LABORATION 1

INTRODUKTION TILL VIVADO

Under laborationerna kommer vi att konstruera/beskriva ett antal kretsar med hjälp av VHDL (Very high speed integrated circuit Hardware Description Language). För att skapa verklig hårdvara kommer vi att överföra konstruktionens beskrivning till en FPGA (Field Programmable Gate Array) från tillverkaren Xilinx med hjälp av programmet Vivado, som är Xilinx utvecklingsmiljö för FPGAer.

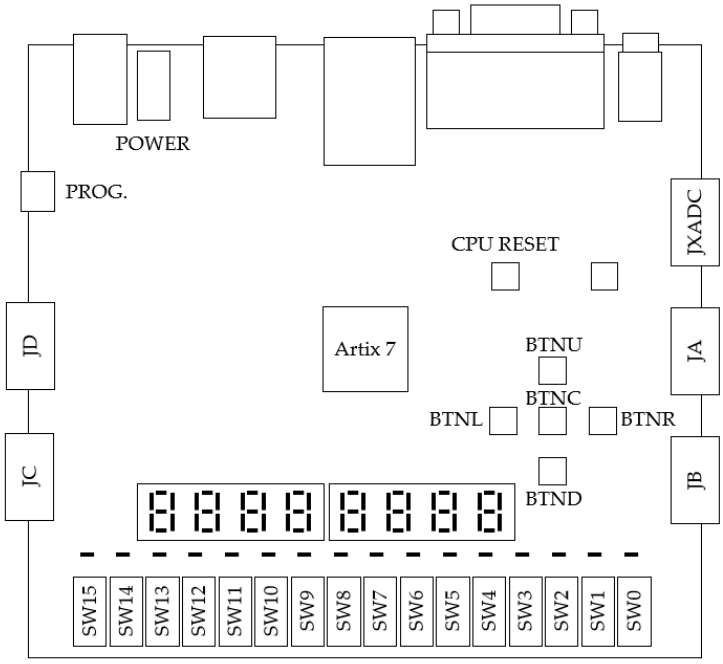
I detta appendix förklaras hur man med Vivado överför en digital konstruktion beskriven i VHDL till en FPGA. Först skapas ett nytt projekt i Vivado. Källfiler som beskriver konstruktionen skapas och läggs till i projektet. Därefter verifieras beteendet hos konstruktionen genom simulering. Efter det följer ett antal steg som syftar till att översätta konstruktionens beskrivning till hårdvara. Slutligen konfigureras FPGAn så att den beter sig enligt beskrivningen.

För att demonstrera designflödet i Vivado kommer alla stegen för att skapa en enkel konstruktion att gås igenom i detta avsnitt.

Introduktion

En FPGA kan förenklat ses som en komponent innehållande ett stort antal logiska grindar och D-element. Kopplingar mellan de olika grindarna och D-elementen kan helt konfigureras så att önskat beteendet fås.

FPGAn som används i denna kurs är Artix7 från tillverkaren Xilinx. Den sitter monterad på utvecklingskortet Nexys4, se figur 1.1. På utvecklingskortet finns även ett antal kringkomponenter. Vissa av dessa kan användas för att generera insignaler till konstruktionen, som knappar och strömställare, medan lysdioder och 7-segmentsdisplayer kan användas för att visa konstruktionens utsignaler.



Figur 1.1: Bild av utvecklingskortet Nexys4 som används i kursen. Ovanför strömställarna *SW0...SW15* sitter lika många lysdioder med namnen *LED0...LED15*.

Konstruktionen av kretsar kommer att göras i programmet Vivado som är utvecklingsmiljön för Xilinx FPGAer. De interna kopplingarna mellan grindar och D-element i FPGAn programmeras genom att en konfigurationsfil (*bitstream* i Vivado) laddas ner i FPGAn från en dator. För att skapa konfigurationsfilen använder vi oss av VHDL som är ett hårdvarubeskrivande språk. Med VHDL beskriver vi hur konstruktionen ska bete sig samt vilka in- och utsignaler den har.

Efter att vi beskrivit en konstruktion i VHDL försöker Vivado översätta beskrivningen till hårdvarukomponenter som grindar och D-element. Resultatet är ett RTL (Register Transfer Level) schema där man ser hur beskrivningen tolkats. Det går även att simulera konstruktionen och verifiera att den beter sig korrekt.

Efter verifieringen använder vi Vivados syntesverktyg för att översätta RTL-schemat till ett format passande vår målkrets Artix7. I FPGAn som vi använder finns nämligen inga verkliga grindar utan logiska funktioner implementeras med hjälp av sanningstabeller. Efter syntetiseringen får vi ett schema över konstruktionen bestående av sanningstabeller, muxar, adderare och D-element samt kopplingarna mellan dessa.

I nästa steg ber vi Vivado att implementera konstruktionen. Det som sker nu är att alla sanningstabeller och andra komponenter får en given position i FPGAn samt att alla ledningar inuti FPGAn konfigureras så att rätt kopplingar fås.

