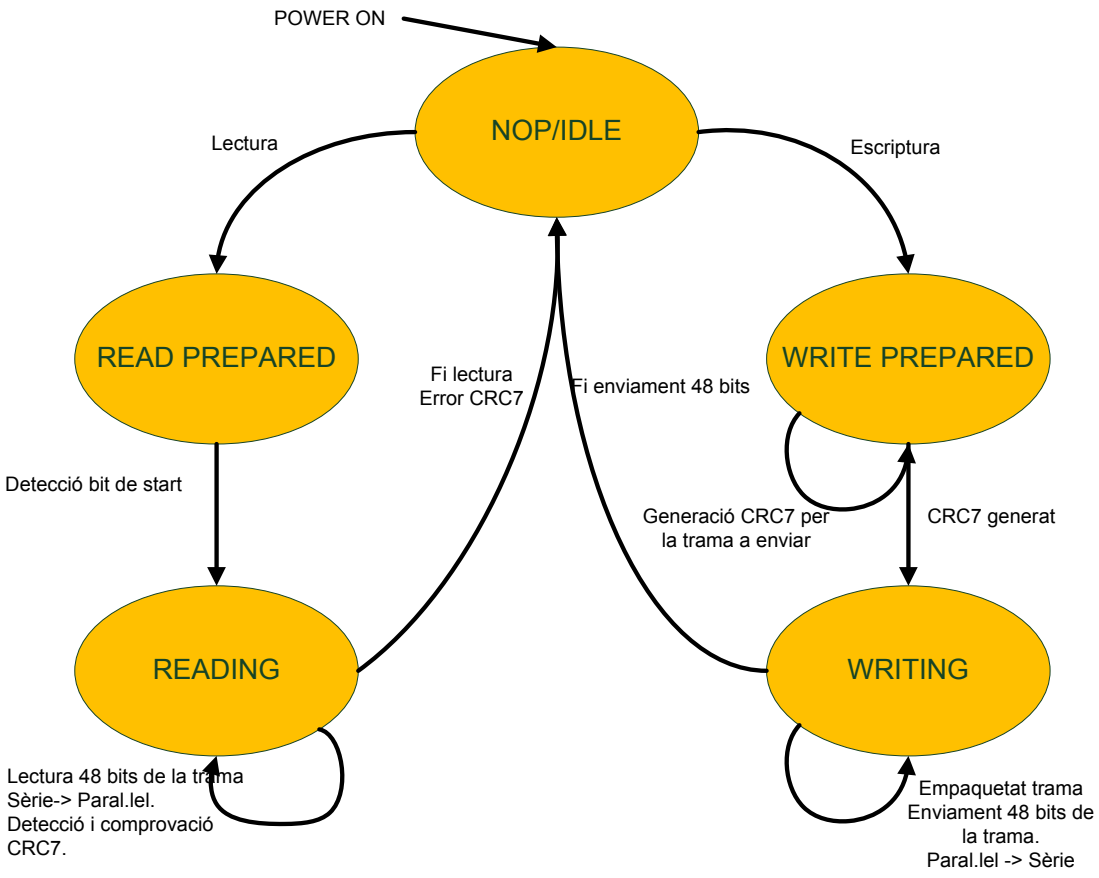


Figura 3.12. Màquina d'estats del mòdul Interface Bus SD comandes

L'estat inicial NOP/IDLE és al que s'arriba en el moment d'inicialitzar el sistema o efectuar un reset. Depenent de l'ordre de rebuda, per part de la màquina d'estats SDIO, es pot produir una situació de lectura o

escriptura. En el primer cas, es llegeix de bus fins a detectar el start bit (READ PREPARED) i a continuació s'efectua la lectura de la trama sencera. Es retorna a l'estat NOP/IDLE quan acaba la lectura o el CRC calcula és erroni. Pel cas d'escriptura, es genera el CRC7 complert (WRITE PREPARED) i s'envia amb la trama empaquetada (WRITING) fins que la transmissió acaba (NOP/IDLE).

- **Interfície BUS SD Dades.** Aquest mòdul és similar a l'anterior en els processos de rebuda i enviament de dades, però en aquest cas realitzats a la línia de dades del bus SD. Per les trames de dades s'utilitza un CRC16 a diferència del mòdul anterior. De nou, una màquina d'estats gestiona la rebuda, emissió i càlcul del CRC sobre les dades transmeses.

- **Màquina d'estats SDIO.** Compren tant la màquina d'estats general del subsistema com els registres associats en tota implementació SDIO. Aquest mòdul determina les accions a realitzar basant-se en el contingut de les comandes que li arriben pel registre ARG i CMD i generar les respostes (de nou registre ARG i CMD) i senyals de control a la resta de mòduls del subsistema. A la Figura 3.13 es mostra esquemàticament el funcionament d'aquesta màquina d'estats. En la implementació real que s'ha fet, molts d'aquest estats han estat segmentats per facilitar la codificació en VHDL.

