Departament de Disseny i Programació de Sistemes Electrònics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Arquitectura de computadors

Pràctica 2

8 de març de 2017

Disseny del banc de registres i del comptador de programa

Antoni Escobet

PRÀCTICA 2 – Disseny del banc de registres i del PC

1. Objectius

En aquesta practica es pretén:

- Repassar els conceptes relacionats amb el banc de registres d'un processador MIPS i el seu comptador de programa.
- Dissenyar un banc de 32 registres de 32 bits.
- Dissenyar el comptador de programa (PC)
- Amplia els coneixements sobre el llenguatge VHDL i l'edició amb QUARTUS.

2. Material

L'únic material necessari per la realització d'aquesta pràctica és el paquet de programari d'ALTERA, el Quartus instal·lat als ordinadors del laboratori, i que us podeu descarregar gratuïtament de la pàgina web d'Altera (http://www.altera.com).

La simulació del vostre disseny s'ha de fer amb el simulador que proporciona el mateix paquet de programari d'Altera (ModelSim-Altera)

3. Problema proposat

En aquesta pràctica s'ha de realitzar el disseny del banc de registres i del comptador de programa que s'ha vist a teoria.

El disseny del processador vist a classe, amb el seu camí de dades i la seva unitat de control permet executar un conjunt reduït d'instruccions relacionades amb la unitat aritmètica i lògica, es a dir, que pot realitzar sumes i restes de nombres sencers de 32 bits, i les operacions lògiques AND i OR sobre 32 bits. També pot comparar dos operants per saber si un és major que l'altre.

Per tal de poder fer els salts condicionals, serà necessari que l'ALU proporcioni a la unitat de control, l'indicador de si el resultat de l'operació val zero (Z).

A la pràctica 1 s'ha dissenyat la unitat aritmètica lògica que permet realitzar aquestes poques instruccions enumerades. El banc de registres ha de subministrar les dades dels programes a la unitat aritmètica lògica perquè realitzi els càlculs requerits. Com recordareu, els operands de les instruccions aritmètiques i lògiques no poden ser variables qualssevol (com les usades en els llenguatges d'alt nivell); han de procedir d'una sèrie limitada de posicions especials denominades **registres**. Cada registre és capaç d'emmagatzemar una paraula de memòria. Així, ja que l'arquitectura MIPS que heu estudiat és de 32 bits, la grandària de cada registre serà també de 32 bits. Com sabeu, el nombre de registres existents en l'arquitectura d'un processador no és il·limitat (la superfície del processador és limitada), i en el cas que ens ocupa és igual a 32. La notació emprada per a referir-nos a ells és $0, $1,…, $31.

Com s'ha dit anteriorment, les instruccions aritmètiques i lògiques necessiten utilitzar operands ubicats en registres. No obstant, inicialment les instruccions i les dades del programa es troben a la memòria. Per tant, abans de poder utilitzar un dada del programa serà necessari portar-ho des de memòria a un registre determinat. A més, és possible que necessitem escriure a la memòria algun resultat emmagatzemat en un registre. Per portar una dada des de la memòria a un registre es necessita una operació de lectura sobre memòria. De la mateixa forma, per emmagatzemar un dada continguda en un registre a la memòria s'ha de fer una operació d'escriptura sobre memòria. L'arquitectura MIPS ofereix dues instruccions específiques per a realitzar la

lectura d'una paraula de memòria i l'escriptura o emmagatzemament d'una paraula en memòria. Aquestes són: lw, *load word i* sw, *store word*, respectivament.

El banc de registres que heu de dissenyar està format per 32 registres de propòsit general, numerats del 0 al 31. **El registre $0 sempre conté el valor 0 perquè així està establert pel maquinari**. En el MIPS hi ha establert una sèrie de convencions sobre com cal utilitzar els registres (per exemple, en quins registres es passen els arguments de les funcions, quin és el punter de pila, els reservats per al sistema operatiu, etcètera), però no afecta en el disseny del maquinari que ens ocupa.

