

Pràctiques de Disseny i Síntesi de Sistemes Digitals Bus I²C

Grau Enginyeria Electrònica de Telecomunicació

Autor

Joan Canals Gil

Contribuidors

Sergio Moreno Martín,
Oscar Alonso Casanovas,
Ángel Diéguez Barrientos



UNIVERSITAT DE
BARCELONA



Índex

Pràctiques de Disseny i Síntesi de Sistemes Digitals Bus I²C	1
Introducció	5
Flux de disseny d'un sistema digital.....	5
Bus de comunicacions I ² C.....	7
Protocol	8
Connexions al Bus.....	10
Llistat de Pràctiques.....	12
Criteris d'avaluació.....	12
Material de Pràctiques	13
Al campus virtual trobareu:	13
Convenis	14
Estructura del directori de treball:	14
Codificació Verilog	14
Pràctica 1: Arquitectura d'un sistema digital	16
Objectius	16
Especificacions tècniques.....	16
Tasques a realitzar	17
Entrega.....	18
Pràctica 2: Introducció al flux de disseny i síntesis en FPGA.....	19
Introducció	19
Objectius	19
Material.....	19
Tasques a realitzar	20
Entrega.....	22
Annex Pràctica 2	24
Simulació netlist	24
Demostració.....	25



Pràctica 3: Disseny RTL i Verificació	26
Introducció	26
Objectius	26
P3 Sessió 1: Temporitzador	26
Introducció	26
Objectius	26
Material.....	26
Tasques a realitzar	26
Entrega.....	28
P3 Sessió 2: Registres de Configuració i Control	29
Introducció	29
Objectius	29
Material.....	29
Tasques a realitzar	29
Entrega.....	31
P3 Sessió 3-4: Unitat de Control del mestre I2C	32
Introducció	32
P3 Sessió 3: Controlador de Bit	32
Objectius	32
Material.....	33
Tasques a realitzar	33
Entrega.....	34
P3 Sessió 4: Controlador de Byte	35
Objectius	35
Tasques a realitzar	35
Entrega.....	37
Annex Pràctica 3	39
Màquines d'estat.....	39



Disseny d'una màquina d'estats pel control d'un semàfor.....	39
Diagrama d'estats.....	40
Definim les variables d'estat	41
Lògica de transició	41
Lògica seqüencial.....	42
Càlcul de les sortides.....	42
Pràctica 4: Estació Meteorològica	44
Introducció	44
Objectius	44
Material.....	44
Requisits tècnics de l'estació meteorològica	45
Tasques a realitzar	46
Entrega.....	49
Annex Pràctica 4	50
Sincronitzadors	50
Introducció	50
Com s'implementen en Verilog?	50
Habilitació de <i>Pull-Ups</i> interns.....	51
Pin Planner	52
Assignment Editor	53

Introducció

Flux de disseny d'un sistema digital

L'objectiu d'aquestes pràctiques és aprendre a dissenyar, sintetitzar i verificar un sistema digital complet. El flux genèric de disseny d'un sistema digital es pot dividir en diverses etapes tal com mostra la **Figura 1**. Cal notar que la síntesi física en el cas de disseny en ASIC inclou més subprocessos (*power planing, IO planning, clock tree*, entre d'altres) i les respectives verificacions en comparació en la implementació en una FPGA. Això és degut a l'estructura interna les FPGAs on el direccionament dels senyals, alimentació, propagació dels rellotges, nombre i tipus de portes lògiques disponibles, disposició d'entrades i sortides estan preestablertes.

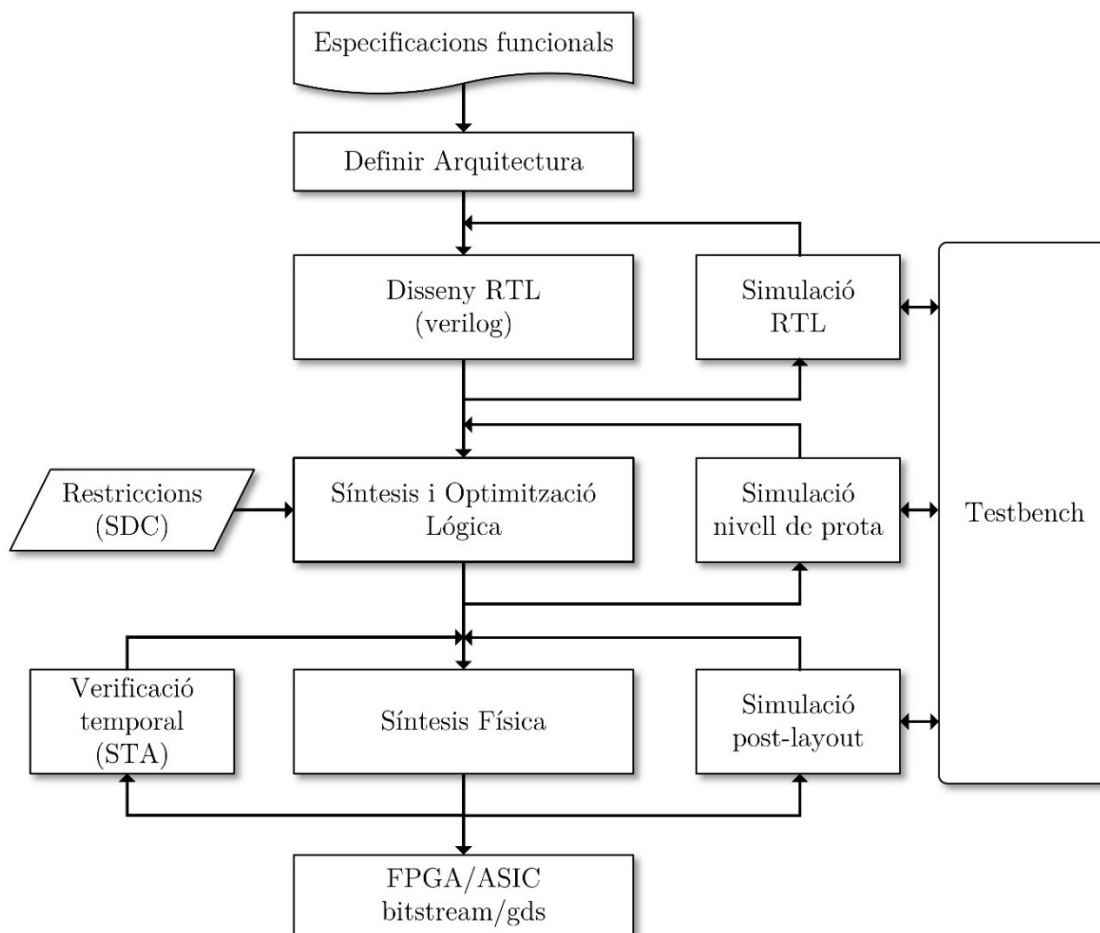


Figura 1 Flux de disseny i verificació genèric d'un sistema digital.



Durant el transcurs de les diferents sessions de pràctiques, veureu cadascuna de les diferents etapes aplicades al disseny d'una interfície de comunicació I²C (de l'anglès, *Inter-Integrated Circuit*), que s'utilitzarà com interfície de comunicació en una estació meteorològica per interrogar el sensor de temperatura, pressió i humitat (model BME280 de Bosch) integrat en una placa de desenvolupament Adafruit-2652.

Bus de comunicacions I²C

El bus I²C és un protocol de comunicació síncron que permet la transmissió de dades entre dispositius electrònics a través de dos línies bidireccionals col·lector/drenador obert (*open-collector/open-drain*): línia de dades en sèrie (SDA) i línia de rellotge sèrie (SCL), amb resistències de *pull-up*. Els voltatges típics utilitzats són +5 V o +3,3 V, encara que es permeten sistemes amb altres tensions. Aquesta tecnologia va ser desenvolupada per Philips l'any 1982, i actualment és utilitzada en una gran varietat d'aplicacions, des de sistemes encastats fins a dispositius mòbils i sensors, per connectar circuits integrats perifèrics de baixa velocitat a processadors o microcontroladors en comunicacions intra-placa de curta distància.

El protocol I²C és un protocol mestre/esclau, on un o més dispositius mestre poden controlar múltiples dispositius esclaus, tal com es mostra a la **Figura 2**. Els dispositius mestres són els encarregats d'iniciar la comunicació, mentre que els esclaus esperen les ordres del mestre per respondre. Per tant, els dispositius esclau no poden iniciar comunicacions ni parlar entre si.

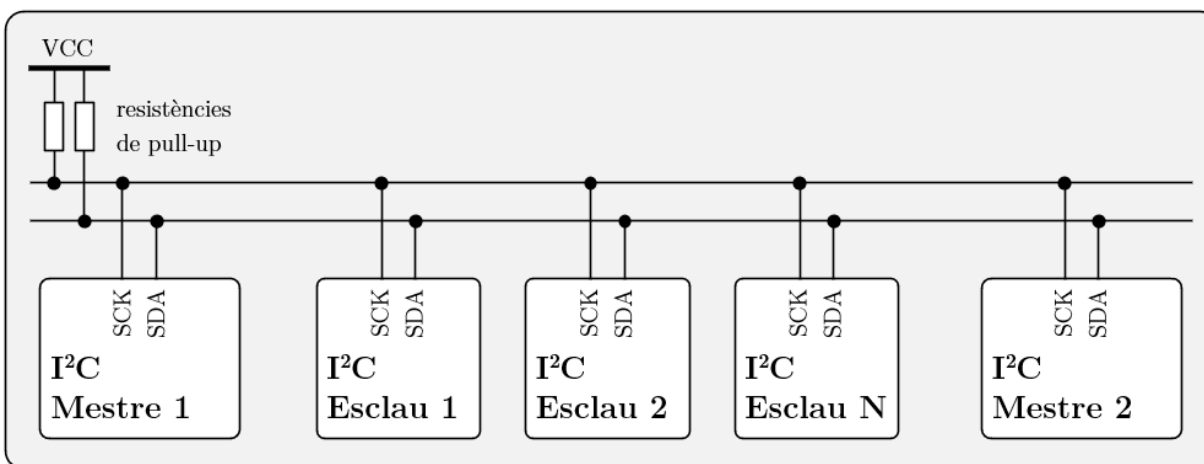


Figura 2 Exemple de bus I²C amb dos nodes mestre i tres nodes esclau.

En general, cada dispositiu I²C s'identifica a través d'una adreça única de 7 bits (amb una extensió de 10 bits rarament utilitzada), que els dispositius mestres utilitzen per seleccionar el dispositiu amb el qual volen comunicar. Tot i que com hem dit és possible tenir més d'un dispositiu mestre connectat al bus I²C, només un d'ells pot exercir de mestre en durant les comunicacions. Degut a l'alta complexitat d'implementació del que es coneix com a canvi de mestre, quan un mestre cedeix el bus a un altre mestre això requereix la monitorització del bus i la detecció de col·lisions, no és habitual trobar un bus I²C amb més d'un mestre.

Així doncs, el nombre de nodes/dispositius que poden existir en un bus I²C determinat està limitat per l'espai d'adreces i també per la capacitat total del bus de 400 pF, que també restringeix les distàncies pràctiques de comunicació a uns quants metres.

Les velocitats d'operació més comuns del bus I²C són el mode estàndard de 100 kbit/s i el mode ràpid de 400 kbit/s. També hi ha un mode de baixa velocitat de 10 kbit/s, però també es permeten freqüències de rellotge arbitràriament baixes. Les revisions posteriors d'I²C poden allotjar més nodes i funcionar a velocitats més ràpides (mode més ràpid d'1 Mbit/s, mode d'alta velocitat de 3.4 Mbit/s i mode ultra ràpid de 5 Mbit/s). Aquestes velocitats s'utilitzen més en sistemes encastats que en ordinadors.

Protocol

- Les dades es transfereixen entre un mestre i un esclau de forma síncrona amb el rellotge SCL a través de la línia SDA byte a byte.
- Cada byte de dades té una longitud de 8-bits.
- Hi ha un pols de rellotge SCL per a cada bit de dades, essent el bit més significatiu (MSB) transmès primer.
- Després de cada byte transferit, hi ha un bit de confirmació (**Ack** o **NAck**).
- Cada bit es mostreja durant la fase alta del senyal SCL; per tant, la línia SDA només pot canviar durant la fase baixa del senyal SCL i ha de romandre estable durant la fase alt del senyal SCL.
- Una transició a la línia SDA mentre el senyal SCL és alt es interpreta com una ordre (**Start** o **Stop**).

La **Figura 3** mostra una comunicació I²C Standard que consta de quatre parts.

1. Generació de la condició/senyal **Start**, indicant l'inici de la transmissió.
2. Transferència de l'adreça de l'esclau (7-bits) objectiu més 1-bit de direcció R/W.
3. Transferència de dades.
4. Generació de la condició/senyal **Stop**.