Efter implementeringen generas så slutligen konfigurationsfilen som sedan laddas ner i FPGAn via USB.

Konstruktionsexempel

För att visa arbetsgången i Vivado kommer nu alla stegen av en enkel konstruktion att gås igenom. Konstruktionen i exemplet kommer att styra lysdioderna på Nexys4 kortet så att olika ljussekvenser visas. Strömställarna **SW[3:0]** kommer att användas som insignaler till konstruktionen för att bestämma ljussekvens enligt tabell 1.1.

SW_3	SW_2	SW_1	SW_0	**Funktion/ljussekvens**
0	0	0	0	Alla dioder släckta
0	0	0	1	Alla dioder tända
0	0	1	-	Alla dioder blinkar med frekvensen 2Hz
0	1	-	-	Rinnande ljus från vänster till höger och sedan tillbaka
1	-	-	-	Rinnande ljus från sidorna till mitten och sedan tillbaka

Tabell 1.1: Beskrivning av konstruktionens funktion. **SW[1]** har högre prioritet än **SW[0]**, **SW[2]** har högre prioritet än **SW[1]** och så vidare.

För att skapa de olika ljussekvenserna kommer vi att använda en i förväg konstruerad modul *led_driver* vars beteende är beskrivet med VHDL i filen *led_driver.vhd*. Modulen har en insignal **C[2:0]** som är 3 bitar bred. Därför måste vi konstruera en krets som avkodar strömställarna och genererar rätt insignal till *led_driver* enligt tabell 1.2.

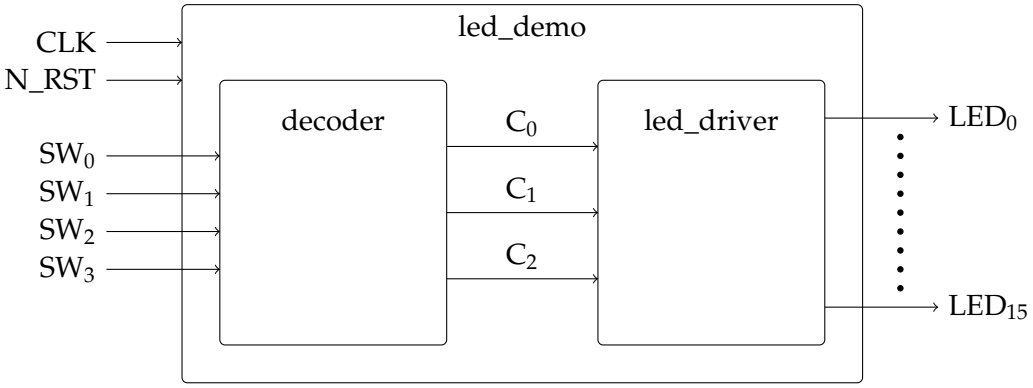
SW_3	SW_2	SW_1	SW_0	C_2	C_1	C_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	-	0	1	0
0	1	-	-	0	1	1
1	-	-	-	1	0	0

Tabell 1.2: Tabellen beskriver funktionen hos avkodaren som vi senare ska beskrivas med VHDL.

Utifrån tabellen får vi fram följande uttryck för signalen **C[2:0]**

$$C_0 = (SW_3' \wedge SW_2' \wedge SW_1' \wedge SW_0) \vee (SW_3' \wedge SW_2)$$
$$C_1 = (SW_3' \wedge SW_2' \wedge SW_1) \vee (SW_3' \wedge SW_2)$$
$$C_2 = SW_3$$

Vi kommer att dela upp konstruktionen i tre delar enligt figur 1.2. Blocket *led_driver* genererar olika sekvenser till lysdioderna och styrs via blocket *decoder* som läser av strömställarna. Kopplingen mellan blocken, lysdioder och strömställare specificeras i *led_demo*. Vi kommer senare att skapa ett projekt i Vivado där varje block beskrivs i varsin fil. Filnamnen kommer att vara samma som blockens namn med filändelsen *.vhd*, till exempel *led_demo.vhd*.



Figur 1.2: Blockschema av konstruktionen. **SW[3:0]** och **LED[15:0]** kommer att anslutas till motsvarande strömställare och lysdioder på Nexys4. **N_RST** kommer att anslutas till knappen *CPU_RESET* och klocksignalen **CLK** till en oscillator på Nexys4.