Així mateix, el comptador de programa és una altra unitat fonamental en el maquinari d'un processador. Bàsicament, el comptador de programa és un registre que es va actualitzant després d'executar una instrucció. Aquesta actualització pot ser un increment constant o bé la càrrega d'una nova adreça en el cas de les instruccions de salt. Al inici de cada instrucció, s'accedeix a l'adreça de memòria apuntada pel contingut del comptador de programa per aconseguir la instrucció que s'ha d'executar. En funció d'aquesta instrucció, el contingut del comptador de programa s'incrementa en 4 unitats (grandària en bytes de les instruccions del processador) o s'actualitza amb un valor nou que s'obté a partir del valor actual del comptador de programa i d'alguns bits codificats en la pròpia instrucció.

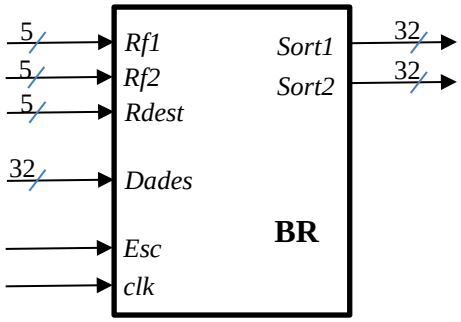
4. Descripció de la practica

En aquesta secció es descriu com realitzar el banc de registres i el comptador de programa descrits a l'apartat anterior. Anirem realitzant un disseny incremental. En primer lloc, després de recordar l'estructura estudiada a classe sobre la relació entre les diverses unitats funcionals del processador, presentarem la definició de les entitats VHDL a realitzar. Començarem pel banc de registres, partint d'un senzill registre de 32 bits que anirem ampliant en la seva funcionalitat fins a la construcció del banc complet. Finalment, implementarem el comptador de programa.

4.1. Relació del banc de registres i del comptador de programa amb la resta d'unitats

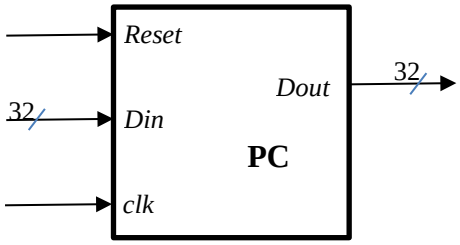
Quina informació hem de tindre present a l'hora de realitzar els dissenys?

Banc de registres:



- Hi ha tres entrades per indicar sobre quins registres volem actuar
  - ***Rf1***, ***Rf2*** indiquen els dos registres a llegir.
  - ***Rdest*** indica el registre a actualitzar.
- Dues sortides per posar-hi les dades que es llegeixen:
  - ***Sor1*** indica el bus de sortida del registre Rf1 que es llegeix.
  - ***Sor2*** indica el bus de sortida del registre Rf2 que es llegeix.
- Una entrada de dades per a les operacions d'escriptura sobre el registre:
  - ***Dades*** indica l'entrada de la informació a escriure sobre el registre *Rdest*.
- ***Esc*** és el senyal d'escriptura per capturar la dada present a *Dades* i emmagatzemar-ho al registre *Rdest*.
- Finalment, no ens podem oblidar del senyal de rellotge del sistema (***clk***) per a determinar o sincronitzar el moment de l'escriptura sobre els registres.

Pel comptador de programa s'ha de considerar:



- Una entrada ***clk***, que és el rellotge del processador. Serveix per sincronitzar els diferents esdeveniments que s'hauran de realitzar sobre el comptador de programa.
- Una entrada ***Din.*** El valor d'aquesta entrada és el que memoritzarà el PC per cada flanc de pujada del senyal de rellotge.
- La sortida del comptador de programa ***Dout***. Aquesta sortida es mostra sempre.
- Afegirem un senyal de ***Reset*** per a inicialitzar a zero el comptador de programa.

4.2. Definició de l'entitat: Banc de registres

A partir dels senyals indicats a l'apartat anterior, la definició de l'entitat del banc de registres serà:

```
entity BancRegistres is
    port (  clk : in STD_LOGIC;
            Esc : in STD_LOGIC;
            Rf1: signed(4 downto 0);
            Rf2: signed(4 downto 0);
            Rdest: signed(4 downto 0);
            Dades : in signed(31 dowto 0);
            Sor1 : out signed(31 dowto 0);
            Sor2 : out signed(31 dowto 0);
         };
end BancRegistres;
```

4.2.1. Guia de disseny del Banc de registres

Hi ha diverses formes de realitzar un desenvolupament. En aquest punt es descriu, a manera d'ajuda, quins passos serien necessaris per a realitzar el banc de registres d'una perspectiva educativa. No obstant, i respectant la definició de l'entitat presentada en la secció anterior, teniu *llibertat* per a enfocar el disseny de la forma que vulgueu.

