Software Radio

2024-2025

Alberto Marquillas

Diego André Gonçalves

Tabla de contenido

Introducció general

Objectiu de la pràctica

La pràctica desenvolupada en aquest treball té com a finalitat fonamental introduir l'estudiant en el llenguatge de descripció de maquinari VHDL, tot proporcionant una base sòlida per al disseny de sistemes digitals mitjançant tècniques formals i estructurades. A través de dotze exercicis successius i una pràctica final, s'ha buscat que l'alumne assimili progressivament els conceptes essencials que regeixen el comportament tant dels circuits combinacionals com dels circuits seqüencials.

L'activitat inclou una àmplia varietat de propostes que abracen des de la simple assignació de senyals fins a la implementació de sistemes més avançats com filtres digitals i generadors de senyals basats en IP Cores. D'aquesta manera, es pretén no només reforçar la sintaxi pròpia del VHDL, sinó també fomentar una comprensió profunda del modelatge temporal, la jerarquia de disseny, l'ús de processos, l'estructuració mitjançant màquines d'estats i la integració de components predefinits dins del flux de treball d'una eina de disseny com Vivado.

Eines utilitzades (Vivado, Simulador, MATLAB, ...)

Per a la realització d'aquesta pràctica s'han emprat diverses eines que han facilitat tant el procés de disseny com la verificació dels sistemes implementats. L'entorn principal de desenvolupament ha estat Vivado Design Suite, proporcionat per Xilinx, que permet crear, sintetitzar i simular codi VHDL dins d'un entorn gràfic completament integrat. Aquesta eina ha estat fonamental no només per escriure el codi sinó també per definir estructures jeràrquiques, gestionar components i generar IPs mitjançant el mòdul IP Integrator.

A més del disseny, Vivado incorpora el seu propi simulador que permet observar el comportament funcional del circuit abans de ser sintetitzat, cosa que resulta essencial per a la depuració i validació. En exercicis més avançats, com el dotzè, ha estat necessari utilitzar MATLAB, especialment per representar gràficament les sortides sinusoidals i cosinusoidals generades per un DDS (Direct Digital Synthesizer) i així poder validar visualment la qualitat de la senyal generada.

Finalment, s'ha fet ús del paquet TextIO del propi VHDL per poder escriure les dades de sortida a fitxers de text durant la simulació, la qual cosa ha facilitat enormement l'anàlisi posterior dels resultats mitjançant eines externes com MATLAB.

Metodologia de treball

El desenvolupament de la pràctica s'ha estructurat seguint una metodologia coherent i iterativa per garantir la comprensió i assimilació progressiva dels conceptes. Per a cada exercici, s'ha començat amb una anàlisi detallada de l'enunciat, identificant clarament els requisits funcionals i les especificacions del comportament esperat del circuit. A partir d'aquest punt, s'ha procedit al disseny de l'arquitectura corresponent, ja sigui mitjançant descripció textual en VHDL o mitjançant eines gràfiques com l'IP Integrator en aquells exercicis que requerien la integració d'IP Cores.

La implementació ha consistit en la codificació acurada del comportament del sistema, seguint les bones pràctiques de programació hardware, amb especial atenció a la correcta gestió del temps, les condicions de sincronisme i l'estructuració modular del codi. Un cop completat el disseny, s'ha elaborat un testbench específic per a cada exercici, amb l'objectiu de verificar la

funcionalitat del circuit en simulació. Aquesta etapa ha resultat clau per assegurar que les especificacions inicials s'han complert.

En els exercicis on s'ha generat sortida numèrica, s'han escrit les dades a fitxers mitjançant TextIO i, si ha estat necessari, s'ha realitzat una representació gràfica dels resultats mitjançant MATLAB. Aquest enfocament ha permès detectar errors lògics i comprovar el comportament temporal dels sistemes de forma visual i precisa.

Finalment, per a cada exercici s'ha redactat una breu anàlisi de resultats i conclusions, amb l'objectiu de consolidar els aprenentatges i reflexionar sobre possibles millores del disseny.

Exercici 1: Assignació directa de senyal

Enunciat i objectiu

Aquest primer exercici té com a objectiu familiaritzar l'estudiant amb la sintaxi bàsica de VHDL i el funcionament de les assignacions simples dins d'una arquitectura. L'enunciat demana crear una entitat amb una única entrada i una única sortida, ambdues de tipus `std_logic`, de manera que la sortida reflecteixi sempre el valor de l'entrada. Aquesta operació s'ha de fer mitjançant una assignació concurrent.

Tot i la seva simplicitat, aquest exercici és essencial per entendre la diferència entre assignacions concurrents i processos seqüencials dins d'una arquitectura, i serveix com a punt de partida per a dissenys més complexos.

Descripció del codi VHDL

El component creat rep el nom de `Exercici_1` i defineix dues portes: `x`, d'entrada, i `y`, de sortida. Ambdues són del tipus `std_logic`. L'arquitectura conté una única assignació concurrent:

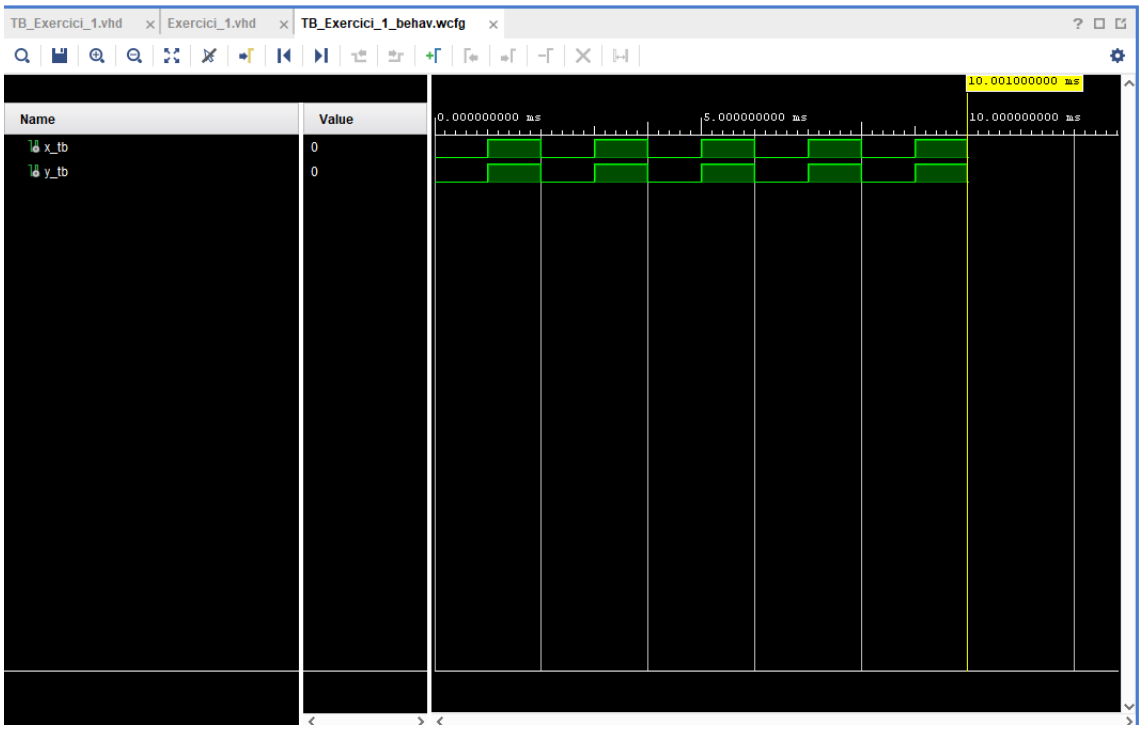
```
x <= y;
```

Aquesta línia indica que el valor de `x` es copia automàticament a `y` de forma immediata i sense necessitat de cap rellotge o control addicional. No s'utilitza cap `process`, ja que l'assignació no depèn de cap condició o seqüència temporal.

Resultats de la simulació

Per verificar el funcionament del component, s'ha creat un testbench anomenat `TB_Exercici_1`. Aquest testbench genera diversos valors per a l'entrada `x` i observa si `y` adopta immediatament el mateix valor. Durant la simulació es pot comprovar que la sortida `y` segueix perfectament el comportament de l'entrada `x`, confirmant així que l'assignació concurrent funciona tal com s'esperava.

El senyal `x` s'ha alternat entre els valors '0' i '1' amb intervals regulars, i en tots els casos `y` ha mostrat el mateix estat sense cap retard observable.



Conclusions

Aquest exercici, tot i ser molt bàsic, ha permès introduir conceptes fonamentals de VHDL com ara la declaració d'entitats, ports i assignacions concurrents. A més, ha servit per comprovar el funcionament del simulador i validar la infraestructura bàsica del flux de treball. El seu propòsit didàctic es compleix plenament, ja que posa l'èmfasi en la comprensió del comportament immediat de les assignacions i la relació entre entrada i sortida sense cap tipus de lògica addicional.

