Disseny i implementació de circuits integrats digitals

Assignatura Disseny i Síntesis de Sistemes Digitals

Grau d'Enginyeria Electrònica de Telecomunicacions

Departament d'Electrònica

Martí i Franquès, 1

08028 Barcelona

Oscar Alonso

Raimon Casanova

Angel Diéguez

INDEX

LABORATORI I - IMPLEMENTACIO D'UN PORT RS232 – PART I

1. Introducció

L'objectiu de les tres primeres pràctiques és la d'implementar un port de comunicacions sèrie RS232 en llenguatge Verilog en una FPGA. Degut a la dificultat del sistema, s'ha dividit el seu desenvolupament en diferents etapes:

-implementació del circuit RX/TX RS232 a nivell RTL.

-test a nivell RTL

-implementació en una FPGA.

Aquestes etapes es realitzaran en diferents pràctiques. En aquest primer laboratori es descriurà el circuit RS232 a nivell RTL amb Verilog. Per fer això caldrà entendre les especificacions del circuit, dividir-lo en diferents submòduls, determinar els ports d'entrada i sortida de cadascun d'ells, fer els diagrames ASM de les diferents màquines, dibuixar els esquemes a nivell RTL de cada bloc i finalment la codificació en Verilog. Per la realització d'aquesta pràctica s'utilitzarà l'eina de simulació ModelSim de Mentor Graphics.

2. UART

Una UART (Universal Asynchronous Receiver and Transmitter) és un circuit que envia dades de varis bits a través d'una línia sèrie. Les UARTs s'utilitzen en conjunció amb l'estàndard RS-232. Aquest especifica les característiques elèctriques, mecàniques, funcionals i de procediment de comunicació entre dos equips.

Una UART inclou un circuit transmissor i un de receptor. El transmissor és essencialment un registre de desplaçament que carrega dades en paral·lel i les treu en sèrie (bit a bit) a un ritme específic. Per l'altra banda, el receptor va desplaçant bit d'entrada un a un i reconstrueix les dades enviades. És un registre de desplaçament de càrrega en sèrie i lectura en paral·lel.

El protocol de comunicació és el següent. La línia de transmissió es manté a 1 mentre no s'envia cap dada. La transmissió s'inicia amb un bit d'inici, que és 0, seguit dels bits de dades, d'un bit de paritat opcional, i finalment amb bits d'aturada que són 1. El nombre de bits pot ser 6, 7, o 8. El bit de paritat és opcional i serveix per detectar possibles errors. Per simplificar el disseny no s'inclourà bit de paritat. El nombre de bits d'aturada pot ser 1, 1.5, o 2. Per simplificar el disseny el deixarem a 1. A la **figura 1** es mostra una trama de 8 bits de dades, sense bit de paritat i amb un bit d'aturada.

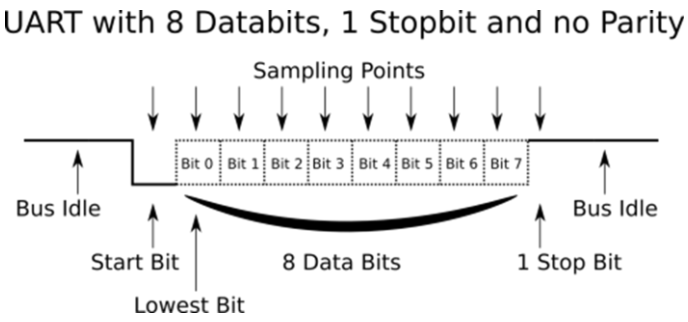


Figura 1.

Com que no s'envia senyal de rellotge a través de la línia sèrie, l'emissor i el receptor s'ha de posar d'acord amb el ritme de transmissió (*baud rate*). Els més comuns són 2400, 4800, 9600, i 19200 bauds.

2.1. Sistema de recepció (uar_rs232_rx)

Com que no s'envia cap rellotge, el receptor ha de generar un rellotge de mostreig a partir del protocol de comunicació establert entre ell i l'emissor (nombre de bits de dades, paritat o no, *baud rate*). La idea és sobre-mostrejar (*oversampling*) el senyal d'entrada amb un rellotge *nBits* cop la freqüència de les dades transmeses. El ritme típic de sobre-mostreig és 16 cops el *baud rate*, el que vol dir que cada bit és sobre-mostrejat 16 cops. En aquest supòsit, el procediment és el següent:

1) esperar que el senyal d'entrada passi a valer 0, inici de recepció del bit d'inici, i després arrancar el rellotge de sobre mostreig (*tick*).
2) comptar 8 cicles del rellotge de sobre-mostreig. En aquest moment som a la meitat del bit.
3) comptar 16 cicles del rellotge de sobre-mostreig i capturar el primer bit.
4) repetir el procediment 3 *nBits* vegades.
5) comptar 16 cicles cicles del rellotge de sobre-mostreig i comprovar que el bit d'aturada és 1. En cas afirmatiu posar el senyal que indica recepció de dada a 1.
6) tornar al pas 1.

El sistema de recepció ha de tenir un senyal d'entrada sèrie RX, un senyal que habiliti el sistema (RxEn), un senyal de sortida que indiqui que s'ha rebut una dada (RxDone), un bus de sortida de 8 bits que contingui la dada rebuda (RxData), un bus d'entrada de 4 bits que indiqui el nombre de bits de dades enviades en cada trama (nBits), un rellotge de sobre-mostreig (tick), el rellotge de sistema (Clk), i el senyal asíncron actiu per nivell baix (Rst_n).

2.2. Generador de rellotge de sobre-mostreig (uar_baudrate_generator)

El circuit generador de rellotge de sobre-mostreig genera una freqüència que és aproximadament 16 cops el *baud rate*. L'afirmació d'aproximadament és degut a que depenent de la freqüència de rellotge a partir de la qual generem el de sobre-mostreig, no serà possible obtenir una freqüència que sigui un múltiple enter del *baud rate*. Per exemple, per obtenir un *baud rate* de 19200, la freqüència de mostreig ha de ser de 307200 (19200x16) commutacions per segon. Si la freqüència de rellotge del sistema és de 50MHz, aleshores el circuit generador de rellotge de sobre-mostreig necessita un comptador mòdul 163 ($50x10^6/307900$)en que és generarà un pols de durada 1/50MHz cada 163 cicles del rellotge de sistema. En general, l'equació a usar serà:

$$REFCOUNT = \left| \frac{f_{clksys}}{16 * baudrate} \right|$$

on f_{clksys} és la freqüència de rellotge del sistema.

2.3. Sistema de transmissió (uar_rs232_tx)

El sistema de transmissió és simplement un registre de desplaçament on a la dada a transmetre caldrà afegir-li els camps del bit d'inici i dels bits d'aturada. El sistema ha de tenir un senyal de sortida sèrie TX, un senyal que habiliti la transmissió (TxEn), un senyal de sortida que indiqui que s'ha enviat una dada (TxDone), un bus d'entrada de 8 bits que contingui la dada a transmetre (TxData), un bus d'entrada de 4 bits que indiqui el nombre de bits de dades a enviar en cada trama (nBits), un rellotge de sobre-mostreig (tick), el rellotge de sistema (Clk), i el senyal asíncron actiu per nivell baix (Rst_n).

3. Realització de la pràctica

Per la realització d'aquesta pràctica heu de treballar en parelles però usar dos ordinadors. Cal procedir de la següent manera:

1) Per cadascun dels mòduls heu de:
    a. crear una taula amb les seves entrades i sortides.
    b. determinar els recursos de càlcul necessaris i màquines d'estats.
    c. dibuixar els diagrames ASM de les màquines si n'hi ha.
    d. dibuixar un esquema a nivell RTL
2) Un cop creat l'esquema de cada mòdul cal realitzar la codificació en Verilog. Hi ha 4 mòduls a realitzar:
    a. uar_rs232
    b. uar_rs232_rx
    c. uar_rs232_tx
    d. uar_baudrate_generator

Cada membre de la parella ha de realitzar dos mòduls. Un ha de fer el uar_rs232_rx i l'altre el uar_rs232_tx. Els altres dos us els repartiu.

3) El mòdul uar_rs232 engloba els altres tres mòduls. Determineu els senyals d'entrada i sortida.