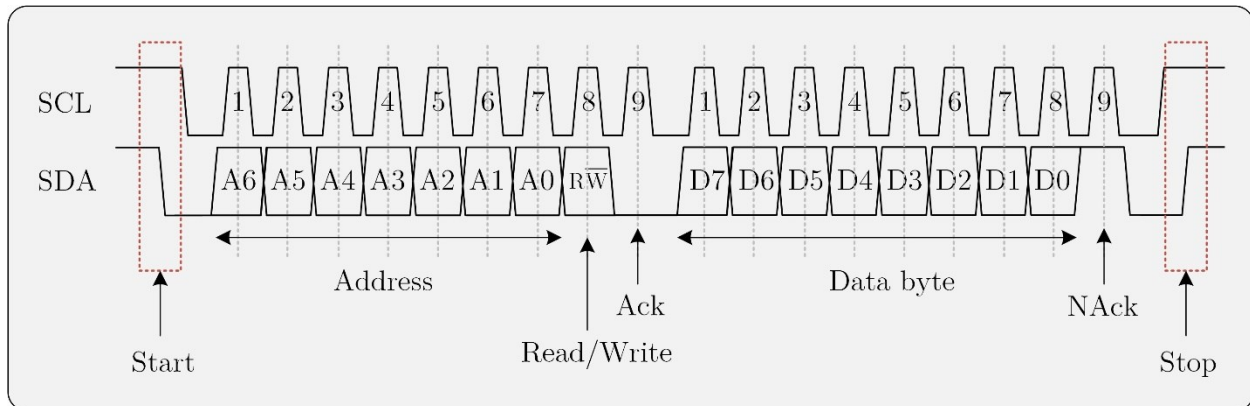


Figura 3 Trama genèrica de comunicació I²C.

- Quan el bus està lliure/inactiu (tant les línies SCL com SDA estan alts) un mestre pot iniciar una transferència enviant amb el senyal de **Start**, per indicar al bus que el mestre vol iniciar una transmissió. A continuació, el mestre envia l'adreça del node esclau objectiu al que vol enviar o rebre dades. Aquesta adreça és un número únic que identifica cada esclau en el bus. L'adreça es transmet en 7 bits seguits del bit d'operació que indica si el mestre vol escriure (0) o llegir (1) dades de l'esclau. Si l'esclau existeix i està disponible, respon amb un bit de confirmació (**Ack**), el mestre envia les dades a l'esclau (en el cas que vulgui escriure) o espera que l'esclau envii les dades (en el cas que vulgui llegir). Les dades s'envien en paquets de 8 bits, i cada paquet es confirma amb un senyal d'**Ack** o **NAck** depenent de si l'esclau ha rebut correctament les dades o no.
- Finalment, quan el mestre ha acabat de enviar o rebre les dades, envia el senyal de **Stop** per indicar al bus que la transmissió ha finalitzat, o torna a genera una senyal de **Start** per iniciar una nova transmissió amb el mateix esclau o un altre.

Consideracions

- Un senyal **Start** (S-bit), es defineix com una transició de nivell alt a nivell baix a la línia SDA mentre el senyal SCL és alt. Un mestre pot iniciar una transferència enviant una senyal **Start**.
- El primer byte de dades transferit pel mestre immediatament després de la senyal **Start** és l'adreça de l'esclau. Aquesta és una adreça de 7 bits seguida d'un bit R/W. El bit R/W indica a l'esclau la direcció de transferència de dades. Si el bit té un nivell lògic alt indica lectura (R) i si té un nivell baix indica escriptura (/W). Dos esclaus en el sistema **no** poden tenir la mateixa adreça.
- Un **Start repetit** és una senyal **Start** sense generar primer una senyal **Stop**. El mestre utilitza aquest mètode per comunicar-se amb un altre esclau o amb el mateix esclau

en una direcció de transferència diferent (per exemple, d'escriure a un dispositiu a llegir des d'un dispositiu) sense alliberar el bus.

- **Ack** bit (de l'anglès *acknowledge*, reconeixement) s'indica mantenint el SDA a nivell baix i el **NAck** (de l'anglès *no acknowledge*, no reconeixement) a nivell alt.
- En cas que el mestre vulgui escriure a l'esclau, llavors envia repetidament un byte i l'esclau respon enviant un bit **Ack** per cada byte. En aquesta situació es diu que, el mestre està en mode de transmissió i l'esclau està en mode de recepció.
- En el cas que el mestre vol llegir (rebre dades) des de l'esclau, rep repetidament un byte de l'esclau, i el mestre envia un bit **Ack** després de cada byte excepte l'últim en que envia un **NAck**. En aquesta situació es diu que el mestre està en mode de recepció i l'esclau està en mode de transmissió.
- Un senyal **Stop** (P-bit), es defineix com una transició de nivell baix a nivell alt a la línia SDA mentre el senyal SCL és a un nivell lògic alt. Quan el mestre genera la senyal **Stop**, això indica que ha acabat amb la transferència de dades i que allibera el bus.
- **Clock Stretching** o estirament del rellotge. Succeeix quan un dispositiu esclau necessita més temps per processar una dada o no pot respondre immediatament al mestre. En aquest cas, l'esclau té la capacitat de mantenir el fil de rellotge en estat baix, bloquejant temporalment la transmissió de dades.

Connexions al Bus

Tal i com sabem la interfície I²C utilitza una línia de dades sèrie (SDA) i una línia de rellotge sèrie (SCL) per a les transferències de dades i rellotge bidireccional. Per això, tots els dispositius connectats a aquestes dues senyals han de tenir sortides de tipus drenador o col·lector obert. Ambdues línies han de ser connectades a VCC mitjançant resistències externes.

En dissenys basats en FPGA, ports d'entrada/sortida inclouen lògica que permet configurar sortides open-drain. L'eina de síntesi facilita la feina i implementa automàticament els ports bidireccionals a través de buffers de tres estats (Taula 1, Figura 4).

Taula 1 Ports dels buffers de tres estats per a les línies SCL i SDA.

Port	#bits	Direcció	Descripció
SclPadIn	1	Entrada	Entrada de la línia de rellotge.
SclPadOut	1	Sortida	Sortida de la línia de rellotge.
SclPadEn	1	Sortida	Habilita la sortida de rellotge.
SdaPadIn	1	Entrada	Entrada de la línia de dades.

SdaPadOut	1	Sortida	Sortida de la línia de dades.
SdaPadEn	1	Sortida	Habilita la sortida de dades.

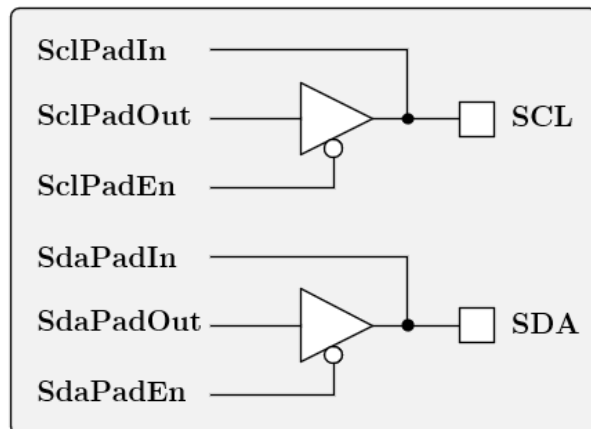


Figura 4 Connexions dels buffers de tres estats.

En dissenys FPGA, el codi Verilog següent pot ser utilitzat per a implementar aquests buffers de tres estats, que s'han d'afegir a un nivell jeràrquic superior (normalment en el fitxer top del disseny a implementar):

```
assign SCL = SclPadEn ? 1'bz : SclPadOut;
assign SDA = SdaPadEn ? 1'bz : SdaPadOut;
assign SclPadIn = SCL;
assign SdaPadIn = SDA;
```

Llistat de Pràctiques

- Les pràctiques es realitzen en sessions de laboratori de 3 hores.
- Cada pràctica consta d'un nombre de sessions i entregues determinats.

Pràctica	Títol	#Sessions	#Entregues	Puntuació
1	Arquitectura d'un sistema digital.	2	1	20 %
2	Introducció al flux de disseny i síntesis en FPGA.	1	1	15 %
3	Disseny RTL i Verificació.	4	4	40 %
4	Disseny i síntesis d'una estació Meteorològica.	3	1	25 %

Criteris d'avaluació

- Per aprovar una pràctica cal fer tots els apartats descrits en els guions i informes.
- Per aprovar les pràctiques s'han d'aprovar totes les pràctiques (mínim de 3 sobre 10).
- L'assistència al laboratori és obligatòria, la falta es considerarà un no presentat.
- Les FPGAs s'han de retornar al final de l'última sessió de pràctiques.
- Els criteris d'avaluació són generals:
 - Codi RTL: estructura i sintetitzable.
 - Codi del banc de proves (*testbench*): estructura i codificació.
 - Verificació completa de la funcionalitat del mòdul.
 - Comentaris al codi tant del *testbench* com del RTL.
 - Informe de pràctiques amb tots els apartats degudament complimentat amb explicacions clares i concises.
 - Implementació i demostració en FPGA.
 - Seguir els convenis d'estructura del directori de treball i codificació Verilog.

Material de Pràctiques

Per la realització de les pràctiques utilitzarem:

- Verilog HDL standard (IEEE 1364-2005) per descriure a nivell RTL els diferents mòduls i els test corresponents.
- ModelSim Intel® FPGA Starter Edition de Siemens que utilitzarem per simular i verificar els diferents mòduls dissenyats tant a nivell RTL com post-implementació.
- Intel Quartus Prime Lite Edition Design Software v20.1 que utilitzarem per sintetitzar el nostre disseny.
- Encara que tant el ModelSim com el Quartus incorporen editors de text, és recomana utilitzar un editor de codi extern. Algunes de les opcions més populars són: [SublimeText](#), [Neovim](#), [VS Code](#) o [GNU Emacs](#).
- Placa de desenvolupament DE0-CV de TerasIC, que consta d'una FPGA Intel 28-nm Cyclone V model 5CEBA4F23C7N, on implementarem alguns dels dissenys al llarg de les sessions pràctiques.
- Es recomana disposar de l'Analog Discovery 2 per comprovar les senyals generades per la FPGA.

Al campus virtual trobareu:

- Enllaç per la descarregar del ModelSim i el Quartus així com els fitxers de la tecnologia necessaris.
- Guia d'instal·lació i ús tant el Modelsim com del Quartus.
- Guia ràpida sobre el llenguatge Verilog.
- Lectures recomanades sobre el codificació, disseny i síntesis de sistemes digitals.
- Fitxers necessaris per la realització de les diferents sessions pràctiques.

Convenis

Estructura del directori de treball:

Prac_x

--- doc	documentació / informe
--- ip	IP generades amb el Quartus
--- misc	mòduls auxiliars i models funcionals utilitzats per verificació.
--- rtl	codi sintetitzable del disseny.
--- sdc	fitxers amb les restriccions del disseny, utilitzat per l'eina de síntesis.
--- sim	directori de treball del ModelSim.
--- syn	directori de treball del Quartus.
--- tb	fitxers de test utilitzats per l'eina de simulació.

Codificació Verilog

1. Un mòdul per fitxer, on el fitxer ha de tenir el mateix nom que el mòdul.
2. Noms de senyals:
 - a. Han de ser descriptius.
 - b. Entrades i Sortides primera lletra en majúscula.
 - c. Senyals internes només de registres en minúscules.
 - d. Paràmetres i directives en majúscules.
 - e. Nom de les instàncies de mòduls en minúscules amb el prefix "i_".

3. Estructura fitxer Verilog:

```
module name #(
    // declaració paràmetres
)(
//llista input outputs comentant la seva funció

);
// declaració de paràmetres locals

// declaració de regs & wires comentant la seva funció

// descripció del disseny: lògica combinacional i seqüencial

endmodule
```

4. Comentaris al codi:

- a. Utilitzeu el comentari en línia // per explicar la funció de les senyals.
- b. Comenteu estructures de codi. No cal comentar línia a línia, a menys que sigui necessari per la correcta comprensió del que fa.

Pràctica 1: Arquitectura d'un sistema digital

Durada: 2 sessions.

Objectius

Aprendre a definir l'arquitectura d'un sistema digital en base a unes especificacions tècniques donades. En aquest cas un mestre I²C que compleixi amb els següents requisits tècnics.

Especificacions tècniques

1. Freqüència del sistema 100 MHz.
2. El mestre I²C ha de permetre mode estàndard, ràpid i alta velocitat.
3. Freqüència de transmissió seleccionable, mitjançant un pre-escalat de 8-bits.
4. Unitat mínima d'informació a transmetre 1 byte.
5. S'ha de poder habilitar i deshabilitar el mòdul.
6. El mòdul ha de poder generar una senyal de sol·licitud d'interrupció.
7. Ha de detectar automàticament quan el bus I²C està ocupat (s'ha generat una senyal d'*Start*).
8. Ha de detectar automàticament quan hi ha una col·lisió (quan el mestre no pot posar un 1 a la línia de dades).
9. Ha de detectar el *clock stretching* produït per l'esclau i esperar per continuar amb la transferència.
10. El control del mòdul ha de ser a través d'un conjunt de registres de 8 bits de dades i 3 bits d'adreces. L'escriptura d'aquests ha de ser síncrona i la lectura asíncrona. Ha d'incloure entre d'altres:
 - Un registre de control on el bit-7 sigui el bit d'habilitació i el bit-6 habiliti la generació de la petició de interrupció.
 - Un registre d'estat on:
 - El bit-7 (més significatiu) indiqui quan s'ha rebut el *Ack* per part de l'esclau.
 - El bit-6 indica si s'ha el bus està ocupat (des de que es detecta *Start* fins que es detecta un *Stop*).
 - El bit-5 indica que s'ha detectat una col·lisió en el bus.