Exercici 2: Half Adder amb if-then-else

Enunciat i objectiu

L'exercici 2 demana implementar un sumador bàsic de 1 bit, també conegut com a *Half Adder*, utilitzant un únic bloc `process` amb una estructura condicional `if-then-else`. Aquest circuit té dues entrades binàries, `A` i `B`, i dues sortides: `SUM`, que representa la suma binària dels dos bits, i `CARRY`, que representa l'acarrerament resultant.

L'objectiu principal d'aquest exercici és introduir l'usuari en la programació seqüencial dins de VHDL mitjançant processos. Això inclou l'ús de la llista de sensibilitat, la declaració de condicions lògiques i la generació d'un comportament equivalent a un circuit combinacional.

Descripció del codi VHDL

L'entitat `Exercici_2` declara dos ports d'entrada, `A` i `B`, de tipus `std_logic`, i dues sortides, `SUM` i `CARRY`, també de tipus `std_logic`. L'arquitectura conté `process(A, B)` que s'activa quan alguna de les dues entrades canvia de valor.

Dins del procés, es fa servir una estructura condicional per analitzar totes les combinacions possibles d'entrades:

```
if (A = '0' and B = '0') then
    SUM <= '0';
    CARRY <= '0';
elsif (A = '0' and B = '1') then
    SUM <= '1';
    CARRY <= '0';
elsif (A = '1' and B = '0') then
    SUM <= '1';
    CARRY <= '0';
else
    SUM <= '0';
    CARRY <= '1';
end if;
```

Aquest conjunt de condicions cobreix totes les combinacions possibles dels bits `A` i `B` i assigna les sortides adequades segons la taula de veritat d'un Half Adder.

Resultats de la simulació

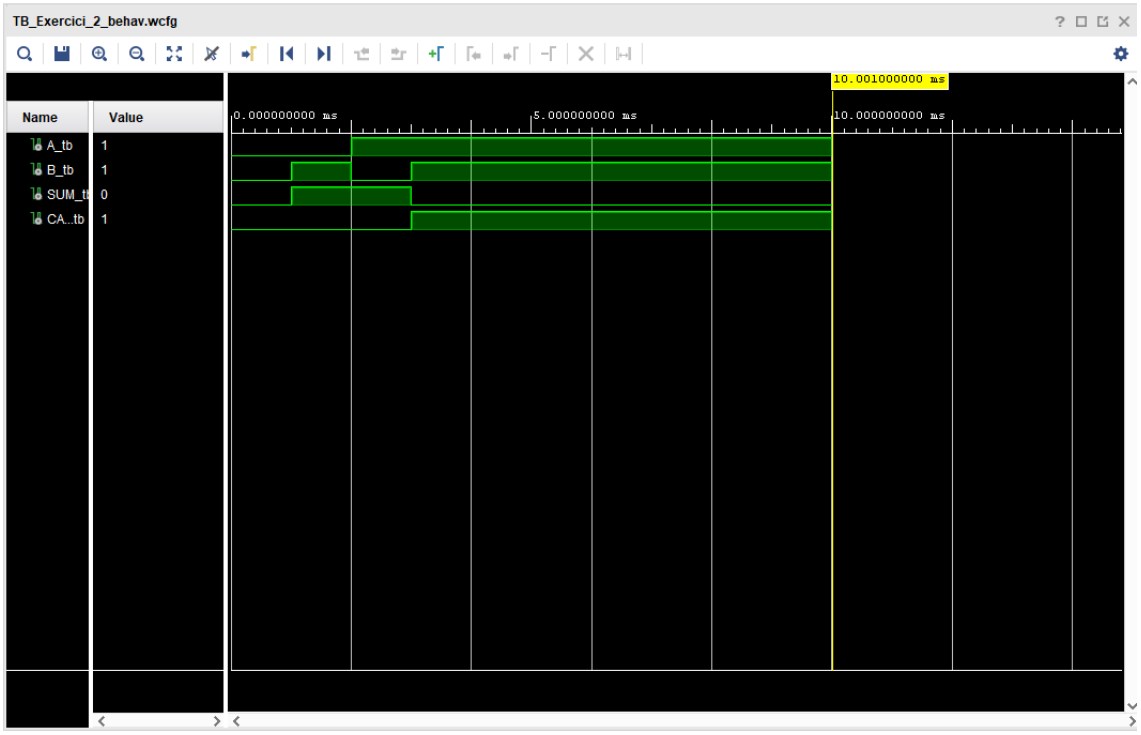
Per verificar el disseny, s'ha creat un testbench anomenat `TB_Exercici_2`, el qual genera totes les possibles combinacions de `A` i `B` amb intervals temporals regulars:

```
A_tb <= '0'; B_tb <= '0'; wait for 1 ms;
A_tb <= '0'; B_tb <= '1'; wait for 1 ms;
A_tb <= '1'; B_tb <= '0'; wait for 1 ms;
A_tb <= '1'; B_tb <= '1'; wait for 1 ms;
```

La simulació mostra que les sortides `SUM` i `CARRY` coincideixen exactament amb el comportament esperat per a un Half Adder:

```
00 -> SUM=0, CARRY=0

01 -> SUM=1, CARRY=0

10 -> SUM=1, CARRY=0

11 -> SUM=0, CARRY=1
```

Aquestes observacions confirmen que el comportament lògic del codi és correcte.



Conclusions

L'exercici 2 ha estat clau per introduir l'arquitectura basada en processos dins de VHDL, així com l'ús d'instruccions de selecció condicional. El Half Adder, tot i ser un circuit senzill, és un bloc fonamental en disseny digital i sovint es fa servir com a base per construir sumadors més complexos.

Aquest exercici també ha mostrat la importància de cobrir totes les condicions possibles en una estructura `if-then-else` per evitar la creació involuntària de memòria implícita. Finalment, s'ha reforçat la idea que un testbench ben estructurat és essencial per a la validació funcional del disseny.

Exercici 3: Lògica combinacional amb variable interna

Enunciat i objectiu

L'exercici 3 introdueix l'ús de variables dins d'un procés seqüencial per implementar lògica combinacional. L'enunciat proposa un circuit amb tres entrades (`A`, `B`, `C`) i una sortida (`Q`), totes de tipus `std_logic`, on la sortida ha de ser el resultat de l'expressió `(A or B) and C`. L'objectiu principal és practicar la manipulació de valors temporals dins d'un `process`, aprofitant la diferència entre l'ús de variables i senyals en entorns seqüencials.

A diferència de les assignacions concurrents, que s'avaluen de forma immediata i paral·lela, les assignacions a variables dins d'un `process` permeten realitzar càlculs intermedis que s'utilitzen dins la mateixa execució del bloc seqüencial, cosa que pot resultar més eficient o clara en determinades situacions.

Descripció del codi VHDL

L'entitat creada, `Exercici_3_v2`, conté tres ports d'entrada (`A`, `B`, `C`) i un de sortida (`Q`). A l'arquitectura, s'utilitza un `process` amb la llista de sensibilitat `A, B, C`. Dins d'aquest bloc, es declara una variable local anomenada `S`, de tipus `std_logic`.

El codi implementa la lògica següent:

```
process(A, B, C)
    variable S : std_logic;
begin
    S := A or B;
    Q <= S and C;
end process;
```

La variable `S` emmagatzema el resultat de l'operació `A or B`, que posteriorment s'utilitza per calcular la sortida `Q` mitjançant l'operació `S and C`. Aquesta aproximació facilita l'organització del codi, sobretot si s'han de fer operacions lògiques intermitges en dissenys més complexos.

També s'ha realitzat una versió alternativa del mateix disseny amb assignacions concurrents, denominada `Exercici_3_v1`, que resol la mateixa funció amb la línia directa:

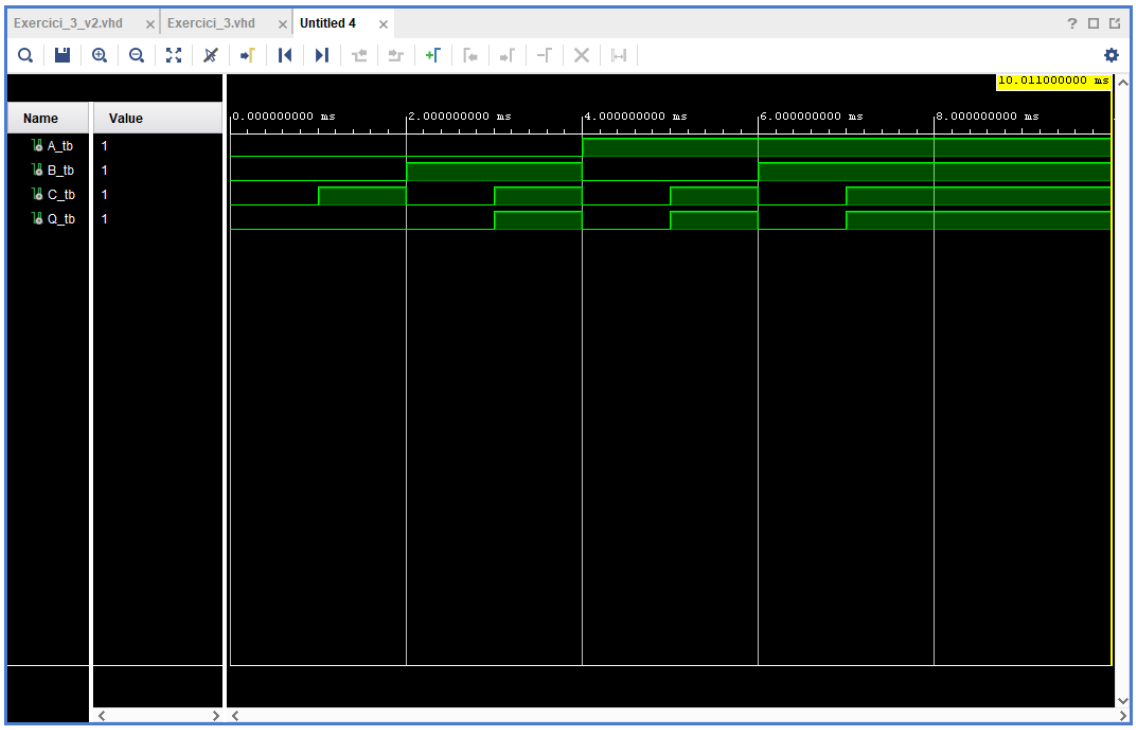
```
Q <= (A or B) and C;
```

Resultats de la simulació

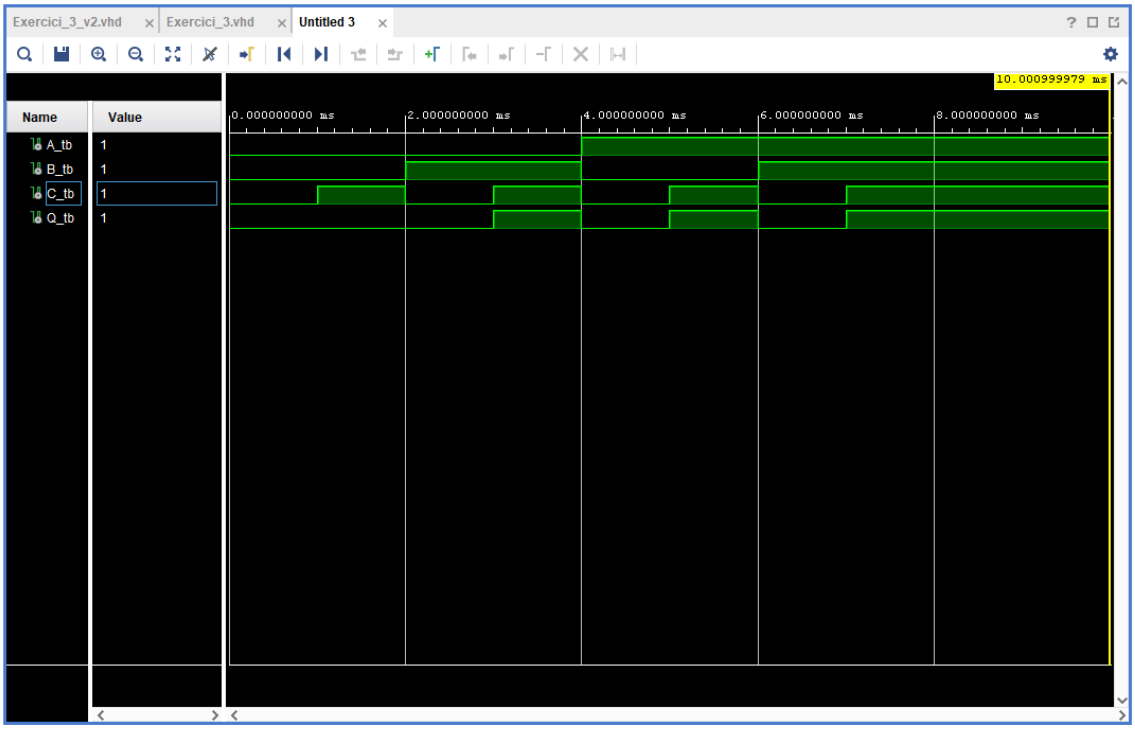
Per a la validació funcional del disseny, s'ha implementat un testbench (`TB_Exercici_3_v2`) que recorre totes les combinacions possibles de les tres entrades binàries (`A`, `B`, `C`). Això es realitza en forma de bucle que aplica seqüencialment els vuit valors binaris corresponents.

La simulació mostra que la sortida `Q` és correcta en tots els casos i coincideix amb el valor esperat de l'expressió `(A or B) and C`. Tant la versió amb `variable` com la versió amb assignació concurrent generen exactament els mateixos resultats, validant la seva equivalència funcional.

S com signa



S com variable



Conclusions

Aquest exercici ha estat especialment Aquest exercici ha estat especialment util per entendre la utilitat de les **variables locals dins de processos** en VHDL. La variable `S` s'ha utilitzat com a

registre temporal per descompondre l'expressió en dues etapes, cosa que pot facilitar la lectura i depuració en dissenys més grans.

També s'ha reforçat la comprensió de com VHDL tracta les **assignacions seqüencials** dins d'un `process` comparat amb les **assignacions concurrents** fora d'aquest. El testbench complet i exhaustiu ha estat clau per garantir la funcionalitat del disseny i per demostrar que ambdós enfocaments (amb `process` i sense) són equivalents des del punt de vista funcional, tot i que presenten diferències estructurals i de llegibilitat.

Exercici 4: Decodificador 2 a 4 amb case

Enunciat i objectiu

L'exercici 4 proposa la implementació d'un decodificador binari de dues entrades i quatre sortides. El comportament desitjat és que, per cada combinació possible de les dues entrades binàries, s'activi exactament una de les quatre sortides, seguint un patró un-hot. El circuit s'ha d'implementar utilitzant una estructura `case` dins d'un bloc `process`, de manera que l'estudiant es familiaritzi amb aquesta construcció de selecció múltiple dins de VHDL.

L'objectiu d'aquest exercici és reforçar el concepte de selecció controlada mitjançant un vector de bits i comprovar el comportament determinista d'un decodificador digital.

Descripció del codi VHDL

L'entitat `Exercici_4` conté una entrada `sel` de dos bits (tipus `std_logic_vector(1 downto 0)`) i una sortida `out_dec` de quatre bits (tipus `std_logic_vector(3 downto 0)`). A l'arquitectura, es fa servir un `process` amb `sel` com a element de sensibilitat. Dins del bloc, una estructura `case` avalua el valor de `sel` i assigna a la sortida el valor corresponent en codi un-hot:

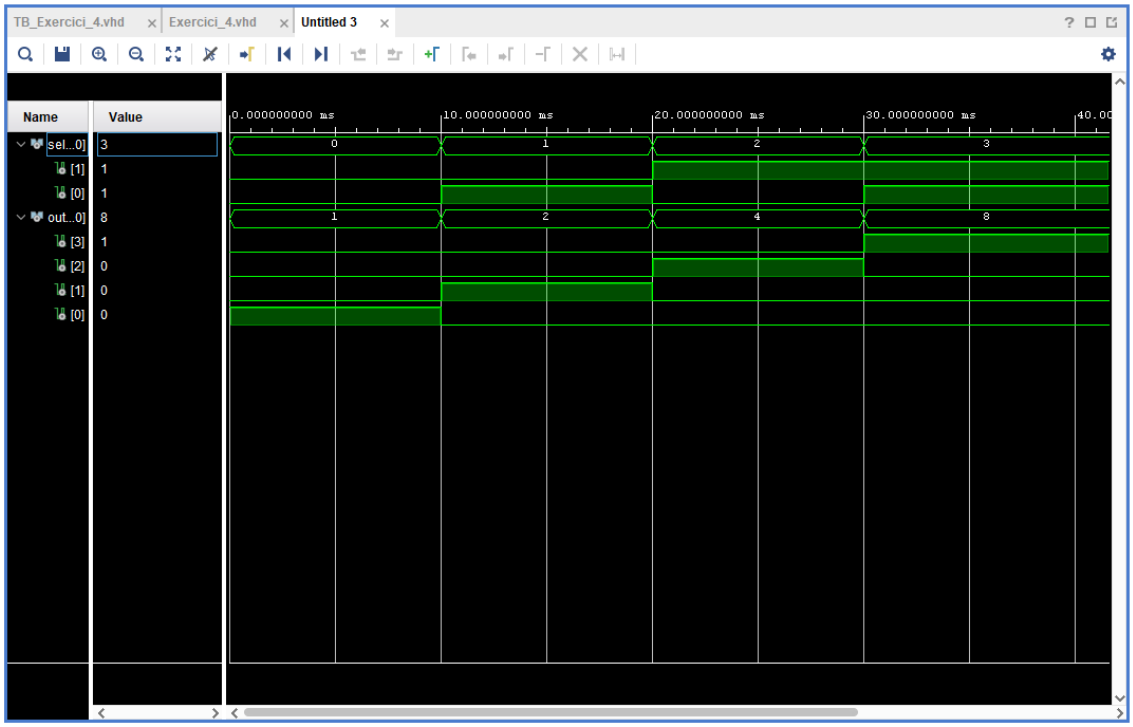
```
case sel is
    when "00" => out_dec <= "0001";
    when "01" => out_dec <= "0010";
    when "10" => out_dec <= "0100";
    when "11" => out_dec <= "1000";
    when others => out_dec <= "0000";
end case;
```

Aquesta estructura assegura que en tot moment només un bit de `out_dec` estigui actiu (a '1') segons la combinació d'entrada, i afegeix un cas `others` per seguretat, tot i que amb una entrada de dos bits no és estrictament necessari.