En aquesta pràctica no realitzareu el test. Això es farà a la pràctica següent.

LABORATORI II - IMPLEMENTACIO D'UN PORT RS232 – PART II

1. Introducció

L'objectiu d'aquesta pràctica és la verificar la correcta funcionalitat de la UART dissenyada a la pràctica 1. Per realitzar aquesta verificació és procedirà a l'ús de models funcionals.

2. Verificació

Per realitzar la verificació de la UART que se l'anomenarà Design Under Test (DUT) s'hi connectarà els models funcionals d'un circuit controlador de la UART que permet configurar-la, una altra uart amb la que es comunica, un rellotge de 50 MHz, i un botó de *reset*. La **figura 1** mostra un diagrama de blocs del sistema de verificació. Cadascun d'aquests models funcionals disposa d'una sèrie de tasques per emular diferents funcionalitats. Tot seguit s'expliquen les diferents tasques de cada mòdul.

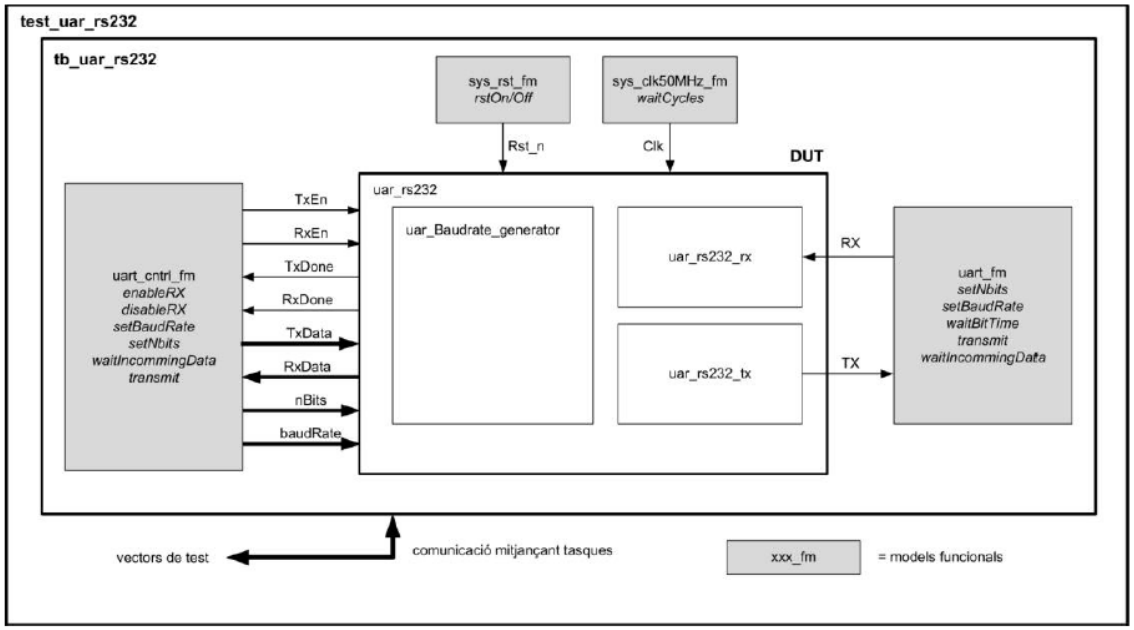


Figura 1

sys_clk50MHz_fm

Aquest mòdul és un model funcional d'una font rellotge configurat amb una freqüència de rellotge de 50MHz. Disposa d'una tasca anomenada waitCycles i que serveix per comptar *N* cicles de rellotge.

sys_rst_n_fm

És un model funcional d'un botó de *reset* actiu per nivell baix. Disposa de dues tasques per tal de fer el *reset* (rstOn) o no fer-lo (rstOff).

uart_cntrl_fm

Aquest és el model funcional d'un controlador de la uart. Permet configurar el nombre de bits i el baud rate mitjançant les tasques setNbits i setBaudRate. També permet activar o desactivar el canal de recepció de dades RX amb les tasques enableRx i disableRx. La tasca waitIncommingData espera que el senyal que indica recepció de dades (RxDone) es posi a 1. En aquest moment captura la dada rebuda (RxData). També disposa una tasca per transmetre una dada mitjançant el canal TX amb la tasca transmit.

uart_fm

És un model funcional d'una uart que pot emetre o rebre dades. El protocol de configuració d'aquest mòdul és pot configurar amb les tasques tasques setNbits i setBaudRate. Noteu que el protocol de comunicació del DUT i d'aquesta uart es poden configurar independentment l'un de l'altre. Ara bé, si volem que es comuniquin entre ells hauran de compartir els mateixos paràmetres. També disposa de dues tasques per enviar (transmit) i rebre dades (waitIncomming Data. Aquesta última l'haureu d'escriure vosaltres.

3. Realització de la pràctica

Descarregueu els fitxers que es troben al campus virtual. Fixeu-vos que hi ha una jerarquia de carpetes:

-fm: conté els models funcionals.

-tb: conté els fitxers de test.

-src: aquí hi heu d'afegir els vostres fitxers de la uart.

-sim: aquí heu de crear el projecte de ModelSim.

-fpga: aquí cal obrir el projecte de Quartus per realitzar la síntesi física.

Copieu tots aquests fitxers al vostre directori de treball. Navegueu pels directoris i doneu-li un cop d'ull als diferents fitxers. Intenteu entendre les diferents tasques i com usar-les. Un cop us hagueu familiaritzat amb elles, llenceu el test i comproveu que el sistema es reseteja.

El següent pas és comprovar que el generador de baud rate funciona correctament. Fixeu-vos que s'ha definit una tasca anomenada baudGeneratorMonitor que mesura el baudrate. Comproveu el correcte funcionament de diferents baudrate mitjançant aquesta tasca.

Un cop hagueu comprovat el correcte funcionament del generador de baud rate procediu a comprovar el canal de recepció RX. Envieu diferents dades a diferents velocitats i amb diferent

nombre de bits per comprovar la correcte funcionalitat amb el model funcional de la uart. Potser útil definir un tasca anomenada test_RXchannel i des d'aquesta crideu a les diferents tasques per realitzar aquest test. Coses a comprovar: que els comptadors de nBits i ticks treballen com s'esperen, comprovar que el senyal RxDone s'activa quan toca, ...

El pas final consisteix en verificar el correcte funcionament del canal transmissor. Envieu diferents dades a diferents velocitats i amb diferent nombre de bits per comprovar la correcte funcionalitat. Potser útil definir un tasca anomenada test_TXchannel i des d'aquesta crideu a les diferents tasques per realitzar aquest test. En aquest cas us farà falta una tasca addicional en el model funcional de la uart, waitIncommingData. Escriviu-la i feu-la servir.

Si us dona temps intenteu implementar el circuit en una FPGA. Heu d'usar el programa Hyperterminal del windows.

LABORATORI III - IMPLEMENTACIO D'UN PORT RS232 – PART III

1. Introducció

En aquesta pràctica és prepararà la UART desenvolupada durant les pràctiques anteriors per tal d'implementar-la en una FPGA. Aquesta pràctica té una durada de 3h i s'avaluarà el resultat al final del laboratori.

En concret sistema s'implementarà en una placa de desenvolupament DE2-115 que es mostra a la **figura 2**. Aquesta placa disposa d'una FPGA Cyclone IV EP4CE115F29C7N, diversos interruptors, leds, displays de 7 segments, memòries SRAM, SDRAM i flash, un lector de targetes SD, un display LCD, un connector ethernet, VGA, RS232, USB i entrades d'audio i vídeo analògics.