- El bit-1 indica si s'està transferint data (1) o s'ha completat la transferència (0).
- El bit-0 indica si hi ha una interrupció pendent, ja sigui per pèrdua d'arbitratge del bus o perquè s'ha completat una transferència.
- Un registre de comandament que s'utilitza per indicar quina transmissió/comanda volem generar:
 - El bit-7 per indicar que volem generar (o repetir) un condició d'*Start*.
 - El bit-6 generar la condició de *Stop*.
 - El bit-5 indica que volem fer una transferència de lectura.
 - El bit-4 indica que volem fer una transferència d'escriptura.
 - El bit-3 per especificar si volem contestar amb un *Ack* o *NAck* en mode receptor.
 - El bit-1 per baixar la bandera de col·lisió.
 - El bit-0 per baixar la bandera d'interrupció.

Els bits 7, 6, 5 i 4 s'esborren automàticament quan la transferència en curs acaba o si s'ha detectat una col·lisió. Els bits 1 i 0 s'esborren automàticament al següent cicle de rellotge.

- Un registre de transmissió on s'escriurà la dada a transmetre.
 - Un registre de recepció que s'actualitzarà amb l'últim byte rebut.
11. El valor dels quatre bits més significatius del registre de comanda s'han de reinicialitzar automàticament un cop finalitzada la transmissió de la comanda.
 12. La seqüència de transmissió/recepció de dades ha d'iniciar-se automàticament després d'escriure en el registre de comanda (en les quatre bits més significatius).
 13. L'ordre de prioritat de generació de comanda és *Start*, *Write*, *Read*, *Stop*. Sempre s'ha d'indicar si la comanda és d'escriptura o lectura.

Tasques a realitzar

En base als requisits tècnics enumerats anteriorment, determineu els mòduls necessaris per implementar el I²C mestre i descriuiu la seva funcionalitat.

1. Definiu les entrades i sortides del mestre I²C a implementar.
2. Definiu els mòduls bàsics necessaris en base a les especificacions.

Ajut: degut a la generació de senyals de *Start*, *Stop*, és interessant tenir un control de la transmissió a nivell "byte" i de bit.

3. Feu el diagrama de blocs (no RTL) de l'arquitectura que mostri les interconnexions entre els diferents blocs i les E/S del mestre I²C.
4. De cada mòdul/bloc, dibuixeu el diagrama RTL detallat i enumereu-ne les seves entrades i sortides, exceptuant les màquines d'estats (on només cal indicar entrades i sortides).
5. Descriviu breument la funcionalitat de la/les màquina/es d'estats.
6. Descriviu la funcionalitat dels registres bit a bit.

Entrega

L'informe que trobareu al campus virtual, que constarà de:

- Descripció breu de l'arquitectura del mestre I²C, enumerant cada mòdul i descrivint la seva funcionalitat.
- Diagrama RTL detallat complet. Indicant noms dels senyals i la seva mida (si són de més d'un bit). Feu l'esquema amb ordinador amb Visio, Draw.io, SchemDraw, InkScape, PowerPoint. Es desaconsella l'ús del Paint o similars.
- Taula amb les entrades i sortides del mòdul I²C i la seva funcionalitat.
- Taula de registres indicant si són d'escriptura i/o lectura i la funció de cada bit.
- Descripció breu l'operació (els passos a seguir) per realitzar la transmissió d'escriptura d'un byte de dades a un esclau amb adreça 0x63.

Pràctica 2: Introducció al flux de disseny i síntesis en FPGA

Durada: 1 sessió

Introducció

La Figura 5 mostra el diagrama RTL simplificat del mestre I²C. Alguns blocs, com el de control de bit y byte a la figura estan representats de forma genèrica como una maquina d'estats amb les entrades y sortides necessàries. En altres, mes senzills, com el bit timer es dona tot el detall d'implementació. Un cop definida l'arquitectura del nostre sistema digital, els següents passos són codificar a nivell RTL i verificar-ne la funcionalitat, per finalment sintetitzar el sistema complet. En aquesta pràctica ens familiaritzarem amb els passos bàsics a seguir pel flux de disseny i síntesis en FPGA. Per fer-ho ens centrarem en la implementació dels registres de desplaçament que es mostren al diagrama RTL de la Figura 5.

Objectius

- Codificar RTL de l'estructura descrita en base a les especificacions donades.
- Familiaritzar-se en l'ús de l'entorn de simulació ModelSim.
- Introducció al *self-checking* mitjançant l'ús de tasques, funcions pròpies del Verilog (*\$display*, *\$monitor*, *\$wait*, *\$force*, *\$release*, ...) i la monitorització de senyals internes del dispositiu a verificar (DUT, de l'anglès Device Under Test).
- Familiaritzar-se en l'ús de l'entorn de síntesis Quartus Prime d'Altera i simulació post-implementació.

Material

El material de suport el podeu descarregar del campus virtual:

- `tb_shiftreg.v` : testbench a completar.
 - Plantilla informe de la pràctica.
-

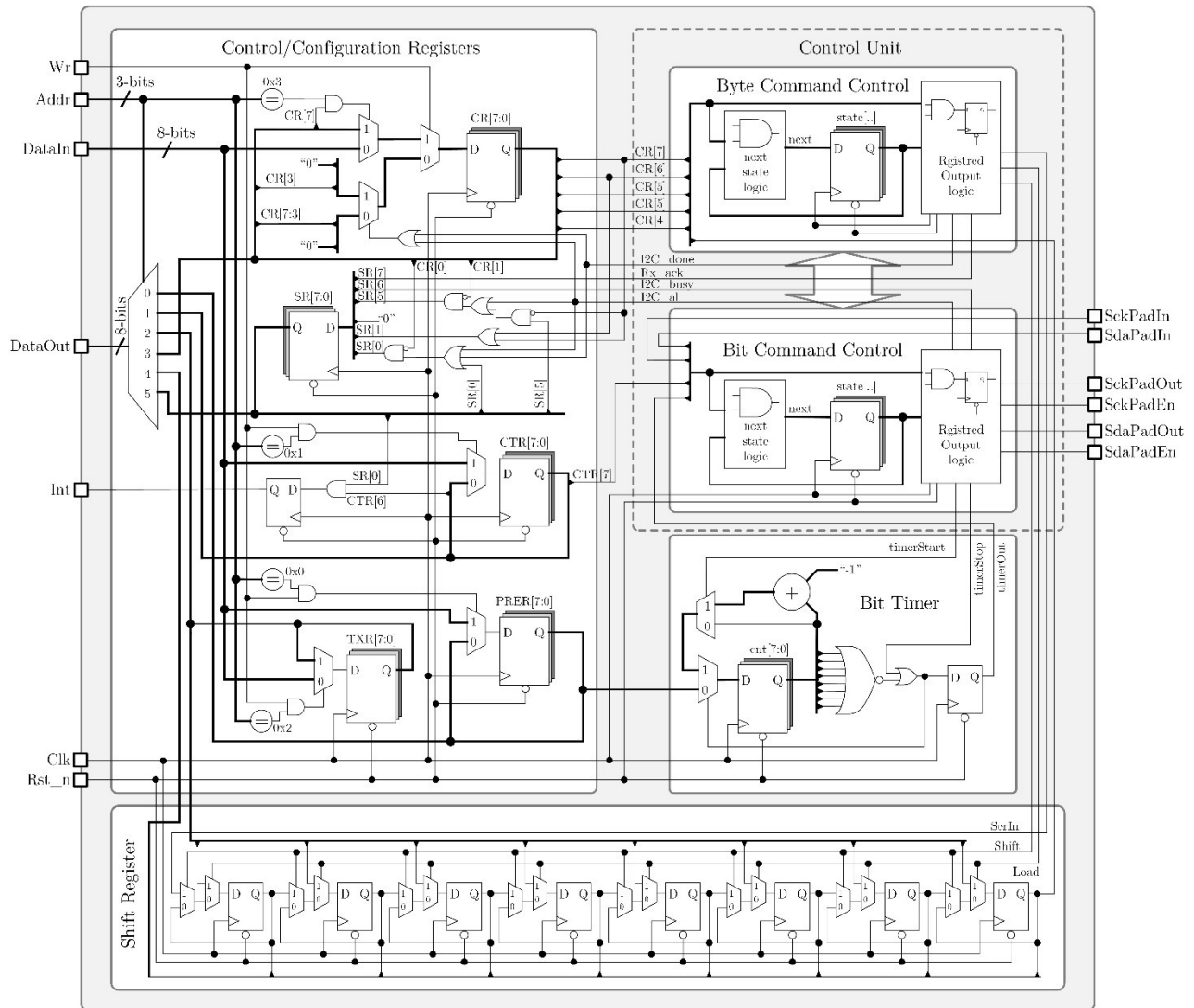


Figura 5 Arquitectura del mestre I2C a implementar.

Tasques a realitzar

1. Dissenyau un registre de desplaçament genèric no-cíclic parametrizable de N-bits, que anomenarem *shiftreg*. El registres de desplaçament ha de permetre la càrrega en paral·lel de dades. Ha de tenir una entrada i sortida en sèrie i la sortida en paral·lel del contingut del registres. El registres ha de tenir una senyal d'habilitació i un reset asíncron actiu per nivell baix. Els ports d'entrada i sortida seran doncs:

Clk	Relotge del sistema.
Rst_n	Reset asíncron actiu per nivell baix.
Load	Habilita la carrega de dades en paral·lel. Actiu per nivell alt.
En	Habilitació del desplaçament. Actiu per nivell alt.
DataIn	Entada de dades en paral·lel de N-bits.
DataOut	Sortida de dades en paral·lel de N-bits.
SerIn	Entrada de dades sèrie, que s'insereixen per el bit menys significatiu (LSB).
SerOut	Sortida de dades sèrie que es correspon amb el bit més significatiu (MSB).

2. Examineu el fitxer *tb_shiftreg.v* que trobareu al campus virtual i instancieu el vostre registre de desplaçament com a DUT i verifiqueu-ne el funcionament.
3. Completeu el test del registre de desplaçament creant la següent tasca:
test_serin, que carregui pel port d'entrada sèrie i comprovar que després de N cicles el valor de dins el registres es correspon al valor esperat. El valor esperat s'ha de calcular automàticament dins la mateixa tasca. Per tant, la tasca ha de tenir com a inputs: la dada a rebre pel port *SerIn*, i el nombre de desplaçaments que volem que fer.
4. Utilitzeu la tasca del sistema *\$monitor* per veure l'estat de les entrades i sortides del registre de desplaçament en tot moment (excloent els senyals de *Clk* i *Rst_n*).
5. Simuleu i verifiqueu el correcte funcionament per un registre de 8-bits per diferents valors d'entrada i nombre de desplaçaments.
6. Verifiqueu visualment mitjançant el diagrama d'ones cada un dels test. Examineu els estímuls externs, les senyals internes i les sortides del vostre registre de desplaçament. També podeu examinar les variable de les tasques.
7. Sintetitzeu el registre de desplaçament de 8 bits amb el Quartus per una Cyclone IVE model EP4CE22F17C6.
 - Creeu un projecte nou. Consulteu la guia de Quartus que teniu al campus virtual.
 - Elaboreu el disseny i comproveu quins missatge d'*error*, *info*, i *alertes* ús dona.
 - Examineu el esquema RTL generat per el Quartus. És l'esperat?
 - Creeu un fitxer de restriccions temporals (*shiftreg.sdc*) dins la carpeta misc, que contingui la següent comanda.

```
create_clock -name clk100MHz -period 10.0 [get_ports Clk]
```

Aquesta comanda defineix un rellotge de 100 MHz al port d'entrada *Clk*. Aquest rellotge serà utilitzat per l'eina de síntesis per realitzar l'anàlisi temporal estàtic i comprova si el disseny implementat pot funcionar aquesta freqüència.

- Afegiu el fitxer SDC al projecte de Quartus.
 - Assigneu les entrades i sortides als pins de la FPGA indicats a la **Taula 2** de l'Annex *Pràctica 2 – Simulació netlist*.
8. Sintetitzeu de nou el disseny i examineu el *Compilation Report*.
- Compleix els requisits temporals? Quina és la freqüència màxima del disseny?
 - Hi ha alguna E/S del vostre mòdul sense definir (*unconstrained*)?
 - Quants i quins recursos s'utilitzen?
9. Configureu el projecte de Quartus per poder simular la *netlist* generada. Consulteu la guia *guia_SimulacióNetlisQuartus* que trobareu al campus virtual.
- Quina diferència veieu respecte a la simulació RTL? Examineu el retard de les senyals respecte al flanc de rellotge.
10. Aquest apartat es optatiu. Per últim comproveu que efectivament funciona el registre de desplaçament amb la placa DE0-CV. Per fer-ho heu de seguir els següents passos:
- A. Canvieu el model de FPGA per el de la DE0-CV una **Cyclone V 5CEBA4F23C7N**. Per fer-ho d'anar al menú *Assignments > Device*.
 - B. Actualitzeu l'assignació de pins d'acord amb la **Taula 3** de l'Annex *Pràctica 2 – Demostració*.
 - C. Sintetitzeu de nou el disseny.
 - D. Carregeu-lo a la placa DE0-CV per comprovar que funciona. Heu de comprovar totes les seves funcions.