Resultats de la simulació

Per comprovar el funcionament correcte del decodificador, s'ha implementat el testbench `TB_Exercici_4`, que simula cadascuna de les quatre combinacions possibles de l'entrada `sel`. Durant la simulació, s'observa que per cada valor de `sel`, la sortida `out_dec` mostra el bit corresponent activat segons l'ordre establert.

Els resultats de la simulació confirmen que el circuit funciona de manera determinista i coherent amb la seva definició funcional. No s'observen errors de sortida ni activacions simultànies incorrectes.



Conclusions

Aquest exercici ha permès consolidar la utilització de l'estructura `case` com a eina clara i eficient per a la selecció múltiple en dissenys combinacionals. La implementació d'un decodificador binari 2 a 4 és un exemple clàssic que il·lustra còm una entrada codificada pot donar lloc a una sortida un-hot mitjançant lògica digital bàsica.

També s'ha remarcat la importància d'incloure el cas `others` per seguretat i robustesa del codi, especialment en dissenys reals on es poden produir situacions inesperades. En resum, l'exercici ha estat ú til per reforçar el pensament lògic seqüencial i la codificació d'estructures de selecció en VHDL.

Exercici 5: Demultiplexor 1 a 4

Enunciat i objectiu

L'exercici 5 proposa la implementació d'un demultiplexor 1 a 4, en el qual una entrada de dades `D` es redirigeix a una de les quatre sortides disponibles (`Y(3 downto 0)`) segons el valor de la senyal de selecció `sel`, que té dos bits. La resta de sortides romanen desactivades. El comportament desitjat és que, si per exemple `sel = "10"`, llavors `Y(2)` rebi el valor de `D`, mentre les altres siguin '0'.

L'objectiu principal és entendre el funcionament bàsic d'un demux i aplicar assignacions **condicionals** mitjançant l'expressió `when ... else`, que és molt comuna en VHDL per a lògica combinacional.

Descripció del codi VHDL

L'entitat `Exercici_5` té tres ports:

- `D`: senyal d'entrada (1 bit)
- `sel`: senyal de selecció de 2 bits (`std_logic_vector(1 downto 0)`)
- `Y`: vector de sortida de 4 bits (`std_logic_vector(3 downto 0)`)

L'arquitectura implementa quatre assignacions concurrents de la forma:

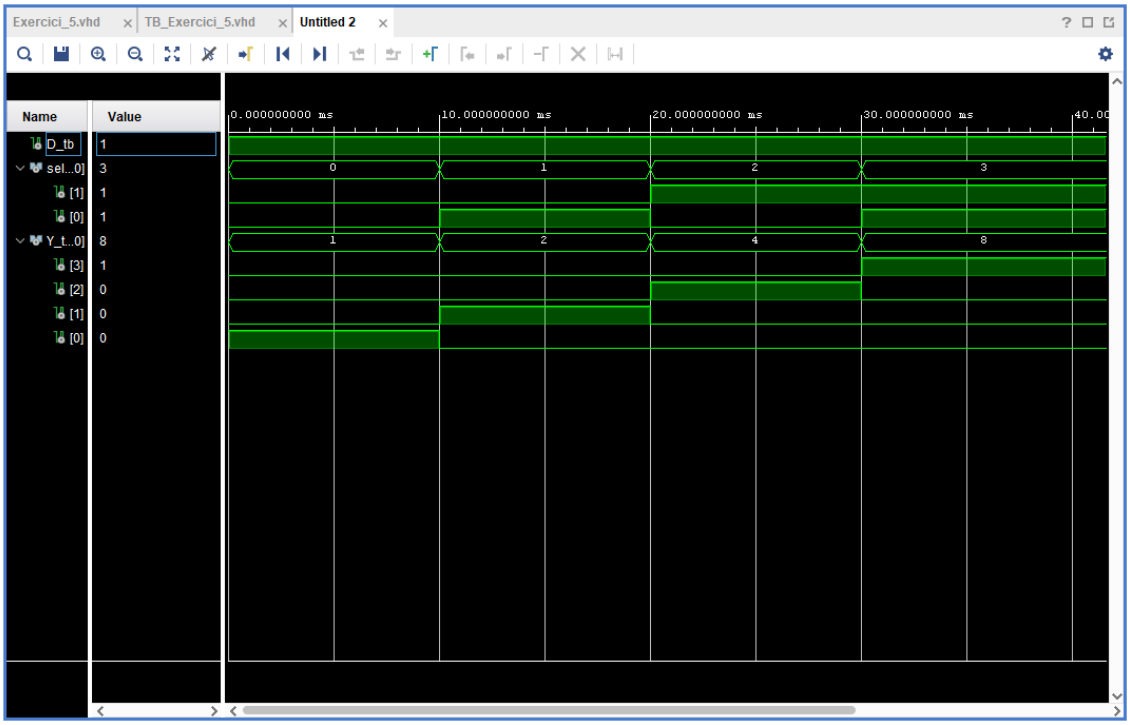
```
Y(0) <= D when sel = "00" else '0';

Y(1) <= D when sel = "01" else '0';

Y(2) <= D when sel = "10" else '0';

Y(3) <= D when sel = "11" else '0';
```

Aquest esquema assegura que **només una de les sortides** pot tenir el valor de `D` en cada moment, mentre la resta estaran a '0'. Es tracta d'una implementació clara, elegant i eficient per un demux simple.

Resultats de la simulació

Per a verificar el funcionament, s'ha creat un testbench `TB_Exercici_5` que aplica les quatre combinacions possibles de `sel`, primer amb `D = '1'` per veure com es propaga a cada sortida, i després amb `D = '0'` per confirmar que cap sortida roman activa.

La simulació confirma que per cada valor de `sel`, només una sortida pren el valor de `D`, i la resta valen '0'. En el cas de `D = '0'`, totes les sortides es mantenen a '0', com s'espera. Això valida el comportament correcte i determinista del demultiplexor.



Conclusions

Aquest exercici ha estat fonamental per entendre la construcció d'un demultiplexor simple utilitzant assignacions condicionals en VHDL. La simplicitat de l'estructura concurrent amb `when ... else` permet modelar circuits combinacionals d'una manera neta i intuïtiva.

També s'ha pogut observar la importància de controlar amb precisió l'activació de cada sortida per evitar condicions incorrectes. El testbench ha estat clau per garantir que la lògica del demux es comporta correctament en tots els casos.

Exercici 6: Comptador de 9 bits amb reset

Enunciat i objectiu

L'exercici 6 planteja la implementació d'un comptador ascendent de 9 bits, que ha d'incrementar el seu valor a cada flanc de pujada del rellotge (`clk`) i disposar d'una senyal de reinicialització (`reset`) que torni el valor del comptador a zero. Aquest tipus de components és essencial en sistemes digitals per comptar esdeveniments, generar temporitzacions o controlar seqüències.

L'objectiu principal és familiaritzar-se amb la descripció de components **seQüencials** en VHDL, on es fa necessari controlar flancs de rellotge i gestionar l'estat intern mitjançant registres.

Descripció del codi VHDL

L'entitat `Comptador` defineix tres ports:

- clk: entrada de rellotge
- reset: senyal de reinici
- comptador: sortida de 9 bits (std_logic_vector(8 downto 0))

A l'arquitectura, es defineix un senyal intern `comptador_reg` de tipus `unsigned` amb 9 bits. Dins d'un `process` sincronitzat amb `clk`, es comprova si `reset` és actiu. En aquest cas, el registre es posa a zero. En cas contrari, s'incrementa el comptador:

```
process(clk)
begin
    if rising_edge(clk) then
        if reset = '1' then
            comptador_reg <= (others => '0');
        else
            comptador_reg <= comptador_reg + 1;
        end if;
    end if;
end process;
```

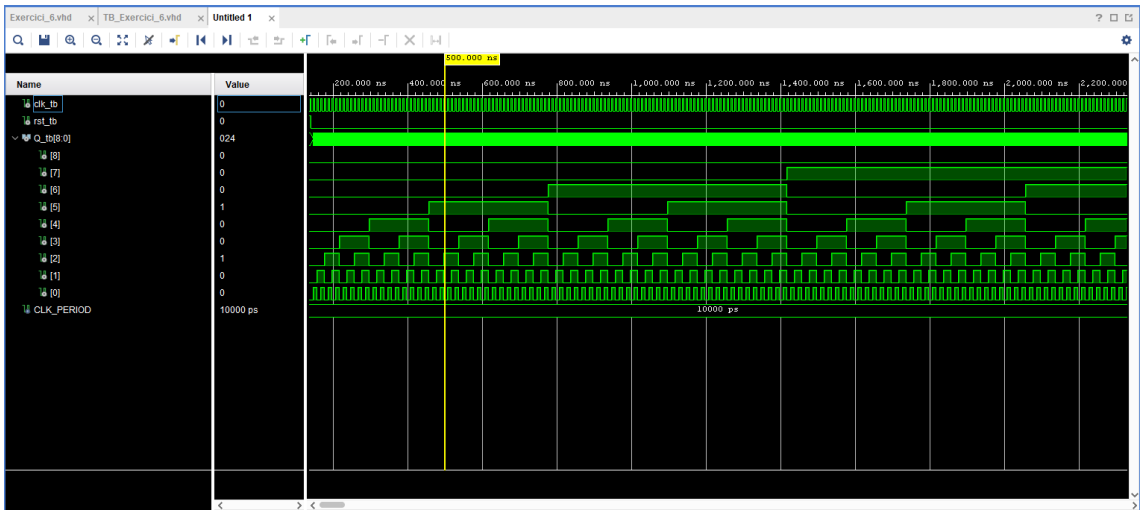
Finalment, el valor del registre intern es converteix a `std_logic_vector` i s'assigna a la sortida del component.