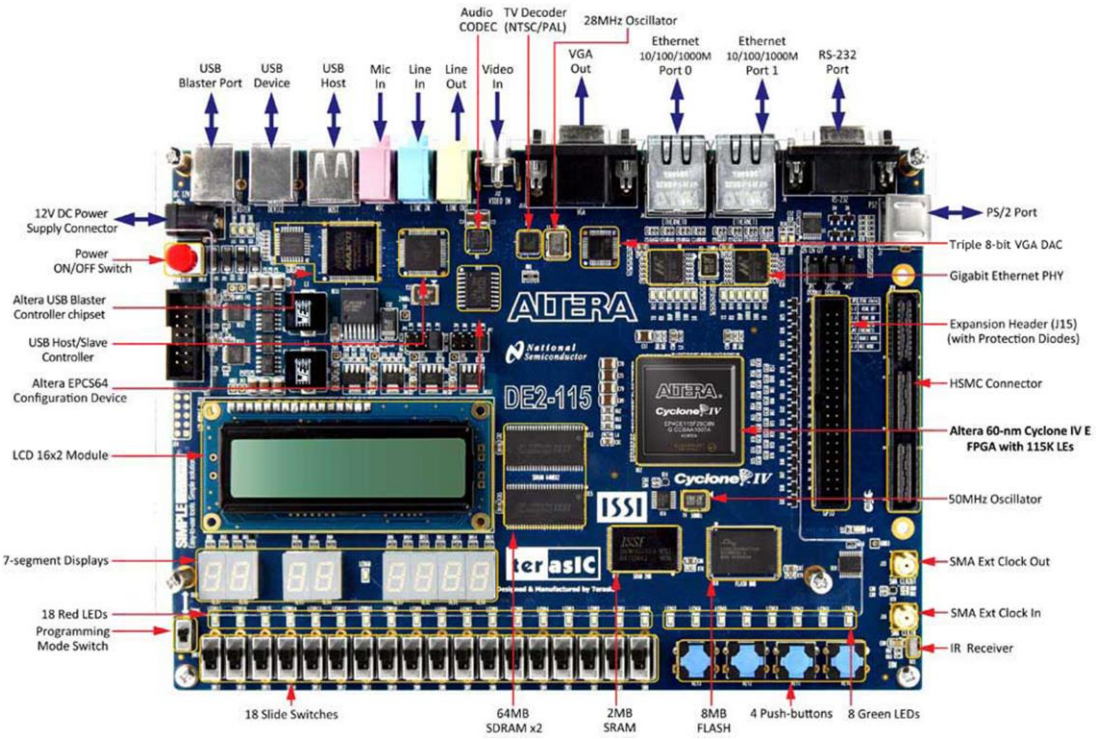


Figura21. **Imatge frontal de la de2-115**

Els objectius d'aquesta pràctica són veure:

- Com preparar el nostre codi per sintetitzar-lo dins d'una FPGA.
- Test del nostre codi sobre la FPGA.
- Familiarització amb el flux d'implementació a una FPGA.

2. UART

El projecte a implementar a la FPGA consisteix en la UART desenvolupada en les pràctiques anteriors a la que hi afegirem els mòduls necessaris per a visualitzar el resultat. Els fitxers a utilitzar els podeu trobar al campus virtual i són:

- Codi amb el mòdul top de la UART (s'ha de modificar per afegir el mòdul "hex2seg")

- Mòduls RX, TX i baud generator

-Mòdul hex2seg. Aquest mòdul transforma 4 bits (en binari) a un resultat que es pugui representar en un display de 7 segments.

3. Realització de la pràctica

Descarregueu els fitxers que es troben al campus virtual. Creeu una carpeta nova on hi posarem tota la pràctica 3. Recordeu seguir l'estructura de carpetes dins la pràctica 3 (codi RTL a la carpeta src i crear la carpeta syn). Dins la carpeta syn és on guardarem el projecte creat amb el Quartus.

1) Obrim el software Quartus. Creem un nou projecte a la carpeta syn de la pràctica 3. El nom del projecte ha de ser el mateix que el nom del mòdul TOP del vostre codi. Hem de configurar el nou projecte per a la FPGA **CYCLONE IV EP4CE115F29C7**.

2) Un cop creat el projecte ens hem de recordar d'afegir els mòduls hex2seg. Els mòduls hex2seg serveixen per representar el resultat que rep el mòdul RX. Com sabeu, les dades rebudes pel mòdul UART RX es troben a DataRX, com aquesta variable és de 8 bits haurem d'afegir 2 mòduls hex2seg, un pels 4 bits més significatius i un pels 4 bits menys significatius.

3) Fent un simple cop d'ull a la FPGA es pot deduir que no hi ha prou "switches" per configurar el nostre mòdul (necessitem 4 bits per Nbits, 8 per la dada a enviar, 16 pel Baudrate, RxEn, TxEn i Rst_n). Per solucionar aquest problema fixarem el valor de Baudrate al codi. Per tant, fixarem el valor de Baudrate i modificarem el codi per tal de que ja no sigui un valor d'entrada.

4) Compileu el projecte. Si no dona cap error observeu els warnings. Podem ignorar-los? Si és així ara mateix tenim el nostre codi sintetitzat per a introduir-lo en una FPGA. L'únic pas que ens queda és relacionar les nostres entrades i sortides amb els pins de la FPGA.

5) Tota la informació relativa a la FPGA es troba al manual d'usuari de la DE2-115 (el fitxer també es troba al campus virtual). Podeu assignar els pins que vulgueu, tot i així recordeu que per facilitar-nos el test hi ha configuracions millors. Per exemple: RxEn l'associem al "switch" 17, Nbits[3:0] als "switches" 11 fins al 8, TxData als "switches" 7 fins al 0, TxEn al botó KEY1 i Rst_n al botó KEY0. Les sortides RxDone i TxDone les associarem a dos LEDs per poder verificar visualment quan s'ha acabat d'enviar i de rebre. RX i TX les assignarem a l'expansion header. Com que utilitzem el nostre codi com a emissor i receptor connectarem externament els senyals TX i RX per fer un test complert del nostre sistema (si algú té alguna senyal per habilitar el clock que recordi configurar-lo per a que el Baudrate estigui funcionant tota l'estona).

Llegint la documentació de la FPGA podreu observar que els botons donen un '1' per defecte i quan els pitgem un '0'. Si heu utilitxat TxEn activa per nivell alt l'haureu d'invertir en el mòdul TOP.

6) Torneu a compilar. Si tot està "ok" podem afegir el nostre codi a la FPGA. Per a fer això hem d'anar a Tools > Programmer.

7) Per a realitzar el test necessitem un company que faci de transmissor/receptor. Recordeu connectar la sortida TX a l'entrada RX. Activeu/desactiveu les entrades d'acord al test que vulgueu realitzar. Si funciona podeu realitzar diferents tests canviant el baudrate (haureu de compilar un altre cop cada cop que el canvieu).

Annex

El software Modelsim serveix per fer simulacions i per tant el seu compilador no és tant complert com el del Quartus (que sí que serveix per implementar hardware). A continuació es llisten una sèrie d'errors típics que es solen cometre i la seva solució (recordeu tornar a simular després de fer qualsevol canvi al codi):

1-) Llista de sensibilitat mal feta: a la llista hi van totes aquelles variables que necessiteu dins el block always. Si teniu quelcom com if (a == NBits) vol dir que NBits també ha d'anar a la llista ja que és una variable externa i no un paràmetre.

2-) Assignacions blocking i non-blocking en un mateix bloc.

3-) Totes les variables s'han de definir en tots els casos, un if sense el seu else infereix latches.

4-) Coses com RxData[count] queden molt bé en programació però això és hardware. Fer això vol dir que deixem llibertat absoluta al compilador per a que posi el necessari per a que funcioni i això no és correcte. Si no enteneu com fer un registre de desplaçament podeu optar per solucions com:

```
case(count)
      4'b0: RxData[0] <= Rx;
      4'b1: RxData[1] <= Rx;
      ....
Endcase
```

D'aquesta manera ens assegurem saber el tipus de decodificació que hi posem.