1. Definició d'un registre de 32 bits

Sembla lògic pensar que la primera cosa que s'ha de realitzar és la definició d'un registre de 32 bits. Aquesta entitat bàsica servirà tant per a la composició del banc de registres com per a altres registres necessaris al camí de dades (acumulador, comptador de programa….).

Per realitzar el registre de 32 bits s'han de tenir en compte les següents consideracions:

- Un senyal d'habilitació o escriptura (*enable*) actiu a nivell alt.

- Un senyal de rellotge (*clk*) per determinar el moment de l'actualització en el cas d'una operació d'escriptura: aquesta sincronització s'efectuarà amb el **flanc de baixada del senyal de rellotge** del processador (suposant que durant el semiperíode positiu del senyal de rellotge la dada ja està generada o es generà perquè al produir-se el flanc de baixada s'actualitzi el contingut del registre).
- Un senyal de *reset* asíncron i actiu a nivell alt.
- I finalment, les dades d'entrada (*Din*) i les de sortida (*Dout*).

```
entity Registre is
        port (  Reset : in STD_LOGIC;
                clk : in STD_LOGIC;
                enable : in STD_LOGIC;
                Din : in STD_LOGIC_VECTOR(31 dowto 0);
                Dout : out STD_LOGIC_VECTOR(31 dowto 0);
                );
end Registre;
```

```
architecture Behavioral of Registre is
begin
-- Es pot realitzar una definició comportamental del registre, basat en un procés
process (clk, reset)
begin
…
…
…
…
end process;
end Behavioral;
```

2. Connexió dels registres amb els busos d'entrada i sortida

Una vegada definida l'entitat *Registre,* s'ha de pensar com es podrà seleccionar i posar el seu contingut a algun dels busos al que està connectat. Hi ha diverses alternatives, unes més costoses en recursos, altres més o menys ràpides, etcètera. Entre elles, en podem destacar dues:

1. Connectar totes les sortides dels registres a un gran multiplexor perquè deixi arribar als busos del processador (connectats a *Sor1* i *Sor2*) el contingut dels registres seleccionats per les entrades *rf1* i *rf2*. En aquest cas serien dos multiplexors que tindrien 32 entrades de 32 bits i una sortida de 32 bits, controlats per *rf1* i *rf2* cadascun.
2. Dotar als registres que s'acaben de definir de dues sortides de tres estats fent que en funció de si es vol llegir un registre determinat per algun dels dos busos s'activi la sortida corresponent. D'aquesta manera es pot connectar directament les sortides dels 32 registres als busos de sortida *Sor1* i *Sor2*.

Entre les dues opcions (podeu decantar-se per qualsevol de les dues), anem seguir l'explicació amb la segona opció:

A partir del registre de 32 bits es realitza una entitat, anomenada *RegSortida3Estats*, que s'afegeix al registre ja dissenyat les dues portes de sortida de tres estats i la lògica per determinar si s'accedeix a un registre per llegir el seu contingut, i per quin bus de sortida s'han d'encaminar les dades, o si s'accedeix per actualitzar el seu contingut.

Per llegir el seu contingut els senyals relacionats seran:

- ***Dout1*** i ***Dout2***: són els dos busos del camí de dades que s'han anomenat *sor1* i *sor2* (és a dir, es passaran directament des de l'entitat *BancRegistres* a la que ens ocupa).
- ***E1*** i ***E2***: dos senyals de lectura de registre, actius a nivell alt, que identifiquen a quin bus s'ha de deixar la informació del registre. Aquests senyals, com es veurà més endavant, s'obtindran d'un descodificador que, a partir de *rf1* i *rf2* determina el registre que es vol accedir.

Per a actualitzar el seu contingut, els senyals relacionats seran:

- ***Din***: dades d'entrada (igual al senyal Dades de l'entitat *BancRegistres*)
- ***Clk***: senyal d'escriptura ja comentada en l'entitat Registre. El flanc de baixada del senyal determina l'instant en què s'actualitza la dada.
- ***WE***: senyal d'habilitació d'escriptura. Determinarà quin dels 32 registres és seleccionat per a realitzar una operació d'escriptura (el $0 no es pot actualitzar, sempre val zero). Aquest senyal s'obtindrà a partir de la descodificació dels senyals *rdest* i de l'activació del senyal d'escriptura *esc* (o e_reg) per part de la unitat de control.