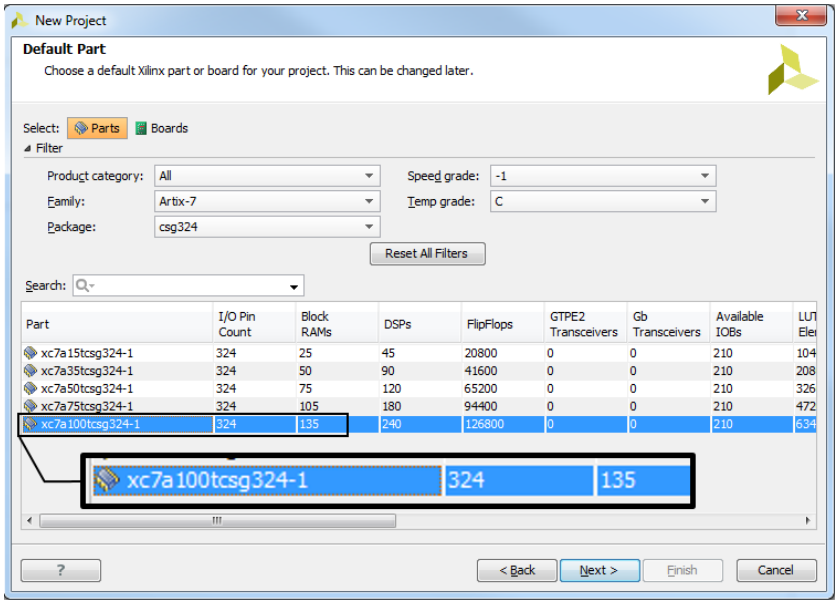
Skapa ett nytt projekt

Vi ska nu börja med att skapa ett nytt projekt i Vivado. Lägga till de färdiga filerna *led_driver.vhd* och *led_demo.vhd* samt skapa filen *decoder.vhd*.

Starta *Vivado 2016.1* genom Windows programmeny. Skapa ett nytt projekt genom att klicka på *Create New Project* alternativt via menyraden *File->New Project...* Guiden för att skapa ett nytt projekt öppnas. Klicka på nästa för att gå vidare och ange projektnamn och var projektet ska sparas. **Projektet måste sparas under "C:\users\<login-id>\Program\" annars kommer inte simuleringar att fungera!**

I nästa steg anges vilken typ av projekt som ska skapas, välj *RTL Project* och se till att *Do not specify sources at this time* är ikryssad.

Därefter måste vi ange vilken FPGA-krets som designen är tänkt att realiseras på. På Nexys4 sitter en Artix7 med beteckningen *xc7a100tcsg324-1*, leta upp den i listan och gå vidare till nästa steg, se figur 1.3. Till sist visas en sammanfattning av det nya projektet, klicka på *Finish* för att skapa projektet.



Figur 1.3: Vid skapandet av ett nytt projekt måste vi välja vilken FPGA som används. FPGAn på utvecklingskortet som vi använder har beteckningen xc7a100tcsg324-1.

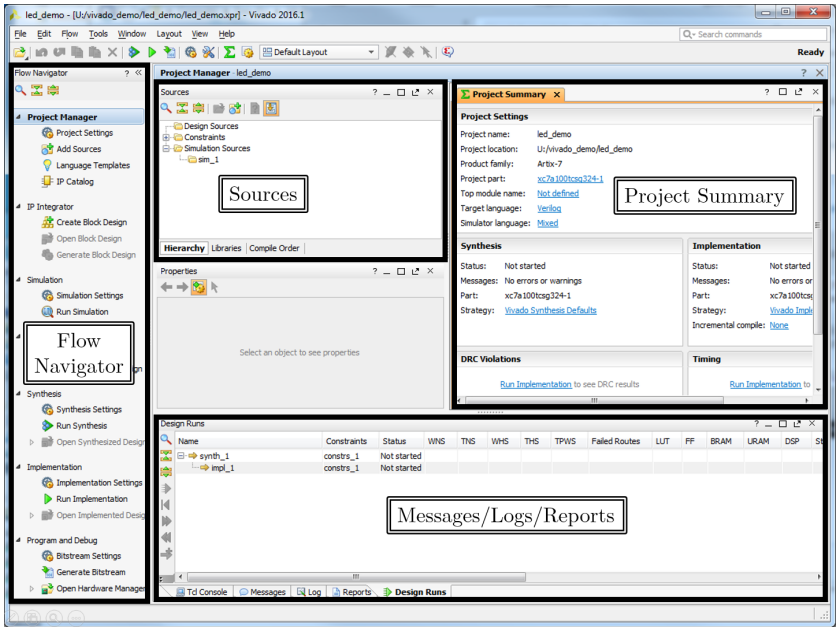
Efter att projektet skapats startas Vivado och ett antal olika fönster visas, se figur 1.4. Till vänster har vi *Flow Navigator*. Här kan vi lägga till källfiler, simulera beteendet, syntetisera, implementera, generera konfigurationsfil och överföra konfigurationen till FPGAn. Det vill säga, härifrån kommer vi åt alla funktioner som behövs för att realisera en design på FPGAn.

Till höger hittar vi *Project Summary*. Här kan vi se om varningar och fel genererats vid de olika designstegen. Efter att vi utför syntetisering och implementering så blir även information om andel utnyttjad hårdvara och effektförbrukning i FPGAn tillgänglig här.

