Linköpings tekniska högskola, ISY, Datorteknik

Laborationskompendium

Digitalteknik I, HT2
Konstruktion av mindre digitala system

Linköping 2020

Laboration 3

Programmerbara kretsar och VHDL

Syftet med laborationen är dels att öva på konstruktion av mindre digitala system, och dels att ge en inblick i moderna konstruktionshjälpmedel för digital konstruktion.

Efter genomförd laboration ska ni:

- Bland annat genom att rita blockschema, kunna konstruera mindre digitala system med hjälp av programmerbar logik.
- Provat på det hårdvarubeskrivande språket VHDL.
- Ha insikt i modern systemutvecklingsmetodik.

Laborationsutrustningen kommer att kompletteras med en modul som innehåller en CPLD (Complex Programmable Logic Device) av typen XC9572 tillverkad av Xilinx. Den innehåller grindar och vippor motsvarande ungefär fyra labplattor.

Först ges en kort introduktion i VHDL med diverse kod-exempel, sedan följer en beskrivning av programvaran som behövs och uppgifterna kommer sist.

> Denna beskrivning fokuserar på programvaran ni använder när ni väl ska programmera CPLD:n. För att göra delar på distans så hänvisas till lektionsmaterialet.

3.1 Kort VHDL-introduktion

Börja med att läsa igenom kapitel 1.4 i läroboken samt läsa igenom följande exempel. Dessa exempel utgör en sammanfattning av den VHDL-kod som ni nu bör klara av att förstå, samt skriva själva.

Det första exemplet visar kod för några vanliga grindar.

Exempel 3.1 Här följer kod för några enkla Booleska funktioner:

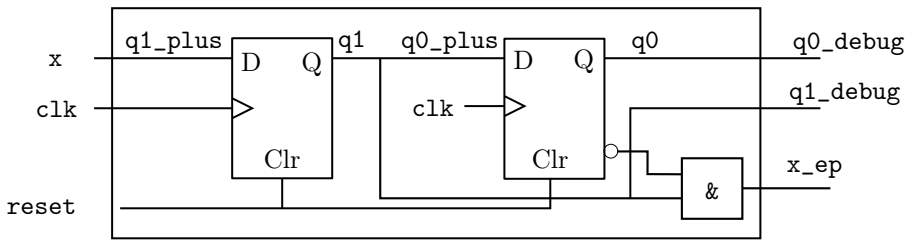
```
library ieee;
use ieee.std_logic_1164.all;

entity K_Net is
  port(x, y, z : in std_logic;
       a, b, c, d : out std_logic);
end K_Net;

architecture equations1 of K_Net is
begin
  a <= not z;                                -- Inverterare
  b <= not (x and y);                        -- NAND-grind
  c <= ((not x) and y) or ((not y) and x); -- XOR-grind
  d <= x xor z;                              -- XOR-grind
end equations1;
```

I nästa exempel introduceras två nya saker:

1. Vippor, vilket skapas med hjälp av en `process`-sats. Med en `process`-sats kan man göra många olika saker, men vi nöjer oss för enkelhetens skull med att bara skapa vippor och senare räknare. Dessa kommer att bli positivt flanktriggade enligt kodexemplet. För mer information om flanktriggning, se avsnitt 2.1.

2. Interna signaler: `q1`, `q1_plus`, `q0`, `q0_plus`. Dessa signaler kommer syntesverktyget att "trolla bort", men brukar underlätta läsförståelsen, samt gynna ett strukturerat tänkande. Eftersom det inte är tillåtet att läsa av värdet på en utsignal (`out`-port) i VHDL så är det dessutom ibland nödvändigt att lägga till dessa.

Exempel 3.2 Figuren visar ett komplett logiskt kopplingsschema med införda signalnamn för en enpulsare.



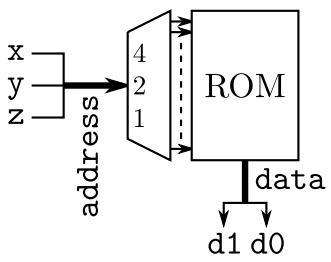
Denna krets översätts till följande VHDL-kod:

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

entity Enpulsare is
  port(clk, reset, x : in std_logic;
       x_ep : out std_logic;
       q1_debug, q0_debug : out std_logic);
end Enpulsare;

architecture equations2 of Enpulsare is

  signal q1, q1_plus : std_logic;
  signal q0, q0_plus : std_logic;
```

```
begin
  -- Vippor
  process(clk, reset) begin
    if reset = '1' then
       q1 <= '0';                -- Asynkron reset
       q0 <= '0';                -- av alla D-vippor
    elsif rising_edge(clk) then
       q1 <= q1_plus;            -- Skapar en D-vippa
       q0 <= q0_plus;            -- Skapar en D-vippa
    end if;
  end process;

  -- Tillståndsuppdatering
  q1_plus <= x;
  q0_plus <= q1;

