Förberedande datorlektioner i Digitalteknik

Petter Källström och Oscar Gustafsson

12 november 2020

Inledning

När man ska göra digitalteknik av en CPLD, så är den rätta vägen att

1. *Konstruera* logiken (rita blockdiagram)
2. *Skriv* VHDL-kod för blocken
3. *Simulera* konstruktionen (koden) i Modelsim för att hitta eventuella fel och *rätta* till dem (genom att göra om tidigare steg)
4. *Syntetisera* koden när den väl fungerar för att få en binärfil som innehåller konfigurationen för CPLDn
5. *Bränn* (programmera) CPLDn
6. *Koppla upp* kretsen och verifiera funktionen

I dessa två lektioner ska vi titta på hur alla dessa steg görs. För att du ska kunna skriva kod, simulera och syntetisera, så börjar vi med hur man kan logga in på distans.

De flesta uppgifter (alla utom 3.1 och 3.2) har *labskal*, som bl.a. innehåller en testbänk för simulering. Den här lektionen är upplagd som en laboration, men med detaljerade instruktioner för hur du löser simulering med mera. "Laborationsuppgiften" är en enkel enpulsare.

Innehåll

1 Fjärrinloggning

När du arbetar på distans så kan du logga in på någon Linux-dator för att editera VHDL-filer, simulera eller syntetisera.

Det finns två olika sätt att koppla upp sig: antingen direkt från din egen dator via ssh till en dator i Muxen eller via ThinLinc till en universitetsdator och sedan via ssh till datorerna i Muxen. ThinLinc är typiskt snabbare men tar över hela skärmen, medan att koppla sig via ssh direkt gör att programmen dyker upp som vilket annat program som helst.

På följande sida finns mer om fjärrinloggning: `https://www.student.liu.se/studentstod/itsupport/linuxdatorsalar/fjarrinloggning?l=sv`

1.1 Inloggning via ssh

För att koppla upp dig direkt med ssh hemifrån så behöver du först ansluta till LiU interna nät via VPN. Detta gäller inte om du sitter på LiU. Läs vidare på `https://www.student.liu.se/itsupport/vpn-forticlient?l=sv`. All trafik kommer nu gå via LiUs datorer, så det kan vara bra att koppla ner sig när arbetsdagen är slut.

Du ska ansluta till någon dator i Muxen 1 eller Muxen 2. De numreras 001 till 016, och heter t.ex. `muxen1-013.ad.liu.se`.

Windows

Om du har Windows, så kan du installera t.ex. MobaXterm, `https://mobaxterm.mobatek.net/`, som har stöd för SSH med grafik. Det finns även andra verktyg. I MobaXterm så väljer du menyn Sessions › New session. I fönstret väljer du SSH och en av datorerna `muxen1-`X`.ad.liu.se` eller `muxen2-`X`.ad.liu.se`, där $X = 001$ till $X = 016$. Port 22 (som bör vara förinställt) Under "Advanced SSH Settings", så ser du till att X11-forwarding och Compression är markerat.

Linux

Om du har Linux, så öppnar du en terminal och skriver:

```
ssh -XC user123@MUXEN-000.ad.liu.se
```

(byt ut `user123` mot ditt användarnamn och `muxen-000` mot en av datornamnen beskrivna ovan).

Mac

Om du har Mac, så kan du testa med programmet XQuartz, `https://www.xquartz.org/`, och därifrån köra samma ssh-kommando som beskrivs i Linux. Har du OSX 10.5 eller äldre så finns motsvarande XQuartz redan installerat.

1.2 Inloggning via ThinLinc

Om du väljer att använda ThinLinc, `https://www.cendio.com/`, så behöver du först ansluta till `thinlinc.edu.liu.se`, öppna en terminal (högerklicka på skrivbordet och väl "Open Terminal Here") och följa instruktionerna för Linux ovan.

Håll koll på ThinLincs "Popup menu key"

Innan du trycker på Connect i ThinLinc, så tryck på Options... och kom ihåg vilken tangent som står som "Popup menu key". Denna tangent kan du trycka på för att få upp en ThinLinc-meny där du t ex kan välja bort helskärmsläget eller minimera ThinLinc och därmed komma tillbaka till ditt vanliga skrivbord.

2 Simulering

Simulering innebär att konstruktionen sätts i en slags virtuel testmiljö. Man stoppar in lämpliga ettor och nollor på ingångarna, och kollar att det som kommer ut beter sig rätt. Detta kan göras manuellt, men det är tidsödande. Enklare är att göra det via en fördefinierad *testbänk*. Testbänken ansluter till konstruktionen som testas (eng. "design under test", DUT). Även testbänken är skriven i VHDL, men använder sig av funktioner i språket som inte går att ha i en CPLD.