Resultats de la simulació

Per validar el funcionament del comptador, s'ha dissenyat un testbench `TB_Comptador` que genera una senyal de rellotge i aplica un pols de `reset` durant el primer microsegon. La simulació mostra com el comptador s'inicialitza a 0 i, a partir de llavors, s'incrementa en cada cicle de rellotge.

- Els resultats de la simulació permeten comprovar que:
- El comptador es posa a zero correctament quan `reset` = '1'.

Després de l'alliberament del reset, el comptador incrementa correctament fins arribar a 511 (màxim valor per a 9 bits).



Conclusions

Aquest exercici ha servit per consolidar els coneixements sobre processos seqüencials en VHDL i sobre l'ús de senyals de control com el `reset`. La utilització del tipus `unsigned` ha facilitat les operacions aritmètiques dins el processador.

També s'ha posat émfasi en la sincronització amb el rellotge i la correcta inicialització del registre intern, dos aspectes fonamentals en el disseny digital. El testbench ha estat essencial per comprovar el funcionament complet del sistema i validar l'arquitectura proposada.

Exercici 7: Comptador amb senyal de comparació

Enunciat i objectiu

En aquest exercici es demana implementar un sistema que combini un comptador de 9 bits amb un comparador, de manera que quan el comptador arriba al valor 256, s'activi una senyal `flag` de sortida. L'exercici s'ha de resoldre mitjançant la instància de components.

L'objectiu principal és entendre com es poden modular circuits digitals a partir de blocs funcionals interconnectats, fomentant el disseny estructural. També es posa en pràctica l'operació de comparació d'enters en VHDL.

Descripció del codi VHDL

El sistema està format per tres entitats:

1. **`Comptador`**: implementa un comptador ascendent de 9 bits amb senyals d'entrada `clk` i `reset`, i una sortida `comptador` de tipus `std_logic_vector(8 downto 0)`.
2. **`Comparador`**: rep la sortida del comptador i activa `flag` quan el valor del comptador és igual a 256. Internament, es fa servir la conversió a `unsigned` per fer la comparació

```
flag <= '1' when unsigned(comptador) = 256 else '0';
```

3. **`Exercici_8`**: entitat principal que instancia els dos components anteriors i connecta les senyals internes.

A l'arquitectura de `Exercici_8` es defineix una senyal `comptador_signal` per interconnectar el comptador i el comparador. El mòdul completa la seva funcionalitat connectant aquesta senyal a la sortida externa `comptador` i a l'entrada del comparador, i dirigint `flag` cap a la sortida.

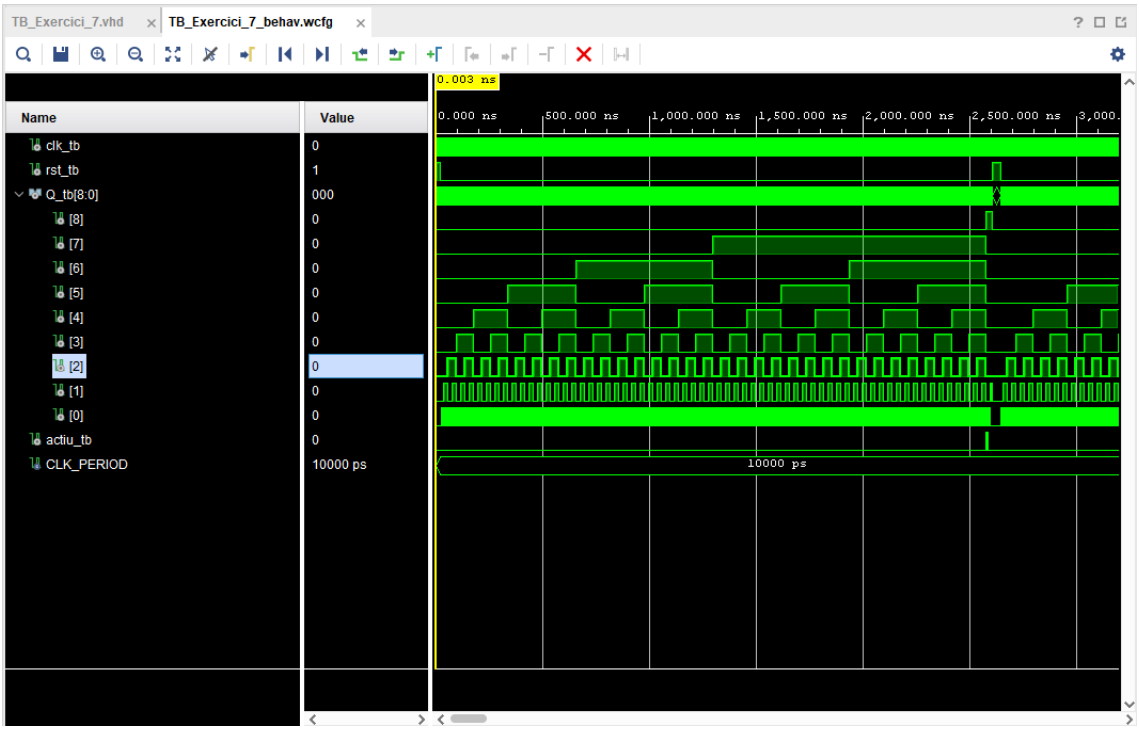
Resultats de la simulació

El testbench `TB_Exercici_8` genera una senyal de rellotge i aplica un reset al començament de la simulació. A continuació, es deixa que el comptador incrementi durant 300 µs, temps suficient per assolir el valor 256 (considerant un rellotge de 1 MHz).

Durant la simulació s'observa que:

- El valor del comptador augmenta correctament a cada cicle de rellotge.
- La senyal `flag` s'activa exactament quan el comptador arriba a 256, i es desactiva en el cicle següent.

Aquest comportament valida la correcta integració dels dos components.



Conclusions

Aquest exercici ha permès posar en pràctica el disseny modular en VHDL, mitjançant la instància de components i la interconnexió de senyals. La separació funcional entre comptador i comparador facilita la reutilització i depuració del codi.

També ha servit per aplicar la comparació entre valors `unsigned` i observar el comportament d'activació puntual d'una senyal de control. Aquest tipus d'estructura és molt comuna en dissenys digitals que requereixen sincronització d'esdeveniments.

Exercici 8: Integració de comptador + comparador

Enunciat i objectiu

Aquest exercici requereix integrar dos blocs VHDL ja implementats en exercicis previs: el comptador de 9 bits i el comparador que activa una senyal quan el comptador arriba a 256. L'objectiu és dissenyar una entitat anomenada `Exercici_8` que combini aquests blocs mitjançant instàncies i senyals internes.

A més de consolidar els coneixements sobre el disseny modular, aquest exercici posa en pràctica la reutilització de codi, la connexió de components i la comprovació funcional d'un sistema compost.

Descripció del codi VHDL

L'entitat `Exercici_8` disposa de les entrades `clk` i `reset`, i de dues sortides:

- `comptador`: valor actual del comptador (9 bits)
- `flag`: senyal d'activació quan `comptador = 256`

A l'interior de l'arquitectura es declaren les instàncies dels dos components:

```
component Comptador
```

```
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        comptador : out STD_LOGIC_VECTOR(8 downto 0)
    );
end component;

component Comparador
    Port (
        comptador : in STD_LOGIC_VECTOR(8 downto 0);
        flag : out STD_LOGIC
    );
end component;
```

També s'inclou una senyal interna `comptador_signal` que serveix d'enllaç entre el comptador i el comparador. Aquesta senyal també es connecta a la sortida externa `comptador`.