  -- Utsignaler
  x_ep <= (not q0) and q1;
  q1_debug <= q1;              -- Vippornas tillstånd blir
  q0_debug <= q0;              -- synliga utanför CPLD:n
end equations2;
```

Nästa exempel visar hur ett ROM kan implementeras samt hur vektorer kan skapas och delas upp.

Exempel 3.3 Följande ROM med storleken 8×2 bitar ska implementeras för de angivna minnesinnehållet.



x	y	z	d_1	d_0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

Motsvarande VHDL-kod är:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;  -- Paket för att räkna

entity ROM1 is
  port (x, y, z : in std_logic;
        d0, d1 : out std_logic);
end ROM1;

architecture arch of ROM1 is
  -- Definition av ROM-minnet:
  type ROM_mem is array(0 to 7) of std_logic_vector(1 downto 0);
  constant ROM_content : ROM_mem := (0 => "00",
                                     1 => "01",
                                     2 to 3 => "10",
                                     4 => "00",
                                     5 => "01",
                                     6 => "10",
                                     7 => "11");

  -- Övriga signaler
  signal address : std_logic_vector(2 downto 0);
  signal data : std_logic_vector(1 downto 0);

begin
  -- Tilldela bitarna in till address-vektorn
  address(2) <= x;
  address(1) <= y;
  address(0) <= z;
```

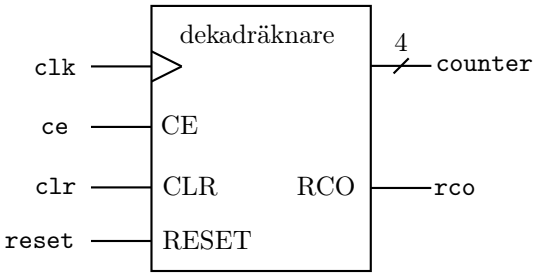
```
  -- Läs ut data
  data <= ROM_content(to_integer(unsigned(address)));

  -- Tilldela utbitar
  d0 <= data(0);
  d1 <= data(1);
end arch;
```

Alternativt används en `with-select`-sats enligt:

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

entity ROM1 is
  port (x, y, z : in std_logic;
        d0, d1 : out std_logic);
end ROM1;

architecture arch2 of ROM1 is
  -- Signaler
  signal address : std_logic_vector(2 downto 0);
  signal data : std_logic_vector(1 downto 0);

begin
  -- Tilldela bitarna in till address-vektorn
  address <= x & y & z;  -- Konkatererning är ett alternativ

  -- Läs ut data från ROM-minnet
  with address select data <=
    "00" when "000" | "100",
    "01" when "001" | "101",
    "10" when "010" | "011" | "110",
    "11" when others;  -- Enklast att ha others på sista

  -- Tilldela utbitar
  d0 <= data(0);
  d1 <= data(1);
end arch2;
```

Det avslutande exemplet visar hur en räknare kan implementeras i VHDL.

Exempel 3.4 Följande dekadräknare med asynkron reset `reset`, synkron clear `clr`, count enable `ce` och ripple carry out `rco` ska konstrueras.



Nedan följer koden för räknaren.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;                -- Paket för att räkna

entity counter is
   port(clr, clk, ce, reset : in std_logic;
        q : out std_logic_vector(3 downto 0);
        rco : out std_logic);
end entity;

architecture rtl of counter is
  signal q_int : unsigned(3 downto 0); -- Typ för att räkna

begin
-- tillståndsuppdatering
process(clk, reset) begin
  if (reset = '1') then                  -- Asynkron reset
     q_int <= to_unsigned(0,4);
   elsif rising_edge(clk) then
    if (clr = '1') then                  -- Synkron clear
      q_int <= to_unsigned(0,4);
    elsif (ce = '1') then                -- Count enable
      if q_int = 9 then
        q_int <= "0000";
      else
        q_int <= q_int + 1;              -- Aritmetisk operation
      end if;
    end if;
  end if;
end process;

-- utsignaler
rco <= '1' when ((ce = '1') and (q_int = 9))
        else '0';
q <= std_logic_vector(q_int);            -- Typkonvertering
end rtl;
```

För att räkna behövs paketet `ieee.numeric_std.all` som inkluderas på rad tre. På första raden i architecture definieras `q_int` som en `unsigned`, detta för att det ska gå att räkna med `q_int` som till exempel på raden `q_int <= q_int + 1;`. Sista raden typkonverterar `q_int` från `unsigned` till `std_logic_vector` som är typen på `q`.

Den asynkrona reset-signalen ska bara användas för att manuellt återstarta räknaren och kopplas typiskt till en resetknapp. Om räknaren ska nollställas under drift används den synkrona clear-signalen.

3.2 Programvara

Mjukvaran för att programmera CPLD:erna består av två delar. Nedan beskrivs:

- XILINX ISE Project Navigator är en programvara för att mata in en VHDL-beskrivning och spara den som en VHDL-fil. Programvaran kan kompilera VHDL-filen och skapa en .jed-fil som exakt beskriver hur den beskrivna kretsen ska passas in i aktuell CPLD. Processen att skapa jed-filen kallas för syntetisering.
- Xilinx ISE iMPACT är mjukvara som används för att programmera CPLD:n enligt jed-filen som anger hur hårdvaran ska konfigureras.

Programvaran finns i ISY:s datorsalar med Windows.

