**DISSENY DIGITAL BÀSIC 2016-2017**

***PRÀCTICA 2: Implementació de circuits combinacionals amb arquitectures estructural i “ifthen”***

L’objectiu d’aquesta tercera pràctica és realitzar la implementació de funcions lògiques utilitzant la filosofia de disseny ***estructural*** i ***ifthen***. La metodologia d’aquest primer disseny (l’estructural) consisteix en construir un sistema digital més o menys complex a partir de components més senzills que realitzen funcions més simples. D’altra banda, la metodologia del disseny ‘***ifthen***’ consisteix en construir clàusules del tipus ‘si succeeix .... amb la variable de entrada, la sortida serà ...’. Es basa, doncs, en la valoració de les diferents situacions que es podem donar en el sistema electrònic i, per tant, és equivalent a escriure la taula de veritat. Això ens servirà, entre altres coses, per poder entendre el comportament d’aquest nou tipus de dispositius i complementar la teoria, comprovant-la.

Dit això, començarem per descriure l’arquitectura estructural considerant la implementació d’una funció lògica simple i després passarem a descriure l’arquitectura *ifthen* posant com a exemple un decodificador de 3 a 8. A la següent pràctica treballarem sobre aquestes noves arquitectures, combinant totes dues.

***Arquitectura estructural***

Com a exemple de la metodologia estructural tenim la realització d’una funció lògica implementada a dos nivells utilitzant components més petits, com són les portes lògiques. Una vegada s’han construït aquests blocs funcionals més simples s’han d’incloure en un sistema superior i connectar-los de forma adequada per que realitzin la funció desitjada. De fet, això correspon a la forma com es va explicant la teoria d’aquesta assignatura.

Fins ara el disseny que hem realitzat s’ha basat en una arquitectura algebraica o lògica. Ara el que volem fer és mostrar que el valor d’una funció lògica, **f**, es pot obtenir connectant sistemes digitals donats per les seves entitats i arquitectures prèviament implementades. Com es mostra a la figura 1, per obtenir la funció **f=/a·b + a·/c** (aquí la barra inclinada la utilitzem per indicar el complementari de **a** o **c**, respectivament) cal que utilitzem un bloc que realitzi la funció suma lògica (OR2) i connectar a les seves entrades les sortides de dos blocs, cadascun dels quals realitzen la funció producte (AND2), alguna entrada dels quals, a la seva vegada, està connectada al ports externs (**a**, **b** i **c**) mitjançant inversors.

La porta AND2 de la part superior del dibuix té com a entrades els senyals del port **b** i el senyal intern **inva**. *És un senyal intern per què, a diferència del tipus de senyals que hem tractat fins ara, no és ni un senyal d’entrada ni un senyal de sortida (externs) i en el llenguatge VHDL cal que introduïm aquest concepte de senyal intern per tal de poder connectar elements internament.* Aquest senyal, a la seva vegada, és la sortida de la funció inversor, que té com a entrada el port **a**. La sortida d’aquesta porta AND2, que torna a ser un senyal intern α, és una de les entrades de la porta lògica OR2.

La porta AND2 inferior està connectada d’una forma similar a la que acabem de descriure per a la porta AND2 superior. Un dels seus dos terminals d’entrada està connectat al port d’entrada **a** i el segon, al senyal intern **invc**. Veiem, per analogia amb el senyal **inva**, que, efectivament, el senyal INVC és un senyal intern i és la sortida d’un component inversor. Finalment la sortida de la porta AND2 inferior torna a ser un senyal intern, β, que és, a la vegada, l’entrada de la porta OR2.



Figura 1. Realització de la funció f de forma jeràrquica (mètode estructural)

A continuació es presenta un codi estructural per fer aquesta funció:

-- Definim primer l’entitat i arquitectura de la porta OR de dues entrades, com s’ha fet a la pràctica anterior.

Entity or2 is

Port (a,b: in bit;

z: out bit);

end or2;

Architecture logicaretard of or2 is

begin

z <= a or b after 3ns;

-- Aquí introduïm un retard en la realització de la funció lògica des de que es presenten els valors de les entrades.

end logicaretard;

-- Definim l’entitat i arquitectura de la porta AND de dues entrades (també feta a la pràctica anterior).

