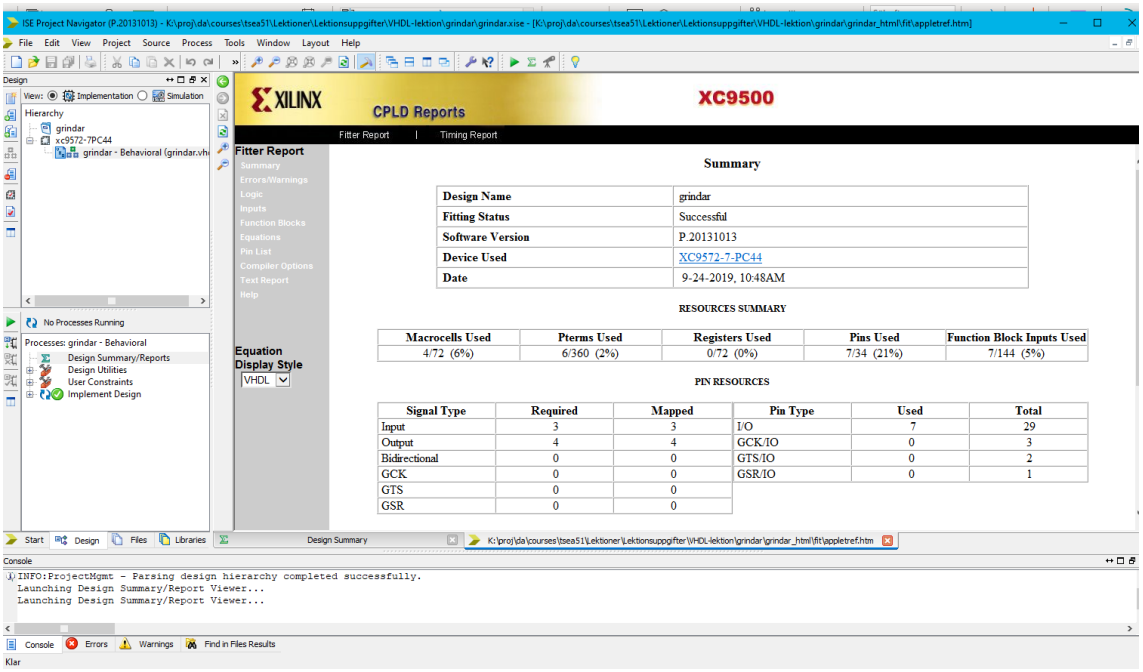
VHDL och Xilinx ISE Project Navigator
Lektionsuppgifter

4 december 2019



Figur 1: Skärmbild av Fitter reportens summary-flik. I det grå fältet syns även flikarna Equations och Pin List som kommer användas i kursen.

Introduktion

Syftet med lektionen är att komma igång med VHDL-programmering och Xilinxs syntesprogram ISE Design Suite som används i kursens VHDL-laboration.

Detta innefattar att:

- Rita blockschema med de digitaltekniska byggblocken listade i Digitaltekniska byggblock.
- Införa signalnamn i blockschemat.
- Översätta blockschema till VHDL-kod med överensstämmande variabelnamn i blockschema och kod.
- Använda ISE Design suite för att skriva, kompilera och felsöka kod samt göra enklare analys av syntesresultatet.

OBS: Det är viktigt att ni alltid skapar ett nytt projekt till varje ny uppgift.

Uppgifter

Uppgift 1.

a) Starta ISE Design Suite enligt anvisningarna i avsnitt 3.2.1 i labkompendiet och skriv koden till labuppgift 3.1.

b) Kontrollera att det går att syntetisera koden och kontrollera att en .jed-filen har genererats i projektkatalogen ni arbetar i. Det är den filen som ska användas på laborationen för att programmera den programmerbara kretsen.

c) Gå in i Fitter Report, under fliken Summary. Då borde det se ut som i figur 1.Hur många Macroceller, Pterms, Register, och Pinnar använder kretsen?

Figur 2: Modul för att koppla ihop CPLD:n med övrig TTL-hårdvara. GND kan kopplas in på någon av 0-pinnarna och VCC på någon av U-pinnarna.

d) Gå in i Fitter Report, under fliken Equations, och skriv upp de logiska uttrycken för utsignalerna.