3.2.1 XILINX ISE Project Navigator

XILINX ISE Project Navigator är ett verktyg för att syntetisera VHDL-kod till programmerbara kretsar så som CPLD och FPGA. Här följer en kortfattad handhavandebeskrivning:

- Starta 64-bitars ISE Project Navigator (ISE Design Suite 14.5)
- Välj File » New » Project. En dialogruta öppnas och du väljer ett filnamn (namn på projektet), en katalog (där du vill spara projektet) samt HDL Top-level Source Type. (Exempelvis kan katalogen döpas till VHDL-lab och projektet till uppgift8.)
- Klicka Next och dialogrutan Project Settings öppnas. Välj Family: XC9500 CPLDs, Device: XC9572 enligt anvisningar om tillåten komponent, Package: PC44
- Klicka Next och därefter Finish.
- Välj Project » New » Source. En dialogruta öppnas och du väljer ett filnamn samt VHDL Module.
- Klicka Next och dialogrutan New Source Wizard öppnas. Här bestämmer du vilka in- och ut-signaler din konstruktion ska ha. (Detta kan i ett senare skede ändras om så behövs.)
- Klicka Next och därefter Finish.
- I den källkodsfil som nu öppnas ska du skriva in dina ekvationer på samma sätt som exemplen har visat.
- När källkodsfilen är klar välj File » Save
- Välj Process » Implement Top Module för att syntetisera koden. En "jed-fil" skapas, samt en rapportfil dyker upp på skärmen. I rapportfilen kan ni titta på Pin List, för att se hur ni ska koppla er konstruktion. Om Fitter Report inte dyker upp kan ni öppna den via fönstret Design Summary.
- Jed-filen ska flyttas över till en annan katalog för att bli tillgänglig på den dator där kretsen programmeras. För detta ändamål finns det monterat en katalog på din dator som heter "L:". Här skapar ni en egen mapp åt er där ni kan placera era jed-filer.
- RTL-viewer, är en funktion där ni kan se hur er konstruktion blev efter syntetiseringen. Välj Tools » Schematic Viewer » RTL, samt följ vidare instruktioner. Här får ni reda på hur syntesverktyget har tolkat er konstruktion på blocknivå. Välj därefter Tools » Schematic Viewer » Technology. Då får ni se hur er konstruktion blir implementerad på CPLD:n. På alla in- och utgångar finns en extra komponent som kallas buffer. Den har till uppgift att elektriskt skydda CPLD:n samt säkerställa en hög signalkvalité. Det logiska värdet på bufferns in- och utsignal är lika.

3.2.2 XILINX ISE iMPACT

XILINX ISE iMPACT är ett verktyg för att programmera bland annat CPLD:er. Be gärna en handledare om att vara med första gången.

- Kontrollera först att spänningsaggregatet är av.
- Sätt därefter i kretsen åt rätt håll, samt slå på spänningsaggregatet.
- Starta programmet iMPACT samt svara på frågorna med OK, No, No och Cancel.
- Dubbelklicka på Boundary Scan.
- Välj Initialize Chain så att verktyget hittar kretsen.
- Klicka på kretsen och välj Assign New Configuration File. Här ska du leta upp din "jed-fil".
- Klicka på kretsen igen och välj Program, samt därefter OK.
- Slå slutligen av spänningsaggregatet och ta ut kretsen genom att trycka på ramen runt kretsen.

3.3 Uppgifter

Uppgift 3.1. Implementera kretsen i exempel 3.1 med hjälp av VHDL och en CPLD av typen XC9572. Använd skjutomkopplare som ingångar och visa utgångarna med hjälp av lysdioder. I rapportfilen Fitter Report, hittar ni bland annat en Pin List som ni behöver för att kunna koppla rätt. Syntesverktyget har valt lämpliga pinnar åt er och detta kan ändras mellan olika syntetiseringar så var uppmärksam på hur ni ska koppla.

Uppgift 3.2. Valfri. Implementera kretsen i exempel 3.3 i en CPLD. Använd skjutomkopplare som ingångar och visa utgångarna med hjälp av lysdioder.

Uppgift 3.3. a) Konstruera sekvensnätet i uppgift 2.2 med hjälp av VHDL och en CPLD av typen XC9572. Till er hjälp har ni exempel 3.1 och 3.2 i VHDL-introduktionen i avsnitt 3.1. Använd de uträknade booleska uttrycken från uppgift 2.2.

b) Under Fitter Report hittar ni en flik Equations, där ni ser vilka ekvationer som syntesverktyget har producerat. Jämför det med det som ni erhöll i uppgift 2.2.

Resultat: .....

c) Lös nu samma uppgift som i uppgift 2.2, men i stället för booleska uttryck ska ni använda er tabell över PROM:et från uppgift 2.1. Till er hjälp har ni exempel 3.3 i VHDL-introduktionen i avsnitt 3.1.

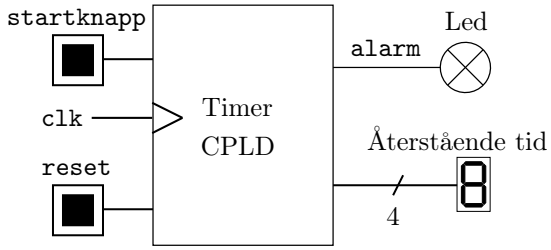
Gå in i syntesverktyget och jämför de ekvationer som syntesverktyget har producerat med det som ni erhöll i uppgift 3.3 (b). Verifiera också att utsignalerna är enligt Moore. (Uppgift 3.3 (c) behöver inte kopplas upp utan är av teoretisk karaktär.)

Resultat: .....

Slutsats: .....

d) **Valfri.** Lös nu återigen samma uppgift som i uppgift 2.2, men skriv nu VHDL-koden som motsvarar en direkt översättning av tillståndsdiagrammet istället. Till er hjälp har ni kapitel 9.3 i Hemert.