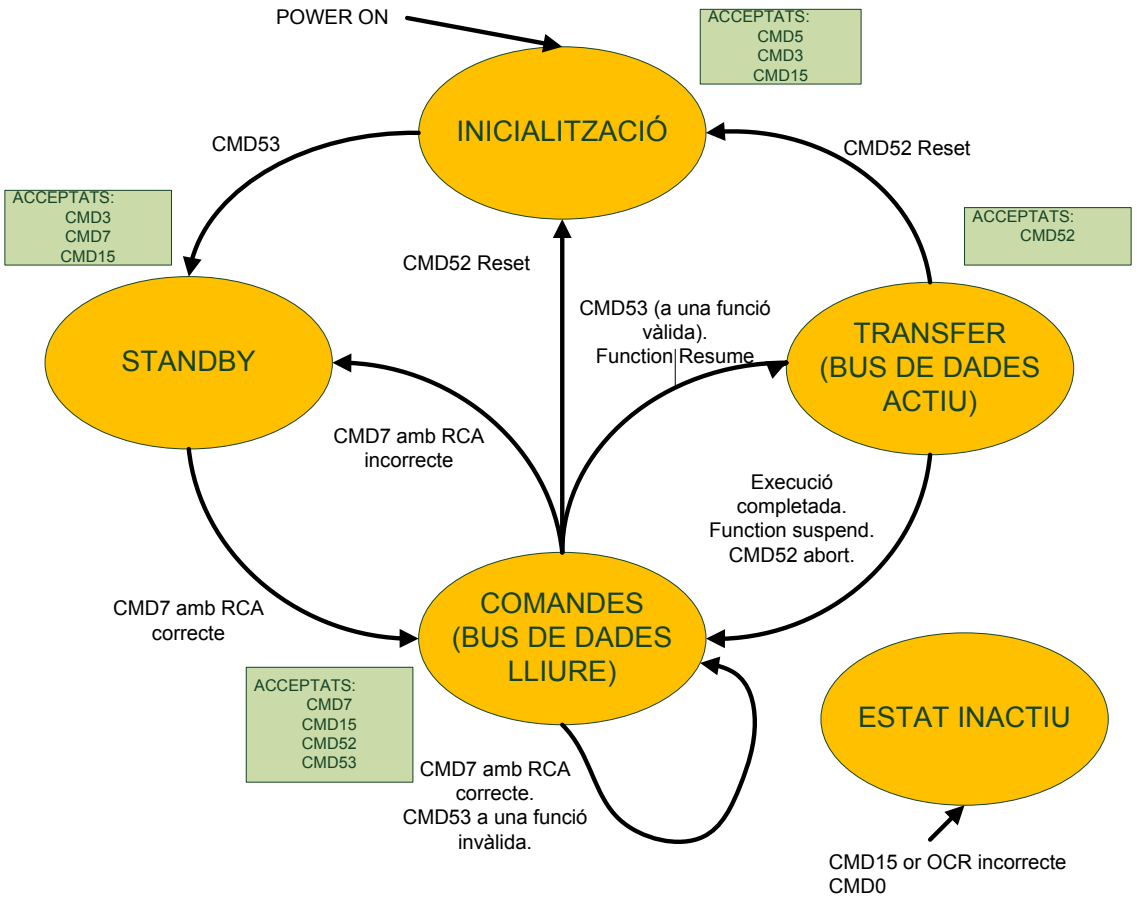


Figura 3.13. Màquina d'estats SDIO.

- **Interfície BUS SPI**. En aquest cas la implementació a la FPGA consisteix en el master del protocol SPI, el qual es la part activa del sistema i que ha de proveir el senyal de rellotge per la transmissió. En la nostra implementació disposem d'un registre de desplaçament de 8 bits i d'un registre temporal (buffer) on es guarden les properes dades a ser enviades pel bus però que encara no es poden escriure al registre de desplaçament abans que el cicle de desplaçament en curs s'hagi acabat complertament. La maquina d'estats associada a la interfície SPI s'encarregarà de gestionar aquest funcionament (double buffered) depenent al seu torn de les ordres de lectura o escriptura per part de la màquina d'estat SDIO.

Capítol 4.
Test i Resultats

4.1. Metodologia de test.

El test del sistema dissenyat s'ha basat en la verificació i validació, per etapes, de cadascun dels subsistemes que composen el projecte, això és des del stream interface driver fins la interfície SPI que comunica amb el microcontrolador.

Del testeig del stream interface driver ja se'n ha parlat a l'apartat 3.2.7, on s'han utilitzat els emuladors proporcionats per l'entorn de desenvolupament (Visual Studio o Platform Builder) en primera instància i posteriorment, la instal·lació real i comprovació dels processos d'enregistrament del driver al sistema basat en Windows CE.

La resta del capítol es centra en el testeig de la part hardware del sistema, que en aquest cas es la implementació a la FPGA de la tarja SDIO. Per dur-lo a terme, ens hem servit d'eines de simulació que, en aquest cas ha estat el programa Modelsim. L'objectiu ha estat el verificar per cada subcomponent que els models VHDL dissenyats compleixen les especificacions teòriques. Aquestes especificacions en el projecte desenvolupat s'han centrat en la funcionalitat i prestacions a nivell de timing i deixant com a oportunitat futura de millora aspectes relacionats amb el consum o l'àrea ocupada a la FPGA.

La verificació del sistema s'ha anat realitzant durant tot el cicle de disseny i ha consistit en el disseny de bancs de prova en forma de generació d'estímuls per fer una simulació funcional o lectura de valors de fitxers. A l'hora d'analitzar les respostes, no només s'ha verificat la correcta seqüència dels senyals de resposta i/o senyals entremitjos sinó que també el comportament repetitiu de les mateixes, i així assegurar-nos que les màquines d'estat implementades parteixen d'un estat inicial a cada iteració.

A mesura que cada component del sistema s'ha anat verificant, al model simulat s'han anat afegint d'altres components a més gran nivell i fins a simular el sistema complert.

A l'últim apartat d'aquest capítol es mostra els resultats de la síntesi del hardware implementat a la FPGA.

4.2. Resultats de simulació.

En aquest apartat es s'il·lustra els resultats simulats per alguns dels subsistemes dels que consta el projecte.

El primer mòdul a verificar correspon al d'interfície amb el bus SD per comandes. En aquest cas, simulem l'entrada d'una comanda pel bus corresponent a la comanda CMD17 amb argument igual a zero. El valor del CRC7 és igual a "0101010". Així la entrada simulada al bus SD seria la trama següent:

CMD17 → **01 010001 00000000000000000000000000000000 "0101010" 1**

Es comprova a la Figura 3.14 com es parteix de la situació que el mòdul està en mode de lectura (*read* (1)), dictat al seu torn per la màquina d'estats SDIO. Tant aviat es rep el bit de "start" (2), el comptador *cont* inicia el recompte dels bits que van entrant des del bus SD i es va calculant el CRC7 dels mateixos al registre *crc_temp* (3). En el moment que la trama es rebuda completament, després de 48 bits, es compara el CRC de la trama llegida amb el valor de *crc_temp*. En cas que aquests coincideixin (4), es procedeix a l'emmagatzematge de la trama i la seva segmentació comanda (*CMD*) i la part d'argument (*ARG*) (5). El senyal *loadreg*, indica a la màquina SDIO que s'ha completat la lectura i que els resultat ja s'han guardat als registres *CMD* i *ARG* per la seva descodificació (5).