Entrega

Fer demostració al professor un cop s'ha implementat a la FPGA. En cas de no poder fer la demostració a l'aula, heu de gravar un vídeo que adjuntareu dins la carpeta doc. Recordeu que la demostració es optativa.

Un fitxer ZIP amb el directori de treball:

1. A la carpeta **rtl**: el codi RTL sintetitzable.
 2. A la carpeta **tb**: el codi del testbench complet.
-



3. A la carpeta **misc**: el fitxer de restriccions temporals (.sdc).
4. Dins la carpeta **doc** hi heu de posar l'informe complimentat que trobareu al campus virtual, que constarà de:
 - Enumerar les tasques del banc de proves i explicar breument què fan.
 - Captures de les simulacions, amb una explicació breu i ressaltant les zones d'interès.
 - Captura del terminal de simulació amb els missatges de l'auto verificació.
 - Captura del esquema RTL resultant de la síntesi amb el Quartus (expandiu les caixetes!).
 - Taula on consti freqüència màxima d'operació, nombre de portes utilitzades de cada tipus i percentatge d'ocupació de la Cyclone V.

Annex Pràctica 2

Simulació netlist

Per la simulació de la netlist amb els retards de la lògica de la pràctica 2 farem servir els següents pins de la Cyclone IV-E muntada en una placa DE0-Nano (no és la mateixa que per la demostració):

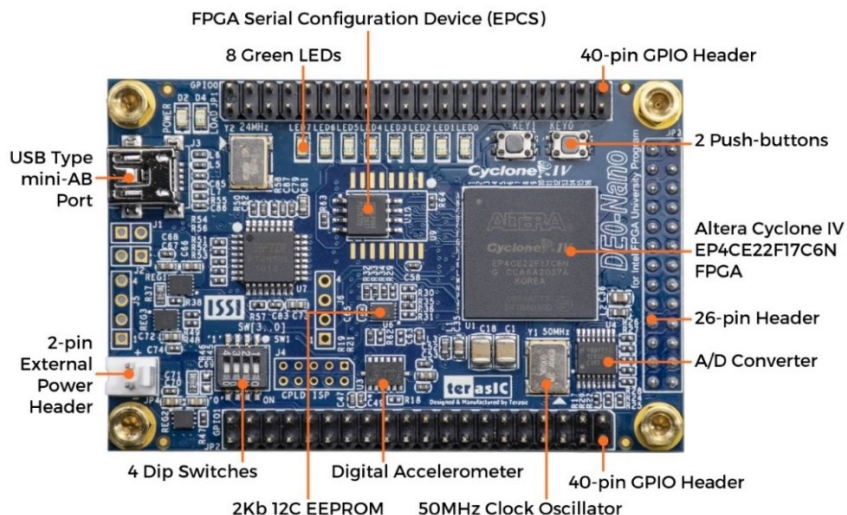


Figura 6 Vista superior de la placa de desenvolupament DE0-Nano amb els diferents elements identificats.

Taula 2 Assignació de pins de la Cyclone IV-E de la DE0-Nano.

Senyal	Element	Pin	Senyal	Element	Pin
En	SW[0]	M1	DataOut[7]	LED7	L3
SerIn	GPIO_0[1]	C3	DataOut[6]	LED6	B1
DataIn[7]	GPIO_0[10]	A6	DataOut[5]	LED5	F3
DataIn[6]	GPIO_0[9]	B6	DataOut[4]	LED4	D1
DataIn[5]	GPIO_0[8]	D5	DataOut[3]	LED3	A11
DataIn[4]	GPIO_0[7]	B5	DataOut[2]	LED2	B13
DataIn[3]	GPIO_0[6]	A4	DataOut[1]	LED1	A13
DataIn[2]	GPIO_0[5]	B4	DataOut[0]	LED0	A15
DataIn[1]	GPIO_0[4]	B3	Clk	CLOCK_50	R8
DataIn[0]	GPIO_0[3]	A3	Rst_n	KEY[0]	J15
SerOut	GPIO_0[2]	A2	Load	GPIO_0[0]	D3

Demostració

Per la demostració de la pràctica 2 farem servir els següents elements de la placa DE0-CV:

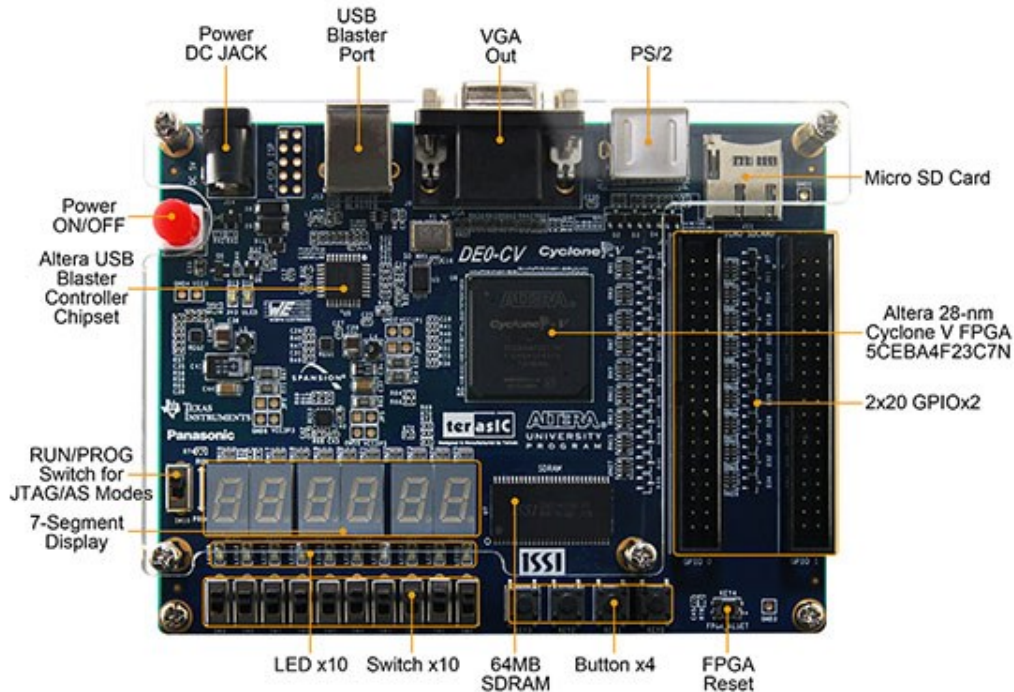


Figura 7 Vista superior de la placa de desenvolupament DE0-CV amb els diferents elements identificats.

Taula 3 Assignació de pins de la Cyclone V de la DE0-CV.

Senyal	Element	Pin	Senyal	Element	Pin
En	SW9	PIN_AB12	DataOut[7]	LED7	PIN_U1
SerIn	SW8	PIN_AB13	DataOut[6]	LED6	PIN_U2
DataIn[7]	SW7	PIN_AA13	DataOut[5]	LED5	PIN_N1
DataIn[6]	SW6	PIN_AA14	DataOut[4]	LED4	PIN_N2
DataIn[5]	SW5	PIN_AB15	DataOut[3]	LED3	PIN_Y3
DataIn[4]	SW4	PIN_AA15	DataOut[2]	LED2	PIN_W2
DataIn[3]	SW3	PIN_T12	DataOut[1]	LED1	PIN_AA1
DataIn[2]	SW2	PIN_T13	DataOut[0]	LED0	PIN_AA2
DataIn[1]	SW1	PIN_V13	Clk	KEY0	PIN_U7
DataIn[0]	SW0	PIN_U13	Rst_n	KEY1	PIN_W9
SerOut	LED9	PIN_L1	Load	KEY2	PIN_M7

Pràctica 3: Disseny RTL i Verificació

Durada: 4 sessions

Introducció

En la pràctica 2 ens vàrem familiaritzar amb el flux de disseny aplicat a un registre de desplaçament. Durant les pròximes 4 sessions dissenyarem i verificarem la resta de mòduls que formen el mestre I²C.

Objectius

- Codificar RTL diferents estructures descrites en base a les especificacions donades.
- Verificació funcional pre- i post-síntesis en FPGA.

P3 Sessió 1: Temporitzador

Introducció

El següent mòdul del mestre I²C que dissenyarem serà el temporitzador, que s'utilitza per gestionar les temps de bit del bus I²C.

Objectius

- Codificar RTL en base a les especificacions donades.
- Verificació pre- i post-síntesis.

Material

El material de suport el podeu descarregar del campus virtual:

- Plantilla informe de la pràctica.

Tasques a realitzar

1. Dissenyu el temporitzador de la **Figura 5** (pràctica 2) anomenat *i2c_bit_timer* basat en un comptador regressiu cíclic amb les següents característiques:



- Restabliment o reset: Ha de ser asíncron i actiu per nivell baix i ha de posar tant el valor del comptador com de la sortida a 0.
 - Parametritzable: El nombre de bits del comptador ha de ser parametrizable.
 - Límit: El valor inicial del ha de ser configurable, mitjançant una entrada de N-bits que anomenarem *Ticks*.
 - Auto-reinici: Un cop el comptador arriba al 0, aquest ha de reiniciar-se per començar un nou cicle de comptatge.
 - Reinici forçat: El temporitzador ha de poder tornar al valor inicial en qualsevol moment i mantenir-se en aquest mentre la senyal de *Start* estigui activa.
 - Parada: El temporitzador ha de poder parar-se sempre que no s'estigui reiniciant. Per fer-ho, heu d'incloure una entrada que anomenarem *Stop*, activa per nivell alt.
 - Sortida: El temporitzador ha de tenir una sortida anomenada *Out*. Aquesta sortida s'ha d'activar quan es reinicia el valor al temporitzador.
2. Dissenyeu el testbench (*tb_timer.v*) per verificar el correcte funcionament del temporitzador que utilitzi tasques i lògica per automatitzar-ne la verificació. Creeu una tasca que:
- Verifiqui el nombre de cicles de rellotge entre dos polsos de sortida consecutius.
 - Indiqui el temps transcorregut entre dos polsos consecutius. Ajuda: *\$realtime \$time \$monitor \$display \$while*.
 - Tingui una entrada per escollir el límit de temporitzador.
Ajut: quan límit és 0 cas especial! Mostreu un missatge que ho digui.
 - Tingui una entrada per escollir quants cicles es para el temporitzador duran el comptatge.
Ajut 1: 0 cicles vol dir que no es para.
Ajut 2: feu que es pari quan el comptador va per la meitat.
3. Simuleu i verifiqueu el correcte funcionament per un temporitzador de 4-bits pels valors 0, 1, 8 i 15.
4. Sintetitzeu un temporitzador de 8 bits amb el Quartus i simuleu la *netlist* generada per verificar-ne el funcionament per una **Cyclone V model 5CEBA4F23C7N**. Pareu atenció als missatges d'informació i alertes. No us oblideu del fitxer de restriccions (SDC). Els pins d'entrada i sortida han d'anar connectats als següents elements.
- SW0 – SW7 per introduir el nombre de cicles entre polsos.
-

- Key0 per generar el rellotge del sistema.
- Key1 com a reset del sistema.
- Key2 com a set del sistema.
- LED0 per connectar el senyal d'interrupció del temporitzador.

Nota: consulteu el manual de la placa de desenvolupament per saber quins pins de configuració es corresponen amb cadascun dels elements esmentats.

5. Comproveu que el disseny compleix els requisits temporals.

Entrega

Fer demostració al professor un cop s'ha implementat a la FPGA. En cas de no poder fer la demostració a l'aula, heu de gravar un vídeo que adjuntareu dins la carpeta doc.

Un fitxer ZIP amb el directori de treball:

1. A la carpeta **rtl**: el codi RTL sintetitzable.
2. A la carpeta **tb**: el codi del testbench complet.
3. A la carpeta **misc**: el fitxer de restriccions temporals (.sdc).
4. Dins la carpeta **doc** hi heu de posar l'informe complimentat que trobareu al campus virtual, que constarà de:
 - Enumerar les tasques del testbench i explicar breument què fan.
 - Captures de les simulacions, amb una explicació breu i ressaltant les zones d'interès.
 - Captura del terminal del ModelSim amb els missatges de l'auto verificació.
 - Captura del esquema RTL resultant de la síntesi amb el Quartus (expandiu les caixetes!).
 - Taula on consti freqüència màxima d'operació, nombre de portes utilitzades de cada tipus i percentatge d'ocupació de la FPGA.

P3 Sessió 2: Registres de Configuració i Control

Introducció

En aquesta sessió ens centrarem el disseny del test per verificar el conjunt de registres de configuració i control que us podeu descarregar del campus virtual.

Objectius

- Planificació del test exhaustiu d'un mòdul digital amb *self-checking*.
- Utilització de **declaracions seqüencials aplicades a testbench** com són, *fork-join*, *wait*, *repeat*, entre d'altres.
- Disseny de tasques i funcions.
- **Accés jeràrquic** a elements com registres, *wires*, *tasques* de les instàncies/mòduls.