Entity and2 is

Port (a,b: in bit;

z: out bit);

end and2;

Architecture logicaretard of and2 is

begin

z <= a and b after 3ns;

end logicaretard;

-- Definim l’entitat i arquitectura de la porta inversora, feta a la pràctica 1.

Entity inv is

Port (a: in bit;

z: out bit);

end inv;

Architecture logicaretard of inv is

begin

z <= not a after 3ns;

end logicaretard;

-- Aquí comença la definició de la funció lògica que volem implementar. Com qualsevol

-- funció, cal definir entitat i arquitectura. La entitat segueix el patró habitual:

entity Funcio\_logica is

port ( a,b,c: in bit;

f: out bit);

end Funcio\_logica;

-- Aquí definim l’arquitectura logica de la funció que volem implementar, que no és

-- l’objectiu d’aquesta pràctica, però que ens servirà per poder comprovar el funcionament

-- correcte del nostre codi. Aquí l’escrivim sense retard.

architecture logica of funcio\_logica is

begin

f <= ((Not a) and b) or ((Not c) and a);

end logica;

**-- Ara comença la definició de l’arquitectura estructural de l’entitat funcio\_logica,**

**-- que és l’objectiu de la pràctica d’avui.**

architecture estructural of funcio\_logica is

-- Primer, a la part declaratòria de l’arquitectura, introduïm tots els components que volem fer servir.

-- Això es fa d’una forma anàloga a com ho hem fet amb l’arquitectura del banc de proves en les

-- pràctiques anteriors. En aquest exemple seran les portes **or2**, **and2** e **inv**. Només cal que

-- definim UN SOL COP cadascun dels components que farem servir, encara que els fem servir diversos cops.

-- Aquesta situació ja s’ha donat a la part puntuable de la pràctica 2.

component portaand2 is

-- Els noms dels components són arbitraris i no tenen per què coincidir amb el nom de l’entitat.

-- Els noms dels ports del component han de coincidir EXACTAMENT amb els de l’entitat a la qual fan referència.

-- Per tant cal que coneguem exactament els noms d’aquestes entitats.

port(a,b: in bit;

z: out bit);

end component;

component portaor2 is

port(a,b: in bit;

z: out bit);

end component;

component portainv is

port(a: in bit;

z: out bit);

end component;

-- Un cop introduïts els tipus de components que utilitzarem (en aquest exemple, les portes lògiques), caldrà afegir

-- **senyals interns** per tal de poder fer les connexions entre els diferents components. En el nostre cas en

-- són quatre: **inva**, **invc**, **alpha** i **beta**. Fan la funció d’entrades i/o sortides dels components.

signal inva, invc, alpha, beta: bit;

-- Definim ara els diferents dispositius que utilitzarem per implementar la funció. De la figura 1 deduïm que ens calen

-- cinc dispositius: dues portes and2, una porta or2 i dos inversors. Per tant, haurem d’escriure 5 DUTs.

for DUT1: portainv use entity WORK.inv(logicaretard);

for DUT2: portainv use entity WORK.inv(logicaretard);

for DUT3: portaand2 use entity WORK.and2(logicaretard);

for DUT4: portaand2 use entity WORK.and2(logicaretard);

for DUT5: portaor2 use entity WORK.or2(logicaretard);

-- Aquí s’acaba la part declarativa de l’arquitectura. Ara passarem al cos de l’arquitectura.