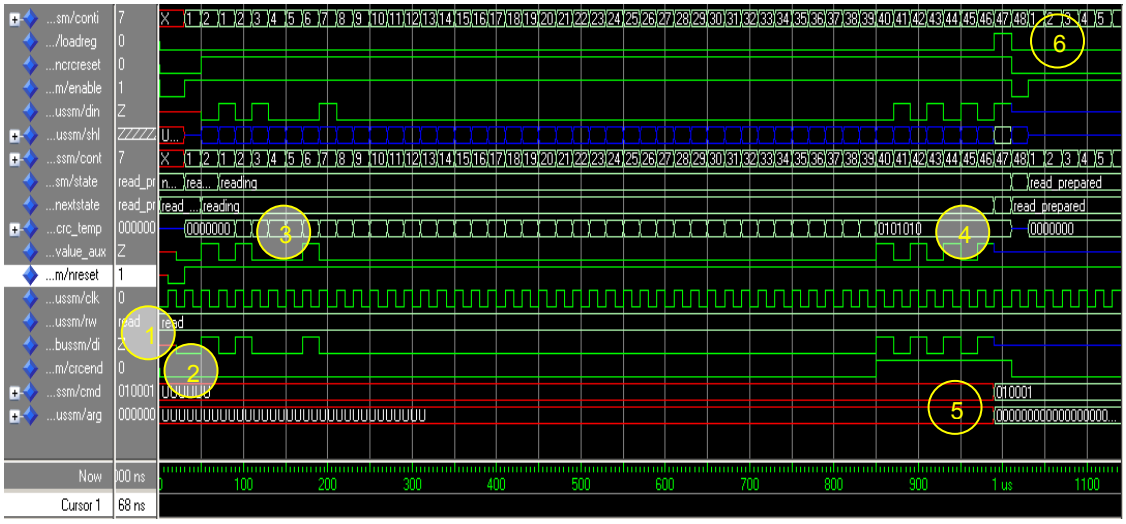


Figura 3.14. Interface de comandes. Lectura del bus SD amb CRC correcte.

A continuació es repeteix l'exercici però en aquest cas la trama a llegir conté un CRC erroni. Farem servir la comanda anterior però en aquest cas el CRC s'ha canviat per "0101011". El resultats de la simulació es poden veure a la Figura 3.15.

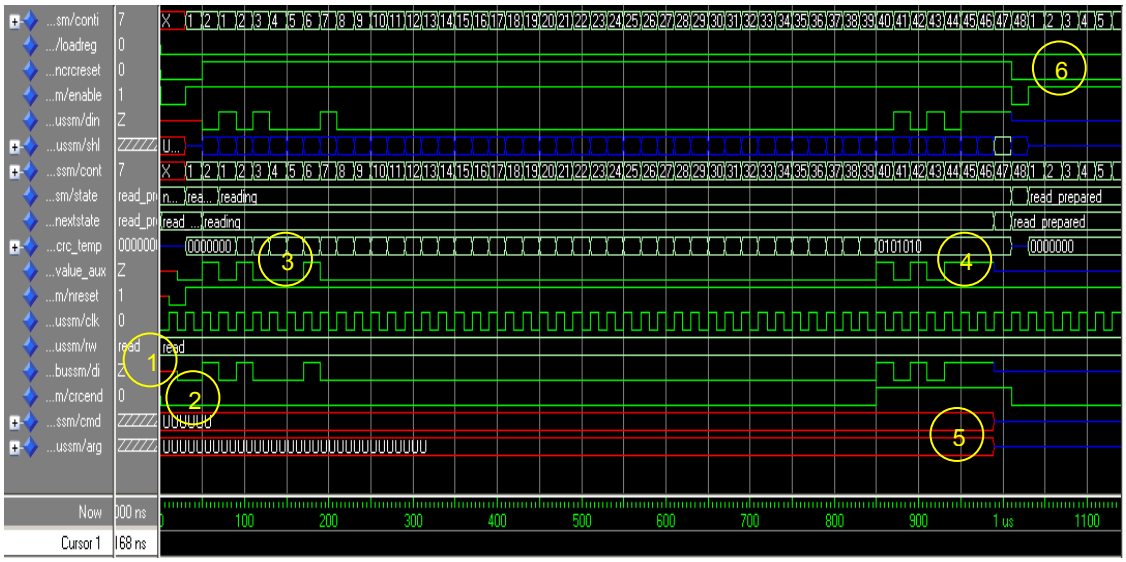


Figura 3.15. Interface de comandes. Lectura del bus SD amb CRC incorrecte.

Es pot observar a la Figura 3.15 com en cap cas s'activa el senyal *loadreg* (6) per indicar a la màquina SDIO que ja es disposa d'una comanda llesta a ésser processada als registres *CMD* i *REG*, els quals romandran en un estat d'alta impedància (6).

A continuació es realitzarà el procés contrari en el mateix mòdul, o procés d'escriptura al bus SD. A la Figura 3.16 es mostra els resultats obtinguts.

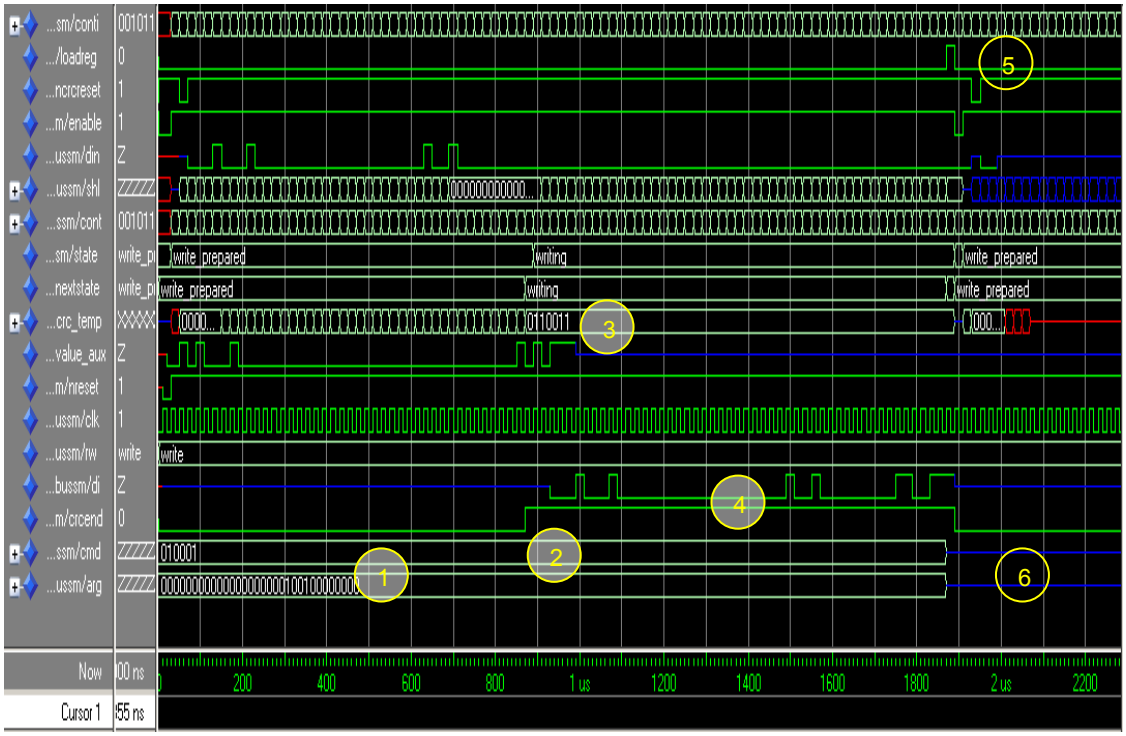


Figura 3.16. Interface de comandes. Escriptura al bus SD.

