Föreläsning 3
VHDL och FPGA-programmering

Erik Agrell 2017-09-01

Innehåll:

1. FPGA-teknik
2. Synkron VHDL
3. Labkortet DE1
4. Mjukvaran Quartus

Bilder av E. Agrell, A. Linde, A. Crayvenn, S. Farinam, C. Fougstedt

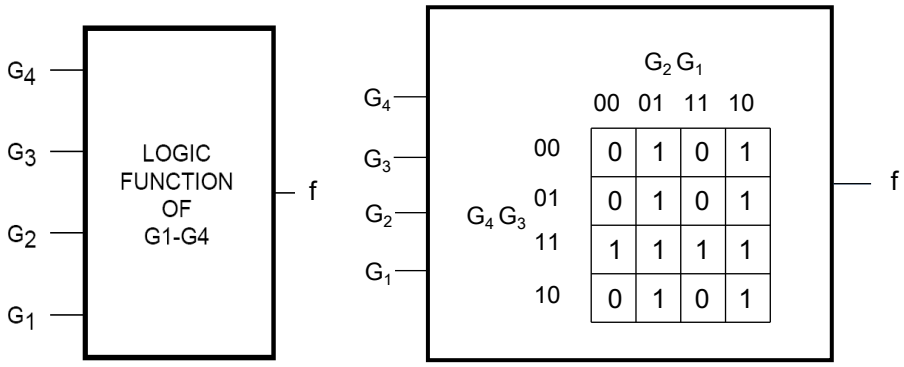
1. FPGA-teknik

En field-programmable gate array (FPGA) består av…

- massor av logikgrindar
- massor av minnesceller (RAM)
- nät av ledningar (interconnects)
- massor av spänningsstyrda kopplingar mellan ledningar
- samt i de mer avancerade modellerna:
  - multiplikatorer
  - processorer (Power PC)
  - AD/DA-omvandlare
  - faslåsta slingor (phase-locked loop, PLL)
  - m.m.

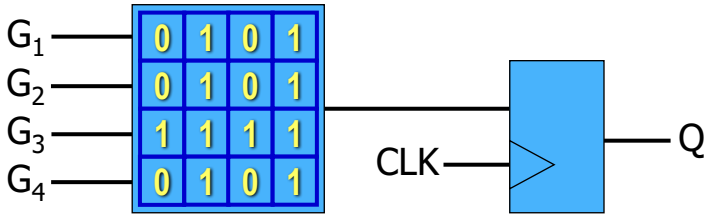
Grindarna i en FPGA

- Grindarna realiseras som en liten tabell (look-up table, LUT)
- Kan ses som ett Karnaugh-diagram
- Kan realisera alla tänkbara grindar



Logikelement

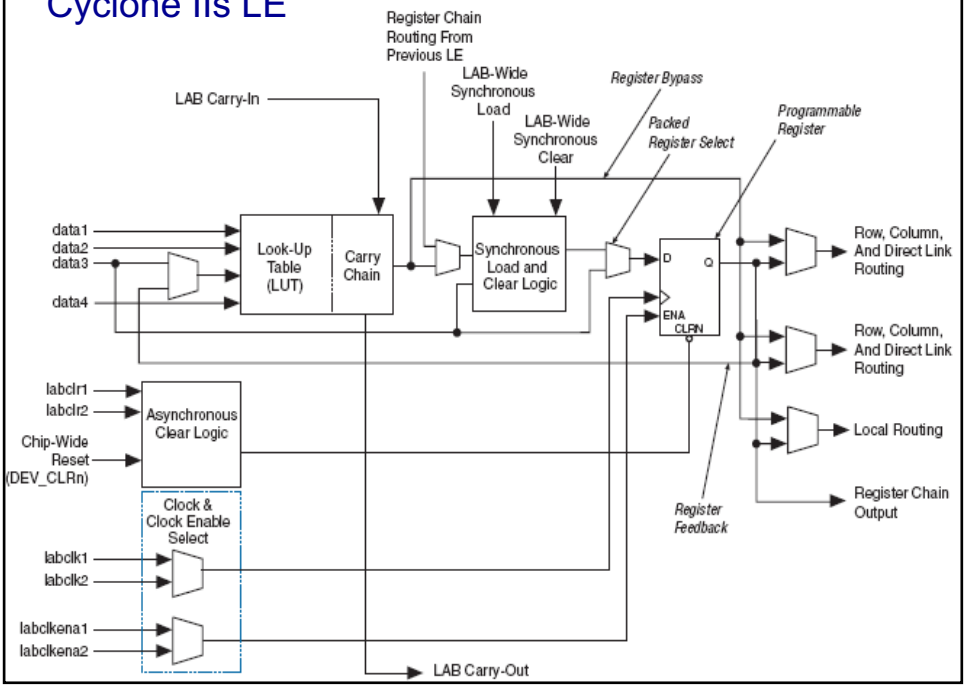
- Kombinationen av grind (LUT) och vippa kallas logikelement (LE)