Uppgift 3.4. En timer ska konstrueras enligt följande skiss



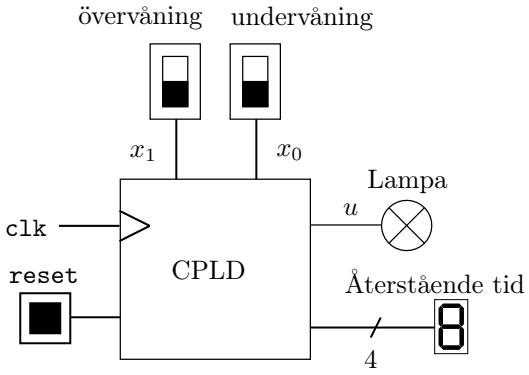
När startknappen trycks ner ska siffran på displayen hoppa till 8 och därefter räkna ner till 0 och stanna där tills nästa gång timern aktiveras. LED-lampan ska lysa om och endast om displayen visar noll. Insignalen från startknappen måste synkroniseras och enpulsas. Timern ska inte kunna startas om med startknappen under pågående nedräkning. Använd klockmodulen med variabel frekvens och ställ in på ca 1 Hz. Det gör inget om nedräkningen börjar någon klockpuls efter det att knappen har tryckts ner. Det ska även finnas en asynkron reset som nollställer alla register. Asynkron nollställning kan även ske under nedräkning.

Förberedelse: Rita blockschema och namnge alla signaler.

Laborationsuppgift: Översätt block för block till kod och använd namnen i blockschemat i koden. Implementera kretsen och verifiera kretsens funktion.

Tips: Modifiera koden i exempel 3.2 och 3.4.

Uppgift 3.5. Konstruera och realisera en trappbelysningslogik som består av en lampa samt två omkopplare placerade högst upp respektive längst ner i trappan. Manövreras en omkopplare när ljuset är släckt skall ljuset tändas och tvärtom. Om båda brytarna byter läge i samma klockintervall ska lyset byta mod. För att spara energi skall belysningssystemet dessutom förses med en timer som automatiskt släcker ljuset cirka 15 s efter det att det tänts. Även om automatisk släckning skett ska inga extra åtgärder behöva vidtagas nästa gång man ska tända. Det ska även finnas en asynkron reset som nollställer alla register.



Använd en CPLD för att realisera kretsen, skjutomkopplare som ”strömbrytare”, en knapp för reset, en sjusegmentsdisplay för timerfunktionen och en lysdiod som lampa. Det är okej att visa återstående tid hexadecimalt.

Extrauppgift: Använd två sjusegmentsdisplayer och visa timern decimalt.

Förberedelse: Rita blockschema med namngivna variabler.

Laborationsuppgift: Koppla upp kretsen och verifiera dess funktion.

3.4 Testbänken för uppgift 3.4

Här följer en beskrivning av testbänken för att förstå hur en testbänk kan byggas upp och vad felen betyder.

Först av allt, så skapas en klocka på 1 MHz. Då blir varje klockcykel 1 mikrosekund (μs), eller 1000 nanosekunder (ns). I verkligheten är klockan 1 Hz, vilket motsvarar 1000000000 ns, men det blir så jobbigt att läsa tidutskrifter då, så vi kör med 1000 ns istället.

Ett antal operationer/tester körs efter varandra:

- Nollställ kretsen några klockcykler.
- Test 0: Ett enkelt test.
  - Sätt `startknapp` till '1'.
  - Vänta på att `tidkvar` > 0 inom 4 klockcykler.
  - Om räknaren inte kommit igång skrivs "Tidkvar fortfarande 0".
  - Kolla att räknaren började på 8, annars skrivs "Vill: Tidkvar = 8".
  - Vänta sju klockcykler.
  - Kolla att räknaren räknat ner till 1, annars "Vill: Tidkvar = 1".
  - Två klockcykler till, så ska räknaren ha stannat på noll. "Vill: Tidkvar stannar paa 0".
- Test 1: Asynkron reset.
  - Sätt `startknapp` till '1' under hela testet.
  - Vänta några klockcykler, så att räknaren börjat.
  - Sätt `reset` till '1' under en kort stund mellan två klockflanker.
  - Testa att `tidkvar` $= 0$, annars "Vill: Tidkvar=0 efter reset"
  - Fler klockcykler. Då enpulsarens vippor också ska ha nollställts, så ska den fortsatta startknappen = 1 trigga en ny start. Kolla att `tidkvar` > 0, annars "Vill: Tidkvar startar om efter reset".
- Test 2: Utdragen startknapp.
  - Nollställ kretsen, och sätt `startknapp` till '1'.
  - Vänta ca 14 klockcykler. Då ska räknaren ha nått 0 och inte startat om. "Vill: Tidkvar=0 efter utdragen startknapp='1'".
- Test 3: Synkron start.
  - Stäng av startknappen. Dra i reset en kort stund. Vänta en klockcykel.
  - Sätt `startknapp` till '1' under fallande klockflank, men ej ända till nästa stigande. Det ska alltså inte registreras.
  - Vänta några klockcykler. Kolla att räknaren inte startat. "Vill: Tidkvar=0 efter en kort startknapp".
- Test 4: Prova att starta om i mitten
  - Sätt `startknapp` till '1' under en klockcykel.
  - Vänta några klockcykler, så att räknaren räknar.
  - Sätt `startknapp` till '1' igen under resten av testet. Nu ska räknaren inte starta om.
  - Vänta några klockcykler. Nu ska räknaren ha kommit ner till 3 eller lägre (om den inte startade om).
  - Kontrollera att `tidkvar` ≤ 3, annars "Vill: tidkvar <= 3".