Per tant, l'entitat *RegSortida3Estats* tindrà la definició:

```
entity RegSortida3Estats is
        Port (
          Reset : in STD_LOGIC;
          clk : in STD_LOGIC;
          WE : in STD_LOGIC;
          Din : in STD_LOGIC_VECTOR (31 downto 0);
          E1 : in STD_LOGIC;
          E2 : in STD_LOGIC;
          Dout1 : out STD_LOGIC_VECTOR (31 downto 0);
          Dout2 : out STD_LOGIC_VECTOR (31 downto 0)
        );
end RegSortida3Estats;
```

```
architecture Behavioral of
      RegSortida3Estats is
component Registre is
        Port (
          Reset : in STD_LOGIC;
          clk : in STD_LOGIC;
          Enable : in STD_LOGIC;
          Din : in STD_LOGIC_VECTOR (31 downto 0);
          Dout : out STD_LOGIC_VECTOR (31 downto 0)
        );
end component;
```

```
signal SortidaReg: STD_LOGIC_VECTOR (31 downto 0);
begin
        reg: Registre port map (…,…, …,…, …);
        Dout1 <= SortidaReg when E1 = '1' else
        "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
        Dout2 <= ……..
end Behavioral;
```

3. Etapa de descodificació dels registres seleccionats i disseny de l'entitat principal del banc de registres

Un cop dissenyada l'entitat *RegSortida3Estats* s'ha de realitzar l'arquitectura de l'entitat general *BancRegistres*. Aquesta entitat serà la que, a partir de components *RegSortida3Estats*, implementi el banc de registres complet.

Falta preparar alguns dels senyals que se li han de passar a l'entitat *RegSortida3Estats*. Bàsicament son els senyals de selecció dels registres implicats en la lectura o escriptura. Per preparar aquest senyals s'han de descodificar els valors de registre codificats al Registre d'Instrucció (IR): *rf1, rf2* i *rdest*. Per a això, cal realitzar una entitat que implementi un descodificador asíncron de 5 a 32 amb entrada d'habilitació per a cadascun d'aquest tres senyals. Les entrades d'habilitació dels descodificadors seran els senyals que genera la

unitat de control *L_SOR1*, *L_SOR2* i *ESC* respectivament. Així, quan la unitat de control no activi cap d'aquests senyals no i haurà cap registre seleccionat ja que el descodificador no generarà cap '1' a les seves sortides. L'única cosa que falta és analitzar en detall l'operació d'escriptura. S'ha dit anteriorment que el senyal d'escriptura serà el flanc de baixada del senyal de rellotge i que el registre realitzat ha de rebre un senyal d'habilitació d'escriptura. D'altra banda, la descodificació de *rdest* indica el registre a actualitzar i el senyal *ESC* (o *e_reg*) de la unitat de control indica o habilita l'escriptura.

A continuació, es mostren totes aquestes idees juntament amb l'esquelet. Com es pot veure el ***registre0*** té un comportament especial: sempre val 0, és a dir, no es pot actualitzar i el seu contingut és sempre el mateix (0). Podeu implementar aquest registre (entitat RegistreZero) d'una forma comportamental o estructural.

```
reg0: RegistreZero port map (…,…,…,…,…,…,…,…,…);
reg1: RegSortida3Estats port map (…, …,…, …, …, …, …, …);
reg2: RegSortida3Estats port map (…, …,…, …, …, …, …, …);
-- completar fins el reg31….
end Behavioral;
```

```
architecture Behavioral of BancRegistres is
    component RegSortida3Estats is
        Port (      Reset : in STD_LOGIC;
                    clk : in STD_LOGIC;
                    WE : in STD_LOGIC;
                    Din : in STD_LOGIC_VECTOR (31 downto 0);
                    E1 : in STD_LOGIC;
                    E2 : in STD_LOGIC;
                    Dout1 : out STD_LOGIC_VECTOR (31 downto 0);
                    Dout2 : out STD_LOGIC_VECTOR (31 downto 0)
                );
        end component;

    component RegistreZero is
        Port ( ……..);
    end component;

- si es fa el registre de forma comportamental dins de la definició d'arquitectura no farà falta la definició del component

    component Descodificador5 is
        Port (      Habilitacio: in STD_LOGIC;
                    Ent : in STD_LOGIC_VECTOR (4 downto 0);
                    Sort : out STD_LOGIC_VECTOR (31 downto 0));
        end component;

signal RegSel1: STD_LOGIC_VECTOR (31 downto 0);
signal RegSel2: STD_LOGIC_VECTOR (31 downto 0);
signal RegEsc: STD_LOGIC_VECTOR(31 downto 0);
begin
dec1: Descodificador5 port map (…, …,…);
dec2: Descodificador5 port map (…,…,…);
dec3: Descodificador5 port map (…,…,…);
- el registre 0 sempre té un 0. No és actualitzable.
- Si s'ha realitzat una entitat s'instanciarà. En cas contrari s'escriurà aquí el codi comportamental
```