5-) Controleu sempre el número de bits del contador. Alguns definiu 4 bits per un contador i despréu dieu count <= count + 3'd1;

6-) Fixeu-vos bé en les simulacions. Que aparentment funcionin no vol dir que realment funcionin. Alguns heu fet coses com:

```
READ : begin
      if (count_tick16 < 5'd16) begin
            rstCountTick16_n = 1'b1;
      end else begin
            rstCountTick16_n = 1'd0;
      end
end
```

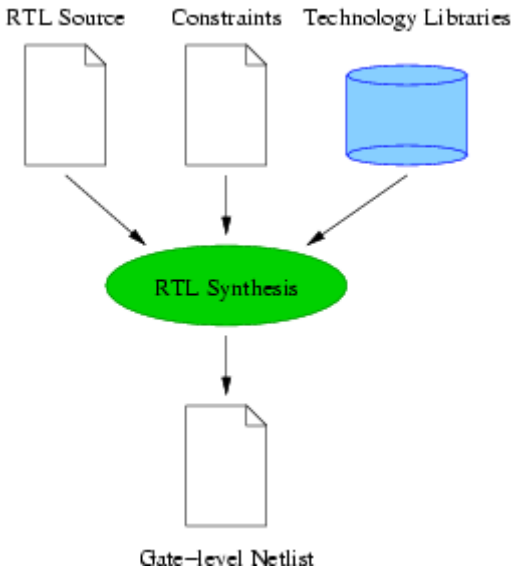
La vostra idea és que quan count_tick16 == 5'd16 el contador es torni a posar a 0. A la simulació segurament veureu el reset i veureu que el contador es posa a 0, el problema és que segurament també veureu que el contador no arriba mai a 5'd16. La solució en el vostre cas és fer un contador cíclic (si el feu de 4 bits cada cop que pasi per 4'd15 el següent número serà 4'd0). En altres sistemes podeu intentar fer un reset del contador amb lògica combinacional.

7-) Una senyal només pot venir d'un bloc (un driver per senyal). La sortida d'un flip-flop no pot ser també la sortida d'un altre bloc (una porta and, or, nand ...o fins i tot un altre flip-flop). Vigileu que la vostra màquina d'estats no estigui posant valors a una senyal a la que també assigneu altres valors en un altre bloc always.

LABORATORI IV – SINTESIS I

1. Introducció

L'objectiu d'aquesta pràctica és la introducció a les eines de síntesi lògica enfocades per ASIC. La síntesi de circuits descrits a nivell RTL és una tasca de disseny automatizada en que un circuit descrit amb un llenguatge de descripció de hardware com Verilog o VHDL és transformat en un circuit descrit a nivell de porta (*netlist*). Les dues representacions són funcionalment equivalents. Una *netlist* és bàsicament una implementació del disseny fet de components de les llibreries (amb portes combinacionals i seqüencials) de la tecnologia amb la que es treballa. L'elecció de portes la realitza l'eina de síntesis lògica en funció de les restriccions, d'àrea, velocitat i consum especificades pel dissenyador. L'eina de síntesi lògica que s'usarà és el DC Compiler de la casa Synopsys. Recordar que aquesta pràctica utilitza els fitxers que es troben al campus virtual i la estructura de carpetes definides pel professor.



El procés de síntesi lògica consisteix en diferents etapes que s'expliquen tot seguit.

Fluxe de síntesi

La síntesi és una tasca complexa que consisteix en varies fases i requereix varies entrades per tal de produir una *netlist* amb la funcionalitat correcte. Els punts basics del procés de síntesi amb Design Compiler es presenten tot seguit. Aquests són:

- Lectura/captura del disseny
- Especificació de restriccions
- Optimització del disseny
- Anàlisi dels resultats
- Guardar resultats i disseny resultant.

Lectura del disseny

La primera tasca en el procés de síntesi és introduir el disseny en el Design Compiler. Aquest procés de lectura consisteix en dues tasques: anàlisi (*analyzing*)i elaboració (*elaborating*) el disseny. El procés d'anàlisi consisteix en:

- llegir el codi font en HDL i cercar errors sintàctics.

- crear una llibreria d'objectes en un format HDL intermedi independent de la tecnologia. Aquesta llibreria es crea per defecte a la carpeta WORK.

Si l'anàlisi falla, els errors s'han de corregir i cal tornar a analitzar el codi HDL. La comanda per analitzar el disseny és *analyze*.

En el procés d'elaboració s'executen els següents passos:

- transforma el disseny a nivell RTL en un disseny basat en equacions i independent de la tecnologia (GTECH) a partir dels fitxers intermedis generats durant l'anàlisi del disseny.
- Permet canviar els valors dels paràmetres definits al codi font.
- Substitueix els operadors aritmètics HDL en el codi per components DesignWare.

Especificació de restriccions

Les restriccions són instruccions que dona el dissenyador a l'eina de síntesi per indicar el que pot fer o no amb el disseny, com s'ha de comportar aquest i quins requeriments en termes d'àrea, temps i potència ha de complir. Hi ha dos tipus bàsics de restriccions:

- Restriccions de regles de disseny: són restriccions que estan definides per la *foundry* de la tecnologia que s'està utilitzant i es troben dins de les llibreries de la tecnologia. No es poden evitar. Restriccions típiques són:
    - Maximum transition time.
    - Longest time allowed for a driving pin of a net to change its logic value .
    - Maximum fanout.
    - Maximum fanout for a driving pin
    - Maximum (and minimum) capacitance
    - The maximum (and minimum) total capacitive load that an output pin can drive. The total capacitance comprises of load pin capacitance and interconnect capacitances.
    - Cell degradation

- Restriccions d'optimització: aquestes són imposades pel dissenyador. Descriuen els requeriments que ha de tenir el disseny (àrea, velocitat, ...) i que l'eina de síntesi ha de tenir en compte. Restriccions típiques són:
    - Definició del rellotge de sistema i definició del retard de rellotge.
    - Camins multi-cicle (*Multicycle paths*).
    - Retards d'entrada I sortida.

- Retard minim I maxim d'un camí.
- Transició d'entrada I capacitate de sortida (*Input transition and output load capacitance*).
- Camins falsos (*False paths*).

És important fer notar que l'eina de síntesi intenta complir tant les restriccions de disseny com les d'optimització. Tot i això, les de disseny tenen preferència sobre les d'optimització pel que pot succeir que alguna restricció d'optimització es violi per tal de poder mantenir les restriccions de disseny.

Definir l'entorn de disseny

També cal definir les condicions d'entorn a les se suposa que ha de treballar el disseny. N'hi ha dues:

- Condicions d'operació: aquestes consideren les variacions en el rang de procés, voltatge, i temperatura (PVT) que s'espera que el disseny es trobi.
- Wire load models: aquests serveixen per estimar les resistències i capacitats paràsites de les línies d'interconnexió abans d'obtenir els valors reals a partir del *layout*. Aquests models són estadístics i extrapolen la longitud de les interconnexions a partir del seu *fanout*.

Optimització del disseny

Aquest pas transforma la descripció GTECH del disseny en una *netlist* de porta utilitzant les cel·les disponibles en les llibreries de la tecnologia. El procés d'optimització es realitza en diverses fases. En cada fase s'apliquen diferents tècniques d'optimització d'acord amb les restriccions del disseny. En el cas del Design Complier es realitzen tres fases:

- Optimitzacions a nivell d'arquitectura:
  - Optimitzacions aritmètiques: l'eina de síntesi utilitza les regles de l'àlgebra de Boole per millorar la implementació del disseny. És a dir, l'eina pot tornar a organitzar les operacions en expressions aritmètiques d'acord a les restriccions per reduir al mínim l'àrea o augmentar la velocitat.
  - Compartició de recursos: l'eina intenta reduir el nombre de recursos de *hardware* fent que diferents operadors en la descripció HDL els comparteixin. Sense compartició de recursos, cada operador en el codi HDL se li associa un recurs de càlcul.
  - Selecció de recursos Designware: això vol dir que es deixa a l'eina de síntesi la implementació de recursos en el codi HDL de la llibreria DesignWare. Per exemple, la llibreria bàsica conté dues implementacions per l'operador + (*ripple* and *carry-lookahead*). En canvi DesignWare en té moltes més. Per tant, si s'usa aquesta llibreria, l'eina seleccionarà automàticament la implementació de l'operador + que millor compleixi les restriccions imposades al disseny.
- Optimitzacions a nivell lògic:

- Estructuració: avalua les equacions de disseny representades a la *netlist* GTECH i tracta, mitjançant l'ús d'àlgebra de Boole, eliminar subexpresions comunes en aquestes equacions. Les subexpresions que han estat identificades i simplificades es poden compartir entre equacions. Per exemple:

Abans d'estructuració	Després d'estructuració
$P = ax + ay + c$	$P = aI + c$
$Q = x + y + z$	$Q = I + z$
	$I = x + y$

- *Flattening*: intenta convertir lògica en una representació de dos nivells (suma de productes). *Flattening* produeix lògica més ràpida (ja que minimitza el nombre de nivells lògics entre les entrades i les sortides) a expenses d'incrementar l'àrea.