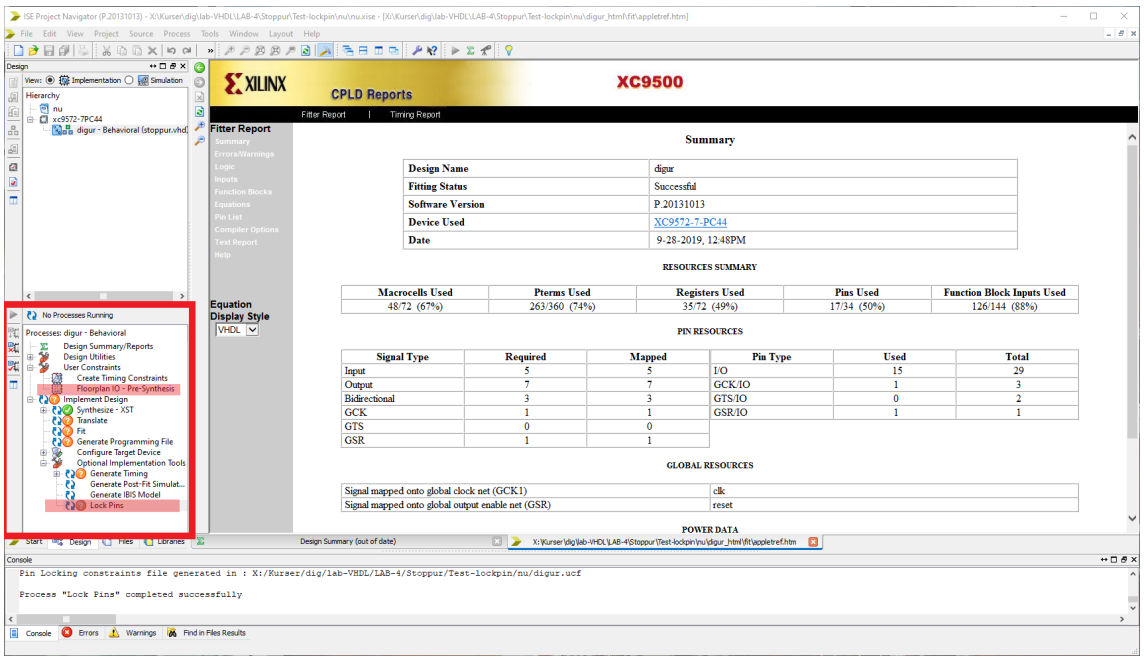
e) På laborationen kommer ni att konfigurera ett chip med hjälp av .jed-filen. För att koppla in- och ut-gångar på chipet används modulen i figur 2. För att veta hur pinnarna ska kopplas in öppnar ni fliken Pin List i Fitter Report. Där listas vilka pinnar kretsens in- och ut-signaler finns på. Gå in i Fitter Report, under fliken Pin List, och se vilka pinnar respektive signaler ska kopplas in på.

f) Efter eventuell felsökning kan det bli aktuellt att ändra i VHDL-koden. Då riskerar man att fittern väljer andra pinnar till de olika signalerna och kretsen måste kopplas om. För att undvika detta kan man begränsa fitterns möjligheter till att välja pinnar, dvs man kan låsa pinvalet. Detta görs på följande vis.

   1. Klicka fram de menyer som syns i det rödmarkerade `Process`-fönstret i figur 3. Då visas bland annat olika aktiviteter/processer som designverktyget kan genomföra och de vanligaste är: Synthesize, Translate, Fit och Generate.

   2. Dubbelklicka på det rödmarkerade processen: `Foorplan IO` och svara `yes` på diaglogrutan som dyker upp. Detta kommer skapa en .ucf-fil och koppla filen till ert projekt för att lagra pintilldelningen.

   3. Stäng ner fönstret som dyker upp.

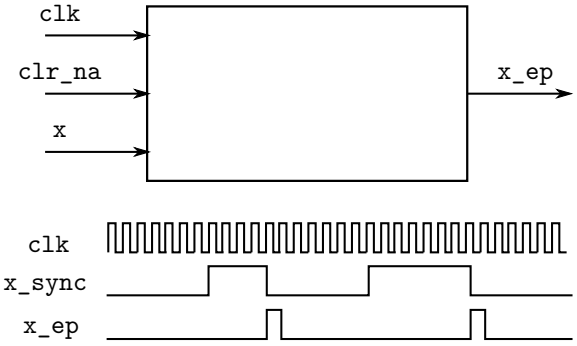


Figur 3: Den rödmarkerarde processmenyn i designfönstret kan användas bland annat för att låsa signaler till specifika pinnar.

