**DISSENY DIGITAL BÀSIC 2019-2020**

***PRÀCTICA 2: Implementació estructural d’un sistema digital***

***(dies 8, 11, 12, 13, 14 de novembre)***

L’objectiu d’aquesta segona pràctica és realitzar la implementació de funcions lògiques utilitzant la filosofia de disseny estructural. La metodologia d’aquest disseny consisteix en construir un sistema digital més o menys complex a partir de components més senzills que realitzen funcions més simples. Com a exemple d’aquesta metodologia 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.

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 3 ns;

-- 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 3 ns;

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 3 ns;

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 200 ns;

senyalB <= NOT senyalB AFTER 100 ns;

senyalC <= NOT senyalC AFTER 50 ns;

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 100 ns;

senyalC <= NOT senyalC AFTER 50 ns;

end process;

-- Aquesta ha estat la variació dels senyals periòdics.

process

begin

senyalA <= ‘0’; WAIT FOR 200 ns;

senyalA <= ‘1’; WAIT FOR 200 ns;

senyalA <= ‘0’; WAIT FOR 200 ns;

senyalA <= ‘1’; WAIT FOR 200 ns;

senyalA <= ‘0’; WAIT FOR 200 ns;

end process;

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

**Treball a desenvolupar de forma autònoma:**

**(a entregar 48 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 funció **funció\_2** de 4 variables d’entrada, **a**, **b**, **c** i **d**, i una sortida **f** tal com està escrita (és a dir, sense simplificar) i utilitzant la metodologia lògica i l’estructural descrita anteriorment i sense realitzar la simplificació de la funció:

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

1. Per tal de poder fer això caldrà que utilitzeu les portes lògiques de la pràctica anterior, amb arquitectures ‘**logica**’ i ‘**logica\_retard**’. En aquesta darrera arquitectura, imposeu un retard de 3 ns.
2. Escriviu l’entitat i arquitectura **estructural** corresponent a la funció que heu implementat. Per tal de comprovar el funcionament correcte, primer genereu-vos en un paper la taula de veritat corresponent a la funció. Després, elimineu els retards de tots els components que feu servir (és adir, feu servir inicialment l’arquitectura ‘**logica’**), i comproveu que sota les 16 combinacions de les variables d’entrada la sortida de la simulació coincideix amb la taula de veritat.
3. Ara considereu l’arquitectura retardada ‘**logica\_retard**’ i obteniu el comportament de la funció en aquestes condicions, fent que els senyals externs variïn cada 50 ns. Comproveu que hi ha diferències amb l’apartat b) que no són, exclusivament, un endarreriment de tota la funció de sortida. Justifiqueu aquestes diferències, afegint un comentari en el codi.
4. Ara feu que al banc de proves els senyals d’entrada variïn cada 5ns. Compareu aquest comportament amb el que s’esperaria per la funció lògica que realitza el circuit, que seria el donat a l’apartat d). Per què són diferents? Penseu-hi i justifiqueu-nos-ho (amb un altre comentari).

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\_Cognom1\_Cognom2\_Nom\_portes.vhd** (a on **y** és el número de la pràctica). 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\_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** i **estructural**. 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** i **sort\_estructural**. Respongueu les dues preguntes que es plantegen als apartats 2c) i 2d), posant-les com a comentaris en el codi.

**Recordeu que aquestes entitats, arquitectures i bancs de proves es faran servir en properes pràctiques**