-- Un cop introduïts tots els dispositius i senyals, passem a realitzar les connexions i, d’aquesta forma, fer la definició

-- de la funció lògica en funció de les variables A, B i C. Això es fa d’una forma anàloga al que hem fet al banc de

-- proves de les pràctiques anteriors. Per cada dispositiu, mirem quins senyals s’han de posar en els seus terminals

-- per poder fer la funció demanada. Haurem de fer servir els senyals definits a l’entitat que volem implementar,

-- funcio\_logica, i els senyals interns. Serà, doncs:

begin

DUT1: portainv port map(A,inva);

DUT2: portainv port map(C,invc);

DUT3: portaand2 port map(inva,B,alpha);

DUT4: portaand2 port map(A,invc,beta);

DUT5: portaor2 port map(alpha,beta,f);

end estructural;

-- Finalment, un cop definida l’entitat i arquitectures del circuit digital que volem implementar, cal que definim

-- l’entitat i l’arquitectura del banc de proves, és a dir, de l’entitat en què es proven que els circuits funcionen

-- correctament sota la presència de senyals externs donats, és a dir, en què es simulen les funcions.

entity bancdeproves is

end bancdeproves;

architecture test\_de\_proves of bancdeproves is

-- Ara introduïm el component que volem testejar, que és el propi circuit. Per a nosaltres aquest component

-- és el circuit digital de la figura 1, és a dir, té tres entrades i una sortida. El nom del component és el que

-- nosaltres vulguem.

-- Fixem-nos que no ens cal tornar a introduir els components que estan dins del circuit a simular, ja que aquests

-- formen part de l’arquitectura de la funció lògica que simulem.

component bloc\_que\_simulem is

port(A,B,C: in bit;

f: out bit);

end component;

--Definim quins són els senyals externs que apliquem o que obtenim com a resultat de la/les funció/ons, que

-- anomenarem **senyalA**, **senyalB**, **senyalC**, **sortida\_f\_logica** i **sortida\_f\_estructural.** De les dues sortides, una

-- correspondrà a la realització funcional i l’altre, a l’estructural. D’aquesta forma podrem veure, simultàniament,

-- les dues realitzacions.

signal senyalA,senyalB,senyalC,sortida\_f\_logica,sortida\_f\_estructural: bit;

for DUT1: bloc\_que\_simulem use entity WORK.funcio\_logica(logica);

for DUT2: bloc\_que\_simulem use entity WORK.funcio\_logica(estructural);

-- Aquí s’acaba la part declarativa de l’arquitectura. Ara passarem al cos de l’arquitectura.

begin

-- Associem les entrades i sortides externes amb els ports que té el component que volem simular i que, tal com ja hem

-- vist, no cal que tinguin el mateix nom. El que si que és important és que l’ordre dels senyals externs sigui el mateix

-- que l’ordre de les variables del component i que no intercanviem entrades i sortides ni que posem més senyals externs

-- al dispositiu que els que estan definits (o menys senyals).

DUT1: bloc\_que\_simulem port map(senyalA, senyalB, senyalC, sortida\_f\_logica);

DUT2: bloc\_que\_simulem port map(senyalA, senyalB, senyalC, sortida\_f\_estructural);

process (senyalA, senyalB, senyalC)

begin

senyalA <= NOT senyalA AFTER 200ns;

senyalB <= NOT senyalB AFTER 100ns;

senyalC <= NOT senyalC AFTER 50ns;

end process;

-- Utilitzar aquesta forma de variar els senyals d’entrada, a més de ser més compacte, té l’avantatge afegit, si

-- s’escriu correctament, que ens permet de fer variar els senyals d’entrada recorrent tots els valors possibles.

-- Ens permet, doncs, de fer l’equivalent a la taula de veritat, presentada a la teoria de l’assignatura.

end test\_de\_proves;

-- A les dues primeres pràctiques es van mostrar dues formes de fer variar els senyals, la que s’indica aquí dalt

-- i la consistent en la instrucció “WAIT FOR xx ns”. Es poden combinar totes dues formes de fer variar els

-- senyals per visualitzar la resposta a senyals periòdics i a senyals no periòdics. Per a això, cal fer dos processos,

-- un per cada tipus de senyal. A continuació es mostra com es podrien substituir les línies en verd del codi

-- que acabem d’escriure:

process (senyalB, senyalC)

begin

senyalB <= NOT senyalB AFTER 100ns;

senyalC <= NOT senyalC AFTER 50ns;

end process;

-- Aquesta ha estat la variació dels senyals periòdics.

process

begin

senyalA <= ‘0’; WAIT FOR 200ns;

senyalA <= ‘1’; WAIT FOR 200ns;

senyalA <= ‘0’; WAIT FOR 200ns;

senyalA <= ‘1’; WAIT FOR 200ns;

senyalA <= ‘0’; WAIT FOR 200ns;

end process;

-- Aquí hem fet la variació arbitrària (en aquest cas, per poder comparar els resultats, s’ha fet periòdica, també).