Com es pot veure a la Figura 3.16, es parteix de la situació en la que als registres CMD i ARG disposem de la comanda i arguments a enviar (1). Aquesta informació és dipositada per la màquina d'estats SDIO que a més activa el senyal *rw* a *write* per tal d'iniciar la operació d'escriptura sobre el bus SD. És en aquest moment en el que es mouen el continguts d'aquests dos registres al registre de desplaçament de sortida *shl*, gestionat per la màquina d'estat de la interfície de comandes. En aquest punt s'afegeix una capçalera a la trama, els bits de comanda i els bits d'argument i mentrestant es va calculant el CRC corresponent a aquesta trama generada. El senyal *crcend* (2) s'activarà per indicar que el càlcul del valor de CRC s'ha completat (3) i aquest s'afegeix a la trama en construcció al registre *shl.* En aquest punt, la trama està llesta per ser enviada al bus SD (4) i això acaba amb l'activació del senyal *loadreg* (5) per indicar a la màquina de control SDIO que la transmissió s'ha

acabat. En aquest moment s'esborren els continguts inicials dels registres *CMD* i *ARG* (6).

La resta de mòduls s'han simulat de manera similar a la dels mòduls anteriors mitjançant l'entrada de vectors de test coneguts, que seran contrastats amb els resultats esperats tot seguint la traçabilitat de tots els senyals que hi intervenen.

4.3. Validació a nivell de sistema.

Per validar tot el subsistema hardware s'ha fet una simulació amb tots els mòduls que el composen i d'aquesta manera ens assegurem de la correcta sincronització entre ells i la validesa dels resultats obtinguts. A la Figura 3.17 es mostra el resultat de la simulació que s'ha dut a terme.

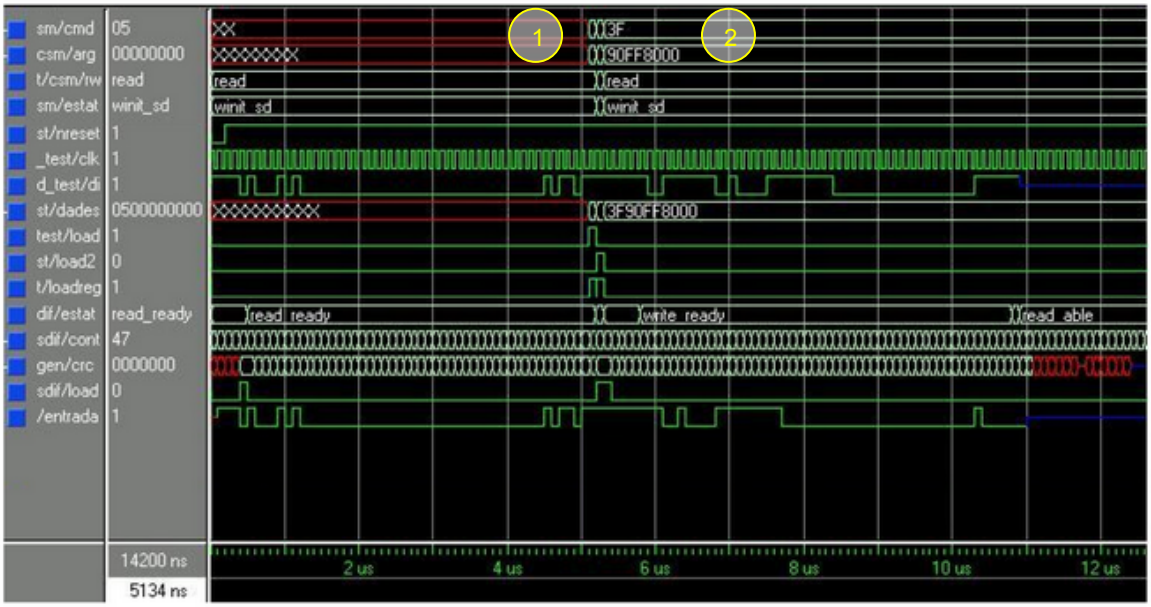


Figura 3.17. Simulació a nivell subsistema.

En aquest cas, s'envia una trama corresponent a una comanda CMD5 amb argument igual a zero i que, seguint els procediments explicats en l'apartat anterior es guarda al registre temporal composat per *ARG* i *CMD* (1). Tant aviat la comanda es llegida per la màquina d'estats SDIO i descodificada, aquesta envia com a resposta la comanda CMD63 amb el contingut del registre OCR (2).

4.4. Resultats de síntesi

Per fer la síntesi s'ha fet servir l'eina Quartus II v.7.0 proporcionada pel fabricant de la FPGA. En aquest cas, es tracta d'una FPGA d'Altera, model Cyclone EP1C6T144. A la Figura 3.18 es mostra els resultats obtinguts.

Subsistema	Logic Cells	Registres	LUT Only LCs	Register Only LCs	LUT/Register
Sistema Total	**912**	**281**	**631**	**66**	**215**
Interfase BUS SD Comandes	217	79	138	10	69
Interfase BUS SD Dades	209	68	141	12	56
Interfase BUS SPI	115	52	63	4	48
Màquina d'estats SDIO	315	56	259	19	37
Registre Temporal	42	42	0	21	21

Figura 3.18. Resultats de síntesi.

La màxima freqüència de rellotge obtinguda és de 90.05 Mhz, la qual queda molt per sobre de les restriccions imposades per targetes SDIO amb mode 1-bit SD.

Capítol 5.
Conclusions

5.1. Conclusions.

Com a conclusió principal, s'ha assolit l'objectiu proposat de crear una solució que permeti la comunicació end-to-end entre una PDA i una tarja SDIO que munta un mòdul Zigbee. S'han creat tots els ponts de comunicació necessàris en el sistema seguint les especificacions del protocol SDIO i SPI. Les prestacions del sistema final permeten gestionar comunicacions bidireccionals fent servir el mode 1-bit SD per SDIO i preparat per suportar una tassa de transferència màxima de 25Mbits/s, la qual cosa "dóna marge" davant les tasses de transferència del protocol Zigbee que com a màxim arriba a 250Kbits/s.