Det här med att simulera sin VHDL-kod *är* ett erkänt krångligt kapitel, och lite av en överkurs i det här skedet. Därför har vi skapat s.k. laborationsskal till de uppgifter som ska simuleras, med bland annat färdig testbänk.

2.1 Labskalet

Börja med att ladda ner laborationsskalet för lektionen, det är en .zip-fil med fyra filer. `first_time.do` och `rerun.do` är hjälpskript, som vi återkommer till. `enpulsare.vhd` är själva design-filen, som är påbörjad, men saknar själva logiken. `enpulsare_tb.vhd` är testbänken.

Placera filerna i lämplig katalog, t.ex. `X:\TSEA52\Lektion` i Windows, eller `$HOME/TSEA52/Lektion` i Linux.

2.2 Starta Modelsim

I Windows kan du starta Modelsim från startmenyn. Öppna menyn och skriv "Modelsim", så får du upp förslag.

I Linux kör du först kommandot `module add prog/modelsim` i kommandoraden, och därefter `vsim`.

Stäng eventuell välkommen-ruta.

Navigera till mappen där du har dina filer: File > Change Directory. Figur 1 visar resultatet, och hintar om var du så småninom ska skriva in kommandon för att simulera.

2.3 Designfilen

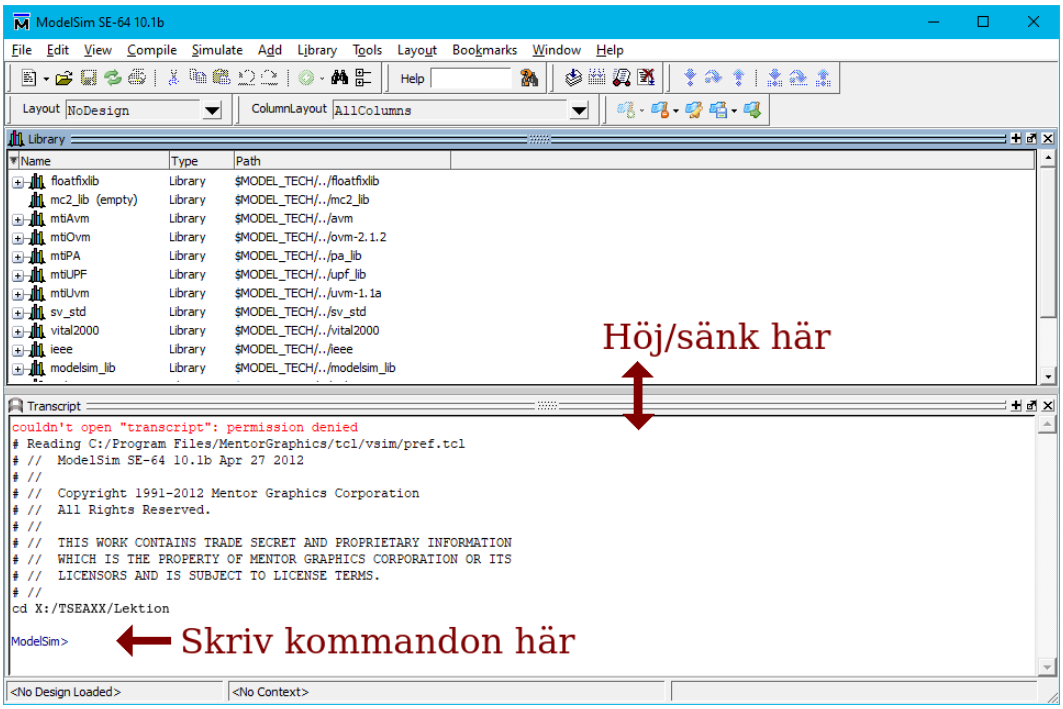
Öppna `enpulsare.vhd` i valfri textredigerare.

Textredigerare

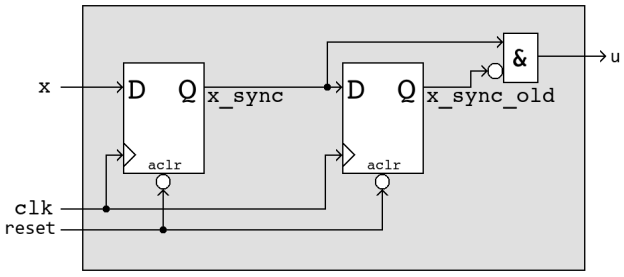
I Windows finns t.ex. Programmers Notepad eller Notepad++. I Linux finns t.ex. `emacs`, `gedit` eller `vim`. Att alternativ är även `atom`, men då måste ni lägga till modulen via `module add prog/atom`. Dessutom har både Modelsim och Xilinx ISE en egen editor.