Parallellt med detta kollas hela tiden att `alarm` = '1' om och endast om `tidkvar` $= 0$, annars skrivs "Alarm = 0 medan tidkvar = 0" eller "Alarm = 1 medan tidkvar > 0". Dessutom kollas att utsignalen är synkron, och endast ändrar sig när klockan har positiva klockflanker. Annars skrivs "Output active between rising_edge(clk)".

Som förstås, så kan ett enda fel i hårdvaran orsaka många felutskrifter. Börja med översta och försök lösa det, och simulera sedan om kretsen.

OBS. Utskrifter i VHDL kan inte hantera å, ä, ö, vilket begränsar formuleringarna i utskrifterna.

Laboration 4

Konstruktion av mindre digitala system

Syftet med laborationen är dels att öva på konstruktion av mindre digitala system, och dels att ge en utökad inblick i moderna konstruktionshjälpmedel för digital konstruktion.

Efter genomförd laborations ska ni:

- Genom att rita blockschema, kunna konstruera mindre digitala system med hjälp av programmerbar logik.
- Ha befäst grunderna i det hårdvarubeskrivande språket VHDL.
- Kunna använda ModelSim för enklare simuleringar.
- Ha insikt i modern systemutvecklingsmetodik.

Laborationsutrustningen kommer att kompletteras med en modul som innehåller en CPLD (Complex Programmable Logic Device) av typen XC9572 tillverkad av Xilinx, en tidbasmodul som kan skicka ut pulser med 1, 10, 100 eller 1000 Hz, samt en pulsgivare. Extrakomponenterna beskrivs i de uppgifter där de används.

4.1 Examinationskrav

För godkänd laboration krävs att uppgift 4.1 är godkänd samt en av uppgifterna 4.2 och 4.3. Laborationen är uppdelad på tre tillfällen om två timmar var.

I den här kursen ligger fokus på grundläggande digitalteknik med enkla tillämpningar som exemplifierar kursinnehållet. Den VHDL som ingår i kursen är därför bara ett absolut minimum. För att bli godkänd på en uppgift krävs:

- Ett detaljerat blockschema över konstruktionen där man på ett tydligt sätt kan se kopplingen till er VHDL-kod.
- VHDL-kod som motsvarar blockschemat.
- En fungerande krets.

Varje uppgift ska lösas i ett eget VHDL-projekt i en VHDL-fil. Varje block i blockschemat ska vara någon av de block som är definierade i dokumentet Digitaltekniska byggblock eller grindar/inverterare. För varje block i blockschemat ska motsvarande kod finnas i VHDL-filen. Motsvarande kod definieras i Digitaltekniska byggblock. Det ska alltså vara en ett-till-ett-matchning mellan blocken

i blockschemat och motsvarande kod-snuttar i VHDL-filen. På detta sätt blir det en process-sats per block innehållande register. Blocken får modifieras efter behov, t ex kan en räknare anpassas så att den får önskat antal bitar och önskade standardfunktioner för en räknare. Om ett block realiserar en egendesignad sekvenskrets ska även tillståndsdiagram ingå i dokumentationen.

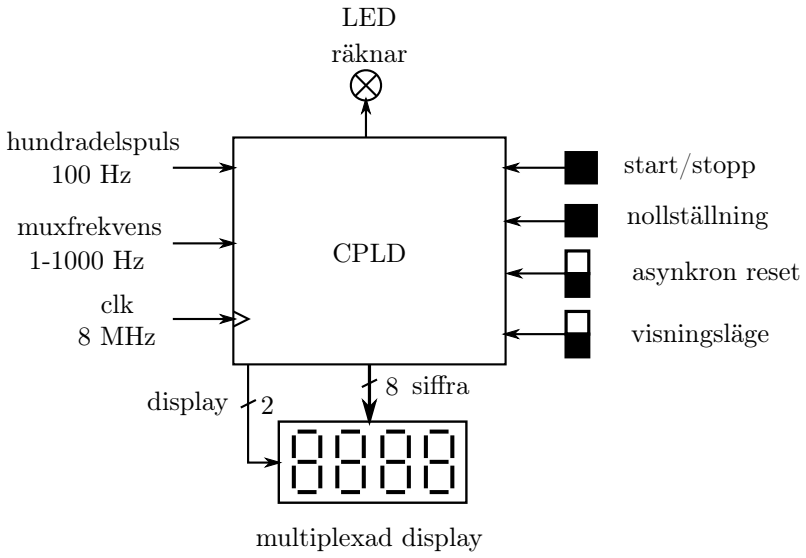
4.2 Förberedelser

Det ska finnas ett ritat blockschema till varje uppgift som ni har för avsikt att göra på laborationstiden. Det är även bra om ni har skrivit den mesta av koden före laborationen. Till uppgift 4.1 och 4.2 finns det även simuleringsövningar som ska redovisas. Inför det avslutande passet ska även lösningarna vara provsyntetiserade i förväg.