A continuació s'enumeren les principals contribucions i reflexions que s'han extret després de la realització d'aquest projecte:

- L'estàndard SD i SDIO corresponen a especificacions molt tancades i de les que es disposa molt poca informació de manera gratuïta. La informació existent és parcial i sovint pot arribar a ser confusa. Malgrat això, el potencial d'aplicacions i desenvolupament de nous dispositius per sistemes mòbils es enorme.
- Els stream driver interface ens permeten d'una manera senzilla i versàtil incorporar nous dispositius en els nostres sistemes mòbils o reemplaçar els drivers existents per uns altres de comportament modificat.
- Els stream driver interface gràcies a la seva interfície comú de funcions exportades, permeten que les aplicacions desenvolupades puguin abstraure al màxim nivell el hardware específic que s'estigui fent servir.
- La utilització de FPGAs permet accelerar el desenvolupament i prototipatge alhora de donar un grau molt alt de flexibilitat en els

dissenys. Permet deslligar en gran mesura el disseny de la placa amb la implementació del codi que s'hi faci, al tenir uns dispositius de propòsit general reconfigurables.

- Gràcies a eines de simulació i emulació en totes les fases del projecte, ha facilitat la depuració dels desenvolupaments abans d'esser traslladats definitivament als dispositius on han de córrer. Això permet centrar-se ne el disseny de les solucions i abstraure's fins l'últim moment de les particularitats del hardware on s'instal·la.

Les restriccions del projecte dut a terme han estat mínimes en el sentit de no haver d'assolir uns requeriments gaire estrictes quant a tassa de transmissió, volum de dades enviades o mida del hardware (en termes de lògic cells a la FPGA). Per tant, els costos associats a la tarja prototipus poden mantenir-se reduïts.

5.2. Experiència personal i professional.

Des d'una òptica professional, el projecte dut a terme és un bon exemple, quant a plantejament, del model que es pot trobar en empreses del sector de la consultoria. En aquest cas, el client ens proveeix d'un disseny propi, ja creat, pel qual se'ns demana efectuar un projecte d'integració de sistema. Basat en els seus requeriments, se li entrega una solució "claus en mà" i que consisteix en el desenvolupament del software i hardware fins obtenir el sistema funcional.

El projecte també ha suposat la superació d'una sèrie de reptes tècnics causats principalment pel la manca de "tota" la informació referent al bus SD, ja que la part pública del estàndard és força limitada, mentre que l'accés a l'estesa comporta el pagament d'uns drets en forma de quantitats importants de diners que queden fora de l'abast assumible per un projecte d'aquestes dimensions. Aquest fet, segurament limita o perjudica dintre la comunitat de petits i mitjans desenvolupadors, la existència de més dispositius al mercat que treguin tot el profit d'aquest bus.

En el nostre cas, molta de la informació s'ha anat obtenint a partir d'estudiar el comportament i la funcionalitat de dispositius comercials que ja fan ús del bus, com per exemple una càmera SDIO o una tarja de memòria SD. Per tant, molta de la informació ha estat deduïda i contrastada a partir d'aquests dispositius en un veritable procés d'enginyeria inversa que malgrat tot, ha resultat gratificant a cada passa en que s'avançava. En altres casos, s'ha optat per models de "força bruta" i seguir procediments de prova-error fins trobar la solució que funciona. Aquest procés aplica sobretot a la part de desenvolupament del stream driver com el disseny del controlador SDIO a la FPGA.

Una altra veritable font d'informació ha estat Internet, ja sigui en format de pàgina web o bé els grups de discussió on desenvolupadors de tot el món, sovint comparteixen problemes semblants i aporten solucions o nous punts de vista per abordar-los. En molts casos, aquest tipus d'informació ha estat molt útil per poder prendre decisions davant les opcions que es podien plantejar en el desenvolupament del projecte.

A nivell de coneixements, ha estat especialment interessant i particularment innovador, el desenvolupament en l'entorn del sistema operatiu Windows CE que fins aquest moment havia estat completament desconegut per a mi. L'arquitectura particular d'aquests sistemes ha forçat un canvi de mentalitat i costums adquirides quan prèviament la majoria de desenvolupaments els havia realitzat per versions de Windows per sistemes no mòbils. Així mateix, la experiència del projecte m'ha servit per conèixer noves tecnologies i eines de desenvolupament, ja sigui a nivell de software, com la tecnologia .NET, o entorns de desenvolupament com el Visual Studio o el Platform Builder.

A nivell de desenvolupament de la part de hardware, la experiència m'ha servit per treballar amb FPGAs les quals permeten un flexibilitat enorme en el disseny i a baix cost comparat amb dissenys hardware totalment customitzats. Així mateix, m'he introduït en les metodologies, fluxos i eines de disseny per la programació de FPGAs com l'entorn Quartus II de Altera per la part de programació i ModelSim per la part de simulació. La utilització de tantes i diferents eines ha incrementat de manera considerable el temps invertit en el

desenvolupament abans no s'aconsegueix un grau de coneixement i agilitat suficients per utilitzar-les convenientment. Tanmateix tenint en compte l'aplicació directa per la qual està dissenyada la tarja SDIO, m'ha permès conèixer de ben a prop les característiques principals del protocol Zigbee i pensar en les enormes possibilitats d'aplicació que se'n poden derivar del mateix.

A nivell personal, el fet de no disposar de tota la informació precisa pel desenvolupament de manera unificada sinó que a partir de múltiples informacions parcials m'ha forçat a fer una tasca de filtratge important per discernir la part útil de la part supèrflua aplicable al projecte .

Al llarg del projecte, també he hagut d'assumir determinats riscos i decidir-me davant de diferents opcions en el desenvolupament. De vegades, decantar-se per una solució o via, implica pagar un preu massa car en cas d'haver de tornar enrere, bàsicament en temps de desenvolupament perdut. Això per exemple ha estat així en la decisió del software de programació a utilitzar pel desenvolupament del stream driver sense conèixer inicialment les possibilitats de cada entorn i les eines disponibles de testing i debugging.

A nivell de gestió del projecte, tenint en compte les dificultats exposades en els anteriors paràgrafs, es podria resumir que el 30% del temps s'ha emprat en la recerca i tria de la informació, aprenentatge de noves tecnologies i també l'aprenentatge de nous entorns de desenvolupament. Un 30% ha estat el disseny de cadascun dels mòduls del projecte i avaluació i tria de les diferents opcions. Un 30% ha estat la codificació real tant del driver, l'aplicació i els mòduls en VHDL així com el seu testeig i depuració. L'elaboració de la memòria ha representat el 10% del temps restant.