***Arquitectura ifthen***:

El segon tipus d’arquitectura que tractarem, l’arquitectura ‘***ifthen***’, presenta un disseny molt més simple i directe que l’arquitectura lògica, on podem expressar de forma directa la taula de la veritat de la funció que es realitza. Aquí us exposem el codi per a la implementació d’un decodificador de 3 a 8 bits *active-low* fent servir l’arquitectura ‘***ifthen***’. La paraula d’entrada estarà constituïda per tres senyals tipus bit ordenats en binari natural segons a2, a1, a0, mentre que la paraula de sortida estarà composta per vuit senyals tipus bit ordenats en binari natural segons z7, z6, z5, z4, z3, z2, z1, z0.

-- Realitzem un decodificador 3 a 8 bits   
-- Definim l’entitat  
ENTITY decodificador\_3a8 IS  
PORT (a2, a1, a0: IN BIT; z7, z6,z5,z4,z3,z2,z1,z0: OUT BIT);  
-- Els vectors de bits aquí els numerem de 2 a 0. Això vol dir que el primer

-- bit del vector és el de més pes i l’últim, el de menys pes. Això vol dir

-- que en la representació ens apareixeran en aquest ordre, que és el ordre

-- de pes decreixent cap a avall, que és com representem els números.

-- També ho podríem fer a l’inrevés, del bit de menys pes al de més pes.

-- En aquest cas seria IN BIT\_VECTOR(0 TO 2).

END decodificador\_3a8;

-- Definim una arquitectura que direm tipus ‘ifthen’  
ARCHITECTURE ifthen OF decodificador\_3a8 IS  
BEGIN  
 PROCESS (a2, a1, a0)  
-- Analitzem els canvis a la única variable, que aquí és el vector a. No cal

-- indicar que varia cada component de la variable.

BEGIN  
-- Iniciem el cos de l’arquitectura.

IF a2='0' AND a1='0' AND a0='0' THEN

Z7<='0'; Z6<='1'; Z5<='1'; Z4<='1'; Z3<='1'; Z2<='1'; Z1<='1'; Z0<='1';  
 ELSIF a2='0' AND a1='0' AND a0='1' THEN

Z7<='1'; Z6<='0'; Z5<='1'; Z4<='1'; Z3<='1'; Z2<='1'; Z1<='1'; Z0<='1';  
 ELSIF a2='0' AND a1='1' AND a0='0' THEN

Z7<='1'; Z6<='1'; Z5<='0'; Z4<='1'; Z3<='1'; Z2<='1'; Z1<='1'; Z0<='1';  
 ELSIF a2='0' AND a1='1' AND a0='1' THEN

Z7<='1'; Z6<='1'; Z5<='1'; Z4<='0'; Z3<='1'; Z2<='1'; Z1<='1'; Z0<='1';  
 ELSIF a2='1' AND a1='0' AND a0='0' THEN

Z7<='1'; Z6<='1'; Z5<='1'; Z4<='1'; Z3<='0'; Z2<='1'; Z1<='1'; Z0<='1';  
 ELSIF a2='1' AND a1='0' AND a0='1' THEN

Z7<='1'; Z6<='1'; Z5<='1'; Z4<='1'; Z3<='1'; Z2<='0'; Z1<='1'; Z0<='1';  
 ELSIF a2='1' AND a1='1' AND a0='0' THEN

Z7<='1'; Z6<='1'; Z5<='1'; Z4<='1'; Z3<='1'; Z2<='1'; Z1<='0'; Z0<='1';  
 ELSIF a2='1' AND a1='1' AND a0='1' THEN

Z7<='1'; Z6<='1'; Z5<='1'; Z4<='1'; Z3<='1'; Z2<='1'; Z1<='1'; Z0<='0';