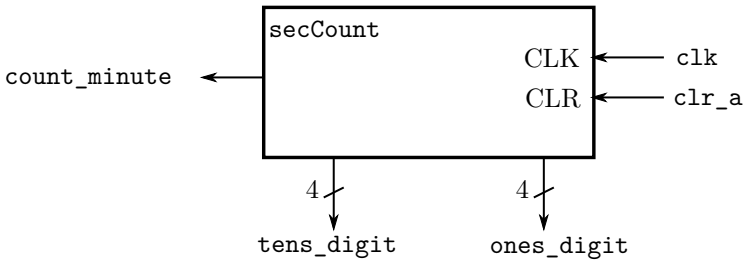
4. Återgå till den rödmarkerade rutan i figur 3. Dubbelklicka på den rödmarkerad processen `Lock Pins` för att låsa pintilldelningen.

Öppna .ucf-filen i er projekkatalog, den finns med i `Hierarchy`-fönstret i vänstra övre hörnet, och se om ni förstår innehållet. Observera att vid större förändringar av VHDL-koden kan det vara nödvändigt att ta bort pinlock för att fittern ska lyckas konfigurera chipet. Öppna i så fall `Floorplan IO`-fönstret, gå in i `Edit`-menyn och välj `Remove All Constraints`.

Uppgift 2. En synkron sekvenskrets med in- och ut-signaler enligt figur ska konstrueras. Utsignalen `x_ep` ska vara hög i ett klockintervall då insignalen `x` går från hög till låg enligt tidsdiagrammet. Insignalen `x` är asynkron och ska synkroniseras. I tidsdiagrammet kallas den synkroniserade insignalen `x_sync`. Den asynkrona insignalen `clr_na` ska aktivera asynkron reset då `clr_na` = 0. Rita kretsschema med signalnamn, skriv och kompilera motsvarande VHDL-kod. Undersök gärna resultatet på samma sätt som i första uppgiften genom att gå in i Fitter report och se hur många Macroceller, Pterms, Register, och Pinnar som kretsen använder. Gå även in i Schematic Viewer och titta på resultatet.



Uppgift 3. Lös uppgift 2 i lektionen speciella sekvenskretsar med VHDL. Namn på in- och utsignaler definieras i figuren.



Den asynkrona clearsignalen `clr_a` ska asynkront nollställa alla register om `clr_a` = 1. Rita blockschema och inför signalnamn, skriv och syntetisera kod. Utgå från standardblocken när ni ritar och kodar kretsen. Om ni inte löst uppgift 2 på lektionen för speciella sekvenskretsar ännu och vill fokusera på VHDL-programmering så kan ni utgå från lösningen i facit till uppgiften.

Uppgift 4. Fortsätt med att förbereda laborationsuppgifterna i lab 3.

Facit

Uppgift 1.

c) Macroceller: 4, Pterms: 6, Register: 0, Pinnar: 7.

d) Uttryck för kod och syntesresultat listas nedan.

$$a_{\text{VHDL}} = z' \qquad a_{\text{Fitter}} = z'$$
$$b_{\text{VHDL}} = (xy)' \qquad b_{\text{Fitter}} = (xy)'$$
$$c_{\text{VHDL}} = x'y + xy' \qquad c_{\text{Fitter}} = x \oplus y$$
$$d_{\text{VHDL}} = x \oplus z \qquad d_{\text{Fitter}} = x \oplus z$$

e) Nedan listas ett exempel på hur pinnarna kan tilldelas:

signal	pinne/pinnar
GND	0, (10, 23, 31)
VCC	U, (21, 32, 41)
a	1
b	11
c	35
d	24
x	43
y	38
z	28

På laborationen behöver ni bara koppla in en av 0-pinnarna till GND och en av U-pinnarna till VCC.