5.3. Evolució futura.

El projecte presentat és només un punt de partida on s'han desenvolupat les eines a partir de les quals tenim el sistema funcional. S'ha implementat un model bàsic d'enllaços (bridges) entre els diferents blocs dels que consta el

sistema i que permeten la comunicació final amb el mòdul Zigbee controlat a partir d'una PDA.

Una evolució interessant es troba en l'àmbit del desenvolupament del stream driver, ja que la nostra implementació només ha tingut en compte un únic protocol de comunicació basat en l'estàndard suportat de 1-bit SD. En el cas d'haver de necessitar un flux de dades superior es podria ampliar la funcionalitat implementant l'estàndard 4-bit SD o fins i tot pensant en la existència d'una memòria externa (per exemple, implementada a la FPGA). En aquest cas es faria us complert de totes les línies de dades del bus SD. Un pla de test més extens, hauria d'assegurar-nos també que la implementació del driver és fidel i complerta segons l'estàndard SDIO i que qualsevol tarja d'aquestes característiques podria ser funcional amb el nostre disseny.

De totes maneres, a partir de la versió de Windows CE 5.0 ja es compta amb stack SDIO implementat, amb la qual cosa es pot esprémer fins el límit les capacitats del driver SDIO i focalitzar-se en la millora de prestacions globals del sistema.

Pel cantó de la tarja SDIO, el model de programació de la FPGA, s'ha realitzat mitjançant la definició de codi VHDL sintetitzat en la mateixa mitjançant el software Quartus II. Amb l'avantatge que la FPGA del nostre sistema és el model Cyclone d'Altera, una altra aproximació més versàtil que s'hauria pogut abordar per aquest disseny es la d'integrar un processador virtual (Nios II) a la FPGA i realitzar sobre ella la programació del nostre sistema. En aquest cas, la programació de la FPGA és més semblant a la programació d'un microprocessador on els llenguatges HDL queden substituïts per llenguatges estructurats tipus C.

Referències i bibliografía.

Publicacions.

Tacke, C. Basset, T. (2001). *eMbedded Visual Basic: Windows CE and Pocket PC Mobile Applications.* Sams Publishing.

Nicolaisen, N. (2002). *Making Win32 Applications Mobile. Porting To Windows CE.* Wiley Publishing, Inc.

Perry, Douglas L. (2002). *VHDL: Programming By Example. 4rth. Edition.* McGraw-Hill.

Avery, J. (2005). *Visual Studio Hacks.* O'Reilly.

Boling, D. (2001). *Programming Microsoft Windows CE. 2nd. Edition.* Microsoft Press.

Pong P. Chu. (2002). *RTL Hardware Design Using VHDL. Coding for Efficiency, Portability and Scalability.* John Wiley & Sons, Inc.

Álvarez Ruiz de Ojeda, L.J. (2004). *Diseño Digital con Lógica Programable.* Editorial Tórculo

Pàgines web.

http://www.sdcard.org

Pàgina web de la SD Card Association

Documents:

Simplified Version of SDIO Bluetooth Type-A, versió 1.00, april 3, 2006

Simplified SD Host Controller Spec versió 2.00 February 8, 2007

Simplified SD Physical Layer Spec versió 2.00 September 25, 2006

Simplified SD Physical Layer Spec versió 1.01 April, 15, 2001

Simplified Version SDIO Card Spec versió 1.00 October, 2001

Simplified Version SDIO Card Spec versió 2.00 February 8, 2007

http://www.msdn.com.

Pàgina web de Microsoft Developer Network

Informació específica sobre plataformes Windows.
Desenvolupament de Stream Interface Drivers.
Tutorials de Platform Builder.
Desenvolupament d'aplicacions sobre entorns Windows CE

http://www.embedded-code.com
Exemples de codi font per sistemes embedded.

http://www.arasan.com
http://www.atheros.com
SDIO IP Cores comercial.

http://www.zigbee.org
Pàgina web de la Zigbee Alliance.
Informació genèrica i tutorials sobre el protocol Zigbee.

http://www.freescale.com.
Informació genèrica i tutorials sobre el protocol Zigbee.
Manuals i datasheets del microcontrolador del projecte.

http://www.pocketpcdn.com
Pocket PC Developer Network

http://pocketpccentral.net
Pocket PC Central

http://www.opencores.org/
Recursos i projectes compartits en VHDL

http://www.fga4fun.com
Projectes realitzats amb FPGAs.

http://www.altera.com
Pàgina web del fabricant de FPGAs Cyclone. Manuals i datasheets de les FPGAs.

Annexos.

Apèndix 1. Extensions mecàniques de SDIO.