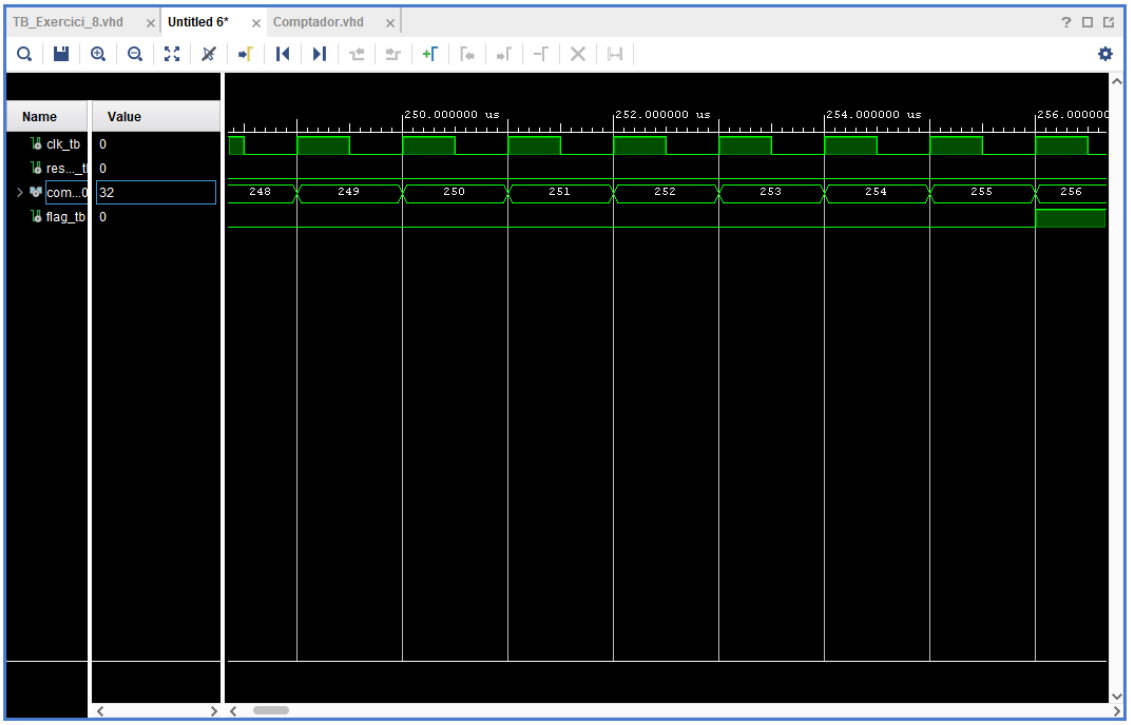
Resultats de la simulació

La simulació es duu a terme amb el testbench `TB_Exercici_8`. Aquest genera una senyal de rellotge de 1 MHz i activa el `reset` inicialment. A partir d'aquest moment, es deixa funcionar el sistema durant 300 µs.

Es comprova que:

- El comptador comenca a comptar des de 0
- En el cicle on el comptador assoleix el valor 256, `flag` s'activa per un sol cicle

Aquest comportament coincideix amb l'esperat, confirmant la correcta connexió entre els blocs.



Conclusions

Amb aquest exercici es completa la cadena de desenvolupament modular, mostrant com es poden integrar diversos blocs de codi VHDL mitjançant senyals internes. Aquest estil de disseny és habitual en projectes reals, on la reutilització i l'organització jeràrquica dels components facilita el manteniment i escalabilitat dels sistemes.

A nivell funcional, el sistema respon correctament al rellotge i a la senyal de `reset`, i produeix l'activació puntual de `flag` al valor esperat. Aquesta integració serà també fonamental per al desenvolupament de sistemes més complexes en exercicis posteriors.

Exercici 9: Serialitzador FSM (8 bits)

Enunciat i objectiu

Amb aquest exercici es completa la cadena de desenvolupament modular, mostrant com es poden integrar diversos blocs de codi VHDL mitjançant senyals internes. Aquest estil de disseny és habitual en projectes reals, on la reutilització i l'organització jeràrquica dels components facilita el manteniment i escalabilitat dels sistemes.

A nivell funcional, el sistema respon correctament al rellotge i a la senyal de `reset`, i produeix l'activació puntual de `flag` al valor esperat. Aquesta integració serà també fonamental per al desenvolupament de sistemes més complexes en exercicis posteriors.

Descripció del codi VHDL

L'entitat `Exercici_9` conté les entrades `clk`, `rst`, `start` i `data_in`, i dues sortides: `serial_out` i `done`.

El cor del sistema és una **FSM amb 10 estats**:

- `sIDLE`: estat de repòs
- `SEND_bit7` a `SEND_bit0`: enviament seqüencial dels bits de `data_in`, del més significatiu al menys significatiu
- `sDONE`: estat final de transmissió

La transició entre estats es fa a cada cicle de rellotge, començant per `SEND_bit7` quan `start = '1'`. Cada estat assigna el valor corresponent de `data_in` a `serial_out`, excepte `sIDLE` i `sDONE`.

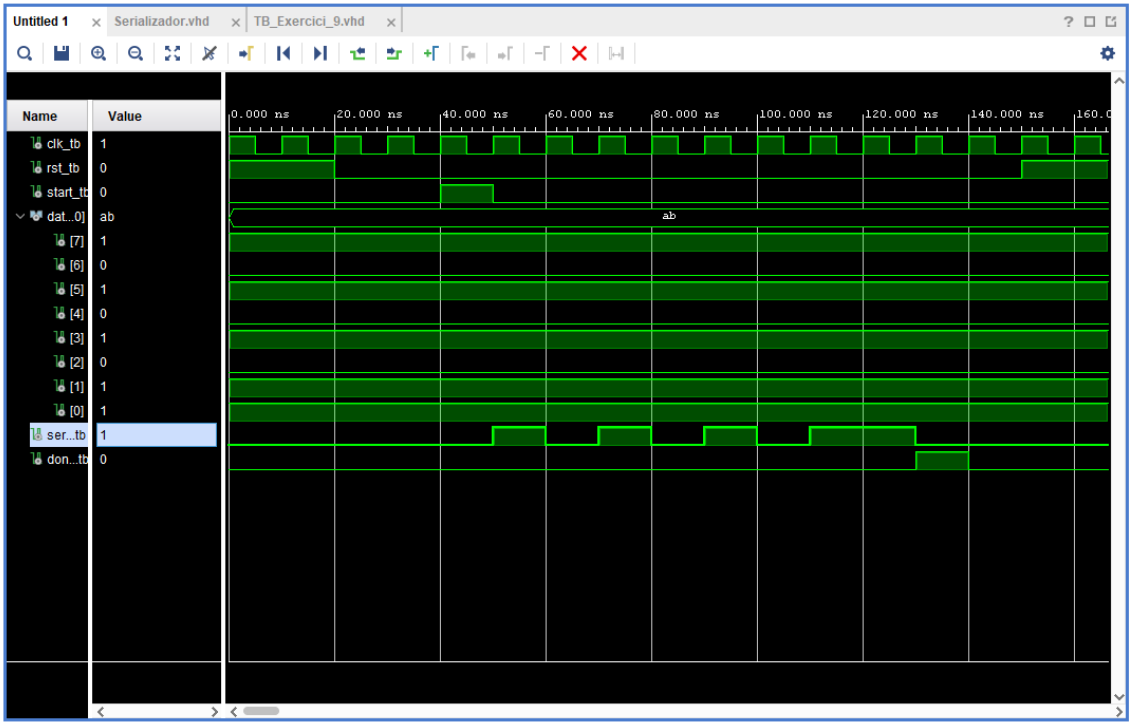
També s'empra un registre `data_reg` per desar la paraula d'entrada quan comenca la transmissió.

Resultats de la simulació

El testbench `TB_Exercici_9` genera el rellotge i la senyal de `reset`. Després de desactivar `rst`, s'aplica `start` amb la paraula `10101011`.

Durant 9 cicles de rellotge:

- S'observen a `serial_out` els bits: `1`, `0`, `1`, `0`, `1`, `0`, `1`, `1`, en aquest ordre.
- En l'ultim cicle, `done` s'activa, i el sistema torna a l'estat `sIDLE`.



Conclusions

Aquest exercici ha servit per entendre la implementació d'una màquina d'estats en VHDL, controlant el flux de dades d'un registre paral·lel a una línia sèrie. La gestió dels senyals `start` i `done` també mostra com senyals de control poden coordinar el funcionament d'un sistema seqüencial.

El disseny es pot adaptar fàcilment per a paraules de longitud diferent o per afegir protocols com bits de paritat o senyals d'inici/fi de trama.

Exercici 10: Registre de desplaçament de 128 x 16 bits

Enunciat i objectiu

L'objectiu d'aquest exercici és implementar un registre de desplaçament (shift register) de 128 posicions, on cada posició és una paraula de 16 bits. Aquest registre emmagatzema dades en sèrie i les va desplaçant a cada cicle de rellotge. La sortida del sistema és l'últim valor emmagatzemat (el més antic) al final de la cadena.

Aquesta estructura s'utilitza habitualment en sistemes de tractament de senyal per retardar dades, construir filtres FIR o emmagatzemar una finestra temporal d'informació.