Material

El material de suport el podeu descarregar del campus virtual:

- `i2c_master_regs.v` : RTL dels registres de configuració del I²C.
- `i2c_master_defines.v` : definicions d'adreces del I²C.
- `sys_model.v` : model del sistema, genera el rellotge base i el reset actiu per nivell baix.
- `dbus_master_model.v` : model de mestre del bus de dades del sistema que escriu/llegeix dels registres de configuració.
- `tb_i2c_master_regs.v` : testbench a completar.
- `timescale.v` : defineix l'escala temporal pel fitxer on està inclòs.
- Plantilla informe de la pràctica.

Tasques a realitzar

1. Examineu el contingut del fitxer `i2c_master_regs.v`, identifiqueu les diferents estructures i comenteu el codi.
 2. Feu una llista amb les verificacions/testes a fer. Un cop feta comenteu-la amb el professor.
 3. Examineu el contingut del fitxer `tb_i2c_master_regs.v`.
-

4. Simuleu i verifiqueu el conjunt de registres (*i2c_master_regs.v*) tot completant el testbench (*tb_i2c_master_regs.v*) on s'indica (TODO). Seguiu l'ordre establert que s'indica en el mateix fitxer.
5. Sintetitzeu i verifiqueu el disseny per la FPGA **Cyclone IV E EP4CE22F17C6**.
 - Creeu el fitxer de restriccions de disseny (*i2c_master_regs.sdc*) on hi heu de definir un rellotge del sistema de 100MHz.
 - Creeu un projecte de Quartus a la carpeta (*syn*) anomenat *i2c_master_regs*.
 - Afegiu els fitxer verilog i el SDC al projecte.
 - Configureu el Quartus per simular a nivell de porta (gate level). Seguiu la guia que trobareu al campus virtual.
 - Sintetitzeu el conjunt de registres (*Processing>Start>Start Analysis & Synthesis*). Pareu atenció als missatges d'informació i alertes.
 - Assigneu els pins d'entrada i sortida als GPIOs (a la vostra elecció). Mireu la fulla d'especificacions de la **DE0-Nano** que trobareu al campus virtual.
 - Abans de compilar el disseny assegureu-vos de desactivar l'opció de llençar la simulació automàticament. Premeu a *Assignments>Settings>EDA Tool Settings* **desmarqueu** l'opció *Run gate-level simulation....* Ja podeu compilar el disseny.
 - Comproveu que el disseny compleix els requisits temporals.
 - Inspeccioneu el esquemàtics generats (*Tools → Netlist Viewers*) a nivell:
 - *RTL Viewer*
 - *Technology Map Viewer (Post-Fitting)*.
 - Quines diferències hi trobeu? Mireu el control dels bits del registre de comandament (CR) per exemple.
 - Examineu la netlist (.vo) i el fitxer de retards (.sdo) generats al directori del vostre projecte de Quartus: `\Lab3s2\syn\simulation\modelsim\`
6. Simuleu la netlist generada amb el mateix fitxer de testbench. Per simular a nivell de porta i veure els retards interns seguiu la guia (Simulació Netlist Quartus) que trobareu al campus virtual.
 - Un cop guardats els canvis en el fitxer de testbench, inicieu la simulació. Per fer-ho cliqueu a *Tools > Run Simulation Tool > Gate Level Simulation...*
 - Examineu el diagrama d'ones i la finestra de transcripció de l'eina de simulació.
 - Examineu el comportament de la senyal de registre de comandament (CR).



Entrega

Un fitxer ZIP amb el directori de treball:

1. A la carpeta **rtl**: el codi RTL.
2. A la carpeta **tb**: el codi del testbench complet.
3. A la carpeta **sdc**: el fitxer de restriccions temporals (.sdc).
4. Dins la carpeta **doc** hi heu de posar l'informe complimentat que trobareu al campus virtual, que constarà de:
 - Enumerar les tasques del *testbench* i explicar breument què fan.
 - Captures de les simulacions funcionals, amb una explicació breu i ressaltant les zones d'interès.
 - Captura del terminal del ModelSim amb els missatges de l'auto verificació.
 - Captura del esquema RTL resultant de la síntesi amb el Quartus i taula on consti freqüència màxima d'operació, i recursos utilitzats de la FPGA.
 - Captures de les simulacions post-síntesis, amb una explicació breu i ressaltant les zones d'interès.

P3 Sessió 3-4: Unitat de Control del mestre I²C

Introducció

Un cop dissenyats els mòduls bàsics del nostre mestre I²C, només ens falta dissenyar la unitat de control encarregada de gestionar la transmissió dels bits d'informació i control així com la generació del senyal rellotge. Per simplificar el seu desenvolupament, la unitat de control es divideix en dos mòduls: el controlador de comandes a nivell de byte i el controlador de comandes a nivell de bit, que anomenarem controlador de byte i controlador de bit.

El controlador byte, gestiona el trànsit I²C a nivell de byte. Prenen les dades del registre de comandament i les tradueix en seqüències basades en la transmissió d'un sol byte. Això ho aconsegueix dividint cada operació de byte en operacions de bit separades, que s'envien al controlador de bit encarregat de gestionar la transmissió real de les dades. Per exemple, per a una lectura d'un sol byte, el controlador de bit rep vuit instruccions de lectura del controlador de byte.

Per tant, el disseny de la unitat de control del mestres I²C es dividirà en dues fases. La primera verificar un controlador de bit donat, i la segona el disseny des de zero del controlador de byte basat en una màquina d'estats.

P3 Sessió 3: Controlador de Bit

Com hem dit el controlador de bit gestiona la transmissió real de cada bit, això implica la generació dels nivells específics per a les senyals *Start*, *Stop*, *repetició d'Start*, així com la temporització controlant les línies SCL i SDA mitjançant les senyals dels buffers de tres estats (veure secció Connexions al Bus de l'Introducció). Per fer-ho, cada operació de bit es divideix en quatre etapes (A, B, C i D), a excepció de *Start* i repetició de *Start* que consten de sis etapes tal i com mostra la **Figura 8**. La transició entre cada estat es controlada per un temporitzador extern.

Objectius

- Verificació a nivell RTL d'una màquina d'estats donada.
- Planificació de testbench.

Material

El material de suport el podeu descarregar del campus virtual:

- `i2c_master_bit_ctrl.v` : RTL dels controlador de bit del I²C.
- `i2c_slave_model.v` : model d'esclau I²C.
- `sys_model.v` : model del sistema, genera el rellotge base i el reset actiu per nivell baix.
- `tb_i2c_master_bit_ctrl.v` : testbench amb bàsic del controlador de bit del I²C.
- `timescale.v` : defineix l'escala temporal pel fitxer on està inclòs.
- Plantilla informe de la pràctica.

Tasques a realitzar

1. Examineu el contingut del fitxer `i2c_master_bit_ctrl.v`, identifiqueu les diferents estructures i comenteu el codi.
2. Examineu el contingut del fitxer `tb_i2c_master_bit_ctrl.v` i comproveu si verifiquen totes les funcionalitats del controlador de bit.
3. Simuleu i verifiquen el controlador de bit (`i2c_master_bit_ctrl.v`).
4. Enumereu i descriviu els errors que heu trobat i com els heu corregit.
5. Contesteu les següents preguntes a l'informe de la pràctica.
 - Perquè la generació del senyal *Start* és de 6 etapes en lloc de 4?
 - Quina és la fórmula que ens dona la freqüència de rellotge o temps de bit?
 - Com s'implementa la detecció de col·lisions? I el *clock stretching*?
Dibuixeu l'esquema RTL de la lògica i mostreu una captura de simulació a nivell RTL que ho mostri com funcionen.
6. Verifiquen que el codi es sintetitzable (sintetitzeu-lo per la FPGA Cyclone IV-E EP4CE22F17C6) per una freqüència de rellotge del sistema de 100 MHz (genereu el fitxer de restriccions temporals bàsic!).

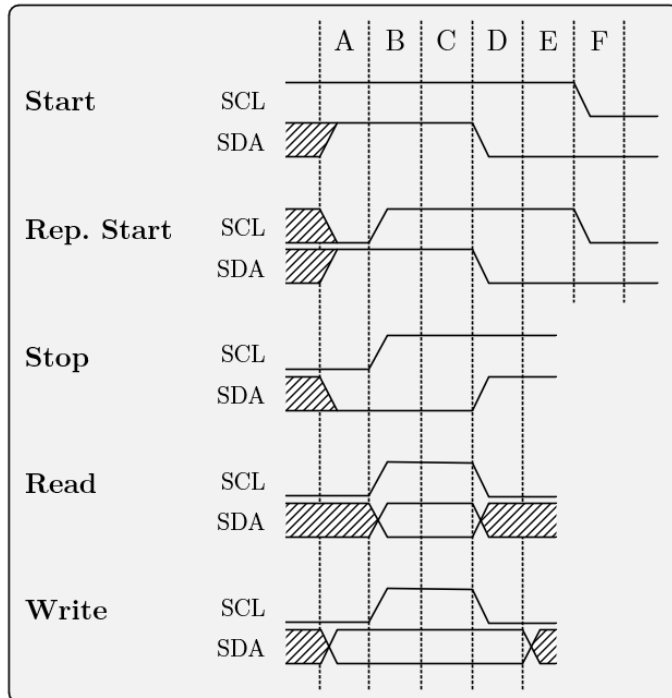


Figura 8 Les cinc operacions possibles de bit i la seva divisió en etapes.

Entrega

Un fitxer ZIP amb el directori de treball:

1. A la carpeta **rtl**: el codi RTL.
2. A la carpeta **tb**: el codi del testbench complet.
3. Dins la carpeta **doc** hi heu de posar l'informe complimentat que trobareu al campus virtual, que constarà de:
 - Verificació funcional: Enumereu i indiqueu les línies de codi que heu modificat perquè funcioni correctament el Controlador de Comandes de Bit.
 - Tipus i explicació de la màquina d'estats i diagrama d'estats (correcte).
 - Responen les qüestions.

P3 Sessió 4: Controlador de Byte

Un cop verificat el controlador de bit, ja tenim tots els mòduls bàsics i podem procedir a dissenyar el controlador de byte que serà el cervell del nostre mestre I²C. El controlador de byte s'encarregarà de gestionar la resta de mòduls en funció de les ordres rebudes a través del bus de dades. Per facilitar-ne la verificació, en aquest punt generarem el nostre fitxer top del nostre mestre I²C, on definirem totes les connexions entre mòduls.

Objectius

- **Disseny i síntesis d'una màquina d'estats** que control l'enviament del byte
- Disseny del top del sistema amb les instàncies dels diferents mòduls del mestre I²C seguint l'arquitectura definida.
- **Auto-Test i verificació del mestre I²C**, reaprofitant tasques i funcions ja desenvolupades.

Tasques a realitzar

1. Determineu les entrades i sortides de la màquina d'estats, en base a l'arquitectura del sistema.
2. Identifiqueu i enumereu la seqüència que ha de seguir per enviar (escriure) un byte i per rebre (llegir) un byte.
Ajut 1: Primer establiu què s'ha d'escriure als registres i en quin ordre per genera cada tipus de transferència (escriptura o lectura).
Ajut 2: Dibuixeu el diagrama temporal de les entrades i sortides de la controlador de byte.
Ajut 3: Enviar l'adreça de 7-bits més el bit de direcció és el mateix que enviar un byte. Es considera una operació d'escriptura.
Ajut 4: Per generar una senyal de *Start*, repetició de *Start* i *Stop* obligeu que estigui definit si la transmissió és d'escriptura o lectura al registre de comandament.
Ajut 5: Ha de ser possible que s'activin el bits *Start*, *Write*, i *Stop* del registre de comandament, i es generi la seqüència correcta completa de transmissió d'un sol byte, el mateix per bit de *Read*.
3. Dibuixeu el diagrama d'estats que implementi la seqüència establerta.
Feu que la ordre de lectura tingui prioritat sobre l'ordre d'escriptura, bits *Read* i *Write* del registres de comandament.

4. Genereu un fitxer anomenat *i2c_master_top* que serà el top del nostre mestre I²C on heu d'instanciar tots els mòduls generats en les sessions anteriors incloent el *i2c_master_byte_ctrl*.
5. Codifiqueu la màquina d'estats que anomenarem *i2c_master_byte_ctrl*.
6. Dissenyau el test que verifiqui la transmissió i recepció de dades el DUT és el *i2c_master_top*.

- Recordeu afegir la descripció Verilog dels buffers de tres estats.

```
assign scl = sclPadEn ? 1'bz : sclPadOut;
assign sda = sdaPadEn ? 1'bz : sdaPadOut;
assign sclPadIn = scl;
assign sdaPadIn = sda;
```

- Per simular les resistències de pull-up afegiu en el test testbench s'utilitza la funció de verilog: pullup nom_instancia (nom_linea).

```
pullup i_P1(scl);
pullup i_P2(sda);
```

- Feu el test utilitzant els registres de configuració i control per realitzar transferències de escriptura i lectura. Primer alliberant el bus (generant senyal **Stop**) i després sense alliberar el bus, és a dir, repetint el senyal **Start** entre operacions.
- Primer comproveu que escriviu i llegiu correctament els registres de configuració i control. *Podeu reutilitzar les tasques dissenyades durant les sessions anteriors.*
- Dissenyau una tasca (*waitEnd*) que monitoritzi el bit d'estat *busy* del registre de control, és a dir, ha d'esperar que el bit *busy* passi de 1 a 0.
- Al campus virtual trobareu un model d'esclau I²C molt senzill anomenat *i2c_slave_model*. Recordeu adjuntar el model als projectes de ModelSim.