I fönstret för *Messages/Logs/Reports* kommer vi åt mer detaljerad information rörande varningar och fel som uppstår. Här finns även en del annan information tillgänglig som maximal klockfrekvens för designen med mera.

Efterhand som vi lägger till filer i projektet kommer dessa att listas under *Sources* i olika mappar beroende på vad filen används till.

Om man råkat stänga något fönster eller av någon annan anledning vill återställa vilka fönster som ska visas så görs detta via menyraden *Layout->Reset Layout*.



Figur 1.4: Överblicksbild av Vivado efter att ett projekt skapats. Via menyn *Layout->Reset Layout* återgår vi till standardinställningen för vilka fönster som ska visas.

Lägga till och skapa filer

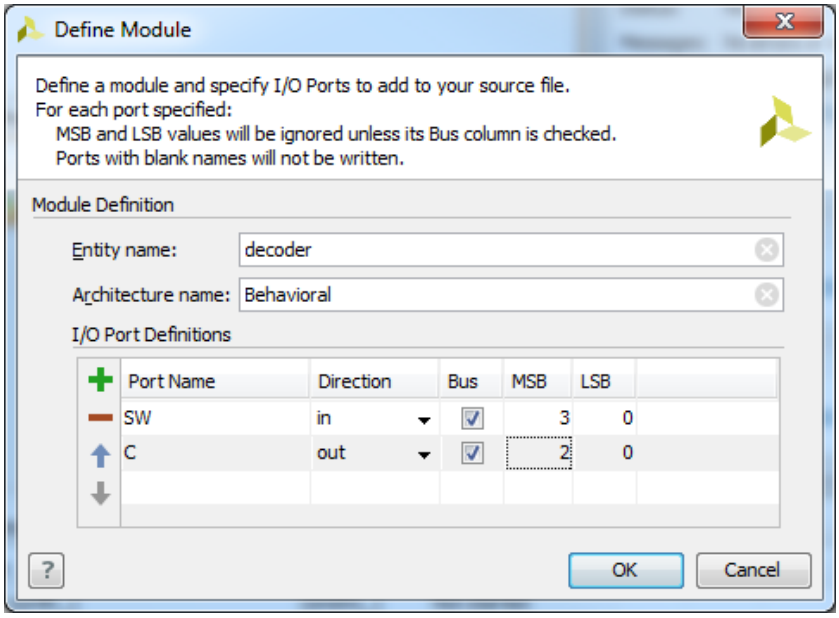
Efter att projekt skapats är det dags att lägga till källfiler som beskriver kretsen vi vill konstruera. Till att börja med så lägger vi till källfiler för *led_driver* och *led_demo*. Därefter skapar vi en ny källfil för *decoder* modulen och fyller den med kod.

Välj *Add sources* under *Project Manager* i *Flow Navigator* fönstret. Eftersom vi vill lägga till en källfil för designen väljer vi *Add or create design sources* och går vidare till nästa steg där vi klickar på *Add files*. Gå till mappen *S:\Courses\EIT\EITA15\Vivado_demo* och välj *led_driver.vhd* och *led_demo.vhd*. Se till att rutan **Copy sources into project** är markerad.

Vi ska även skapa en ny fil för *decoder* modulen. Klicka på *Create File*, se till att *File type* är satt till VHDL och ange *decoder* som *File name*. Klicka på *OK* och därefter *Finish*. Ett nytt fönster öppnas och där fyller vi i vilka in- och utsignaler *decoder* modulen ska ha. Se till att allt ser ut som i figur 1.5 och klicka på *OK*.

Efter att filer lagts till läser Vivado av innehållet och listar alla moduler (entity) i ett hierarkiskt träd under *Sources*. I det här exemplet har vi tre moduler *led_demo*, *decoder* och *led_driver*.

I det här projektet är redan källfiler för de två modulerna *led_demo*, och *led_driver* klara. Vi ska nu beskriva beteendet för den tredje modulen *decoder*. Öppna källfilen för *decoder* modulen genom att dubbelklicka på modulen under *Sources*. När källfilen öppnats ser vi att Vivado redan har fyllt i de delar som beskriver hur modulens gränssnitt med in- och utsignaler ser ut. Om allt gått rätt till så ska in- och utsignaler enligt figur 1.5 finnas med.



Figur 1.5: Efter att vi skapat en ny källfil frågar Vivado vilka in- och utsignaler modulen ska ha. Här anger vi att insignalen **SW** är 4 bitar bred och utsignalen **C** är 3 bitar bred.

Beteendet för *decoder* modulen gavs i tabell 1.2. Ändra källkoden så den ser ut som i figur 1.6. Glöm inte att spara filen efter modifieringen.