Designen visas i figur 2. Designfilen har tom arkitektur, precis som i labskalen på Lab 3 och 4. Din uppgift är att fylla den med kod som definierar insidan av kretsen.

Komplettera VHDL-filen med det som står nedan. Det *går* att kopiera, men du lär dig bättre av att skriva av.



Figur 1: Modelsims fönster i Windows, efter att ha navigerat till `X:\TSEAXX\Lektion` (eller där du valt att lägga filerna).



Figur 2: Vår enpulsare, med interna strukturen.

```
library ieee;
use ieee.std_logic_1164.all;

entity enpulsare is
port(clk, reset : in std_logic;
     x : in std_logic;
     u : out std_logic);
end entity;

architecture behav of enpulsare is
   signal x_sync : std_logic;
   signal x_sync_old : std_logic;
begin
-- The two D-flip flops
process(clk, reset) begin
   if reset = '1' then
      x_sync <= '0';
      x_sync_old <= '0';
   elsif rising_edge(clk) then
      x_sync <= x;
      x_sync_old <= x_sync;
   end if;
end process;

u <= x_sync and x_sync_old;

end architecture;
```

Paketet `std_logic_1164` definierar typen `std_logic`, som är våra binära bitar.

Porten definierar pinnarna på chippet.

Arkitekturen beskriver insidan på chippet. Ovanför `begin` deklarerar vi två interna "sladdar".

Kommentarer börjar med `--`.

Med processen definierar vi två D-vippor. Båda klockas med `clk` och nollställs av `reset`.
Den ena läser från `x` och skriver till `x_sync`. Den andra läser från `x_sync` och skriver till `x_sync_old`.

Slutligen skapar vi AND-grinden som skriver till utsignalen `u`. **Notera att det finns en avsiktlig bugg här**, som testbänken ska få hitta.

2.4 Simulera

Modelsim beter sig helt olika, beroende på om det har en design "laddad" i simulatorkärnan eller inte. Nu är det dags att kompilera din kod, testbänken, samt ladda in dem i simulatorn.

Gå till ramen "transcript" i Modelsim, och kör kommandot:

```
do first_time.do
```

Detta kompilerar VHDL-filerna, laddar in testbänken i simulatorn (och via den även din design), lägger till filerna i ett vågfönster, och kör simuleringen. När Modelsim laddar in designen så kommer fönstrena att hoppa runt lite.

Om du får kompileringsfel, så försök rätta till det, och kör sedan kommandot igen (du kan använda ↑ + ↵ för att köra förra kommandot igen).

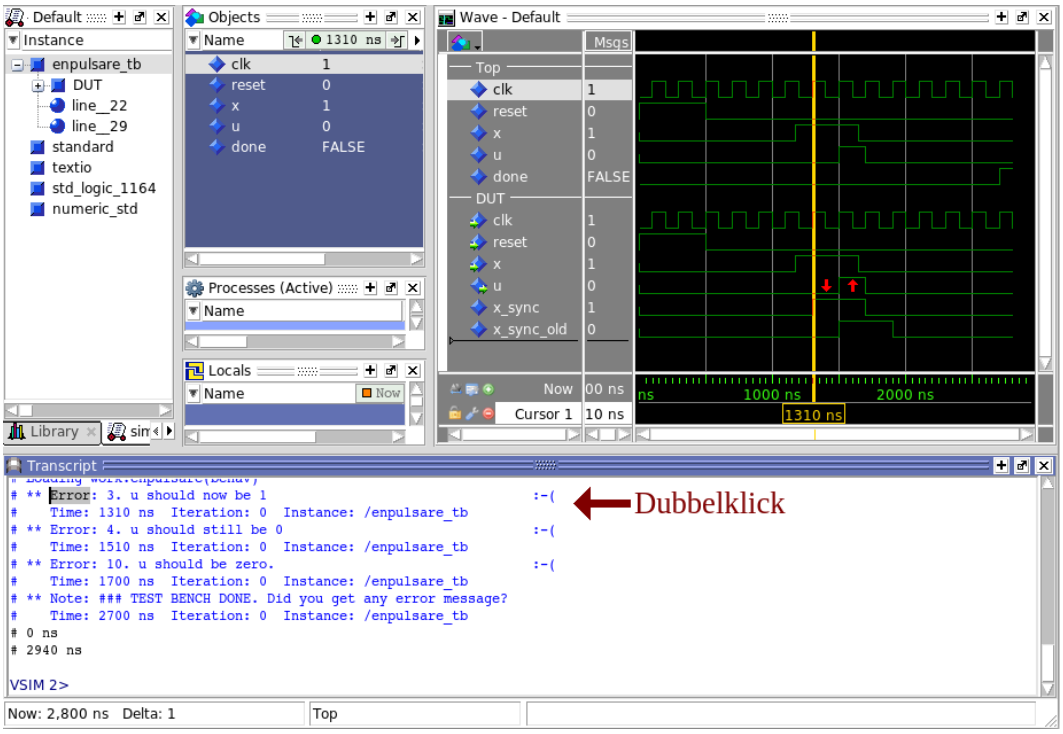
Resultatet ses i figur 3.