- Optimitzacions a nivell de porta: treballa amb la *netlist* GTECH i la mapeja amb les cel·les de les llibreries de la tecnologia per generar una *netlist* a nivell de porta que compleixi les restriccions imposades. L'optimització a nivell de porta consisteix en:
  - *Mapping*:
  - Optimització de retards: corregeix les violacions de temps degudes al procés de *mapping*.
  - Corregir violacions de regles de disseny: bàsicament l'eina de síntesi introdueix *buffers* o canvia portes per poder fer un *driving* correcte. En aquesta etapa es poden produir violacions de temps per tal de poder fer complir les restriccions de regles.
  - Optimització d'àrea: l'eina intenta minimitzar l'àrea canviant unes portes per unes altres de menor àrea sempre i quan no es violin les restriccions.

Escriptura d'informes (*reports*), anàlisi del disseny i salvar el disseny

Un cop que s'ha completat el procés de síntesi, és necessari analitzar els resultats per comprovar que el circuit resultant compleix amb les especificacions requerides.

Finalment, només resta salvar el disseny i la sessió per poder recuperar-la si fos necessària.

2. Arrancar DC Compiler, càrrega de llibreries i síntesi bàsica

El procés de síntesi lògica

Aneu al directori ./dssd07/scripts i editeu el fitxer dc_setup.tcl. Aquest és un fitxer de comandes que seran interpretades per l'eina de síntesi. La funció d'aquest fitxer és la de carregar les llibreries de la tecnologia, llibreries sintètiques de la pròpia eina i especificar alguns paràmetres de com s'ha de sintetitzar el codi en Verilog.

De la línia 1 fins a la 9, s'especifiquen els *paths* on es troben les llibreries necessàries per sintetitzar el circuit. A la línia 12 s'indica quins llibreries s'han de carregar. En aquest cas se'n carreguen 3:

-h35_CORELIB.db: llibreria de *standard cells* (0.35um i 3.3V).

- h35_IOLIB_4M.db: *pads* d'entrada i sortida (0.35 um i 3.3V).

-dw_foundation.sldb: llibreria de recursos de Synopsys.

A la carpeta misc podreu trobar informació sobre aquestes llibreries. Per arrencar DC Compiler, aneu a la carpeta syn i escriviu a la consola:

>design_vision

Important, no poseu & després de la comanda per fer córrer el programa en *backgroud*. Aquest no pot funcionar en aquest mode. Un cop hagi arrencat el programa, us apareixerà la següent finestra:

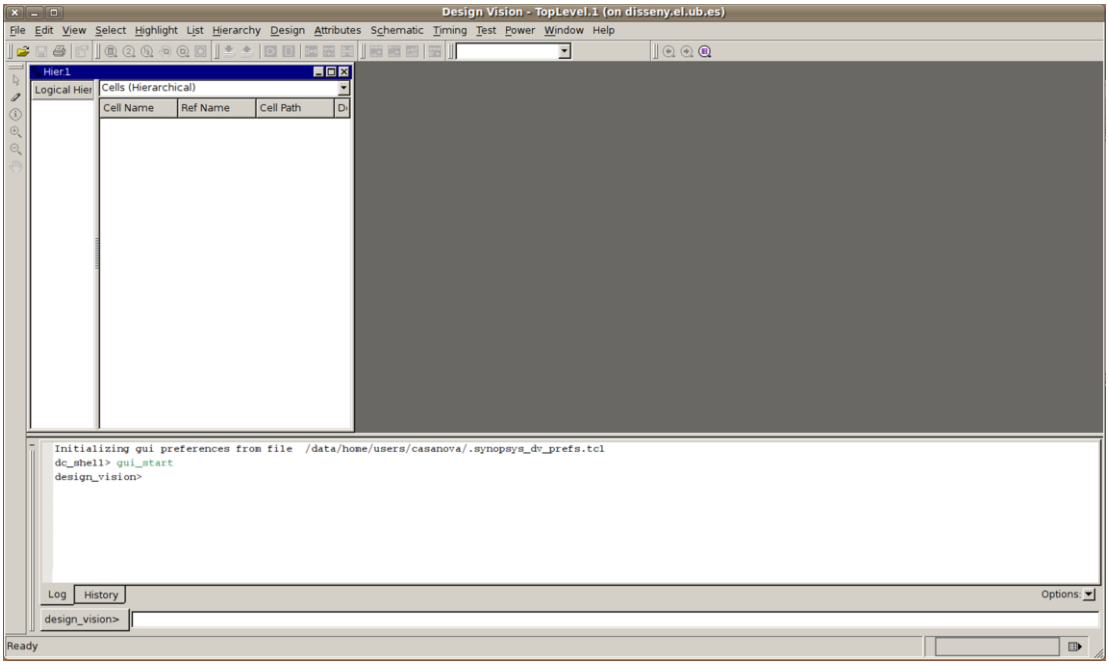


Figura 1.

Per carregar el fitxer dc_setup.tcl aneu a File > Execute Script. Un cop inicialitzada l'eina i amb les llibreries de la tecnologia carregades, cal carregar un disseny descrit a nivell RTL que vulguem sintetitzar. Aneu a Files> Read. S'obrirà la següent finestra:

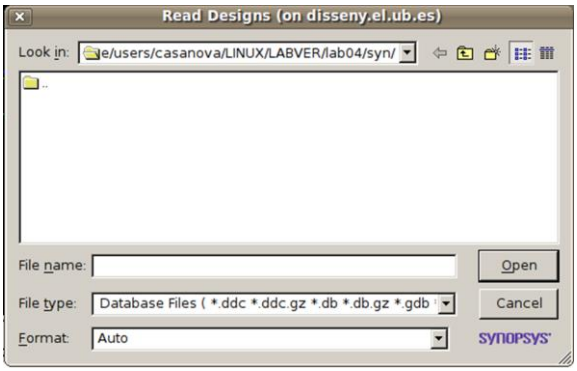


Figura 2.

Aneu a /files/rtl i seleccioneu el fitxer exemple1.v i premeu el botó Open. Automàticament, DC Compiler llegirà el fitxer, comprovarà que no hi ha errors de sintaxi i convertirà la descripció RTL en una *netlist* implementada en una tecnologia genèrica (GTECH). Si premeu el botó  us apareixerà un esquema de la netlist:

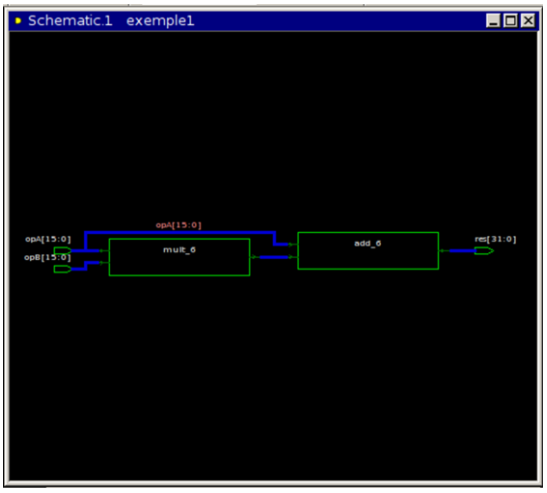


Figura 3.

Editeu el fitxer verilog i compareu el disseny RTL amb la netlist. Què veieu? Què creieu que ha fet l'eina de síntesi? Per a què creieu que serveix GTECH?

Abans de continuar amb el procés de síntesi, analitzarem una mica més en detall que són les llibreries de la tecnologia i per a què serveixen. Per veure les llibreries que s'han carregat escriviu a la consola del Design Compiler:

>list_libs

Apareixeran llistades 5 llibreries, 2 de les quals corresponen a la tecnologia de 0.35 µm CMOS d'AustriaMicrosystems. Per saber el contingut, escriviu:

>report_lib h35_CORELIB.db:c35_CORELIB

Veureu que a la consola apareixerà un munt d'informació. La més important la trobareu al principi de tot. Fixeu-vos amb l'escala de les unitats. Per a què creieu que serveixen les condicions d'operació? Per extreure les condicions d'operació d'una llibreria escriviu:

>report_operating_conditions -library h35_CORELIB.db:c35_CORELIB

Si no s'han especificat cap condició, l'eina de síntesi agafa per defecte la primera que troba llistada dins de la llibreria. En aquest cas WORST_MIL. Per canviar les condicions d'operació es pot usar la comanda:

>set_operating_conditions <condicions d'operació>

Creeu una taula amb les diferents condicions de treball.