(förenklad bild – ett LE innehåller normalt även andra komponenter)

- Storleken hos FPGAer mäts i antal LE

Cyclone IIs LE



Labbets FPGA

Cyclone II 2C35 FPGA

- 18,752 LEs
- 52 M4K RAM blocks
- 240K total RAM bits
- 26 embedded multipliers
- 4 PLLs
- 315 user I/O pins
- FineLine BGA 484-pin package

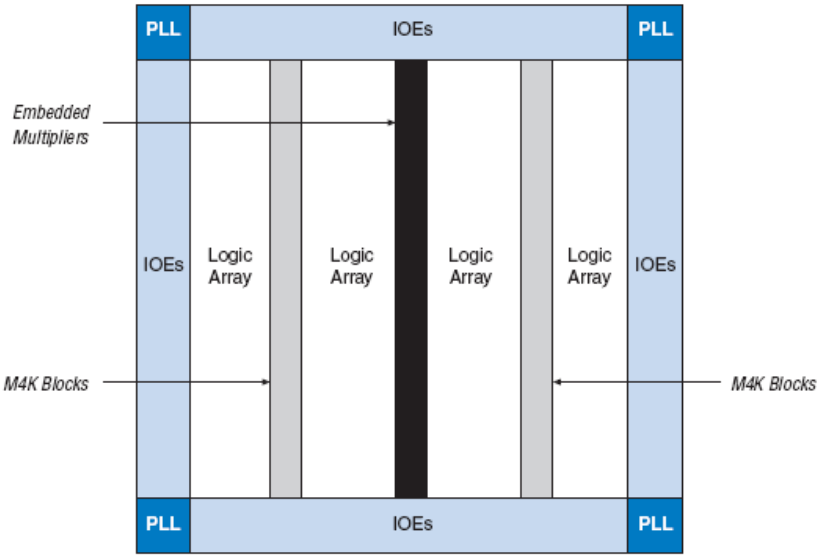
Cyclone är lågprismodellen

Finns nu i version V

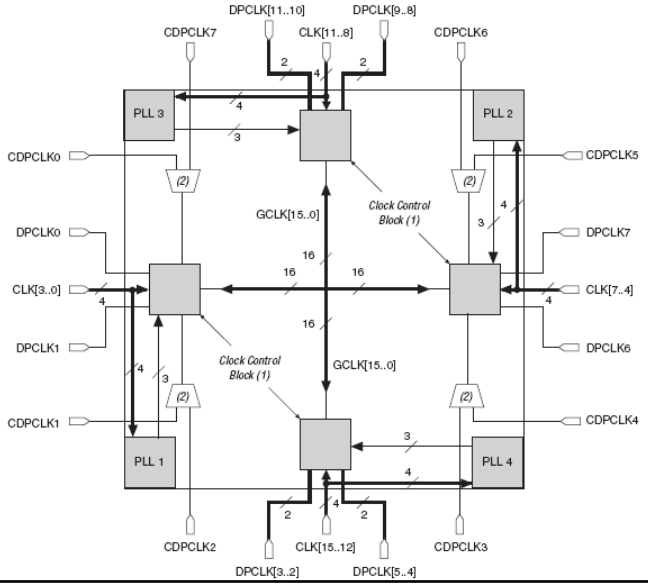
Från ca 300 kr/styck

Cyclone II

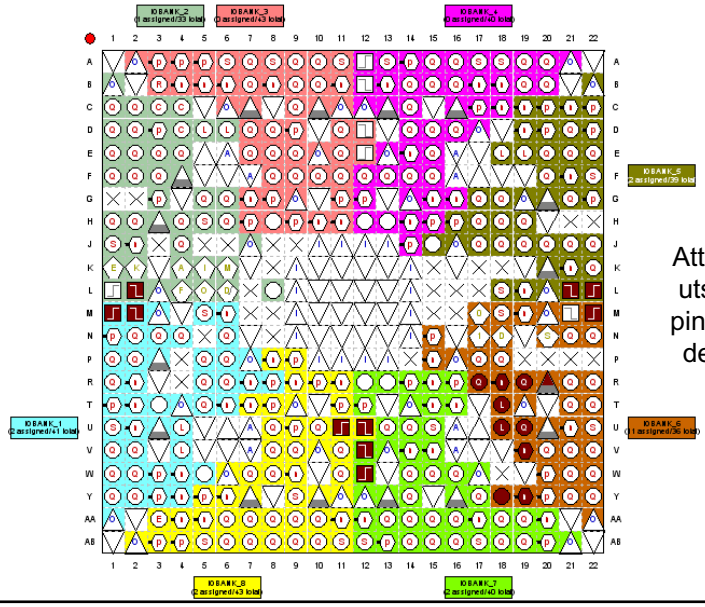
FPGA-kretsens interna layout



Specialiserat klocknät

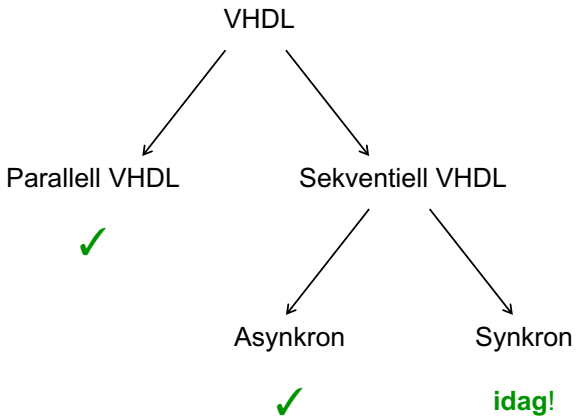


Kretsens pinnar



Att koppla in- och utsignaler till rätt pinnar är en viktig del av syntesen

2. Synkron VHDL



Parallell VHDL:

- I architecture, utanför processer
- All exekveras samtidigt, ordningen är oväsentlig
- Signaltilldelning med <=
- with

Sekventiell VHDL:

- Sker i processer
- Allt exekveras sekventiellt inom processer
- Alla processer exekveras samtidigt
- Variabeltilldelning med :=
- if, case

- Kod som står inom
  `process( ) ... end process`
  är sekventiell VHDL

- Kod som står inom
  `if rising_edge( ) then ... end if`
  eller liknande är synkron (klockad) VHDL

Synkron VHDL är en sorts sekventiell VHDL, där processen aktiveras av klockan

```
process(clk)
begin
      if rising_edge(clk) then
         ...
      end if;
end process;
```

Sensitivitetslista

Trigga på positiv klockflank

Från F2:

- I processens sensitivitetslista ingår de signaler som skall påverka processens igångsättning vid simulering.
- Sensitivitetslistan ignoreras vid syntes

