**DISSENY DIGITAL BÀSIC 2016-2017**

***PRÀCTICA 3: Ús de paraules (vectors de dades)***

En aquesta pràctica ens iniciarem a la utilització de paraules de bits (o, tal i com ho considera el QuestaSim, de vectors de dades) com a senyals d’entrada, de sortida o interns i dissenyarem diferents tipus de circuits combinacionals utilitzant la combinació de les arquitectures que ja coneixem (*logica*, *estructural* i *ifthen*). El primer que hem de fer és definir-li al QuestaSim quines són les paraules i de quina mida. Hi ha dos punts fonamentals a on hem de definir aquests paràmetres: a la definició dels ports i dels senyals interns. A continuació posarem un exemple per a una funció amb una sortida de tipus bit i una entrada d’una paraula de 3 bits:

ENTITY funcio IS

PORT(a: IN BIT\_VECTOR (2 DOWNTO 0);

z: OUT BIT);

END funcio;

ARCHITECTURE logica OF funcio IS

BEGIN

z <= ((NOT a(2)) AND a(1)) OR ((NOT a(0)) AND a(2)) AFTER 2 ns;

END logica;

Aquesta funció és la mateixa que ja vàrem implementar a la pràctica anterior, però considerant ara l’entrada com a un vector de tres bits en compte de tres bits individuals. Per tant, el banc de proves corresponent serà molt semblant a l’anterior:

entity bancdeproves is

end bancdeproves;

architecture test\_de\_proves of bancdeproves is

component bloc\_que\_simulem is

PORT(a: IN BIT\_VECTOR (2 DOWNTO 0);

--hem de considerar els senyals d’entrada i sortida de l’entitat a la que ens referirem

z: 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** (paraula de 3 bits) i **sortida.**

signal senyalA: BIT\_VECTOR (2 DOWNTO 0);

SIGNAL sortida: bit;

--Cada tipus de senyal el definim segons sigui un vector o de tipus bit

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

-- 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,sortida);

process (senyalA)

begin

-- la variació dels senyals la podem fer component a component

senyalA(2) <= NOT senyalA(2) AFTER 200 ns;

senyalA(1) <= NOT senyalA(1) AFTER 100 ns;

senyalA(0) <= NOT senyalA(0) AFTER 50 ns;

end process;

end test\_de\_proves;

L’ús de paraules és molt avantatjós per sistemes combinacionals definits amb l’arquitectura ***ifthen***. A continuació presentem el codi per a la implementació d’un decodificador de 3 a 8 bits *active-low* fent servir vectors (és exactament el mateix de la pràctica anterior):

-- Realitzem un decodificador 3 a 8 bits   
-- Definim l’entitat  
ENTITY decodificador\_3a8 IS  
PORT (a: IN BIT\_VECTOR(2 DOWNTO 0); z: OUT BIT\_VECTOR (0 TO 7));  
-- 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 (a)  
-- 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 a="000" THEN z<="01111111";  
 ELSIF a="001" THEN z<="10111111";  
 ELSIF a="010" THEN z<="11011111";  
 ELSIF a="011" THEN z<="11101111";  
 ELSIF a="100" THEN z<="11110111";  
 ELSIF a="101" THEN z<="11111011";  
 ELSIF a="110" THEN z<="11111101";  
 ELSIF a="111" THEN z<="11111110";  
-- Els valors en un vector de bits s’indiquen amb doble cometes, no amb

-- cometes simples, com es fa amb els bits. Aquestes doble cometes no són les

-- que te definides el Word. Si calgués posar-hi retard s’ha d’afegir al final

-- de la instrucció: IF a="000" THEN z<="01111111" 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 (a: IN BIT\_VECTOR(2 DOWNTO 0); z: OUT BIT\_VECTOR (0 TO 7));  
-- 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 vector\_entrada: BIT\_VECTOR (2 DOWNTO 0);  
SIGNAL sortida: BIT\_VECTOR (0 DOWNTO 7);  
-- Com que els dos vectors són de tipus (longituds) diferents, la instrucció

-- SIGNAL és diferent per als dos. Caldrà, doncs, utilitzar diferents

-- instruccions SIGNAL per cada tipus de senyals.  
  