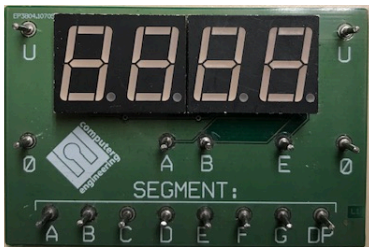
4.3 Uppgifter

Uppgift 4.1. Konstruktion av ett stoppur.

Ett stoppur som styrs av två knappar och två skjutomkopplare ska konstrueras. Uret ska räkna minuter, sekunder och hundradelar. Den ena knappen används som start/stopp och den andra för nollställning. Om tiden är stoppad och startas igen fortsätter räkningen från visad tid. LED-lampan indikerar statusen på start/stopp på så sätt att tänd lysdiod betyder att klockan går. Den ena skjutomkopplaren styr vilka fyra siffror som ska visas på den multiplexade displayen. Om skjutomkopplaren är i nedre läget visas sekunder+hundradelar och om skjutomkopplaren är i sitt övre läge visas minuter+sekunder. När 1 timma har gått börjar tiduret om från 0. Den andra skjutomkopplaren ska i övre läget aktivera asynkront reset.

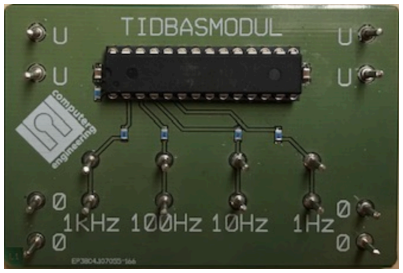


Konstruera det digitala systemet. Använd en CPLD av typen XC9572, den multiplexade displayen



en lysdiod och specificerade kappar och skjutomkopplare. Alla insignaler utom den asynkrona resetsignalen **måste synkroniseras.**

Använd kristalloscillatorn inställd på 8 MHz som systemklocka, clk, som finns på matningsskenan till vänster på labplattan. Muxfrekvensen hämtas från den variabla klockpulsgeneratorn och hundradelspulsen från en tidbasmodul



Systemklockan, clk, är den enda signal som får kopplas till klockingångar på register/vippor varför hundradelspuls och muxfrekvens ska behandlas som ”count enable”-signaler.

Förberedelser:

a) Som laborationsförberedelse ska en simulering i ModelSim över tre kaskadkopplade BCD-räknare genomföras, välj sekundsiffrorna och första minutsiffran. (Jämför gärna med uppgift 2.3c) på laboration 2 där ni kopplade upp tre kaskadkopplade BCD-räknare samt uppgift 2 på lektion 7.) Resultatet ska redovisas med hjälp av kod och en sparad bild från simuleringen alternativt direkt i simuleringsfönstret i ModelSim. (Man ska alltså kunna se en rippel-carry vid omslaget 59→00 och ni behöver visa två stycken sådana omslag i simuleringen.)

   Rita blockschema för den simulerade kretsen med namngivna variabler.

b) Rita ett blockschema för hela tidtagaruret och översätt till VHDL-kod.

   Det kan vara bra att utveckla kretsen inkrementellt. Utgå till exempel från den simulerade kretsen i a)-uppgiften. Bygg ut kretsen t ex med hundradelar. Kompilera och simulera för att verifiera att delsystemet fungerar som förväntat. Bygg sedan ut kretsen steg för steg och verifiera genom kompilering och simulering innan ni går vidare. Avsluta med att provsyntetisera kretsen i Xilinx ISE.

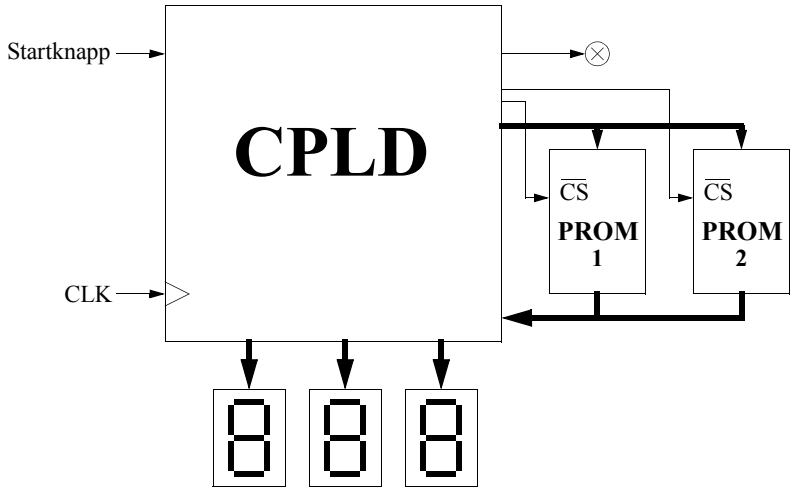
Laborationsuppgifter:

c) Koppla upp och verifiera kretsen.

d) Multiplexning innebär att man visar en siffra i taget och om detta sker tillräckligt fort kommer ögat att uppfatta detta som att alla siffror visas samtidigt. Mät med hjälp av en frekvensräknare upp den displayuppdateringsfrekvens som krävs för att erhålla en flimmerfri visning av siffrorna.

Resultat: .....