-- Si calgués posar-hi retard s’ha d’afegir al final de la instrucció:

-- IF a2='0' AND a1='0' AND a0='0' THEN

-- Z7<='0' AFTER 5 ns; Z6<='1' AFTER 5 ns; Z5<='1' AFTER 5 ns; Z4<='1' AFTER 5 ns;

-- Z3<='1' AFTER 5 ns; Z2<='1' AFTER 5 ns; Z1<='1' AFTER 5 ns; Z0<='1' AFTER 5 ns;  
 END IF;  
 END PROCESS;  
END ifthen;  
  
-- Realitzem el banc de proves  
ENTITY banc\_de\_proves IS  
END banc\_de\_proves;  
  
ARCHITECTURE test OF banc\_de\_proves IS  
  
COMPONENT el\_meu\_decodificador\_3a8 IS  
PORT (a2, a1, a0: IN BIT; z7,z6,z5,z4,z3,z2,z1,z0: OUT BIT);  
-- Recordeu que els noms de les variables del component han de ser

-- exactament les mateixes que a l’entitat a que es refereixen.  
END COMPONENT;  
  
SIGNAL ent2, ent1, ent0: BIT;

SIGNAL sort7,sort6,sort5,sort4,sort3,sort2,sort1,sort0: BIT;

FOR DUT: el\_meu\_decodificador\_3a8 USE ENTITY WORK.decodificador\_3a8(ifthen);  
  
BEGIN  
  
DUT: el\_meu\_decodificador\_3a8 PORT MAP (ent2,ent1,ent0,sort7,sort6,sort5,sort4,sort3,sort2,sort1,sort0);

PROCESS (ent2,ent1,ent0)

BEGIN

ent0<= NOT ent0 AFTER 50 ns;  
ent1<= NOT ent1 AFTER 100 ns;  
ent2<= NOT ent2 AFTER 200 ns;  
-- Realitzem la variació de les entrades de forma ‘ordenada’, de manera que

-- les variables recorren tots els valors possibles.

-- Així indiquem que la component 0 del vector, la última, és la complementària

-- de si mateixa cada 50 ns. La component 1, la intermitja, cada 100 ns; i la

-- component 2, cada 200ns. D’aquesta manera ens apareixen els senyals d’entrada

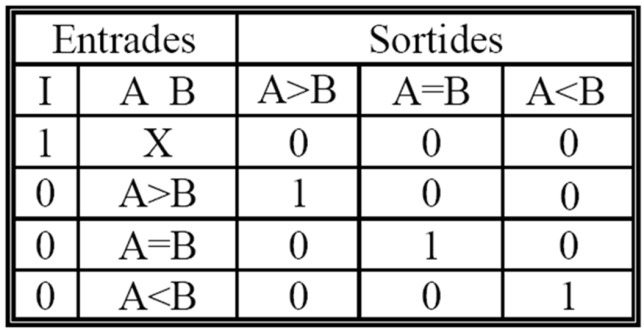
-- representats de manera que podem “llegir”, directament, la taula de veritat

-- de manera ordenada.  
END PROCESS;

END test;

-- Simulem 500ns com a mínim, per poder veure totes les variacions de l’entrada

Un altre exemple el tenim a continuació, a on presentem un comparador de 2 bits, amb senyal enable, i que segueixi la descripció de la taula següent.



Reviseu aquest codi i comproveu el funcionament amb el banc de proves.

ENTITY comparador\_2bits IS

PORT (enable: IN BIT; a1,a0,b1,b0: IN BIT; z2,z1,z0: OUT BIT);

END comparador\_2bits;

-- Definim l'arquitectura tipus 'ifthen'

ARCHITECTURE ifthen OF comparador\_2bits IS

BEGIN

PROCESS (enable,a1,a0,b1,b0)

BEGIN

IF enable='1' THEN Z2<='0'; Z1<='0'; Z0<='0';

ELSE

IF a1='1' AND b1='0' THEN Z2<='1'; Z1<='0'; Z0<='0';

ELSIF a1='0' AND b1='1' THEN Z2<='0'; Z1<='0'; Z0<='1';

ELSIF a1=b1 THEN

IF a0='1' AND b0='0' THEN Z2<='1'; Z1<='0'; Z0<='0';

ELSIF a0='0' AND b0='1' THEN Z2<='0'; Z1<='0'; Z0<='1';

ELSE Z2<='0'; Z1<='1'; Z0<='0';

-- La sortida z2,z1,z0 representa les 3 sortides del comparador: A>B, A=B i A<B.