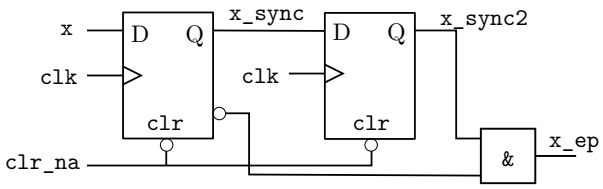
e) .ucf-filen:

```
#PINLOCK_BEGIN

#Mon Sep 30 12:03:35 2019

NET "x"            LOC = "S:PIN43";
NET "y"            LOC = "S:PIN38";
NET "z"            LOC = "S:PIN28";
NET "a"            LOC = "S:PIN1";
NET "b"            LOC = "S:PIN11";
NET "c"            LOC = "S:PIN35";
NET "d"            LOC = "S:PIN24";
#PINLOCK_END
```

Uppgift 2. Kretsschema:



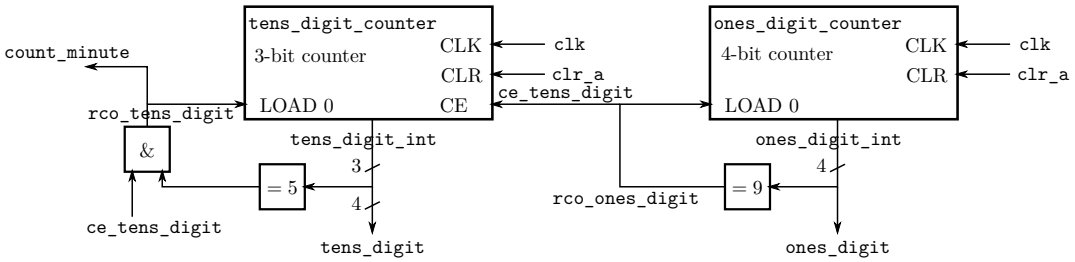
Motsvarande kod:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity enpuls is
       Port ( clk : in STD_LOGIC;
                  clr_na : in STD_LOGIC;
                  x_ep : out STD_LOGIC;
                  x : in STD_LOGIC);
end enpuls;

architecture Behavioral of enpuls is
signal x_sync, x_sync2 : std_logic;
begin
       process(clk) begin
              if clr_na = '0' then
                     x_sync <= '0';
                     x_sync2 <= '0';
              elsif rising_edge(clk) then
                     x_sync <= x;
                     x_sync2 <= x_sync;
              end if;
       end process;
       x_ep <= (not x_sync) and x_sync2;
end Behavioral;
```

Uppgift 3. Nedan visas ett exempel på ett blockschema.



Motsvarande kod är:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity secCounter is
        Port ( clk, clr_a : in STD_LOGIC;
               count_minute : out STD_LOGIC;
               ones_digit : out STD_LOGIC_VECTOR (3 downto 0);
               tens_digit : out STD_LOGIC_VECTOR (3 downto 0));
end secCounter;

architecture Behavioral of secCounter is

-- Ones digit counter signals
signal ones_digit_int : unsigned(3 downto 0);
signal rco_ones_digit : std_logic;

-- Tens digit counter signals
signal ce_tens_digit : std_logic;
signal tens_digit_int : unsigned(2 downto 0);
signal rco_tens_digit : std_logic;

begin
-- Ones digit counter
process(clk,clr_a) begin
        if clr_a = '1' then
               ones_digit_int <= "0000";
                      elsif rising_edge(clk) then
               if rco_ones_digit = '1' then
                      ones_digit_int <= "0000";
               else
                      ones_digit_int <= ones_digit_int + 1;
               end if;
        end if;
end process;
rco_ones_digit <= '1' when (ones_digit_int = 9)
        else '0';

-- Tens digit counter
ce_tens_digit <= rco_ones_digit;
```

```
process(clk,clr_a) begin
	if clr_a = '1' then
		tens_digit_int <= "000";
	elsif rising_edge(clk) then
		if ce_tens_digit = '1' then -- ce
			if rco_tens_digit = '1' then -- load 0
				tens_digit_int <= "000";
			else
				tens_digit_int <= tens_digit_int + 1;
			end if;
		end if;
	end if;
end process;
rco_tens_digit <= '1' when ((tens_digit_int = 5) and (ce_tens_digit = '1'))
	else '0';

-- Outputs
count_minute <= rco_tens_digit;
ones_digit <= std_logic_vector(ones_digit_int);
tens_digit <= '0' & std_logic_vector(tens_digit_int); -- add a fourth bit
end Behavioral;
```