FOR DUT: el\_meu\_decodificador\_3a8 USE ENTITY WORK.decodificador\_3a8(ifthen);  
  
BEGIN  
  
DUT: el\_meu\_decodificador\_3a8 PORT MAP (vector\_entrada,sortida);

PROCESS (vector\_entrada)

BEGIN

vector\_entrada (0)<= NOT vector\_entrada (0) AFTER 50 ns;  
vector\_entrada (1)<= NOT vector\_entrada (1) AFTER 100 ns;  
vector\_entrada (2)<= NOT vector\_entrada (2) 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

**Treball a desenvolupar de forma autònoma:**

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

1. Reviseu el codi anterior i el seu correcte funcionament.
2. Implementeu un comparador de 2 paraules de 2 bits descrites com a vectors (arquitectura ***ifthen***), utilitzant com a paraules d’entrada **‘paraula\_a’** i **‘paraula\_b’**, de 2 bits cadascuna (1 DOWNTO 0), un senyal de control d’un bit **‘enable’** actiu per nivell baix i la sortida, un vector de 3 bits **‘z’** (2 DOWNTO 0). Considereu l’estructura que ja vàreu treballar a la pràctica 2. Poseu el retard de **5ns** a cada condició, segons el model. Feu el banc de proves **bdp\_comp2**, amb arquitectura **‘test’**. Recordeu que teniu a la pràctica anterior el codi d’un comparador de 2 bits, però sense considerar les entrades ni les sortides com vectors.
3. A partir de dos comparadors com els de l’apartat 2 i les portes lògiques adequades, implementeu de forma estructural un comparador de 2 paraules de 4 bits (de forma similar a la pràctica anterior i considerant la implementació que es fa al problema 14 del bloc 2). Per a les portes lògiques carregueu el codi **‘portes.vhd’**, i utilitzeu la seva arquitectura **‘logica\_retard’**. Les dues paraules seran **‘paraula\_a’** i **‘paraula\_b’**, de 4 bits cadascuna (3 DOWNTO 0), el senyal de control serà un bit **‘enable’** i la sortida, un vector de 3 bits **‘z’** (2 DOWNTO 0).Feu el banc de proves **bdp\_comp4**, amb arquitectura **‘test’**. Guardeu aquest codi en el fitxer:

**Py\_xxx\_Cognom1\_Cognom2\_Nom\_comp4bits.vhd**.

Per fer aquesta tasca, necessitem introduir també el codi per poder dividir un vector en diverses paraules més petites que l’original (o fins i tot en *bits*). En el codi adjunt us donem unes pautes de com es faria: cadascuna de les paraules que entrarien al comparador de 4 bits s’haurien de dividir en 2 paraules de 2 bits cadascuna:

ENTITY comparador\_4bits IS

PORT (enable: IN BIT; paraula\_a,paraula\_b: IN BIT\_VECTOR(3 DOWNTO 0); z: OUT BIT\_VECTOR (2 DOWNTO 0));

END comparador\_4bits;

ARCHITECTURE estructural OF comparador\_4bits IS

COMPONENT comparador\_2bits IS

PORT (enable: IN BIT; a,b: IN BIT\_VECTOR(1 DOWNTO 0); z: OUT BIT\_VECTOR (2 DOWNTO 0));

END COMPONENT;

COMPONENT ...

...

SIGNAL alfa1,alfa2,beta1,beta2: BIT\_VECTOR (1 DOWNTO 0);

-- Aquests seran els dos vectors en que dividirem cadascuna de les

-- dues paraules de 4 bits: alfa1 i alfa2 contindran els dos bits

-- de més pes i de menys pes, respectivament, de la paraula a.

-- Anàlogament, per beta1 i beta2 respecte a la paraula b.

SIGNAL ...

-- Els altres senyals que ens calguin ...

FOR DUT1: ...

-- Els dispositius que necessitem ...

BEGIN

alfa1(1)<=a(3);

alfa1(0)<=a(2);

-- Aquesta és la forma d’assignar els dos bits de més pes del vector a

-- als dos bits del vector alfa1.

alfa2(1)<=a(1);

alfa2(0)<=a(0);

-- I aquesta, els dos de menys pes utilitzant concatenació.

beta1<=b(3) & b(2);

beta2<=b(1) & b(0);

-- Igual però per el vector b.

DUT1: ...

-- Els PORT MAP que calguin ...