Una altre forma de crear el banc de registres, és utilitzant els "array" de vhdl per crear una matriu de dades. Les matrius son una col·lecció d'elements del mateix tipus de dades als que s'accedeix mitjançant un índex. N'hi ha de monodireccionals (un sol índex) o multidimensionals (varis índex) i poden estar emmarcades en un rang o pot ser lliure, donant una matriu d'una dimensió infinita.

Exemples:

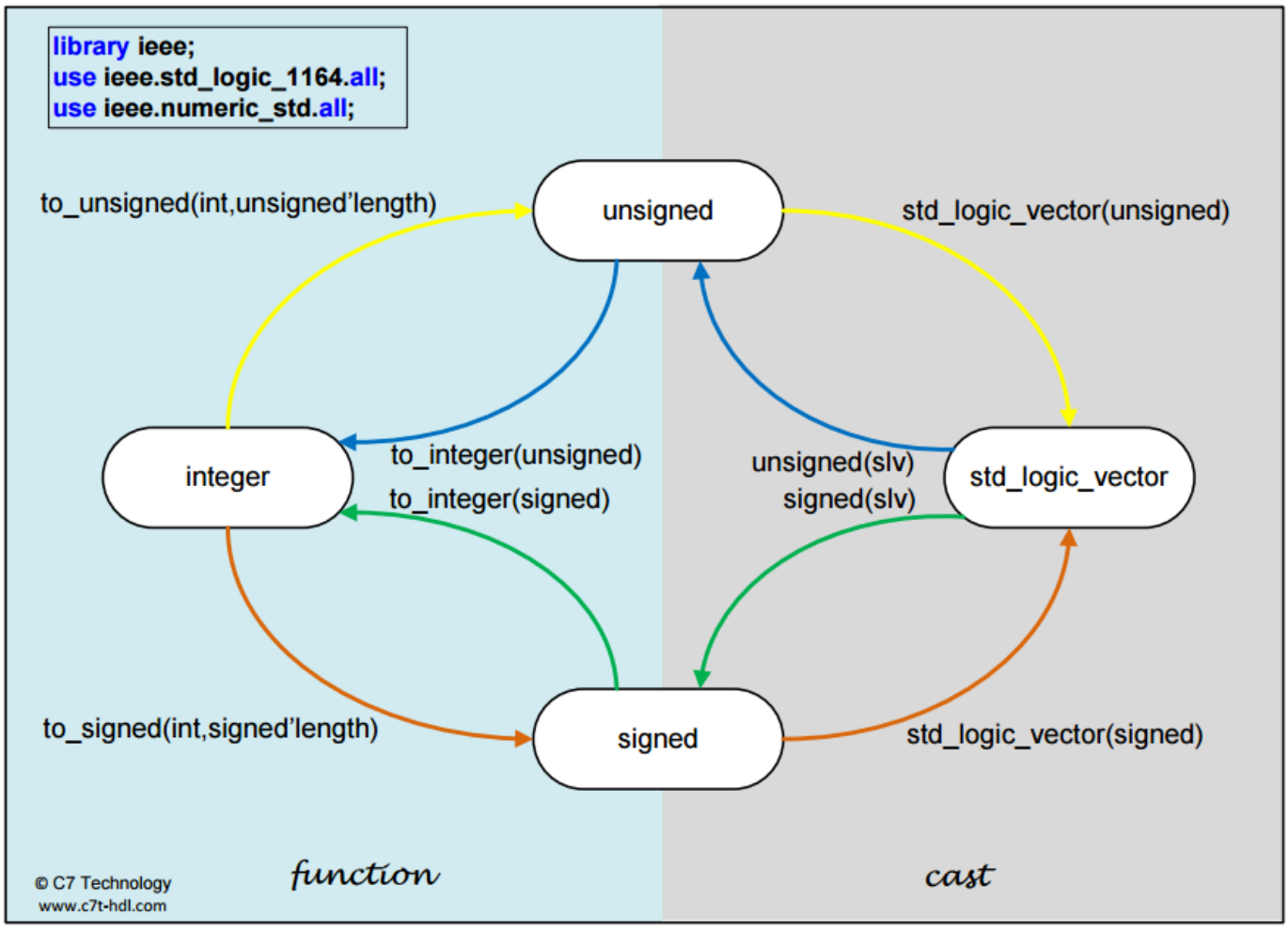
```
package matrius is
        type word is array (31 downto 0) of bit;
        type vector is array (natural range <>) of integer;
        type byte_array is array (integer range <>) of std_logic_vector(7 downto 0);
        ....
end;
```