Ara convertirem aquesta *netlist* en una tecnologia genèrica en una *netlist* amb la tecnologia desitjada. Aneu a Design > Compile Design i premeu OK (useu els paràmetres per defecte). Un cop acabada la síntesi obriu l'esquema del circuit resultant. Quines diferencies veieu?

Amb la comanda report_area, podeu saber en primer aproximació quina àrea ocupa el circuit. Quan ocupa?

3. Estudi d'arquitectures

L'objectiu d'aquest apartat és la de entendre la importància de les restriccions temporals is les diferents estratègies per aconseguir complir les restriccions de temps.

Per fer aquest laboratori es farà servir un circuit molt simple que realitza la següent operació:

res = opA + opA*opB

on opA i opB són dos operands de 16 bits i res és el resultat de l'operació. Té una longitud de 32 bits.

Ara us heu de moure al directori dssd08. Al directori *./rtl* hi trobareu els següents fitxers:

-timing1.v

-timing2.v

-timing3.v

-timing4.v

Tots els fitxers corresponen a diferents circuits però amb la mateixa funcionalitat. Cada arquitectura és un exemple de les diferents estratègies que es poden seguir per fer complir les restriccions temporals. En tots els casos, la sortida està registrada.

El fitxers de test per la simulació a nivell RTL i a nivell de porta els trobareu a dins de la carpeta *./tb*.

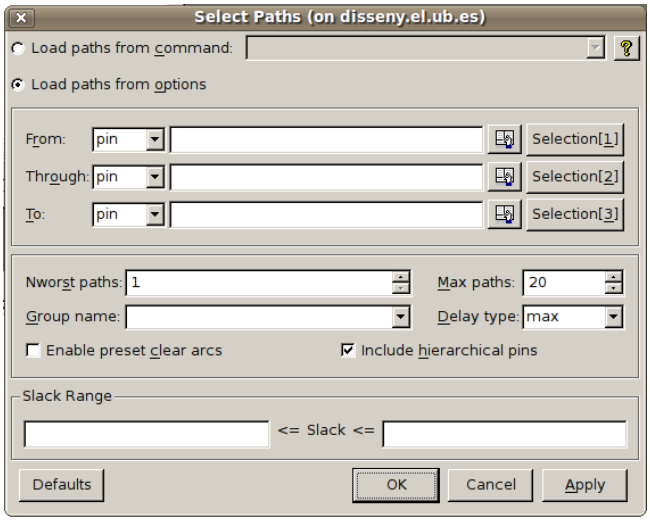
3.1. Arquitectura 1

Aquesta arquitectura és la implementació directa de l'operació desitjada. Editeu el fitxer timing1.v per comprovar-ho. Feu una simulació a nivell RTL del circuit. Aneu al directori *./sim* i a la consola escriviu-hi:

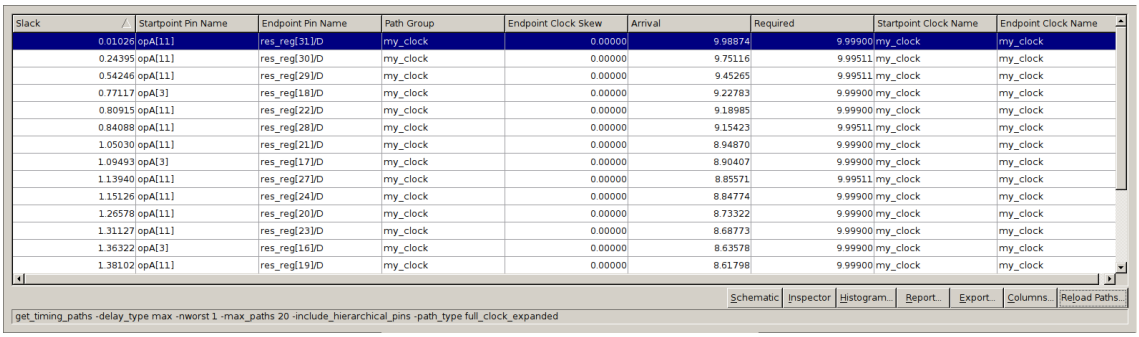
>make sim_gui_rtl

Doneu un cop d'ull a la simulació. Un cop hagueu vist que tot funciona correctament ja posem passar al procés de síntesi. Abans però, editeu el fitxer que imposa restriccions temporals al circuit (*./scripts/constraints.tcl*).Intenteu esbrinar que fa el fitxer. Discutiu-ho amb el professor. Un cop hagin quedat clar els conceptes, aneu al directori de síntesi *./syn* i arranqueu el Design Compiler. Carregueu el disseny timing1.v, el fitxer de restriccions temporals i el fitxer compileSave.tcl.

Un cop sintetitzat el circuit analitzeu els resultats. Primer mirareu si es compleixen les restriccions de temps. Aneu a Timing > Timing Analyzes Driver. Us sortirà la següent finestra.



Aquesta finestra ens permet indicar entre altres coses el número de pitjors (lents) camins qué té el circuit. Per defecte és 20 i ja ens està bé. Premeu el botó OK. Aleshores us sortirà la següent finestra:



Slack	Startpoint Pin Name	Endpoint Pin Name	Path Group	Endpoint Clock Skew	Arrival	Required	Startpoint Clock Name	Endpoint Clock Name
0.01026	opA[11]	res_reg[31]/D	my_clock	0.00000	9.98874	9.99900	my_clock	my_clock
0.24395	opA[11]	res_reg[30]/D	my_clock	0.00000	9.75116	9.99511	my_clock	my_clock
0.54246	opA[11]	res_reg[29]/D	my_clock	0.00000	9.45265	9.99511	my_clock	my_clock
0.77117	opA[3]	res_reg[18]/D	my_clock	0.00000	9.22783	9.99900	my_clock	my_clock
0.80915	opA[11]	res_reg[22]/D	my_clock	0.00000	9.18985	9.99900	my_clock	my_clock
0.84088	opA[11]	res_reg[28]/D	my_clock	0.00000	9.15423	9.99511	my_clock	my_clock
1.05030	opA[11]	res_reg[21]/D	my_clock	0.00000	8.94870	9.99900	my_clock	my_clock
1.09493	opA[3]	res_reg[17]/D	my_clock	0.00000	8.90407	9.99900	my_clock	my_clock
1.13940	opA[11]	res_reg[27]/D	my_clock	0.00000	8.85571	9.99511	my_clock	my_clock
1.15126	opA[11]	res_reg[24]/D	my_clock	0.00000	8.84774	9.99900	my_clock	my_clock
1.26578	opA[11]	res_reg[20]/D	my_clock	0.00000	8.73322	9.99900	my_clock	my_clock
1.31127	opA[11]	res_reg[23]/D	my_clock	0.00000	8.68773	9.99900	my_clock	my_clock
1.36322	opA[3]	res_reg[16]/D	my_clock	0.00000	8.63578	9.99900	my_clock	my_clock
1.38102	opA[11]	res_reg[19]/D	my_clock	0.00000	8.61798	9.99900	my_clock	my_clock

Schematic | Inspector | Histogram... | Report... | Export... | Columns... | Reload Paths...

get_timing_paths -delay_type max -nworst 1 -max_paths 20 -include_hierarchical_pins -path_type full_clock_expanded

En aquesta finestra podeu veure els 20 camins més lents. Indica per cadascun d'ells el *slack*, el punt d'inici del camí i el punt de final del camí, el retard del camí (*arrival*) i el que es desitjaria (*required*). La diferència d'aquests dos valors és el *slack*. Els camins es poden analitzar de moltes maneres. Si premeu el botó *schematic* us pintarà el camí i podreu veure totes les portes lògiques. Amb *report* podreu veure en detall el retard de cada porta del camí. Apunteu el pitjor valor de *slack*.

També és possible calcular l'àrea. Escriviu a la consola:

>report_area

Ara simulareu la netlist generada. Per fer això aneu al directori *./sim* i a la consola escriviu-hi:

>make sim_gui_syn

En principi tot hauria d'haver anat bé. Ara el que fareu és veure el que passa quan es fa treballar el circuit a una freqüència major que 100 MHz. Editeu el fitxer de test i canvieu la freqüència del rellotge a 200 MHz. Torneu a simular, que succeeix?