7. Creu un fitxer SDC on heu d'especificar:

- Un rellotge del sistema de 100 MHz amb la comanda *create_clock*.
- Els senyals d'entrada i sortida del xip (FPGA/ASIC) tenen un retard associat. Aquest retard és pot especificar en el fitxer de restriccions SDC per tal que les eines de síntesis el tinguin present a l'hora de sintetitzar i de "mapejar" i implementar el nostre disseny. Per fer-ho, s'utilitzen les comandes *set_input_delay* i *set_output_delay*.

```
set_input_delay -clock <clockName> <delay value in ns> [get_ports
<portName>]
```

```
set_output_delay -clock <ClockName> <delay value in ns> [get_ports  
<portName>]
```

La ordre *set_input_delay* especifica el temps d'arribada de dades als ports d'entrada especificats en relació amb el rellotge especificat per l'opció *-clock*. On el rellotge ha de fer referència al nom del rellotge al disseny. Per tant, determinarà el temps disponible per dur la senyal d'entrada des del port fins el registres intern (Tclk - InputDelay). De la mateixa manera l'ordre *set_output_delay* especifica el temps requerit en que dades han d'estar als ports de sortida especificats en relació amb el rellotge especificat per l'opció *-clock*. (Noteu que les comandes tenen més paràmetres que els esmentats. Es recomana consultar l'ajuda del Quartus referent a les ordres TCL.)

En el nostre cas modelitzarem totes les entrades i sortides amb un retard de 2 ns. Per fer-ho heu d'incloure les següents comandes al fitxer SDC, on heu de substituir el <nomClk> pel nom que heu donat al rellotge de 100 MHz que heu definit.

```
set_input_delay -clock <nomClk> 2.0 [all_inputs]  
set_output_delay -clock <nomClk> 2.0 [all_outputs]
```

8. Sintetitzeu i verifiqueu el disseny per FPGA **Cyclone IV E EP4CE22F17C6**. No us oblideu d'incloure el fitxer de SDC. Definiu els pins d'entrada i sortida al GPIO.
9. Comproveu que el disseny està completament definit (no té cap *unconstrained path*) i que compleix els requisits temporals.
10. Simuleu la *netlist* generada amb el mateix fitxer de testbench. Per simular a nivell de porta i veure els retards interns seguiu la guia (Simulació Netlist Quartus) que trobareu al campus virtual.

Entrega

Un fitxer ZIP amb el directori de treball:

1. A la carpeta **rtl**: el codi RTL.
2. A la carpeta **tb**: el codi del testbench complet.
3. A la carpeta **sd**: el fitxer de restriccions temporals (.sdc).
4. Dins la carpeta **doc** hi heu de posar l'informe complimentat que trobareu al campus virtual, que constarà de:
 - Tipus i explicació de la màquina d'estats implementada i diagrama d'estats.
 - Enumerar les tasques del testbench i explicar breument què fan.



- Captures de les simulacions funcionals, amb una explicació breu i ressaltant les zones d'interès.
- Captura del terminal del ModelSim amb els missatges de l'auto verificació.
- Captura del esquema RTL resultant de la síntesi (no oblideu el diagrama d'estats) amb el Quartus i taula on consti freqüència màxima d'operació, i recursos utilitzats de la FPGA Cyclone IV E EP4CE22F17C6.
- Captures de les simulacions post-síntesi, amb una explicació breu i ressaltant les zones d'interès.

Annex Pràctica 3

Màquines d'estat

Una màquina d'estats (en anglès, *Finite State Machine*) és un model que utilitzem per controlar el comportament d'un sistema. En el cas d'aquesta assignatura, utilitzem les màquines d'estat per controlar el comportament de circuits digitals.

En la **Figura 9** es presenta el diagrama de blocs d'una màquina d'estats. Qualsevol màquina d'estats consisteix d'un bloc de lògica combinacional per al càlcul de l'estat futur (*i.e.* definir les transicions d'estats), un bloc de lògica seqüencial per actualitzar l'estat amb l'arribada del flanc de rellotge, i un últim bloc per al càlcul de les sortides generades per la màquina d'estats. Aquest últim bloc pot ser combinacional o seqüencial segons les nostres necessitats.

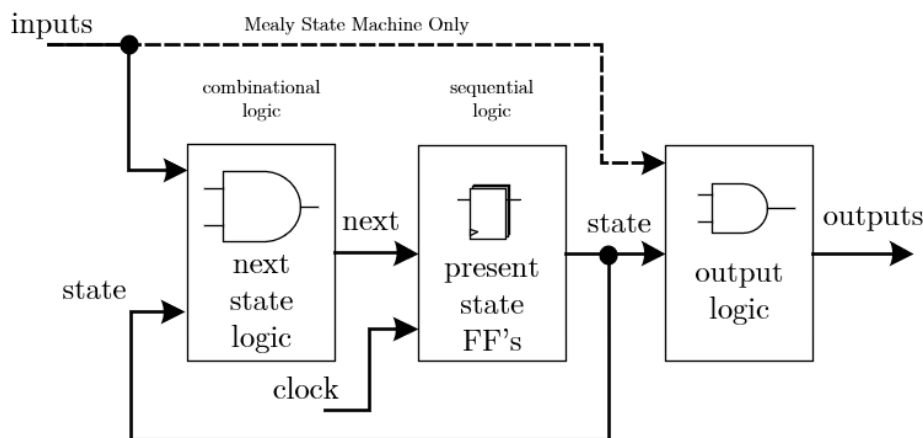


Figura 9. Diagrama de blocs d'una màquina d'estats (imatge extreta de l'article "*State Machine Coding Styles for Synthesis*", SNUG 1998)

Disseny d'una màquina d'estats pel control d'un semàfor

A continuació detallarem els diferents passos per a codificar, en llenguatge Verilog, una màquina d'estats per a controlar un semàfor. Cada bloc descrit en la **Figura 10** el definirem emprant un bloc *always*. Així doncs, tota màquina d'estats codificada en **Verilog** utilitzarà **3 blocs *always***.

Els requisits del sistema són:

- Quan pitgem un botó de la FPGA es genera la senyal "Boto" que estarà a 1 durant un cicle de rellotge i al següent cicle tornarà a 0. Canviarem d'estat quan pitgem el botó (senyal "Boto" igual a 1)
- L'ordre d'il·luminació de LEDs ha de ser verd, groc i per últim vermell, podent repetir aquesta seqüència els cops que es vulgui.
- Rellotge d'1MHz

Diagrama d'estats

La definició del diagrama d'estats és un pas primordial ja que ens permet fixar les diferents variables de la màquina d'estats. És en aquest pas on fem una reflexió sobre les diferents entrades de la màquina d'estats, definim el nombre d'estats, definim les sortides i fins i tot definim si dissenyarem una màquina d'estats de Moore (*i.e.* les sortides només depenen de l'estat actual) o de Mealy (*i.e.* les sortides depenen de l'estat actual i de les senyals d'entrada).

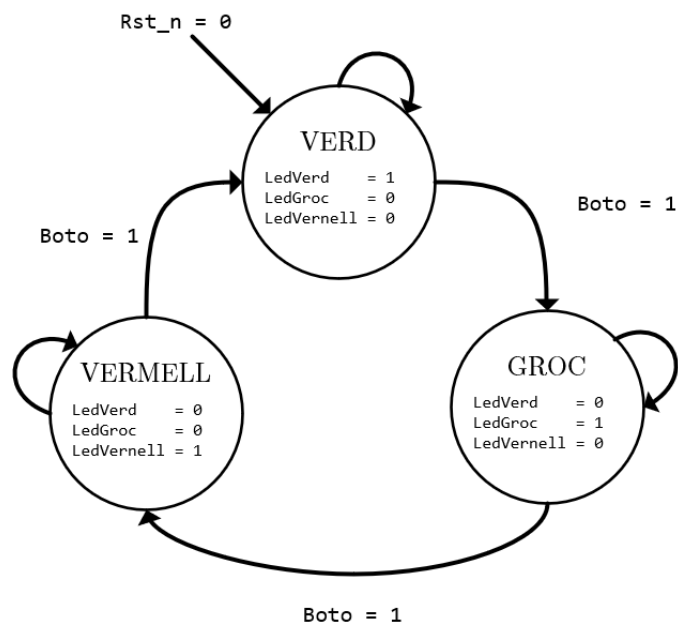


Figura 10 Diagrama d'estats.

En el cas del semàfor, podem resoldre el problema definint **3 estats**, 1 per cada color de LED del semàfor. A més, en aquest exemple només hi ha **una senyal d'entrada**, "Boto", que ens permet passar d'un estat a un altre. Com sabem, el semàfor té 3 colors diferents,

així que definim **3 sortides** independents per a que il·luminin el LED que toca. La senyal d'entrada només ens indica el moment quan fer la transició d'un color a un altre, així doncs, podem definir la màquina d'estats com una **màquina d'estats de Moore**. Tot el que acabem de definir ho podem simplificar com el diagrama d'estats presentat a la **Figura 10**, on cada rodona defineix un estat, on podem veure que a cada estat s'hi defineixen les sortides i podem observar quines transicions entre estats podem realitzar i quina senyal les provoca.

Un cop tenim el diagrama d'estats podem passar a la codificació de la màquina d'estats.

Definim les variables d'estat

Al definir qualsevol màquina d'estats estem definint dues variables d'estat: l'**estat actual** i l'**estat futur**. A ambdues variables se'ls hi assignaran valors dins de blocs *always*, així doncs es definiran com variables tipus registre.

En aquest cas, "state" i "next_state" son variables de 2 bits ja que s'utilitza una codificació binària. Per a aquesta assignatura és suficient utilitzar la codificació binària però n'hi ha d'altres com la codificació *gray*, *one-hot*, etc.

A banda de definir les variables d'estats també es defineixen variables internes. Aquestes es defineixen amb "parameter" i serveixen per assignar un nom que es pugui reconèixer a cada estat. Això ho fem per facilitar la lectura del codi i el seu test, sobretot quan dissenyem màquines d'estats amb un nombre elevat d'estats.

En l'exemple tindriem el següent:

```
// Definició variables d'estat
reg [1:0] state, next_state; // Tenim 3 estats, codificació binària
                                // = > necessitem 2 bits

// Definim variables internes
parameter [1:0] VERD = 2'b00,
                GROC = 2'b01,
                VERMELL = 2'b10;
```

Lògica de transició

La lògica de transició és el bloc combinacional que ens calcula el valor de l'estat futur segons el valor de l'estat actual i el valor de certes senyals d'entrada. És un bloc combinacional, per tant senyals com el rellotge o el *reset* del sistema no han d'aparèixer a la llista de sensibilitat; sí que hi ha d'aparèixer l'estat i aquelles senyals d'entrada que afecten a les transicions. En l'exemple del semàfor, el codi és el següent:

```
// Definim bloc comb. càlcul estat futur
always @(state, Boto) begin
  case(state)
    VERD :   if(Boto)   next_state = GROC;   //definim transicions estat VERD
            else       next_state = VERD;
    GROC :   if(Boto)   next_state = VERMELL; //definim transicions estat GROC
            else       next_state = GROC;
    VERMELL: if(Boto)   next_state = VERD;   //definim transicions estat VERMELL
            else       next_state = VERMELL;
    default: next_state = VERD;             //definim default, no hem definit totes
possibilitats state
  endcase
end
```

Amb la codificació binària emprada podríem definir fins a 4 estats. En aquest exemple en tenim 3, per aquest motiu definim l'estat *default*.

Lògica seqüencial

La lògica seqüencial en la màquina d'estats serveix per actualitzar el valor de l'estat. Així doncs, en l'exemple del semàfor tenim la següent codificació:

```
// Definim lògica seqüencial (estat futur => estat actual)
always @(posedge Clk or negedge Rst_n)
  if (!Rst_n)   state <= VERD;           //en cas de reset anem al VERD
  else         state <= next_state;     //actualitzem l'estat a cada cicle
nelloatge
```

En aquest cas, "state" actualitza el seu valor quan arriba un flanc positiu de rellotge. A més, també definim l'estat inicial de la màquina d'estats en cas de *reset*.