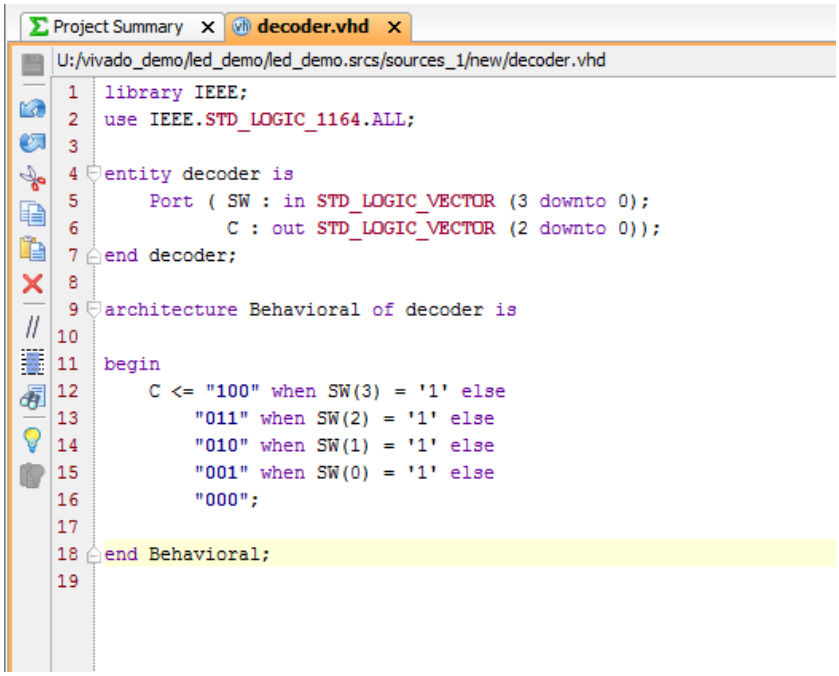
Simulering

Projektet måste vara sparat under "C:\users\<login-id>\Program" annars kommer inte simuleringar att fungera!

För att kontrollera att *decoder* modulen uppför sig korrekt ska vi nu använda Vivados inbyggda simulator för att verifiera att rätt utsignaler genereras vid olika insignaler.

Verifieringen kan göras på lite olika sätt men i den här kursen nöjer vi oss med att använda en testfil som genererar insignaler till modulen vars beteende vi vill studera. Sedan tittar vi på utsignalsvärden och avgör om beteendet är korrekt.

Innan vi kan starta simuleringen måste vi lägga till källkoden för testfilen. Klicka på *Add Sources* och välj alternativet *Add or create simulation sources*. Vivado ser då till att de filer som läggs till endast används under simulering och inte då hårdvara ska genereras. I nästa steg väljer vi *Add Files*, gå till mappen *S:\Courses\EIT\EITA15\Vivado_demo* och välj filen *decoder_tb.vhd*. Se till att alternativet **Copy sources into project** är markerat och klicka på *Finish*.



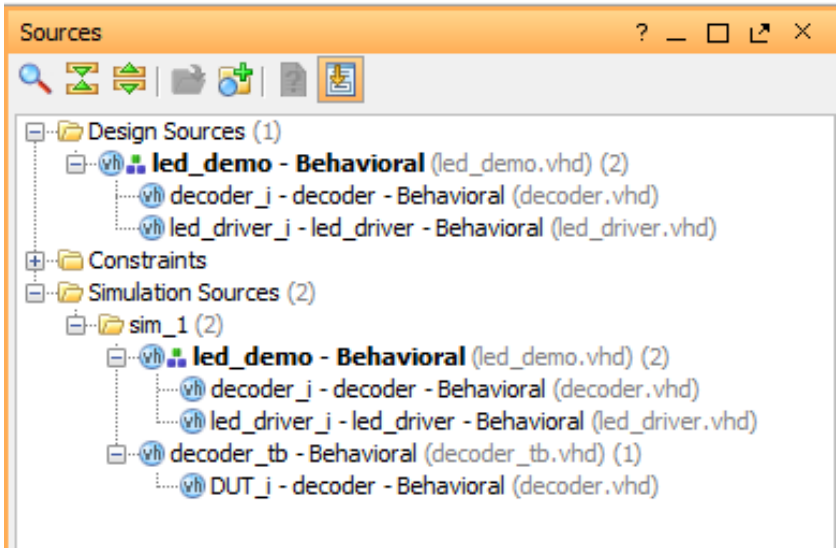
Figur 1.6: Källkod för *decoder* modulen som används för att avkoda strömställarna. När Vivado skapat filen innehåller den ett antal rader som börjar med - -. Dessa rader är endast kommentarer och har tagits bort från källkoden som visas.

Filen som importerades finns nu under *Sources->Simulation Sources*, se figur 1.7. Här finns även de moduler som beskriver hårdvaran listade (*led_demo*, *decoder* och *led_driver*). I figur 1.7 visas *led_demo* i fetstil. Det betyder att simuleringen kommer att utgå från den här modulen när den körs. Men nu vill vi simulera hur *decoder* modulen beter sig. Vi måste därför ställa in att simuleringen ska utgå från *decoder_tb* i stället. Vi ska även ange tiden för hur länge simuleringen ska köras.