2.5 Åh nej, något är fel!

Vi har fått tre felmeddelanden (test nummer 3, 4, 10). Det är i regel bra att ta hand om första felmeddelandet först.

Dubbelklicka på första felutskriften, så får du upp en gul markör i vågfönstret.

Enpulsaren ska ju detektera att `x` just har tryckts ner (alltså inte att `x` *är* nertryckt), genom att se till att `x` är 1, samtidigt som den var 0 förra klockcykeln. Detta är markerat med två små röda pilar i vågformen i figur 3. Så har vi gjort i designen (lilla cirkeln i figur 2).



Figur 3: Simuleringsresultatet, innan buggen rättats.

Men det stämmer ju inte vid gula markören när `x_sync` = 1 och `x_sync_old` = 0. Istället kommer utsignalen när `x_sync` = 1 och `x_sync_old` = 1.

Våra register verkar stämma, men utsignalen `u` gör det inte. Det verkar vara något i den ekvationen. Lägg in den glömda inverteraren på raden för `u` i designfilen:

```
u <= x_sync and not x_sync_old;
```

Spara filen.

I Modelsim måste vi nu simulera om. Det kan vi göra med samma kommando som förut, men när designen nu är laddad så finns ett snabbare alternativ:

```
do rerun.do
```

Detta kompilerar (om) koden och startar om simuleringen "på plats" i simulatorkärnan.

Om du får kompileringsfel, så rätta dessa och kör om (↑ + ⏎).

Om du bara får en röd utskrift som säger att något är kompileringsfel, så kan det hjälpa att dubbelklicka på den för att se mer i detalj vad kompilatorn klagade på.

Nu ska alla felutskrifter vara borta. Dags att börja syntetisera.

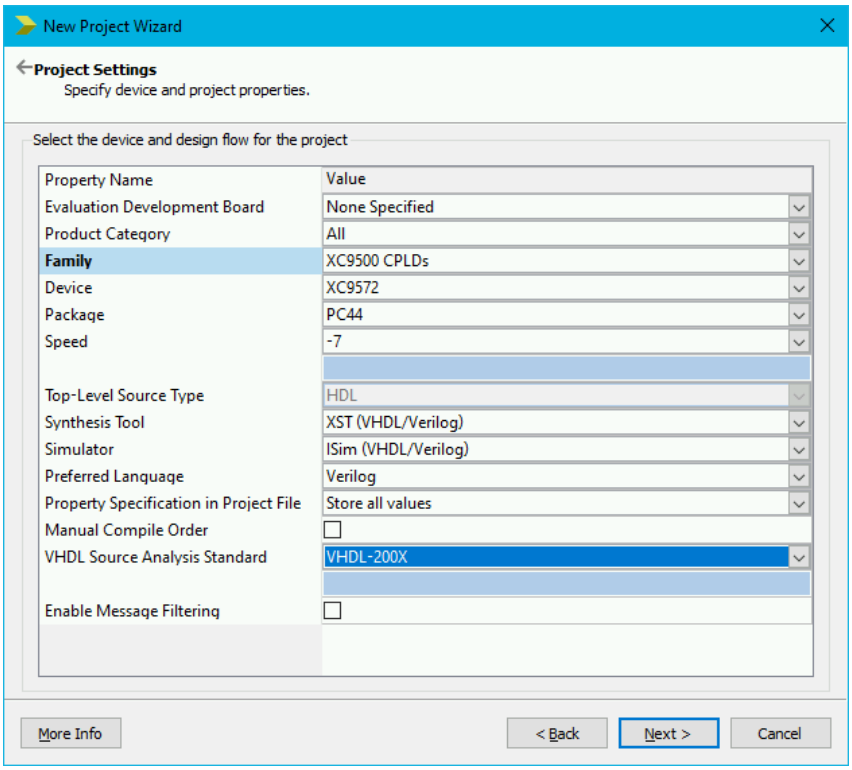
3 Syntes

När kod *syntetiseras* så översätts den till en binärfil, som kan brännas in på en CPLD. Syntesprogrammet tolkar din kod, och gör efter bästa förmåga om den till and/or-uttryck och register. Dessutom bestäms hur dessa ska spridas ut på CPLDs resurser, samt hur de ska kopplas ihop. Slutligen skapas den *.jed*-fil som definierar innehållet i CPLDn.