Descripció del codi VHDL

L'entitat `ex_10` conté:

- Entrada `clk`: rellotge del sistema
- Entrada `data_in`: dada d'entrada de 16 bits
- Sortida `data_out`: valor de la posició 127 (el que porta més temps emmagatzemat)

A l'interior, es declara un array de 128 elements de 16 bits:

```
type shift_reg_array is array (127 downto 0) of STD_LOGIC_VECTOR (15 downto 0);

signal shift_reg : shift_reg_array := (others => (others => '0'));
```

A cada flanc de pujada del rellotge:

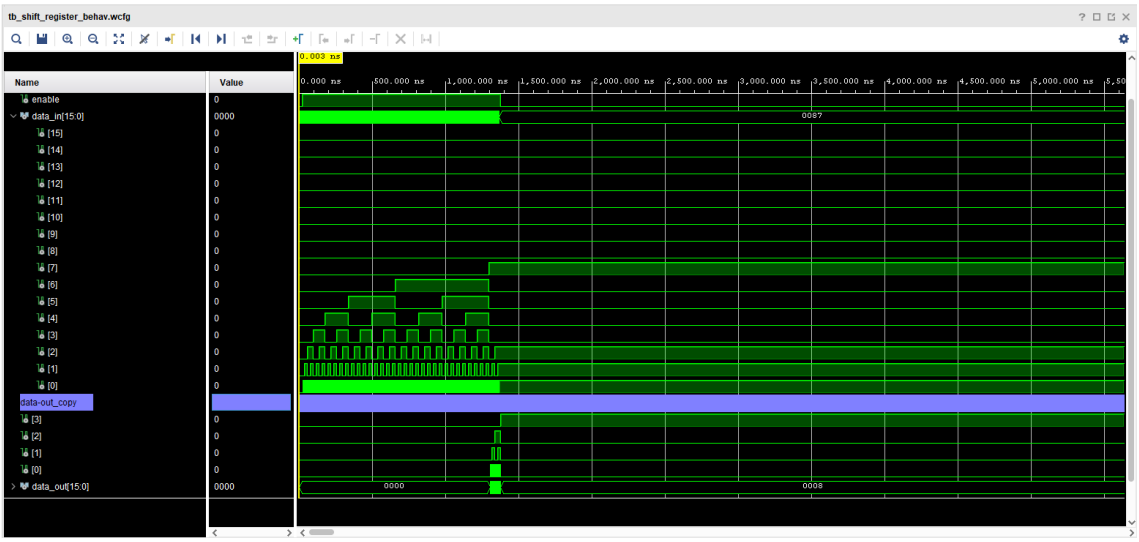
1. Es desplacen tots els valors cap a la dreta
2. S'introdueix `data_in` a la posició 0
3. S'assigna la sortida com el valor de la posició 127

Resultats de la simulació

El testbench `tb_ex10` proporciona:

- Un rellotge amb període de 2 ms
- Un conjunt de valors d'entrada seqüencials amb bits a 1 a diferents posicions

A mesura que s'introdueixen valors nous a `data_in`, aquests van desplaçant-se pel registre. Després de 128 cicles, els primers valors apareixen per `data_out`, confirmant el correcte funcionament del desplaçament.



Conclusions

Aquest exercici posa en pràctica una estructura molt habitual en sistemes digitals: el registre de desplaçament. Tot i que la seva implementació en VHDL és senzilla, el control de la mida, sincronització i propagació correcta de les dades és fonamental.

Aquests registres tenen aplicacions en filtres digitals, comunicacions en sèrie i sistemes de memòria temporal. El codi pot adaptar-se per incloure senyals de `reset` i `enable`, augmentant-ne la flexibilitat i control.

Exercici 11: Filtre FIR de 4 coeficients

Enunciat i objectiu

Aquest exercici demana implementar un filtre FIR (Finite Impulse Response) de 4 coeficients amb els valors:

```
b[n] = [-8, 5, 3, -4]
```

L'objectiu és aplicar aquests coeficients a les quatre mostres més recents d'un senyal d'entrada per obtenir una sortida filtrada, fent servir aritmètica amb nombres enters en representació signed. Aquest tipus de filtre es fa servir habitualment en processat digital de senyals per eliminar soroll, suavitzar o extreure característiques d'un senyal.

Descripció del codi VHDL

L'entitat `ex_11` conté:

- Entrada `clk`: rellotge del sistema
- Entrada `x`: dada d'entrada de 8 bits en format `signed`
- Sortida `y`: sortida filtrada de 16 bits en format `signed`

A l'interior de l'arquitectura es defineix:

- Un registre de desplaçament `shift_reg` de 4 posicions per mantenir les mostres anteriors
- Un array constant `coef` amb els coeficients
- Una senyal `accumulator` que acumula el resultat del producte de coeficients i mostres

El funcionament és el següent:

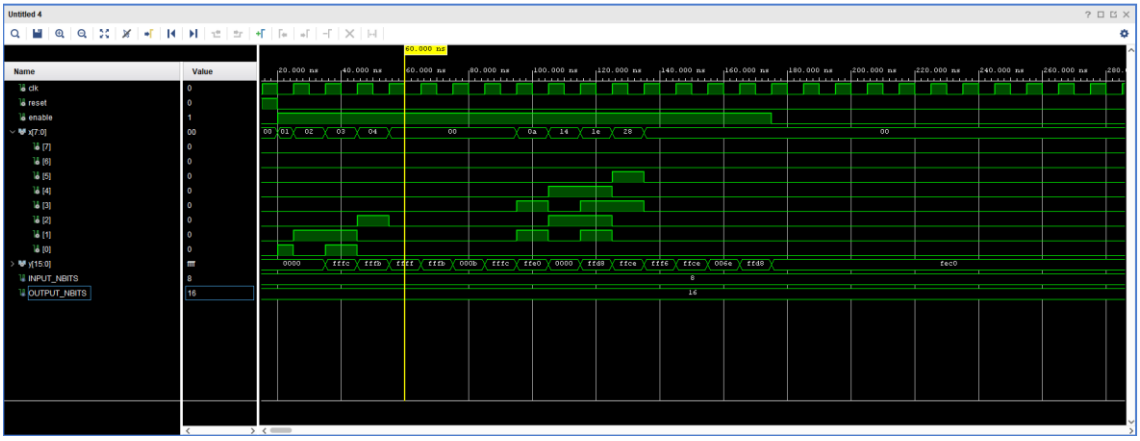
- Es multiplica `x` pel primer coeficient
- Es multipliquen les tres mostres anteriors pels coeficients corresponents
- Es fa la suma acumulativa de tots els productes
- S'actualitzen les mostres antigues desplaçant el registre
- La sortida `y` s'assigna amb el valor final acumulat

Resultats de la simulació

El testbench `tb_ex11` envia quatre valors consecutius a `x`, alternant valors positius i negatius (com ara `15`, `-15`, `30`, `-30`), i espera 4 ms entre cada valor. El rellotge té un període de 1 µs.

S'observa que la sortida `y` evoluciona segons les mostres anteriors i els coeficients. En concret:

- Quan `x` rep un nou valor, la sortida canvia segons les tres mostres anteriors i la nova mostra.
- L'impacte de cada coeficient es veu reflectit en la suma acumulada.



Conclusions

Aquest exercici mostra la implementació directa d'un filtre FIR de 4 taps en VHDL, una eina fonamental en el processament digital. L'actualització del registre de desplaçament i el càlcul en temps real de la resposta filtrada permeten entendre com s'apliquen coeficients en sistemes discrets.

Tot i que l'exemple empra nombres enters, una extensió natural seria treballar amb nombres en coma fixa o fins i tot flotants si el sistema ho permet. També seria possible carregar els coeficients dinàmicament o fer servir una estructura més eficient si es busqués optimització per a FPGA.

Exercici 12: DDS/NCO amb IP Core DDS Compiler

Enunciat i objectiu

Aquest exercici consisteix en la integració d'un Direct Digital Synthesizer (DDS) **o** Numerically Controlled Oscillator (NCO) dins d'un projecte VHDL, utilitzant un IP core proporcionat per Vivado anomenat *DDS Compiler*. L'objectiu és generar dues senyals sinusoidals digitals, en fase (cos) i en quadratura (sin), que poden utilitzar-se per aplicacions de modulació o processat digital de senyals.

Aquest component genera mostres digitals d'una sinusoide de forma eficient i precisa, i és molt comú en comunicacions digitals (QAM, QPSK, mixers digitals, etc).

Càlculs previs i configuració del DDS

Abans de crear el bloc, cal configurar correctament els paràmetres del DDS:

- Resolució de fase: 24 bits (permet definir la freqüència de sortida amb precisió)
- Resolució de dades: 16 bits (amplitud de les mostres)
- Freqüència de rellotge: 100 MHz
- Freqüència de sortida desitjada: es pot configurar mitjançant el paràmetre *Phase Increment*

Aquests paràmetres es defineixen a l'assistent de configuració del IP *DDS Compiler*.

Implementació del Block Design i Wrapper

L'estructura del sistema inclou:

- L'entitat `ex_12_wrapper`, que encapsula el block design generat automàticament per Vivado.
- El block design inclou:
  - L'IP `dds_compiler`
  - Dos blocs `xlslice` que extreuen 16 bits dels 32 disponibles del bus `m_axis_data_tdata`
- Entrades i sortides:
  - Entrada: `aclk` (rellotge)
  - Sortides: `Dout_cos`, `Dout_sin` (cos i sin de 16 bits, respectivament)

Aquest disseny genera dues senyals sinusoidals amb 90° de desfasament, útils per demodulació I/Q.