Els dos últims exemples mostren una matriu amb un índex sense rang i serveix qualsevol sencer. Quan es declari la dada, s'haurà de posar límits a la matriu:

```
signal dada: vector(1 to 64);
```

Als elements d'una matriu s'accedeix mitjançant l'índex. dada(3) és l'element 3 del vector dada.

Per utilitzar les matrius és important recordar la taula de conversió de tipus de dades:



4.3. Definició de l'entitat: Comptador de programa

Després de crear el banc de registres s'ha de fer un nou projecte per a realitzar el comptador de programa. Aquesta entitat, tal com s'ha indicat tindrà la següent definició:

```
entity ComptadorPrograma is
    Port (  Reset : in STD_LOGIC;
            clk: in STD_LOGIC;
            Din: in STD_LOGIC_VECTOR (31 downto 0);
            Dout: out STD_LOGIC_VECTOR (31 downto 0)
    );
end ComptadorPrograma;
```

4.3.1. Guia de disseny del Comptador de programa

El comptador de programa només ha de memoritzar el contingut de l'entrada cada vegada que hi hagi un flanc de baixada al senyal de rellotge. El que té memoritzat, és el que s'ha de mostrar a la sortida. Afegim l'entrada de "Reset" per poder inicialitzar el registre a zero.

5. Realització pràctica

Aquesta part depèn de com realitzeu el disseny del banc de registres. Si ho feu de la primera forma que s'ha explicat, heu de realitzar tot un seguit de passos (els tres primers) per verificar que el vostre disseny funciona correctament. Si realitzeu el disseny d'alguna altre forma, podeu passar directament a l'exercici 3.

5.1. Exercici 1

Realitzeu el disseny corresponent a l'entitat *Registre* i verifiqueu el seu funcionament correcte amb el simulador.

5.2. Exercici 2

Realitzeu el disseny corresponent a l'entitat *RegSortida3Estats* i verifiqueu el seu funcionament correcte amb el simulador.

5.3. Exercici 3

Realitzeu el disseny corresponent a l'entitat *BancRegistres* i verifiqueu el seu funcionament correcte amb el simulador.

5.4. Exercici 4

Realitzeu el disseny corresponent a l'entitat *ComptadorPrograma* i verifiqueu el seu funcionament correcte amb el simulador.

5.5. Exercici 5 (Opcional)

Amb els dissenys del banc de registres i del comptador de programa creats als apartats anteriors creeu uns símbols nous. Amb l'editor d'esquemàtics del "Quartus" dissenyeu uns esquemes que utilitzin aquests nous components (només els heu de connectar a unes entrades i sortides) i verifiqueu el seu correcte funcionament amb el "ModelSim".

En tots els casos, heu de presentar el codi VHDL realitzat per a cadascun dels exercicis, així com les simulacions que demostren el funcionament correcte dels circuits realitzats.

Recordeu que s'ha de demostrar el funcionament correcte de la pràctica el dia assenyalar. En aquests cas el dia que s'ha de presentar la pràctica és el 22 de març.