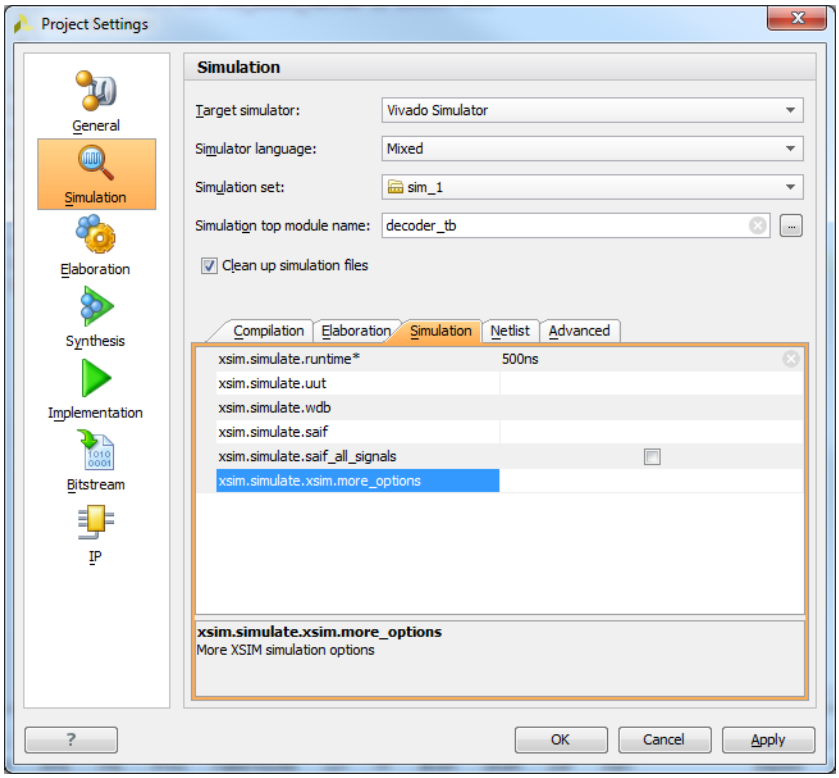
Klicka på *Simulation Settings* i *Flow Navigator*. För att ändra modul som simuleringen ska utgå ifrån klickar vi på ... vid *Simulation top module name* och väljer *decoder_tb*. Gå därefter till fliken *Simulation* och ändra simuleringstiden till 500ns så att det ser ut som i figur 1.8, klicka därefter på *OK*.

Vid större projekt är det vanligt att man har flera olika testfiler som används för att verifiera olika delar av en konstruktion. Man väljer då vilken testfil som simuleringen ska köras ifrån på samma sätt som ovan, under *Simulation Settings->Simulation top module name*. Tiden för hur länge en simulering ska köras beror på hur testfilen är utformad. Det är dock möjligt att förlänga tiden medan simuleringen körs.

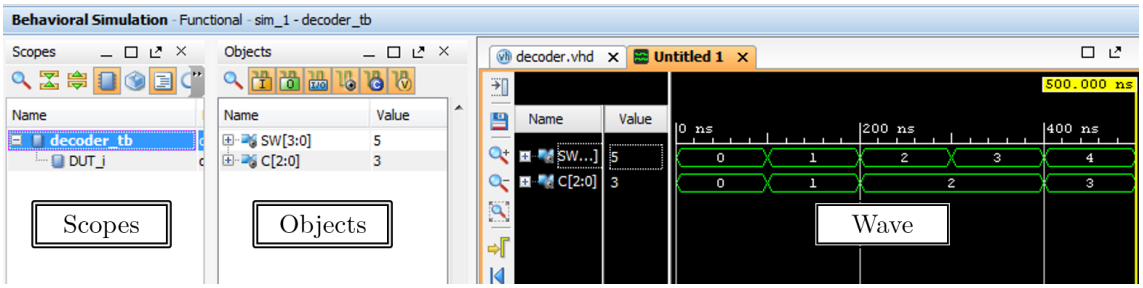
För att starta simuleringen väljer vi *Simulation->Run Simulation->Run Behavioral Simulation* under *Flow Navigator*. Vivado kommer att simulera konstruktionen från tiden 0ns till 500ns och generera insignaler till *decoder* modulen enligt beskrivningen i filen *decoder_tb*. Efteråt visas vågformer av signalerna som tillhör testfilen, se figur 1.9. Några nya menyknappar kommer även att bli tillgängliga, se figur 1.10.