Recupereu la freqüència de 100 MHz i elimineu els retards de 1 ns de la tasca *operate*. Torneu a simular el circuit. Per què el circuit ha deixat de funcionar?

Aneu un altre cop al Design Compiler i sintetitzeu el circuit per uns períodes de rellotge de 6ns (161MHz), 5.5ns (181.82MHz) i 6ns (200MHz). Anoteu l'àrea i el *slack*.

3.2. Arquitectura 2

Amb el resultats de l'anàlisi temporal estàtic heu vist que el circuït pot treballar fins als 100 MHz. Com podem fer-lo anar més ràpid? Una opció és dividir el *datapath* en dos, de mmanera que l'operació es realitza en dos cicles de rellotge. Aquesta tècnica que s'anomena *pipeline*. En el primer cicle s'executa l'operació opA*opB i es desa en un registre. La sortida d'aquest s'opera al segon cicle es fa la suma.

Obriu el fitxer *timing2.v* i doneu-li un cop d'ull. Després simuleu-lo a nivell RTL. Per fer això cal que modifiqueu el fitxer de test. Comenteu la tasca *test_timming1* i descomenteu la tasca *test_timing2*. També heu de modificar el fitxer *RtlFiles* de la carpeta *./scripts*. Canvieu el fitxer *timing1.v* per el fitxer *timing2.v*. Finalment aneu al directori *./sim* i a la consola escriviu-hi:

>make sim_gui_rtl

Comproveu que el test funciona correctament. Quina és la diferència entre l'arquitectura 1 i l'arquitectura2? Quins canvis han calgut fer al test per tal que aquest funcioni correctament?

Sintetitzeu el circuit. Cal fer els mateixos passos que per l'arquitectura 1. Analitzeu els resultats. Mireu el *time_slack* i apunteu-ne el pitjor valor. Quines diferències veieu amb l'arquitectura 1. Val la pena fer *pipelining*?

Ara sintetitzeu el circuit per una freqüència de 166 MHz (període de 6ns). Anoteu el *time_slack*.

3.3. Arquitectura 3 i 4 (reordenació d'operands)

Aquí reordenem els operands ja que es possible fer:

res = opA + opA*opB = opA*(1+opB)

Sintetitzeu els circuits timing3.v i timing4.v per uns períodes de rellotge de 10ns (100MHz), 6ns (161MHz), 5.5ns (181.82MHz) i 6ns (200MHz). Anoteu l'àrea i el *slack*.

Compareu tots els resultats. Quines conclusions en traieu?

4. Síntesi ràpida d'una màquina que calcula el màxim comú divisor de dos nombres enters de 8 bits.

Ara sintetitzareu una màquina que calcula el màxim comú divisor de dos nombres enters de 8 bits cadascun. Us heu de moure un altre cop al directori dssd07. Els fitxers del circuit els trobareu a *./rtl/mcd*. Hi ha tres fitxers:

-up.v: unitat de procés de la màquina.

-uc.v: unitat de control de la màquina.

-mcd.v: mòdul top que connecta la unitat de procés amb la de control.

Abans de sintetitzar el circuit, simulareu el circuit a nivell RTL. El fitxer de test el trobareu a la carpeta *./tb*. Doneu un cop d'ull al fitxer test_mcd.v per veure que fa. Un cop hagueu esbrinat el funcionament del test ja podeu llançar la simulació. Aneu a la carpeta *./sim* i escriviu:

>make sim_gui_rtl

Quan hagi aparegut l'eina de simulació simvision, afegiu els senyals del bloc mcd al visualitzador d'ones. Llanceu la simulació. Doneu un cop d'ull a la consola, al visualitzador d'ones i analitzeu els resultats.

Ara passem a fer la síntesi lògica del circuit. Per fer-la usarem els següents scripts:

-load_files.tcl: carrega el disseny RTL.

-constraintsMcd.tcl: imposa restriccions de temps.

-compileSave.tcl: compila el disseny i salva la netlist.

Per realitzar el procés executeu cadascun d'aquests fitxers en l'ordre en el que estan llistats. No us oblideu d'executar primerament el fitxer dc_setup.tcl si heu començat una nova sessió del Design Compiler. Un cop finalitzat el procés de síntesi s'han generat una sèrie de fitxers segons indica el fitxer compileSave.tcl. Quins són? Per a què serveixen? Doneu un cop d'ull al fitxer Verilog de la netlist. Compareu aquest amb l'esquema al Deisgn Compiler. Veureu que són equivalents. Ara obriu el fitxer amb extensió .sdf.

Editeu el fitxer que imposa restriccions temporals al circuit (*./scripts/constraints.tcl*).Intenteu esbrinar que fa el fitxer. Un cop sintetitzat el circuit analitzeu els resultats. Primer mirareu si es compleixen les restriccions de temps.

Un cop tenim generada la netlist, la simularem per comprovar que el circuit funciona tal com esperem. Per simular-la cal afegir la descripció en Verilog de les portes lògiques de la tecnologia. És interessant donar-les-hi un cop d'ull. Editeu els fitxers:

/data/soft/cadence/tech/AMS/AMS_4.0/verilog/h35b4/h35_CORELIB.v

/data/soft/cadence/tech/AMS/AMS_4.0/verilog/h35b4/UPD.v

Per simular la netlist aneu al directori ./sim i escriviu a la consola:

>make sim_gui_syn

Quan hagi aparegut el simvision, afegiu els senyals del bloc mcd al visualitzador d'ones. Llanceu la simulació. Veieu alguna diferència amb la simulació a nivell RTL?

LABORATORI V – SINTESIS II

1. Introducció

L'objectiu d'aquesta pràctica és la introducció a les eines de síntesi lògica enfocades per ASIC. Fins ara hem vist la generació de la netlist a nivell de porta, a continuació farem el layout i el tancament del xip emprant la tecnologia H35 d'Austria microsystems. Per aquesta pràctica emprarem un disseny ja fet d'un heater. Aquest consisteix en un disseny mixte on hi ha blocs digitals, entre ells un microcontrollador 8051, que s'utilitzen per controlar unes memòries RAM i uns circuits analògics. Recordar que aquesta pràctica utilitza els fitxers que es troben al campus virtual i la estructura de carpetes definides pel professor.

Fluxe de síntesi

Un cop creada la netlist a nivell de porta (amb el design vision) fem servir el programa encounter per tancar el xip. El flux és el següent:

- Lectura/captura del disseny a nivell de porta
- Lectura de les "LEFs"
- Floorplan
- Power plan
- Placement i Trial Routing
- Clock i Reset tree
- Optimització del disseny i Routing final
- Generació de fitxers i tancament del diseny

Lectura del disseny

Dins de la carpeta de treball (./implement) trobareu un fitxer anomenat heater_v1.conf. Editeu per a la lectura aquest fitxer. En ell hi podeu veure que es criden els següents blocs:

- Codi digital a nivell de porta i un top (heater_v1.v). Aquest últim fitxer conté la informació de connectivitat entre el bloc digital i els blocs analògics, a part de la informació de com es connectaran els pads.

- Llibreries que contenen la informació dels "timings" de les cel·les digitals i de les memòries.

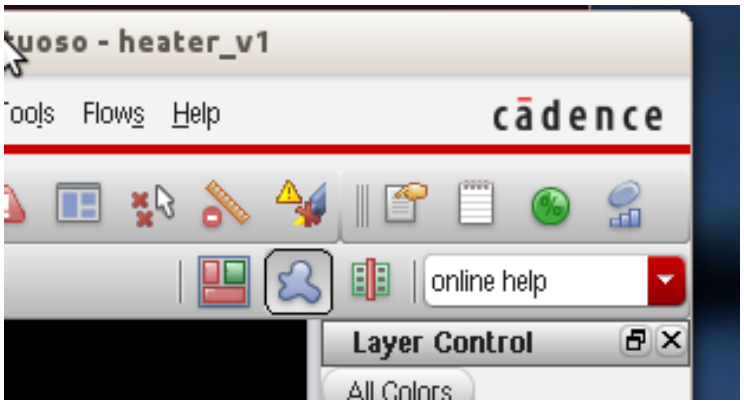
- Arxius "LEF". Aquests arxius contenen la informació a nivell de layout de tots els blocs (tant digitals com analògics), bàsicament indiquen al programa amb quins metalls es pot "routejar" per sobre aquella cel·la.