Starta verktyget ISE

I Windows ligger det i startmenyn under "ISE Design Suit", och heter Project Navigator – välj 64-bitars-versionen.

I Linux får du köra kommandona `module add prog/xilinx_ise` följt av `ise`.



Figur 4: ISE proj

Om du får upp rutan "Did you know...", så stäng den!

Starta ett nytt projekt

Knappen New Project... . Sätt ett bra namn, t.ex. enpulsare. "Location" och "Working Directory" sätter du till där du har dina filer. Se till att Top-level source type är HDL. Klart – klicka på Next > .

Skriv namnet först

OBS! När du skriver in ett namn så kan Location ändras. Skriv därför namnet först.

Nu ska vi berätta för syntesprogrammet vilken CPLD vi har.

- Family = XC9500 CPLDs
- Device = XC9572
- Package = PC44
- Speed = –7
- VHDL Source Analysis Standard = VHDL-200X

Klart – Next > , sen Finish .

Nu kanen ny VHDL-fil skapas, men det är ju redan gjort. Istället ska den befintliga läggas till. Välj Project > Add Source... . Därmed har din konstruktion blivit importerad i projektet, och du ser den som *toppmodul*.

Syntetisera

Klicka på den gröna "play"-ikonen ▶, eller välj Process » Implement Top Module. Det tar ett tag. Om allt går bra, så ska du få upp ett fönster med syntesrapport. Om inte, så kan du klicka på summa-ikonen Σ eller välj Project » Design Summar/Reports.

Design Summary dyker upp som en flik i ISE, med summa-ikonen. Till vänster i den hittar du "Design Overview" / "CPLD Fitter Report".

I Fitter report så finns några noterbara delar (i menyn till vänster).

Under **Summary**, så får man se en summering av hur många makroceller, produkttermer etc. som används. I ett and-or-uttryck, så är det en produkt-term per AND-grind, och en makrocell består av flera produkttermer och en OR-grind. Våran enpulsare använder två D-vippor (kallat register), samt en produktterm.

Under **Equations** så får du se en lista på alla ekvationer som används i CPLD:n. Denna lista är i regel svår att förstå, då den använder konstiga uttryck, som ".LFBK". Med kvalificerade gissningar går det ofta att räkna ut vad som sägs.

Equations finns bara på Windows

Equations finns bara på Windows. Under Linux finns all information under CPLD fitter report (text), men är inte lika enkelt läsbar som på Windows, så det kan vara idé att vänta med den delen tills ni är i labbsalen.

Under **Pin List** får du reda på vilka in- och utgångar som hamnar var på det fysiska chippet. Detta behöver du veta när du ska koppla upp sen.

Kolla runt lite på olika delar här. Speciellt så vill ni titta på ev. fel och varningar. Dessa finns antingen via flikarna längst ned eller under Summary där det finns en länk i tabellen.

Varningar

Vissa varningar går inte att få bort och en del är OK att ha. Det ni ska hålla speciellt utkik efter är varningar om **Latches** och **sensitivity list**. Dessa kan innebära att er konstruktion inte fungerar likadant i simuleringen som när ni har syntetiserat. Det bör inte finnas några sådana varningar för er konstrukion!

Slutligen så har syntesverktyget producerat en .jed-fil (som heter t.ex. `lektion.jed`) i projektmappen. Denna fil innehåll hela binären som definierar innehållet i CPLD:n.

4 Bränning och uppkoppling

På denna lektion ska vi inte bränna eller koppla upp någonting. På labben ska vi göra det, och då ska vi använda .jed-filen.

Labassistenten kommer berätta hur du bränner och kopplar upp. Om du är sugen på att se det i förväg, så finns det en video om det på Sharepoint Video, kanalen VANHEDEN. Se `https://liuonline.sharepoint.com/portals/hub/_layouts/15/PointPublishing.aspx?app=video&p=p&chid=73d46fbc-6ab3-495b-b3bd-12bc047ee2f8&vid=4d62431a-c4f4-4db3-ae14-9b42a43`