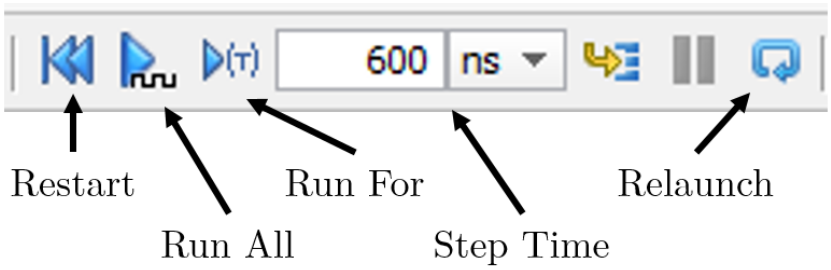
Figur 1.7: Filer som är tillgängliga för simulering finns listade under *Simulation Sources* i *Sources*. Simuleringen kommer att utgå från topp-modulen som är markerad i fetstil då den startas.



Figur 1.8: Inställningar för Vivados inbyggda simulator. Se till att *Simulation top module name* är satt till *decoder_tb* och *xsim.simulate.runtime* är satt till 500ns.



Figur 1.9: När simuleringen körts visas vågformer för signalerna som tillhör testfilen. Det går även att visa vågformer för signaler som tillhör modulen som testas genom att högerklicka på modulens namn i *Scopes* och välja *Add To Wave Window*. Observera att *decoder* modulen är instansierad med namnet *DUT_i* i testfilen



Figur 1.10: Menyknappar som används för att styra simuleringen i Vivado. *Restart* nollställer simuleringen. *Relaunch* startar om simuleringen och kör den till inställd tid i *Simulation Settings*. *Run All* kör simuleringen tills inga nya insignaler genereras. *Run For* stegar fram simuleringen enligt tiden *Step Time*.

Testfilen som vi kör kommer att generera insignaler till *decoder* modulen motsvarande de binära talen 0 till 10 med en paus på 100ns mellan varje tal. Eftersom vi endast körde simuleringen i 500ns så har vissa insignaler inte genererats än, då simuleringstiden var för kort. Ändra *Step Time* till 600ns och klicka på *Run for ....*

För att få en bättre överblick av simuleringsresultat högerklickar vi i *Wave* och väljer *Full View* för att se hela tidsförloppet. De enskilda bitarna hos en signal kan visas via plustecknet framför signalens namn i *Wave*.

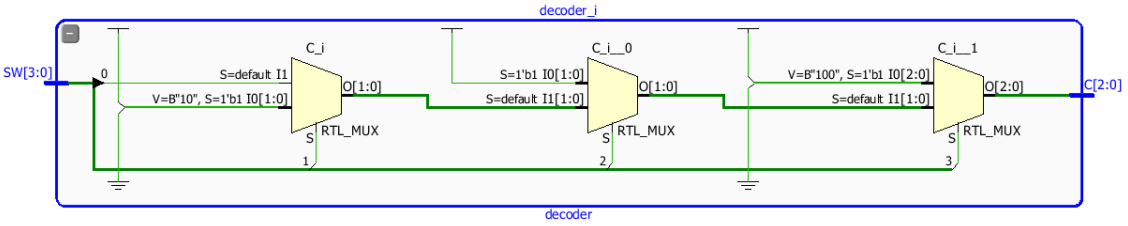
Som standard lägger Vivado till de signaler som tillhör topp-modulen (testfilen) i *Wave*. Genom att markera en annan modul under *Scopes* blir dess signaler synliga i *Objects*. Därifrån kan man högerklicka på en signal och välja *Add to Wave Window*.

Verifiera att *decoder* modulen beter sig korrekt genom att jämföra simuleringsresultatet med specifikationen i tabell 1.2

RTL-schema

Efter att designens beteende beskrivits kan vi se hur Vivado tolkar och översätter detta till hårdvara i ett RTL-schema. Schemat som visas byggs upp med olika logiska block som grindar, muxar, buffrar, adderare och så vidare.

Klicka på *Open Elaborated Design* under *Flow Navigator*. Ett nytt fönster öppnas där vi ser hur de olika modulerna är sammankopplade. Genom att dubbelklickar på *decoder* blocket så går vi ner en nivå och kan se hur avkodarens beteende har översatts till hårdvara, se figur 1.11.



Figur 1.11: RTL-schema över *decoder* modulen. När RTL-schemat först öppnas visas hela konstruktionen med sina olika block. Genom att dubbelklicka på *decoder* blocket går vi ner en nivå och kan se RTL-schemat för blocket

Syntetisering

Efter att designen har simulerats och beteendet verifierats är det dags för syntetisering. Det som sker då är att de olika delarna i RTL-schemat översätts till resurser som finns tillgängliga i den aktuella FPGAn.

I denna kurs använder vi en Artix7 FPGA som innehåller ett stort antal sanningstabeller, D-element, adderare samt ett par andra komponenter. Dock så innehåller den inga grindar som behövs för att forma logiska uttryck. Detta löses under syntetiseringen genom att sanningstabellerna konfigureras så att de realiserar de olika uttrycken i designen.