- Arxiu que conté com estan distribuïts els pads (corners.io).

- Arxiu de constraints

- Definició de les alimentacions

Un cop entès l'arxiu cridem el programa (>velocity). Aquest programa té tres modes de representar diferents: Floorplan view, Amoeba view i Physical view; de moment ens col·locarem en la Floorplan view.



Floorplan

Per a fer el floorplan primer hem de carregar el disseny. Prèviament, la foundry ens diu que hem de córrer una comanda:

> source amsSetup.tcl

Seguidament, anem a File>Import design. Aquí carreguem l'arxiu heater_v1.conf des de l'opció LOAD. Si tot es correcte, veureu que el programa us dibuixa el que sembla un xip amb pads i la resta de blocs (la part digital en rosa). La part "ratllada" del mig del xip és el core i és on aniran totes les cel·les digitals.

Ara el que hem de fer és col·locar cada bloc a on volem que estigui. Considerant que necessitem espai per posar els blocs analògics i les memòries el que fem és reduir l'espai del core. Per això anem a Floorplan> Specify Floorplan. Podeu jugar una estona amb els tamanys del xip. **Què fa cada configuració ?**

Quan considereu que ho teniu, definiu el floorplan per a que sigui igual al de la figura 3.

Figura 3: Dades del Floorplan.

D'aquesta manera podeu col·locar la resta d'elements per a que quedin com a la figura 4 emprant l'eina "move".

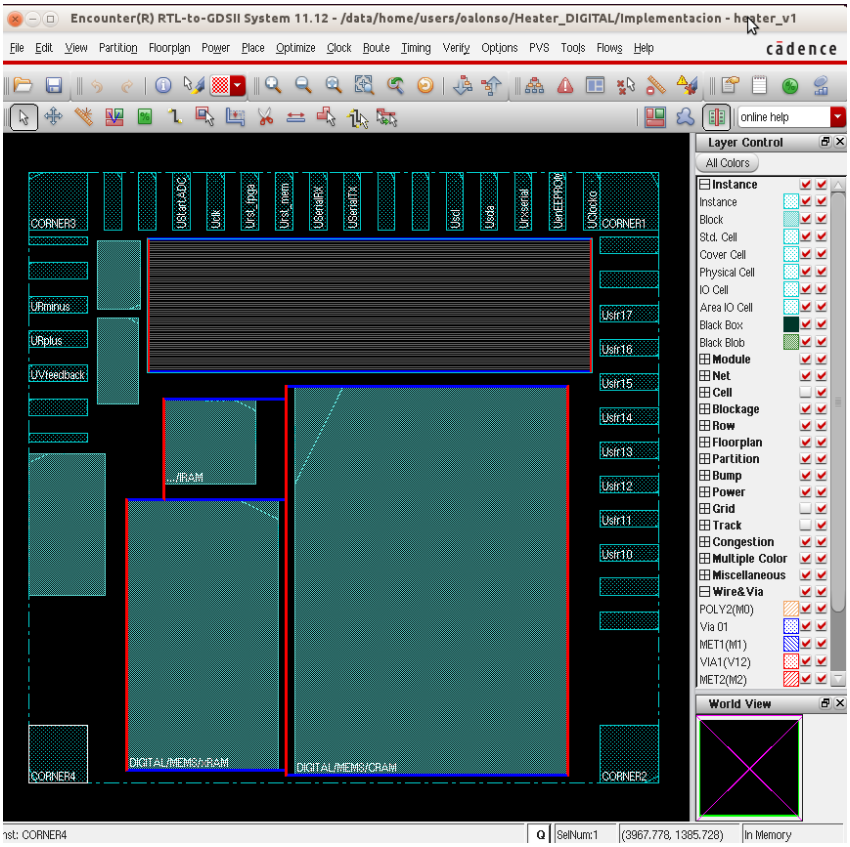


Figura 4 Floorplan final.

Powerplan

Els blocs digitals i les memòries necessiten uns anells de polarització per a connectar les alimentacions. En aquesta etapa anirem a la opció Power> Power Planning> Add ring. Com podeu veure és possible posar anells tant al core (digital) com al voltant dels blocs sel·leccionats. A més, també podeu definir les amplades dels metalls i la separació entre ells. **Quines creieu que són les millors mides ? Per què?**

Quan hagueu acabat el que farem és carregar el floorplan i el power plan d'un disseny ja acabat. A la consola escriurem: >loadFPlan ./guio/heater_v1_PD.fp

Placement i Trial Routing

A continuació el que fem és fer un placement de les cel·les digitals. Abans de fer-ho, des de la foundry ens manen que correm un script abans que ens col·locarà unes cel·les especials per a poder connectar els anells amb el core:

> amsAddEndCaps

Seguidament anirem a Place > Place Standard Cells i li donarem a Ok a la finestra emergent. Un cop finalitzat, en el Physical view, podem veure com ha fet el place de les cel·les digitals en la zona del core. Seguidament anirem a Route> Trial Routing, triem M3 com a metall top i li donarem a Ok per a que finalitzi el trial routing. Fixeu-vos també que les cel·les, com els pads, tenen espais buits entre mig. Aquests buits els omplirem de les anomenades Filler cells més endavant. Aquestes cel·les s'encarreguen de distribuir l'alimentació pels pads i per les cel·les digitals.

Un cop fet el trial Routing és obligatori veure les violacions que tenim al sistema, això es fa des de Timing> Report timing. Tot i així en aquesta pràctica no és necessari.

Clock tree

El clock tree i reset tree es fan per a que les senyals dels clocks i resets arribin al mateix temps a tota la part digital (dins d'un marge d'error). Per a poder-ho fer, el programa insereix unes cel·les especials durant aquesta etapa. És per això que durant la synthesis digital feta amb el design vision vam posar com a constraints que tant el clock com el reset no tinguessin cap cel·la extra en el seu camí.

En aquest exemple només farem el clock tree. El fitxer per a la síntesis del clock es troba a ./scripts/CT.ctstch, editeu-lo per a la seva lectura. Com podeu observar aquest conté les dades mínimes per a fer la síntesis, des del punt on s'ha de fer el clock tree fins a les cel·les que es poden utilitzar. Aquest exemple inclou un generador de clock a la part digital, per tant, farem dos clocks tree, un des de el pad fins a la zona digital i un des de la sortida del generador de clock.

Per a fer la síntesis del clock anirem a Clock > Synthetize clock tree. En la finestra emergent afegim el fitxer que volem utilitzar i li donem a Ok. Un cop acabat torna a ser obligatori reportar quantes violacions de temps de setup i de hold tenim al nostre sistema després del clock tree, ja que si n'hi ha, serà necessari fer una optimització del disseny. Per a fer el càlcul

anirem a Timing> Report timing i demanarem que ens calculi les violacions de setup i hold postCTS. En aquest cas tenim violacions de setup, així que anirem a Optimize > Optimize design i triarem optimitzar post-CTS i el temps de setup. **Quantes violacions de temps hi ha ara?**

Routing

Abans de fer el Routing la foundry ens diu que hem de col·locar les filler cells. Per tant correm les comandes:

> amsFillperi

>amsFillcore

Un cop col·locades les filler cels fem el Routing de les línies d'alimentació. Anem a Route> Special route i triem les nets vdd!, gnd! I psub!. Com a top layer triem M2 i li donem a l'ok. Hauríem de veure que en el core s'han tirat totes les línies d'alimentació però no s'han connectat a l'anell.

Finalment fem el Routing:

> amsRoute

Amb el disseny ja acabat veurem que ens surten unes "X" en blanc, **què volen dir ?**

Quan s'han arreglat tots els possibles errors de timing i de DRC el que es generen són els fitxers que utilitzarem per a importar el disseny i els fitxers amb els retards (SDF) que utilitzarem per la simulació de la part digital. En el nostre cas, un cop importat el disseny al cadence hauríem de fer un pas més degut a que tenim l'anell de pads sobre un dels blocs analògics. Per tant, hauríem de borrar aquells pads i tancar l'anell adequadament.