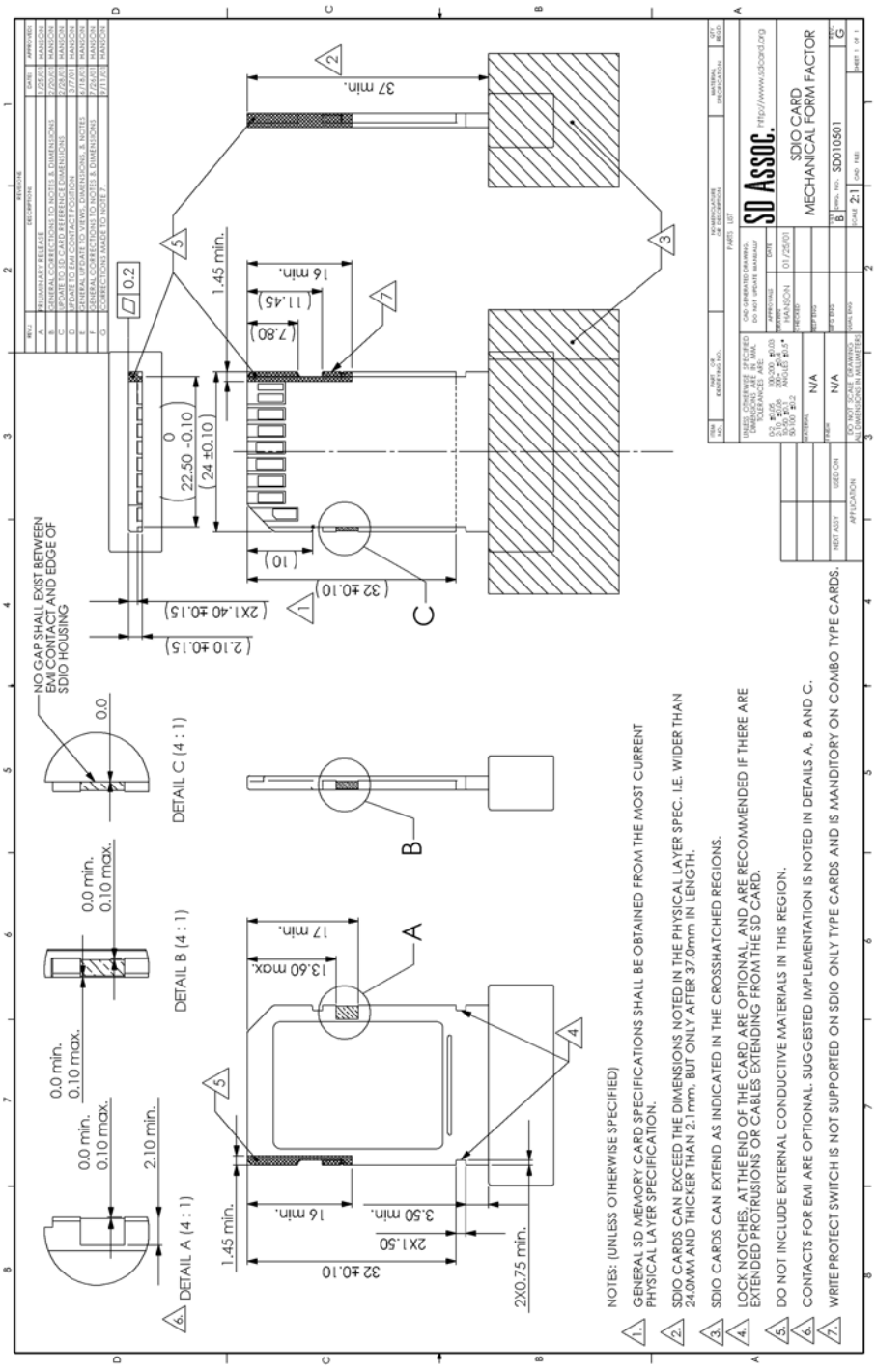


Figura A1. Extensions mecàniques de SDIO.

Apèndix 2. Llista de comandes SD i SPI.

Les taules Taula-1 i Taula-2 mostren les comandes suportades pels dispositius de memòria SD i SDIO en els modes SD i SPI. Si una comanda no és identificada com “Optional” o bé “Mandatory”, aleshores no és suportada pel dispositiu en qestió.

Taula 1. Llista de comandes SD.

Supported Commands	Abbreviation	SDMEM System	SDIO System	Comments
CMD0	GO_IDLE_STATE	Mandatory	Mandatory	Used to change from SD to SPI mode
CMD2	ALL_SEND_CID	Mandatory		CID not supported by SDIO
CMD3	SEND_RELATIVE_ADDR	Mandatory	Mandatory	
CMD4	SET_DSR	Optional		DSR not supported by SDIO
CMD5	IO_SEND_OP_COND		Mandatory	
CMD6	SWITCH_FUNC	Mandatory[1]	Mandatory[1]	Added in Part 1 v1.10
CMD7	SELECT/DESELECT_CARD	Mandatory	Mandatory	
CMD9	SEND_CSD	Mandatory		CSD not supported by SDIO
CMD10	SEND_CID	Mandatory		CID not supported by SDIO
CMD12	STOP_TRANSMISSION	Mandatory		
CMD13	SEND_STATUS	Mandatory		Card Status includes only SDMEM information
CMD15	GO_INACTIVE_STATE	Mandatory	Mandatory	
CMD16	SET_BLOCKLEN	Mandatory		
CMD17	READ_SINGLE_BLOCK	Mandatory		
CMD18	READ_MULTIPLE_BLOCK	Mandatory		
CMD24	WRITE_BLOCK	Mandatory		
CMD25	WRITE_MULTIPLE_BLOCK	Mandatory		
CMD27	PROGRAM_CSD	Mandatory		CSD not supported by SDIO
CMD28	SET_WRITE_PROT	Optional		
CMD29	CLR_WRITE_PROT	Optional		
CMD30	SEND_WRITE_PROT	Optional		
CMD32	ERASE_WR_BLK_START	Mandatory		
CMD33	ERASE_WR_BLK_END	Mandatory		
CMD38	ERASE	Mandatory		
CMD42	LOCK_UNLOCK	Optional		
CMD52	IO_RW_DIRECT		Mandatory	
CMD53	IO_RW_EXTENDED		Mandatory	Block mode is optional
CMD55	APP_CMD	Mandatory		
CMD56	GEN_CMD	Mandatory		
ACMD6	SET_BUS_WIDTH	Mandatory		
ACMD13	SD_STATUS	Mandatory		
ACMD22	SEND_NUM_WR_BLOCKS	Mandatory		
ACMD23	SET_WR_BLK_ERASE_COUNT	Mandatory		
ACMD41	SD_APP_OP_COND	Mandatory		
ACMD42	SET_CLR_CARD_DETECT	Mandatory		
ACMD51	SEND_SCR	Mandatory		SCR not supported by SDIO