END estructural;

**Treball a desenvolupar al laboratori:**

* Finalment, Implementeu un sumador modificat de 3 bits a partir d’un sumador d’1 bit, i el seu banc de proves corresponent. A les transparències del tema 6 teniu l’esquema de com fer les connexions per a aconseguir un sumador de *n*-bits a partir d’un sumador d’un bit (els professors de l’aula us faran una petita explicació). D’altra banda, el codi del sumador modificat d’1 bit el teniu a continuació:

--SUMADOR MODIFICAT D'UN BIT

--Aquesta és la entitat del sumador modificat d'un bit que volem implementar

ENTITY sumador\_modificat\_1bit IS

PORT (a,b,cin: IN BIT; suma\_smod\_1bit,cout\_smod\_1bit,aib\_smod\_1bit,aob\_smod\_1bit: OUT BIT);

END sumador\_modificat\_1bit;

-- Aquesta és la definició del sumador modificat de forma estructural

ARCHITECTURE estructural OF sumador\_modificat\_1bit IS

COMPONENT portaand2 IS

PORT(a,b: IN BIT; z: OUT BIT);

END COMPONENT;

COMPONENT portaor2 IS

PORT(a,b: IN BIT; z: OUT BIT);

END COMPONENT;

COMPONENT portaxor2 IS

PORT (a,b: IN BIT; z: OUT BIT);

END COMPONENT ;

--Ens calen 6 DUTs.

FOR DUT1: portaxor2 USE ENTITY WORK.xor2(logica);

FOR DUT2: portaxor2 USE ENTITY WORK.xor2(logica);

FOR DUT3: portaand2 USE ENTITY WORK.and2(logica);

FOR DUT4: portaor2 USE ENTITY WORK.or2(logica);

FOR DUT5: portaand2 USE ENTITY WORK.and2(logica);

FOR DUT6: portaor2 USE ENTITY WORK.or2(logica);

-- Calen 4 senyals interns

SIGNAL sort\_xor,sort\_or,sort\_and1,sort\_and2: BIT;

-- Un cop introduïts tots els blocs 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 cin.

BEGIN

DUT1: portaxor2 PORT MAP (a,b,sort\_xor);

DUT2: portaxor2 PORT MAP( sort\_xor,cin,suma\_smod\_1bit);

DUT3: portaand2 PORT MAP (a,b,sort\_and1);

DUT4: portaor2 PORT MAP( a,b,sort\_or);

DUT5: portaand2 PORT MAP (sort\_or,cin,sort\_and2);

DUT6: portaor2 PORT MAP (sort\_and1,sort\_and2,cout\_smod\_1bit);

-- ara introduïm quins senyals interns s’utilitzen, també, com externs

aib\_smod\_1bit <= sort\_and1;

aob\_smod\_1bit <= sort\_or;

END estructural;

Feu servir com a entrades les següents: dos vectors de tres bits ***a***,***b*** (2 DOWNTO 0) seran els sumants, un senyal tipus bit ***cin*** correspondrà al *carry in*. Com a senyals de sortida, tindrem els vectors de 3 bits suma\_smod\_3bits, aib\_smod\_3bits, aob\_smod\_3bits, i un senyal tipus bit ***cout*** (*carry out*).

Per al banc de proves, considereu alguns casos particulars per a sumar, com per exemple:

ent1<= "001", "101" AFTER 20ns, "001" AFTER 40ns, "011" AFTER 60ns;

ent2<= "001", "101" AFTER 30ns, "011" AFTER 50ns, "010" AFTER 70ns;

Haureu de pujar 2 fitxers, SENSE COMPRIMIR, que continguin les següents informacions:

* Un fitxer amb l’entitat comp2bits (comparador de 2 bits) amb la seva arquitectura **ifthen** que heu generat i fent ús de paraules (vectors). Incloeu el corresponent banc de proves per a comprovar el seu correcte funcionament. També heu d’incloure les entitats i les arquitectures de les portes lògiques.

El nom del fitxer serà **Py\_xxx\_Cognom1\_Cognom2\_Nom\_comp2bits.vhd**.

* El segon fitxer (**Py\_xxx\_Cognom1\_Cognom2\_Nom\_comp4bits.vhd**) haurà d’implementar el comparador de 4 bits, a partir de dos comparadors de 2 bits i les portes lògiques necessàries. Incloeu també el corresponent banc de proves, indicant quines sumes feu.

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