Uppgift 4.2. Räkna ettor i PROM. Minnesinnehållet i labsatsens PROM ligger kvar även vid spänningsbortfall. Konstruera ett digitalt system som räknar det totala antalet ettor i två parallellkopplade PROM, se figur. Resultatet ska presenteras i BCD-form. Varje räkning ska startas med en knappnedtryckning, vilken alltså inkluderar såväl nollställning som start. En lysdiod ska vara tänd under tiden som räkningen pågår.



Använd VHDL och en CPLD samt lämpliga in- och utsignaler. Använd de tre avkodade sjusegmentsdisplayerna för att visa antalet ettor i PROM:en. Klockfrekvensen bör vara max 100 Hz, ty vi vill kunna se när konstruktionen arbetar. Osynkroniserade insignaler måste synkroniseras. Eftersom adressen till PROM:en är synkron och klockfrekvensen relativt låg, kan utdata från PROM:en räknas som synkrona, dvs PROM räknas som en rent kombinatorisk krets. Parallellkopplade PROM innebär att det är en gemensam adress- och data-buss, och att vi därför behöver utnyttja PROM:ens "tri-state"-funktionalitet på datautgången, vilket styrs med "chip select"-signalerna. Anslut en asynkron reset till kretsen för nollställning efter spänningspåslag.

Tips: Mängden hårdvara som behövs kan minimeras om konstruktionen tar minst 128 klockpulser på sig för att lösa uppgiften.

Förberedelser: Innan ni börjar med att skriva koden till uppgiften läs avsnitt 4.4 som beskriver hur ni kan simulera koden. Rita blockschema, översätt till VHDL-kod, simulera i en testbänk enligt beskrivningen i avsnitt 4.4 samt provsyntetisera koden.

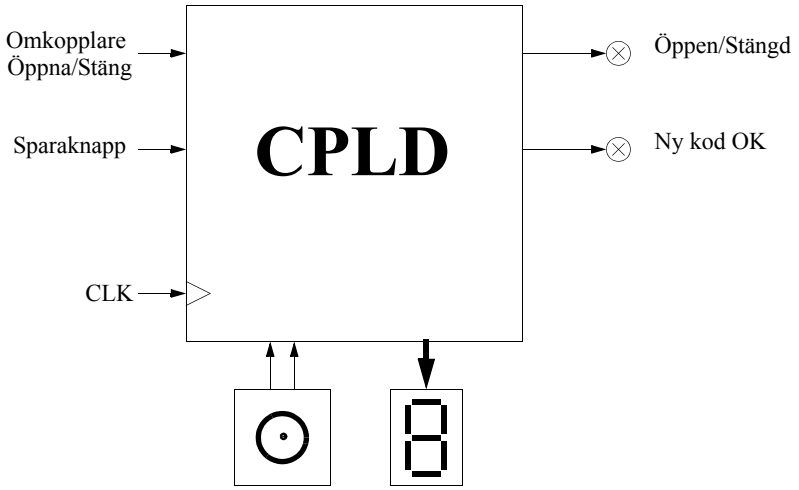
Kretsen ska ge rätt summa oberoende av hur PROMen programmeras. Det finns 2^{128} olika sätt att programmera PROMen, vilket är ett orimligt stort antal tester att utföra. Kan vi med ett rimligt antal tester (olika programmeringar av PROMen) vara säkra på att kretsen fungerar korrekt för samtliga fall? Hur ser dessa tester ut i så fall? Frågorna är viktiga men det finns inget lätt svar på frågorna och svaren beror också på er implementation.

Laborationsuppgifter:

a) Visa resultaten av simuleringarna med testbänken.

b) Visa blockschema, kod och fungerande krets.

Uppgift 4.3. Konstruktion av ett kassaskåpslås. Konstruera styrelektroniken till ett enklare kassaskåpslås, som har en tvåsiffrig kod. Koden skapas bland annat med hjälp av en ratt som sitter på en pulsgivare vars funktion visas på nästa sida.



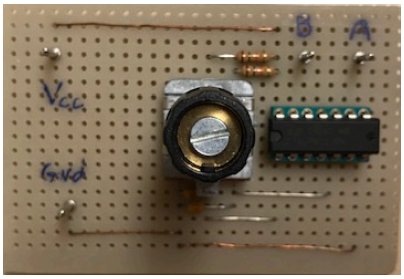
Del a) samt antingen del b) eller del c) ingår i uppgiften.

a) Koda av pulsgivaren (ratten) så att den kan styra den Upp/Ner-räknare som visas på displayen. Siffran är decimal och ska räkna upp vid vridning åt höger samt räkna ner vid vridning åt vänster. Räkningen ska ske med ett steg/puls från pulsgivaren. Siffran ska nollställas varje gång öppnaknappen förs till sitt nedre läge, dvs en övergång från 1 till 0 ska detekteras för nollställningen.