Testbench i generació de fitxers

El testbench `tb_ex12` realitza la simulació del sistema:

- Genera un rellotge amb cicles de 10 ns (freqüència de 100 MHz)
- Instancia `ex_12_wrapper` com a component
- Escriu les sortides a dos fitxers de text (`file_NCO_I.txt` i `file_NCO_Q.txt`), un per a cada component sinusoidal (I i Q)

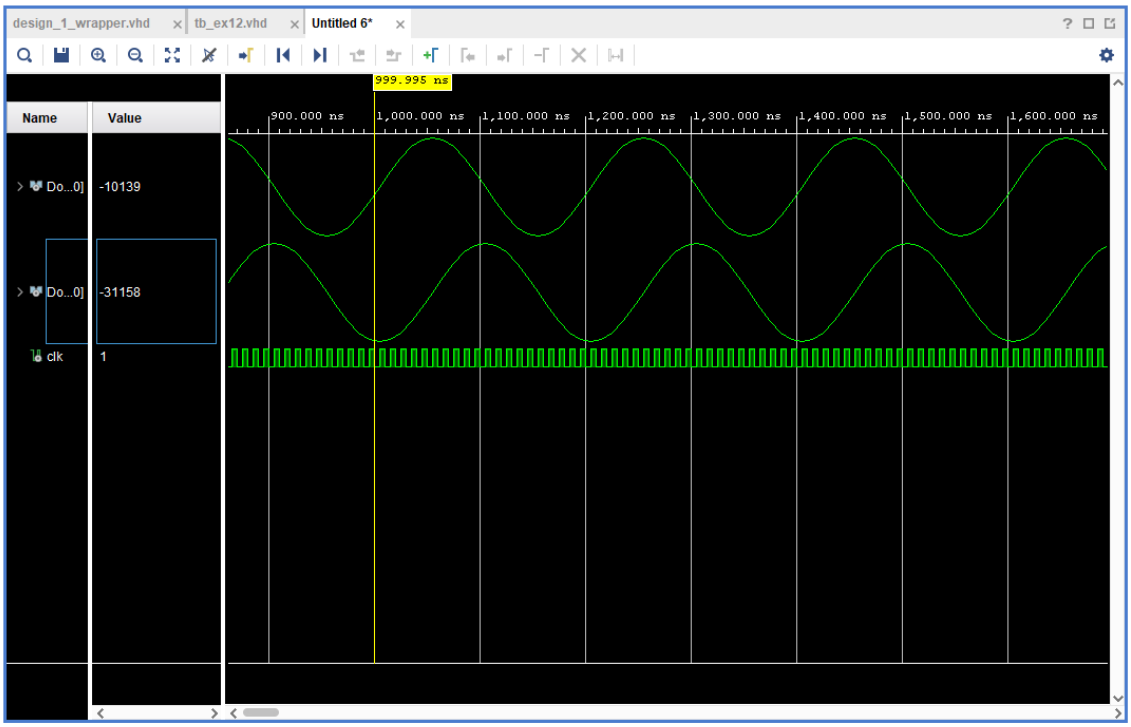
El codi fa servir les llibreries `textio` i `std_logic_textio` per convertir les sortides a enters i escriure'ls a fitxer.

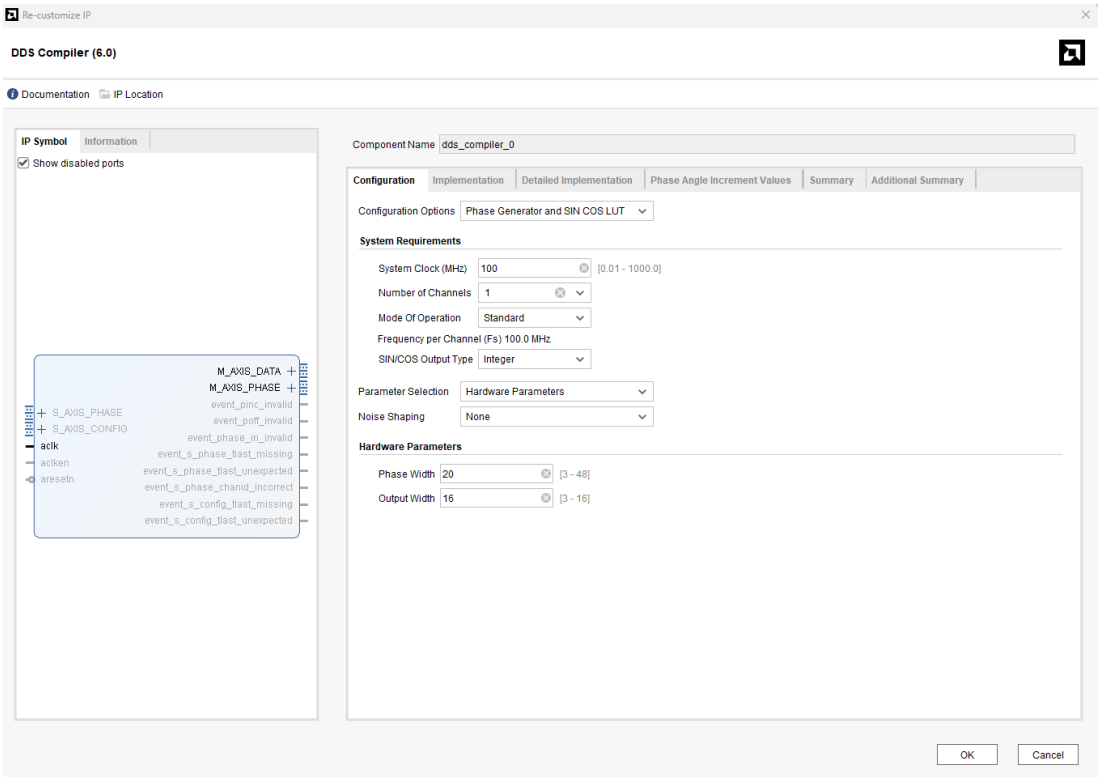
Resultats i validació amb MATLAB

S'utilitza un script MATLAB per:

- Llegir els fitxers `file_NCO_I.txt` i `file_NCO_Q.txt`
- Mostrar els senyals sinusoidals I i Q en dues gràfiques
- Representar els valors en funció del temps (amb una escala temporal en microsegons)

Aquest anàlisi permet comprovar visualment la periodicitat i la fase relativa dels senyals generats. Es confirma que les dues senyals tenen forma sinusoidal i que presenten una separació de 90° en fase, tal com s'espera d'un generador DDS amb sortides I/Q.





Conclusions

Aquest exercici representa un cas pràctic d'integració d'un IP core en un disseny VHDL. Tot i que l'IP és generat per Vivado i no s'ha programat manualment, la seva correcta instància, simulació i visualització requereix entendre el funcionament intern i els senyals de control.

També s'ha introduït la interacció entre VHDL i MATLAB a través de fitxers, una eina molt potent per anàlisi i visualització de dades. Aquesta combinació permet validar sistemes digitals de forma més intuïtiva i automatitzada, i ofereix una base sòlida per futurs sistemes de modulació digital o sintetitzadors digitals de freqüència.

Conclusions finals

Dificultats trobades

Al llarg del desenvolupament dels dotze exercicis, s'han presentat diversos reptes tant de naturalesa tècnica com metodològica. Una de les dificultats més recurrents ha estat la comprensió inicial del funcionament del llenguatge VHDL, especialment pel que fa a l'execució concurrent i les diferències amb llenguatges de programació més tradicionals. L'arquitectura dels testbenchs també ha suposat una corba d'aprenentatge, en particular l'estructuració de processos seqüencials, la generació de rellotges i el control dels temps de simulació.

També s'ha requerit un esforç addicional per entendre la configuració i integració de blocs IP, com en el cas del DDS de l'exercici 12. Aquesta etapa ha implicat interaccionar amb Vivado en un entorn gràfic, comprendre el funcionament del *Block Design*, i verificar la correcta connexió entre mòduls.

Finalment, la lectura i escriptura de fitxers amb `textio` i la representació gràfica dels resultats amb MATLAB ha suposat un desafiament en termes d'integració de tecnologies diverses, requerint una bona comprensió del flux de dades entre el maquinari simulat i l'entorn d'anàlisi.

Aprenentatges adquirits

Aquest projecte ha estat una introducció pràctica i intensa al disseny digital amb VHDL. S'han assimilat conceptes fonamentals com el funcionament de processos, senyals, estructures condicionals, i l'arquitectura general d'un sistema digital seqüencial. S'ha entès també com es construeixen sistemes més complexos a partir de mòduls senzills (jerarquia de components).

La implementació de filtres, multiplexors, comptadors i serialitzadors ha permès posar en pràctica el disseny modular i la simulació de circuits. A nivell metodològic, s'ha guanyat fluïdesa en la planificació, depuració i validació de dissenys digitals mitjançant testbenchs.

També ha estat especialment enriquidor l'exercici 12, on s'ha treballat amb un IP core i s'ha fet una integració completa amb eines externes com MATLAB. Aquest enfocament ha obert la porta a escenaris més reals dins del disseny digital i la simulació hardware-software.

En conjunt, aquest conjunt d'exercicis ha proporcionat una base sòlida per afrontar futurs projectes en sistemes digitals, FPGA i disseny de maquinari amb eines professionals.