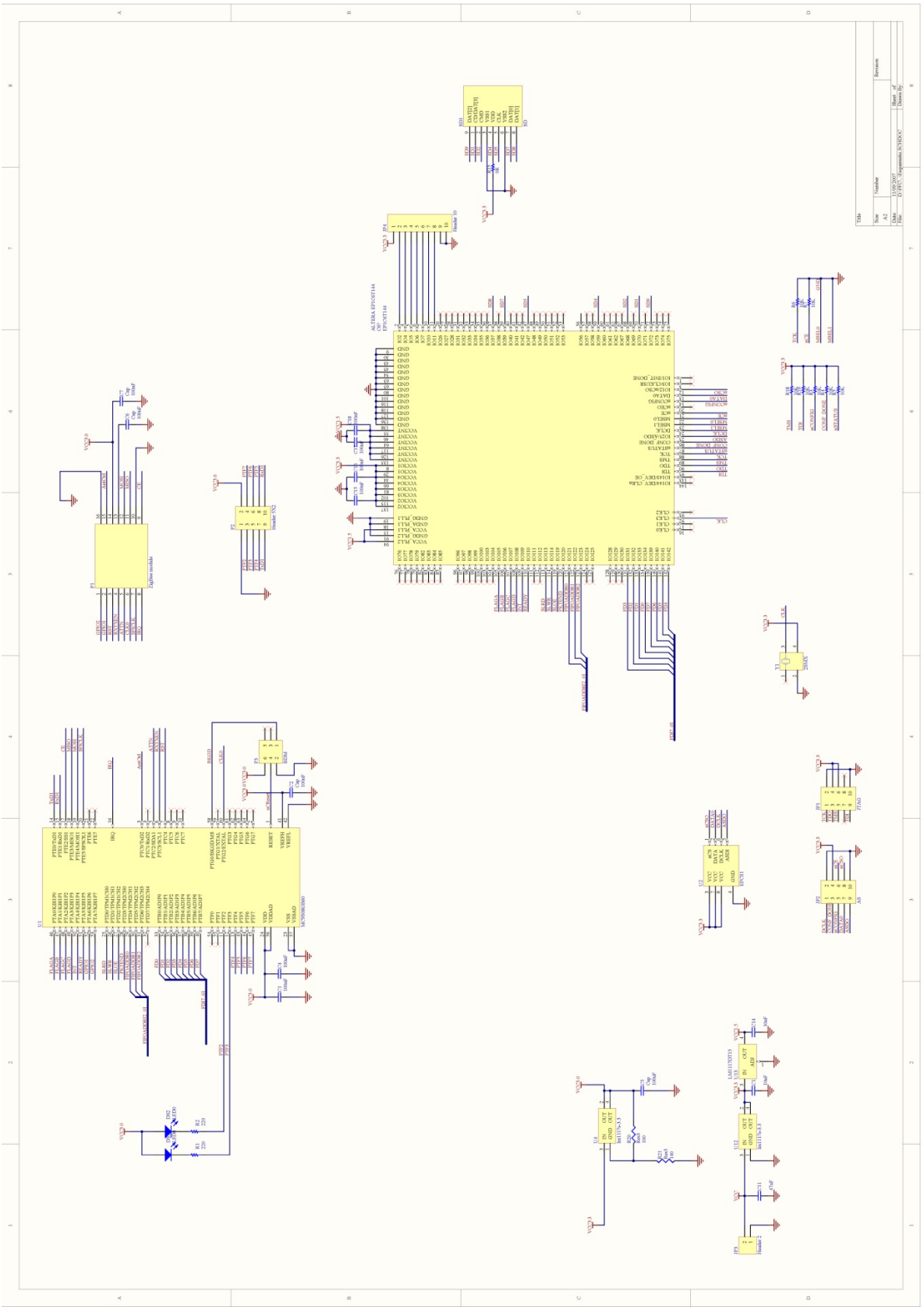
1 For Part 1 v1.10 or higher Memory or Combo Cards

Taula 2. Llista de comandes SPI.

Supported Commands	Abbreviation	SDMEM System	SDIO System	Comments
CMD0	GO_IDLE_STATE	Mandatory	Mandatory	Used to change from SD to SPI mode
CMD1	SEND_OP_COND	Mandatory		
CMD5	IO_SEND_OP_COND		Mandatory	
CMD6	SWITCH_FUNC	Mandatory[1]	Mandatory[1]	Added in Part 1 v1.10
CMD9	SEND_CSD	Mandatory		CSD not supported by SDIO
CMD10	SEND_CID	Mandatory		CID not supported by SDIO
CMD12	STOP_TRANSMISSION	Mandatory		
CMD13	SEND_STATUS	Mandatory		Card Status includes only SDMEM information.
CMD16	SET_BLOCKLEN	Mandatory		
CMD17	READ_SINGLE_BLOCK	Mandatory		.
CMD18	READ_MULTIPLE_BLOCK	Mandatory		
CMD24	WRITE_BLOCK	Mandatory		
CMD25	WRITE_MULTIPLE_BLOCK	Mandatory		
CMD27	PROGRAM_CSD	Mandatory		CSD not supported by SDIO.
CMD28	SET_WRITE_PROT	Optional		
CMD29	CLR_WRITE_PROT	Optional		
CMD30	SEND_WRITE_PROT	Optional		
CMD32	ERASE_WR_BLK_START	Mandatory		
CMD33	ERASE_WR_BLK_END	Mandatory		
CMD38	ERASE	Mandatory		
CMD42	LOCK_UNLOCK	Optional		
CMD52	IO_RW_DIRECT		Mandatory	
CMD53	IO_RW_EXTENDED		Mandatory	Block mode is optional
CMD55	APP_CMD	Mandatory		
CMD56	GEN_CMD	Mandatory		
CMD58	READ_OCR	Mandatory		
CMD59	CRC_ON_OFF	Mandatory	Mandatory	
ACMD13	SD_STATUS	Mandatory		
ACMD22	SEND_NUM_WR_BLOCKS	Mandatory		
ACMD23	SET_WR_BLK_ERASE_COUNT	Mandatory		
ACMD41	SD_APP_OP_COND	Mandatory		
ACMD42	SET_CLR_CARD_DETECT	Mandatory		
ACMD51	SEND_SCR	Mandatory		SCR includes only SD-MEM information.

1 For Part 1 v1.10 or higher Memory or Combo Cards

Apèndix 3. Esquemàtics de la tarja SDIO.



Joan Campderrós i Canas

RESUM

En aquest projecte es presenta el disseny i desenvolupament d'un conjunt d'interfícies per un sistema de comunicació basat en l'estàndard Zigbee. El sistema està composat per una tarja que integra el mòdul Zigbee, un microcontrolador i una FPGA, que es vol controlar des d'un sistema Pocket PC a través del port SD. La implementació consta d'un driver SDIO per Windows CE 4.2, el controlador SDIO a la FPGA i l'enllaç de comunicació entre la FPGA i el microcontrolador.

RESUMEN

En este proyecto se muestra el diseño y desarrollo de un conjunto de interfaces para un sistema de comunicación basado en el estándar Zigbee. El sistema, que está compuesto por una tarja que integra el módulo Zigbee, un microcontrolador y una FPGA, se quiere controlar desde un sistema Pocket PC a través del puerto SD. La implementación consta de un driver SDIO para Windows CE 4.2, el controlador SDIO en la FPGA y el enlace de comunicación entre la FPGA y el microcontrolador.

SUMMARY

In this project it is shown the design and development of a set of interfaces for a comunication system based on the Zigbee standard. The system, which is composed by a card that integrates a Zigbee module, a microcontroller and a FPGA, wants to be controlled from a Pocket PC system by the SD port. The implementation consists on a SDIO driver for Windows CE 4.2, the SDIO controller in the FPGA and the comunication bridge between the FPGA and the microcontroller.