-- Nosaltres hem definit aquí les sortides de manera que z2 indiqui si A>B;

-- z1 si A=B i z0 si A<B.

END IF;

END IF;

END IF;

END PROCESS;

END ifthen;

ENTITY banc\_de\_proves IS

END banc\_de\_proves;

ARCHITECTURE test OF banc\_de\_proves IS

COMPONENT comparador\_2bits IS

PORT (enable: IN BIT; a1,a0,b1,b0: IN BIT; z2,z1,z0: OUT BIT);

-- Recordeu que els noms de les variables del component han de ser

-- exactament les mateixes que a l'entitat a que es refereixen.

END COMPONENT;

SIGNAL enable: BIT;

SIGNAL ent\_a1, ent\_a0, ent\_b1, ent\_b0: BIT;

SIGNAL sort2, sort1, sort0: BIT;

FOR DUT: comparador\_2bits USE ENTITY WORK.comparador\_2bits(ifthen);

BEGIN

DUT: comparador\_2bits PORT MAP (enable,ent\_a1,ent\_a0,ent\_b1,ent\_b0,sort2,sort1,sort0);

PROCESS

BEGIN

enable<='1'; WAIT FOR 207 ns;

enable<='0'; WAIT FOR 309 ns;

enable<='1'; WAIT FOR 500 ns;

enable<='0';

END PROCESS;

PROCESS (ent\_a1,ent\_a0,ent\_b1,ent\_b0)

BEGIN

ent\_a0 <= NOT ent\_a0 AFTER 50 ns;

ent\_a1 <= NOT ent\_a1 AFTER 100 ns;

ent\_b0 <= NOT ent\_b0 AFTER 200 ns;

ent\_b1 <= NOT ent\_b1 AFTER 400 ns;

-- Realitzem la variació de les entrades de forma 'ordenada' dins de cada

-- paraula però cada paraula varia a ritmes diferents.

-- Per poder utilitzar dos tipus de variacions dels senyals, unes de tipus

-- periòdics i altres de tipus no periòdics, hem de fer dos PROCESS, un per cada

-- tipus, com es mostra en aquest exemple.

END PROCESS;

END test;

**Treball a desenvolupar de forma autònoma:**

**(a entregar 24 hores abans de la vostra corresponent sessió de pràctiques a través de CampusVirtual)**

1. Analitzeu el codi sobre l’arquitectura estructural que es subministra més amunt, intentant entendre el seu funcionament. Comproveu que, efectivament, funciona correctament i que les dues arquitectures escrites descriuen correctament la mateixa funció, una amb retard i l’altre, sense. Raoneu per què es produeix un “rebot” en el senyal de sortida de la arquitectura estructural entre els instants 206 i 209 ns.
2. Implementeu la següent entitat **funcio\_2** de 4 variables d’entrada, **a**, **b**, **c** i **d**, i una sortida **f** tal com està escrita (sense simplificar) i utilitzant 4 arquitectures diferents: ***logica***, ***logicaretard***, ***estructural*** i ***ifthen***:

***f=(/a·b·/c+b·/d+a·c·d+a·/d) XOR (a+/d)***

1. Comenceu per implementar l’arquitectura lògica considerant un retard de 5 ns. Heu de desenvolupar la porta XOR per tal d’expressar la funció en termes de portes AND, OR i NOT, **i no utilitzeu la funció XOR que existeix en el programa QuestaSim per defecte**. Construïu el corresponent banc de proves per a comprovar el seu correcte funcionament (feu que els senyals variïn cada 50ns).
2. Escriviu la taula de la veritat de la funció en un paper. Això us permetrà, per una banda, comprovar el funcionament de l’arquitectura lògica i, per l’altra, definir els diferents casos de l’arquitectura ***ifthen***. Arribats aquí, genereu l’arquitectura ***ifthen*** i introduïu també un retard de 5 ns per cada cas. Incloeu al banc de proves un altre dispositiu (DUT) que funcioni amb aquest arquitectura, conjuntament amb el de l’apartat a), i comproveu que la sortida de totes dues coincideix.
3. Ara escriviu l’arquitectura **estructural**, fent servir les portes lògiques generades a la pràctica 1, utilitzant la seva arquitectura ***logica***. Per tal de comprovar el funcionament correcte, afegiu un altre DUT al banc de proves i comproveu que la sortida coincideix amb la taula de la veritat i amb la sortida de les altres dues arquitectures.
4. Ara modifiqueu l’arquitectura ***estructural*** de l’entitat **funcio\_2** per a fer servir l’arquitectura ***logicaretard*** dels components, per tal d’obtenir el comportament de la funció en aquestes condicions. Comproveu que hi ha diferències amb les arquitectures anteriors i que aquestes diferències no són, exclusivament, un endarreriment de tota la funció de sortida. Justifiqueu aquestes diferències.
5. Ara feu que, al banc de proves, el senyal d’entrada que varia més ràpid ho faci cada 5 ns. Compareu aquest comportament amb el que s’esperaria per la funció lògica que realitza el circuit, que seria el donat a l’apartat ***a)***. Per què són diferents? Penseu-hi i justifiqueu-nos-ho (poseu la resposta com un comentari al codi).

**Treball a desenvolupar al laboratori:**

1. A partir de dos comparadors de 2 bits i les portes lògiques adequades, implementeu de forma estructural un comparador de 2 paraules de 4 bits (de la mateixa forma que al problema 2 del bloc 3). Per a les portes lògiques carregueu el codi **‘portes.vhd’**, i utilitzeu la seva arquitectura **‘logica\_retard’**. Les dues paraules seran **ent\_a3,ent\_a2,ent\_a1,ent\_a0** i **ent\_b3,ent\_b2,ent\_b1,ent\_b0** (4 bits cadascuna), el senyal de control serà un bit **‘enable’** i la sortida seran tres bits sort2, sort1, sort0.Feu el banc de proves **bdp\_comp4**, amb arquitectura **‘test’**.

Pel treball autònom, haureu de pujar 2 fitxers, SENSE COMPRIMIR, que continguin les següents informacions:

* Un fitxer amb les entitats i arquitectures **logica** i **logicaretard** de les portes lògiques que heu implementat a la part prèvia de la pràctica 2 i que heu completat aquí. És a dir, les portes **inv**, **and2**, **and3**, **and4**, **or2**, **or3**, **or4** i **xor2**. El nom del fitxer serà **Py\_xxx\_Cognom1\_Cognom2\_Nom\_portes.vhd** (a on **y** és el número de la pràctica i **xxx** serà el vostre grup de pràctiques, per exemple, D01, o A23, ...). Els codis d’aquest fitxer, amb EXACTAMENT aquests noms d’entitats, arquitectures i terminals, seran necessaris per a les properes pràctiques. Per tant, no utilitzeu noms diferents.
* Un segon fitxer (**Py\_xxx\_Cognom1\_Cognom2\_Nom\_funcio.vhd**) que implementi la funció que s’ha descrit a l’apartat 2 (nom de l’entitat **funcio\_2**) amb les seves arquitectures **logica**, **logicaretard**, **estructural** i **ifthen**. Feu l’entitat **banc\_de\_proves** amb l’arquitectura **test**, amb els senyals d’entrada **ent3**, **ent2**, **ent1** i **ent0** i les sortides, **sort\_logica**, **sort\_logica,** **sort\_estructural** i **sort\_ifthen**. Respongueu les dues preguntes que es plantegen als apartats 2d) i 2e), posant-les com a comentaris en el codi.