Innan syntetiseringen kan genomföras måste vi lägga till en fil som talar om för Vivado vilka begränsningar vi har på olika delar i designen. I denna kurs är vi främst intresserade av att bestämma vilka fysiska anslutningspunkter på FPGAn som designens in- och utsignaler ska anslutas till. I den här konstruktionen vill vi till exempel se till att de fyra strömställarna i det nedre högra hörnet på Nexys4 kortet ansluts till designens insignal **SW[3:0]**.

Till vår konstruktion finns det redan en färdig begränsningsfil som vi ska använda. Denna läggs till i projektet genom att välja *Add Source* under *Flow Navigator*. I fönstret som öppnas väljer vi *Add or create constraints* och sedan *Add Files*. Gå till *S:\Courses\EIT\EITA15\Vivado_demo* och välj filen *Nexys4_led_demo.xdc*. Se till att **Copy sources into project** är markerad och klicka på *Finish*. Begränsningsfilen finns nu med i projektet under *Sources ->Constraints*.

Nu kan designen syntetiseras via *Run Synthesis* under *Flow Navigator*. Om inga fel upptäcks frågar Vivado om implementering ska utföras. Här gör vi bäst i att välja *Cancel* och kontrollerna om några varningar har genererats genom att titta under *Project Summary*. Om fönstret inte är synlig kan den öppnas via *Window->Project Summary*. Under rubriken *Synthesis* kan vi se om varningar genererades. Klickar vi på länken för varningar så visas de i fönstret för *Messages/Logs/Reports*.

Det går bra att strunta i varningar om man vet vad det innebär och vilka konsekvenser det får.

Efter syntetisering kan vi se att det har genererats två varningar som båda har samma orsak. Klickar vi på länken för varningarna i *Project Summary* visas *[synth 8-3331] desgin top has unconnected port clk* under *messages*. Varningen beror på att klocksignalen till kretsen inte används.

Tyvärr är det inte alltid lätt att lokalisera orsaken till varningar i Vivado. I det här fallet är det dock enkelt och kan ses i RTL-schemat. Öppna filen för modulen *led_demo* och ändra rad 25 från *CLK => N_RST* till *CLK => CLK*. Spara filen och utför en ny syntetisering. Denna gång ska inga varningar eller fel genereras.

Implementering

Under implementering bestämmer Vivado var i FPGAn som de olika delarna ska placeras samt vilka ledningar som ska användas i FPGAn och hur de ska kopplas samman.

För att starta implementering väljer vi *Implementation->Run Implementation* under *Flow Navigator*. Efter implementering frågar Vivado om vi vill utföra någon mer funktion. Precis som vid syntetiseringen så ser man eventuella fel och varningar under *Project Summary*.

Programmering av FPGA

Efter implementering har vi en design som kan överföras till FPGAn. Men för att konfigurera FPGAn med implementeringsresultatet måste vi generera en konfigurationsfil som kan överföras till FPGAn via USB.

För att generera konfigurationsfilen väljer vi *Program and Debud->Generate Bitstream* under *Flow Navigator*. När processen är klar återstår bara att skicka över filen till FPGAn.

Se till att Nexys4-kortet är anslutet till datorn via USB-kabel. Se till att strömställaren *Power* är i läge *ON*. Välj *Program and Debug->Open Hardware Manager->Open Target* under *Flow Navigator*. I rutan som visas väljer vi *Auto Connect*, Vivado kommer då att leta efter anslutna enheter. Därefter skickar vi över konfigurationsfilen genom *Program and Debug->Hardware manager-> Program Device* under *Flow Navigator* och väljer *xc7a100_0* i rutan som visas.

När FPGAn blivit programmerad kan vi verifiera att hårdvaran beter sig enligt tabell 1.1.

Sammanfattning

Slutligen följer här en liten sammanfattning av designflödet i Vivado som kan användas som referens vid senare projekt.

1. Skapa ett nytt projekt i Vivado. Se till att välja rätt FPGA.
2. Lägg till/skapa källfiler som beskriver hårdvaran.
3. Lägg till/skapa källfiler för testmoduler.
4. Simulera och ändra källfiler iterativt tills konstruktionen beter sig korrekt.
5. Lägg till/skapa begränsningsfilen.
6. Syntetisera designen. Åtgärda eventuella fel och varningar. RTL-schemat kan vara till hjälp.
7. Implementera designen. Åtgärda eventuella fel och varningar.
8. Generera konfigurationsfilen.
9. Programmera FPGAn.
10. Verifiera att hårdvaran beter sig korrekt.
11. Om hårdvaran inte beter sig korrekt kanske beskrivningen av hårdvaran är felaktig. Kanske upptäcktes inte fel vid simuleringen. Eller så struntade någon i varningar som genererades vid syntetiseringen och implementeringen.