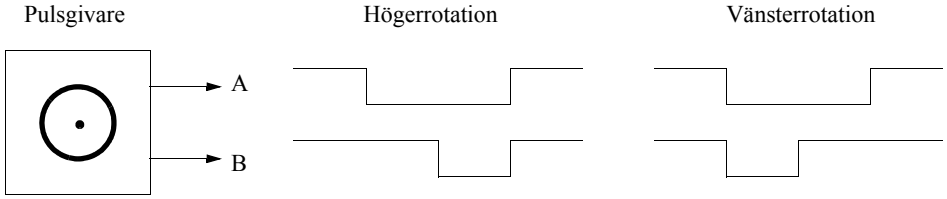
b) För att öppna låset ska först skjutomkopplaren föras till sitt nedre läge varvid en nollställning av inmatad sekvens sker. Därefter används ratten i kombination med en sparaknapp för att mata in sifferkombinationen, dvs när rätt siffra visas på displayen kan den sparas genom ett tryck på sparaknappen. Man kan trycka på sparaknappen hur många gånger som helst i och med att det bara är de två senaste sparade siffrorna som gäller. Om rätt kombination är inmatad när öppnaknappen förs till sitt övre läge ska låset öppnas, vilket indikeras av en tänd lysdiod. När låset är öppet kan man ändra på koden genom att mata in en ny sifferkombination varvid man kan se det som att det skiftas in en ny siffra i koden för varje tryckning på sparaknappen. Låset stängs genom att öppnaknappen förs till sitt nedre läge, vilket medger ett nytt öppningsförsök. Lysdioden för ny kod OK saknar funktion i den här varianten.

c) I ett klassiskt kassaskåp definieras de inmatade siffrorna genom att man vrider ratten åt andra hållet varje gång man vill spara en ny siffra. Sifferkombinationen matas därmed in genom att ratten vrids åt höger tills önskad siffra visas, följt av en vridning åt vänster tills nästa siffra i låskombinationen visas. Slutligen vrids ratten åt höger tills siffran noll visas och när nu skjutomkopplaren förs till sitt övre läge ska låset öppnas. (Under förutsättningen att rätt sifferkombination har matats in.) För att stänga låset förs skjutomkopplaren till sitt nedre läge varvid ett nytt öppningsförsök kan göras. När låset är öppet kan koden ändras genom att man nu matar in en ny kombination följt av att sparaknappen trycks ner. Ett lyckat kodbyte ska indikeras av en tänd lysdiod. När låset är stängt har sparaknappen ingen funktion. Ett öppet lås indikeras av en tänd lysdiod. Om ratten vrids åt fel håll, eller en för lång sekvens matas in ska kombinationen betraktas som felaktig.

Använd VHDL och en CPLD samt föreskrivna in- och utsignaler. Systemklockan väljs till 1000 Hz. Ratten är av pulsgivartyp och har följande utseende:



samt funktion:



Observera att där det i tidsdiagrammet ser ut som om signal A och B ändrar värde samtidigt kan bytet i praktiken ske en klockpuls senare i en av signalerna.

Förberedelser: Rita ett blockschema för kretsen och översätt till VHDL-kod. Simulera gärna kretsen och provsyntetisera koden.

Laborationsuppgift: Koppla upp och verifiera kretsens funktion.

4.4 Simulering av uppgift 4.2 med en testbänk

Uppgift 4.2 är svår att simulera "för hand" i ModelSim, eftersom kretsen ska prata flitigt med utomstående minnen (PROM:en). Det är helt enkelt för jobbigt att manuellt tilldela data-insignalen utifrån adress- och CS-utsignalerna.

Därför har vi gjort en så kallad testbänk som gör detta automatiskt. En testbänk är också skriven i VHDL, men använder funktioner i språket som inte går att stoppa in i en CPLD. Exempel på sådana funktioner är att skapa en klocka i en viss hastighet, eller skriva ut testresultat på skärmen.

Det denna testbänk gör är:

- Den skapar de insignaler som behövs (klocka, reset, start).
- Den emulerar två PROM (så att data-ingången faktiskt beror på adress m.m.).
- Den kontrollerar att resultatet på sjusegmentsdisplayerna är rätt när LED-lampan tänds.

För att detta ska fungera, så är det viktigt att testbänken vet exakt hur din modul ser ut. Alltså vad modulen heter, samt hur alla signaler som kommer in och ut ur den ser ut. D.v.s. allt som står mellan `entity` och `end entity`.

För att detta inte ska bli fel, så får du detta tillsammans med testbänken (förkortat "tb"). Du får alltså en "start-fil" för din kod, där du ska fylla i det som står mellan `architecture` och `end architecture`.

4.4.1 Så här gör du

- Kopiera `Lab4_2.vhd` och `Lab4_2_tb.vhd`.
- Skriv in din design i `Lab4_2.vhd`.
- Starta ModelSim, som vanligt.
- Kompilera både `Lab4_2.vhd` och `Lab4_2_tb.vhd`. Rätta eventuella kompileringsfel, kompilera om etc.
- Simulera modulen `Lab4_2_tb`: `vsim Lab4_2_tb` Eller högerklicka på → "Simulate"
- Lägg till signaler i waveform: `add wave -r *` Eller gör som vanligt.
- Kör simuleringen: `run -a` Eller klicka på ikonen "run all".

Testbänken testar nu din design fyra gånger, med olika innehåll i de två PROM:en. Första gången är PROM:en fyllda med nollor, den andra gången med ettor, och de två senare innehåller PROM:en lite slumpmässig data. Innan första gången aktiveras clear-signalen.

För varje gång så skriver testbänken ut resultatet i transcript-fönstret i ModelSim. `OK` betyder att testet gick bra, `NOK` betyder att resultatet blev fel. Det är troligt att det blir ett antal varnings-meddelanden vid tidpunkt 0 ns i transcript-fönstret men dessa kan ignoreras.

När simuleringen fungerar, så använder du din kod till att göra ett CPLD-projekt, som vanligt.