Càlcul de les sortides

Aquest últim bloc s'utilitza per assignar el valor desitjat a cadascuna de les sortides de la màquina d'estats, i és el bloc on es pot comprovar el tipus de màquina d'estats dissenyada (Moore o Mealy). Aquest bloc pot ser seqüencial o combinacional segons les necessitats del sistema. Pel cas del semàfor, definim un bloc combinacional com el següent:

```
// Definim lògica output
always @(state) //no depèn de Clk = lògica combinacional
  case(state) //definim sortides en cada estat
    VERD: begin //Outputs només canviem amb estat => Moore
      LedVerd = 1'b1;
      LedGroc = 1'b0;
      LedVermell = 1'b0;
    end
    GROC: begin
      LedVerd = 1'b0;
      LedGroc = 1'b1;
      LedVermell = 1'b0;
    end
    VERMELL: begin
      LedVerd = 1'b0;
      LedGroc = 1'b0;
      LedVermell = 1'b1;
    end
    default: begin //en default tot a 0
      LedVerd = 1'b0;
      LedGroc = 1'b0;
      LedVermell = 1'b0;
    end
  endcase
```

Totes les sortides depenen exclusivament del valor de la variable "state", així doncs s'ha dissenyat una màquina d'estats de Moore. El codi complet el trobareu a l'arxiu **fsm_semaphore.v** adjunt. També hi trobareu un arxiu de test per a comprovar el seu funcionament.

Pràctica 4: Estació Meteorològica

Durada: 3 sessions

Introducció

Un cop dissenyat i verificat el mestre I²C, l'utilitzarem com a interfície de comunicacions I²C d'una estació meteorològica simple basada en el sensor BME280. El sensor BME280 és un sensor digital combinat d'humitat, pressió i temperatura, que permet monitoritzar les condicions ambientals en temps real.

Objectius

- **Analitzar i combinar mòduls donats per generar un sistema digital complex amb interfície d'usuari, adquisició, post-processat i visualització de dades.**
- **Definir l'arquitectura del l'estació meteorològica, en base a mòduls donats.** Seguint els criteris de disseny adients, tal com l'ús de sincronitzadors, restabliments per encesa (de l'anglès *power-on-reset*), implementació d'IPs (de l'anglès *Intellectual Property*), entre d'altres.
- Definició de restriccions temporals necessàries per el correcte funcionament del sistema (SDC).
- **Implementació en FPGA Cyclone V 5CEBA4F23C7N de la placa de desenvolupament DE0-CV.**

Material

El material de suport el podeu descarregar del campus virtual:

- **Fitxer Zip amb el directori i fitxers pel projecte**
 - **misc:** models funcionals per verificació del sistema complet.
 - **i2c_slave_model.v** : model funcional de esclau I²C, emulant el sensor BME280.
 - **bme280_regs.mem** : contingut de la memòria /registres del esclau i²c
 - **pll_fm.v** : model funcional PLL.
 - **sys_model.v** : model del sistema, genera el rellotge base i el reset actiu per nivell baix.

- **timescale.v** : defineix l'escala temporal pel fitxer on està inclòs.
 - **rtl**: codi font pels diferents controladors necessaris
 - **top_meteo_de0cv.v** : top del sistema a **completar**.
 - **bme280_compensation.v** :
 - **bme280_reader.v** : Control del procés de configuració i lectura del sensor BME280.
 - **bme280_i2c_ctrl.v** : Controlador del mestre i2c que implementa les seqüències per poder escriure i llegir registres del sensor.
 - **digital_por.v** : power on reset digital.
 - **gray_timer.v** : Temporitzador en format gray.
 - **hex2seg.v** : converso Hexadeciman a 7 segments.
 - **bcd2seg.v** : converso BCD (binary code decimal) a 7 segments.
 - **bin2bcd.v** : converso de binari a BCD (Binay Code Decimal)
 - **shifreg.v** : registre de desplaçament.
 - **sync_reg.v** : registre de sincronisme.
 - **sdc** : fitxers de restriccions de disseny.
 - **tb** : fitxers de testbench a **completar**.
- Plantilla informe de la pràctica.

Requisits tècnics de l'estació meteorològica

1. Ha de fer una adquisició de temperatura, pressió i humitat per segon.
2. Les dades es mostraran per les pantalles de set segments, en format BCD (de l'anglès *Binary-Coded Decimal*)
3. L'usuari ha de poder seleccionar quina magnitud (temperatura, pressió o humitat) es mostra per les pantalles, utilitzant els interruptors (**Taula 4**).

Taula 4 Connexions I²C sensor BME280.

SW2	SW1	SW0	Magnitud
x	x	1	Temperatura
x	1	0	Pressió
1	0	0	Humitat

4. Ha de tenir una bandera d'error en el LED0.
5. Ús d'un PLL per generar dos dominis de rellotge diferents: un de 100 MHz que serà el rellotge del sistema, i un de 1 MHz per controlar quan es realitza una nova mesura.
6. Reset extern asíncron actiu per nivell baix (al polsador Key0).
7. Recordeu d'implementar el *power-on-reset* per cada domini de rellotge.
8. El mòdul del sensor BME280 es connectarà al GPIO 1, tal com indica la **Taula 5**.

Taula 5 Connexions I²C sensor BME280.

Senyal	BME PIN	DE0-CV	Observacions
SCL_io	SCK	GPIO_1_D27	
SDA_io	SDI	GPIO_1_D31	
SlaveAddr_LSbit_o	SDO	GPIO_1_D29	Fixe a "1" o "0"
Slavel2C_en_o	CS	GPIO_1_D33	Fixe a "1"
VDD	VIN	VCC3P3	
GND	GND	GND	

9. Recordeu que el bus I²C les línies de rellotge i dades són bidireccionals que requereixen de resistències *pull-up*. Per tant, cal afegir la descripció Verilog dels buffers de tres estats al top del sistema.

```
assign SCL_io = sclPadEn ? 1'bz : sclPadOut;
assign SDA_io = sdaPadEn ? 1'bz : sdaPadOut;
assign sclPadIn = SCL_io;
assign sdaPadIn = SDA_io;
```

Tasques a realitzar

1. Descarregueu-vos i descomprimiu el directori lab4 del campus virtual.
2. Examineu el contingut de les diferents carpetes i fitxers.
3. Enumereu i descriuiu els estats del mòdul *bme280_reader*, i dibuixeu-ne el diagrama d'estats.
4. Feu el diagrama de blocs (no RTL) de l'arquitectura de l'estació meteorològica que implementareu, en base al requisits anteriors.
5. Feu una taula enumerant els mòduls que utilitzeu i la seva funció principal.

6. Genereu l'entitat de primer nivell (top de l'estació meteorològica) en base a l'arquitectura definida. Tant el mòdul com el fitxer s'han d'anomenar ***top_meteo_de0cv***.
Recordeu d'utilitzar sincronitzadors per les senyals que creuen dominis de rellotge.
Recordeu afegir la descripció Verilog dels buffers de tres estats.
Verifiqueu l'adreça de l'esclau del vostre dispositiu.
7. Penseu i llisteu quins tests i simulacions RTL heu de fer per verificar que el sistema funciona. Especifiqueu-los per cada mòdul.
8. Verifiqueu el correcte funcionament de la vostra estació meteorològica simulant a nivell RTL els tests que heu definit abans. Podeu utilitzar el fitxer de test així com els diferents models funcionals que trobareu a les carpetes *tb* i *misc*.
9. Implementeu el disseny a la DE0-CV (Cyclone V 5CEBA4F23C7N):
 - A. Afegiu el fitxer de restriccions temporals (SDC) al projecte i examineu-ne el contingut. Podeu consultar l'ajuda del mateix Quartus per saber que fa cada una de les comandes.
 - B. Actualitzeu el nom de les entrades i sortides del fitxer SDC així com el nom de la instància del PLL.
 - C. Per generar la IP del PLL, heu de buscar a la catàleg d'IPs del Quartus (*Tools > IP Catalog > PLL Intel FPGA IP*).
 - D. Genereu la IP a la carpeta *ips*, amb el nom de *pll_cv.v*, i configurar-lo com es mostra en la **Figura 11**.
 - E. Aneu a la carpeta *ips* i comproveu els fitxers generats. Per instanciar el PLL al vostre disseny utilitzeu el fitxer *pll_cv.v*. Quan l'instancieu poseu la senyal de reset a zero.
 - F. Per fer l'assignació de pins, consulteu el manual de la DE0-CV que trobareu al campus virtual.

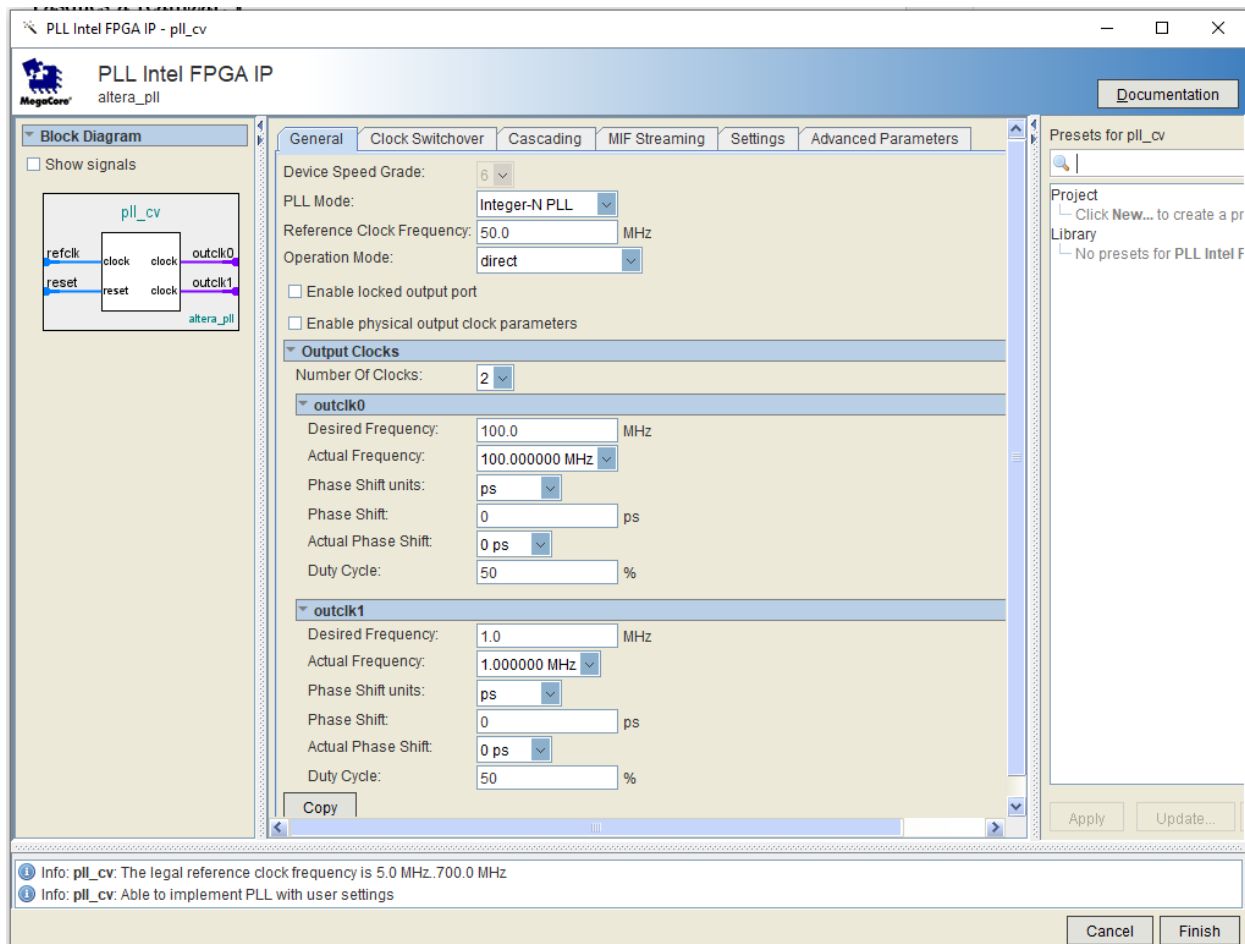


Figura 11 Paràmetres de configuració de la IP del PLL.

10. Per evitar utilitzar resistències de pull-up externes, aprofitarem els recursos dels que disposa la Cyclone V. Els *pads* de propòsit general de la Cyclone-V disposen de resistències de *pull-up* dèbils (25 k Ω) que es poden activar. Seguiu els passos descrits en l'Annex de la Pràctica 4.
11. Comproveu els missatges d'alerta i error així com l'anàlisi temporal, es compleixen els requisits temporals? Quin és el problema? Com es podria solucionar?
12. Connecteu el sensor BME280 a la placa de desenvolupament DE0-CV i comproveu que podeu comunicar-vos amb ell.



Entrega

Fer demostració al professor un cop s'ha implementat a la DE0-CV. En cas de no poder fer la demostració a l'aula, heu de gravar un vídeo que adjuntareu dins la carpeta doc.

Un fitxer zip amb el directori de treball:

1. A la carpeta **rtl**: el codi RTL, amb el *top_meteo_de0cv* i *bme280_reader* comentats.
2. A la carpeta **tb**: el codi del testbench comentat.
3. A la carpeta **sdc**: el fitxer de restriccions temporals (.sdc) comentat.
4. Dins la carpeta **doc** hi heu de posar l'informe complimentat que trobareu al campus virtual, que constarà de:
 - Diagrama de blocs de l'arquitectura de l'estació meteorològica implementada i descripció breu de l'arquitectura, enumerant cada mòdul i descrivint la seva funcionalitat, així com les seves entrades i sortides.
 - Diagrama d'estats del mòdul *bme280_reader*, així com una descripció breu del seu funcionament enumerant els diferents passos que fa fins a realitzar la segona tanda d'adquisició de dades.
 - La llista de tests i les captures de les simulacions RTL corresponents, amb una explicació breu i ressaltant les zones d'interès.
 - Captura (o captures) del esquema RTL generat pel Quartus (expandiu les caixetes) i taula de recursos utilitzats.
 - Preguntes que trobareu a l'informe.

Annex Pràctica 4

Sincronitzadors

Introducció

En qualsevol circuit seqüencial, quan es produeixen violacions de temps de *setup* o de *hold* en un *flip-flop*, aquest entra en un estat on la seva sortida és impredecible. D'aquest estat en diem metastable i es pot propagar per tot el disseny causant pèrdua de dades o un mal funcionament del xip. Els circuits on hi ha creuaments de rellotge, és a dir, el *Clock Domain Crossing* (CDC), són especialment susceptibles a patir metastabilitats, ja que no hi ha ningú que assegurí que es compleixen els temps de *setup* o *hold*.

Per solucionar aquest problema utilitzem sincronitzadors. Els sincronitzadors són circuits utilitzats en disseny VLSI per transferir dades de manera segura i fiable entre dominis de rellotge diferents. El sentit en què es produeix la transferència de dades (del rellotge lent al ràpid o del ràpid al lent) pot tenir implicacions diferents en la metastabilitat i les necessitats del disseny.

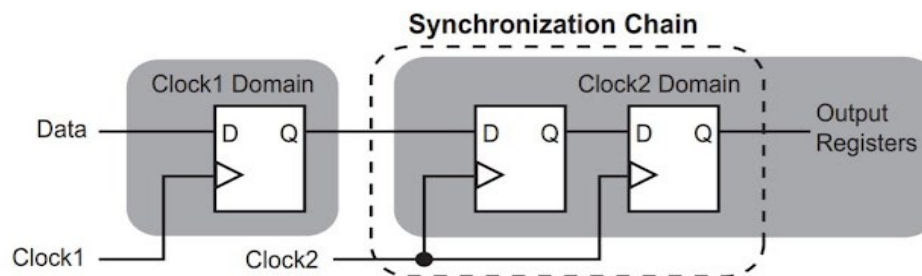


Figura 12 Sincronitzador basat en 2 *flip-flops*.

Com s'implementen en Verilog?

En *Verilog*, es poden utilitzar diferents tipus de sincronitzadors, com ara els basats en flanc de rellotge o els basats en senyal de control. Un exemple bàsic d'un sincronitzador *Verilog* que ens sincronitza una dada d'un domini de rellotge lent (*clk_1*) a un de més ràpid (*clk_2*) és el de la **Figura 12**. La seva implementació en *Verilog* podria ser:

```
module sincronitzador (  
    input wire Clk_1,        // Rellotge del domini lent  
    input wire Clk_2,        // Rellotge del domini ràpid  
    input wire Reset,        // Senyal de reset  
    input wire Data_in,      // Dades d'entrada del domini lent
```

```
output wire Data_out      // Dades de sortida sincronitzades al domini ràpid
);
reg data_sync1;          // Primer registre de sincronització
reg data_sync2;          // Segon registre de sincronització

always @(posedge Clk_1 or posedge Reset) begin
    if (Reset) begin
        data_sync1 <= 1'b0;
    end else begin
        data_sync1 <= Data_in;
    end
end

always @(posedge Clk_2) begin
    data_sync2 <= data_sync1;
end

assign Data_out = data_sync2;
endmodule
```

En aquest exemple, s'utilitzen dos registres de sincronització (*data_sync1* i *data_sync2*). Les dades d'entrada s'agafen en el primer registre mitjançant el rellotge lent (*clk_1*). Després, es traslladen al segon registre amb el rellotge ràpid (*clk_2*). La sortida sincronitzada (*data_out*) s'obté del segon registre. Aquesta estructura de doble registre garanteix que les dades es transmetin de manera segura i redueixen el risc de metastabilitat quan es passa de domini de rellotge lent a ràpid.

Habilitació de *Pull-Ups* interns

Hi ha dues maneres d'habilitar les resistències internes de pull-up al Quartus. No obstant això primer cal habilitar l'opció de *Weak Pull-Up Resistor* al nostre projecte de Quartus. Per fer-ho cal anar al menú *Assignments-> Settings* des de la finestra principal (o premeu Ctrl+Shift+E).

A la finestra emergent, seleccioneu del menú de l'esquerra l'opció *Compiler Settings* i seguidament premeu al botó *Advanced Settings (Fitter)* que trobareu a la part dreta. A la finestra emergent trobareu un llistat d'opcions. Busqueu l'opció *Weak Pull-Up Resistor* i activeu-la seleccionant *On*.

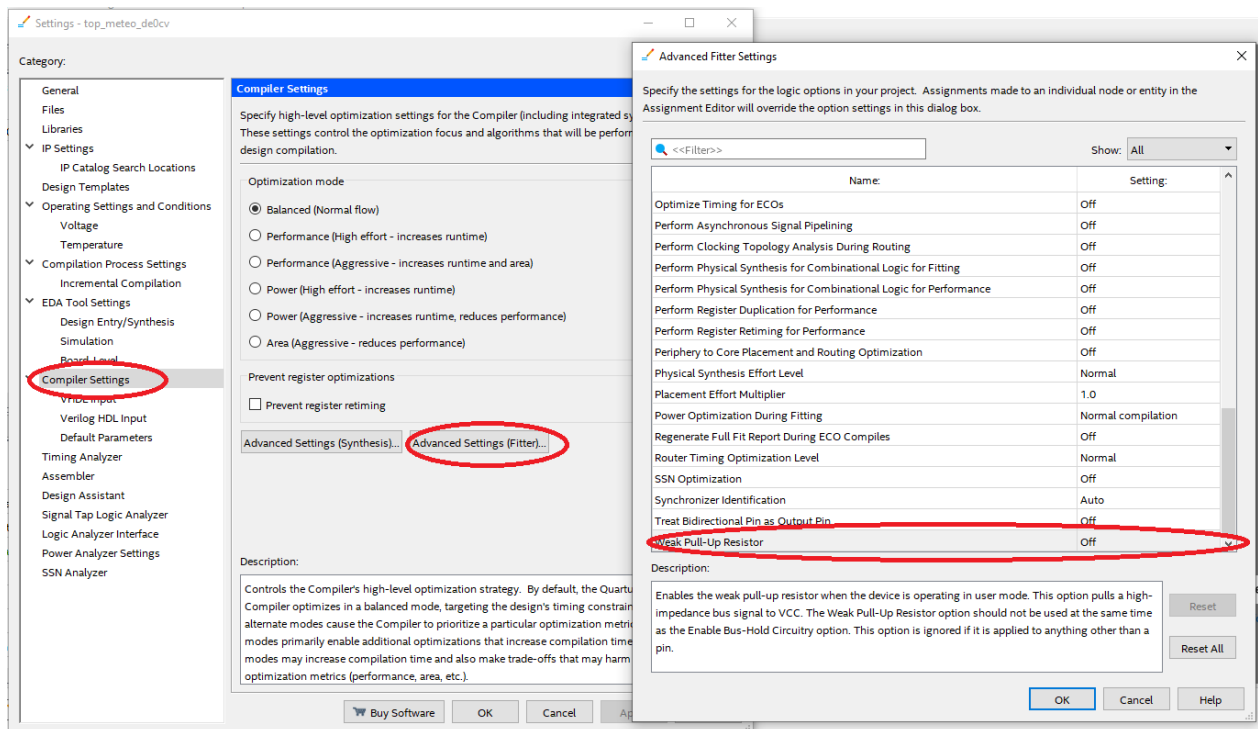


Figura 13 Habilitació de l'opció *Weak Pull-Up Resistor* al projecte de Quartus.

Un cop activada aquesta opció i durant l'assignació de pins podem habilitar els *pull-up* pels senyals del bus I²C *SCL_io* i *SDA_io* mitjançant el *Pin Planner* o l'*Assignment Editor* del Quartus.

Pin Planner

Des de la finestra principal del vostre projecte, aneu al menú *Assignment > Pin Planner* (o premeu Ctrl+Shift+N). A la finestra de planificació de pins, a la vista de *All Pins* a la part inferior, feu clic amb el botó dret del ratolí a qualsevol capçalera de columna i seleccioneu *Customise Columns*. A la finestra emergent, desplaceu-vos cap avall al costat esquerre i trobeu l'opció *Weak Pull-Up Resistor*. Seleccioneu-la i afegiu-la a les columnes visibles pitjant la bot de la fletxa cap a l adreta. Guardeu els canvis clicant al boto *Ok*.

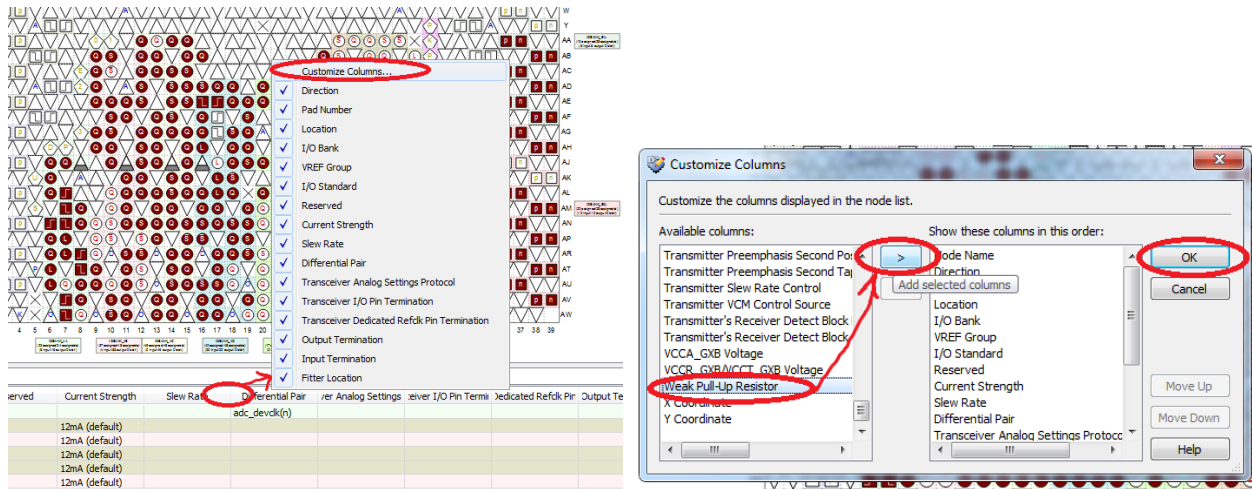


Figura 14 Personalització de les columnes del Pin Planner.

A la finestra de planificació de pins, hauríeu de veure una nova columna anomenada *Weak Pull-Up Resistor*. Per a cada pin que vulgueu activar el *pull-up*, simplement feu clic a la casella de la columna i seleccioneu *On*. Per defecte, està desactivada (en blanc que és equivalent a l'opció *Off*).

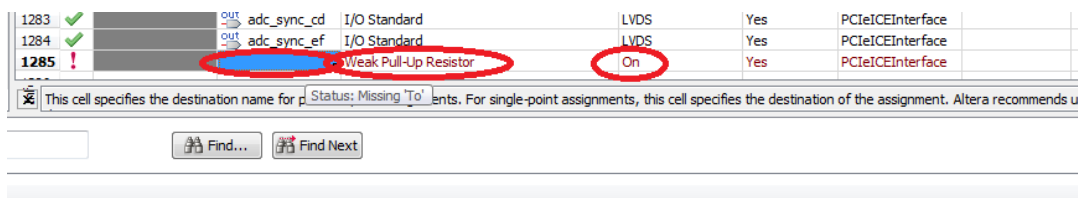
Pin	Output Termination	Input Termination	Fitter Location	Weak Pull-Up Resistor
		Differential	PIN_AR8	
			PIN_AG28	
			PIN_AJ10	
			PIN_AL29	
			PIN_AM29	Off
			PIN_AL12	On
			PIN_AK12	

Figura 15 Columna de per activar el *pull-up* intern.

Assignment Editor

Obriu l'eina de l'editor d'assignacions a la finestra principal anant al menú *Assignment > Assignment Editor* (o premeu *Ctrl+Shift+A*).

A la part inferior de la llista d'assignacions, hi ha una fila on totes les entrades són *<<New>>*. Feu clic a la columna *Assignment Name* i seleccioneu *Weak Pull-Up Resistor*. Després, a la columna *Value*, seleccioneu *On*. Finalment, a la columna *To*, introduïu el nom del pin (que pot incloure el caràcter comodí * per seleccionar busos sencers o fins i tot totes les senyals d'entrada i sortida del nostre projecte).



1283	✓	OUT	adc_sync_cd	I/O Standard		LVDS	Yes	PCIEICEInterface	
1284	✓	OUT	adc_sync_ef	I/O Standard		LVDS	Yes	PCIEICEInterface	
1285	!		Weak Pull-Up Resistor			On	Yes	PCIEICEInterface	

This cell specifies the destination name for pin. Status: Missing 'To' field. For single-point assignments, this cell specifies the destination of the assignment. Altera recommends u

Find... Find Next

Figura 16 Activació del *Weak Pull-Up Resistor* per mitja del *Assignment Editor*.