Exempel: shift-register

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

entity shift_register is
port (
  clk: in std_logic;
  x: in std_logic;
  y: out std_logic);
end entity;

architecture arch of shift_register is
  signal r0, r1, r2, r3: std_logic;
begin
  r0 <= x;
  y <= r0 xor r1 xor r2 xor r3;

  process(clk)
  begin
    if rising_edge(clk) then
      r1 <= r0;
      r2 <= r1;
      r3 <= r2;
    end if;
  end process;
end architecture;
```

Vanligt fel:

```
if clk='1' then
```

Varför är det fel?

Dessa tre rader innebär inte

```
r3 <= r0
```

Demo

… eller implementerat med en vektor

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

entity shift_register is
port (
  clk: in std_logic;
  x: in std_logic;
  y: out std_logic);
end entity;

architecture arch of shift_register is
  signal r: std_logic_vector(3 downto 0);
begin
  r(0) <= x;
  y <= r(0) xor r(1) xor r(2) xor r(3);

  process(clk)
  begin
    if rising_edge(clk) then
      r(3 downto 1) <= r(2 downto 0);
    end if;
  end process;
end architecture;
```

Shiftregister kan också implementeras med inbyggda operationer `sll`, `srl`, `rol`, `ror`, …

Mer om vektorer

- Om alla bitar skall vara samma kan man skriva

  ```
  f <= ( others => '0')
  ```

  vilket betyder detsamma som `f <= "00000000"`

- Jämförelser kan göras binärt eller som heltal:

  ```
  if x=8 then
  if x="1000" then
  ```

- Vektorer kan i ModelSim kan anges binärt, decimalt eller hexadecimalt:

  ```
  force x 2#1111
  force x 10#15
  force x 16#F
  ```

Heltalsberäkningar

Med tillägget i biblioteket

```
use ieee.std_logic_unsigned.all;
```

kan heltalsberäkningar utföras

Exempel:

```
f <= x + 5;
```

Om `f` och `x` är av typen `std_logic_vector(3 downto 0)` och `x` är `"0001"` (heltalet 1), så blir `f` `"0110"` (heltalet 6)

Klockning av processer

Problem: Antag att en snabb klocka

skall styra en långsam process –

hur implementerar man det?

Lösning: Man skapar en triggsignal genom att dela ned klocksignalen med en räknare

Exempel: räknare

Vi har en 50 MHz klocka (clk50) och vill skapa en långsammare signal:

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

entity timer is
port (
  clk50: in std_logic;
  squarewave: out std_logic);
end entity;

architecture arch of timer is
  signal counter: std_logic_vector(26 downto 0);
begin
  process(clk50)
  begin
    if rising_edge(clk50) then
      counter <= counter + 1;
    end if;
  end process;
  squarewave <= counter(26);
end architecture;
```

Nytt bibliotek

Heltalssummering, börjar om vid 0 när summan går i taket

Demo?
Nja, inte än...

Initialisering av variabler

Problem: Föregående VHDL-kod kan inte simuleras, eftersom `counter` aldrig initialiseras. När strömmen slås på till ett chip, har alla minnesenheter (register) slumpmässigt innehåll.

Lösning: Inför en reset-signal.

Varning: VHDL-syntaxen tillåter att signaler och variabler ges initial-värden, men dessa används bara vid simulering, inte syntes. Undvik!

Räknare med asynkron reset

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

entity timer is
port (
  clk50: in std_logic;
  reset: in std_logic;
  squarewave: out std_logic);
end entity;

architecture arch of timer is
  signal counter: std_logic_vector(26 downto 0);
begin
  process(clk50,reset)
  begin
    if reset='1' then
      counter <= (others => '0');
    elsif rising_edge(clk50) then
      counter <= counter + 1;
    end if;
  end process;
  squarewave <= counter(26);
end architecture;
```

reset-signal

reset-signalen måste ingå i sensitivitetslistan. (Vad händer annars?)

reset-signalen kollas utanför den klockade delen av processen

Räknare med synkron reset

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

entity timer is
port (
  clk50: in std_logic;
  reset: in std_logic;
  squarewave: out std_logic);
end entity;

architecture arch of timer is
  signal counter: std_logic_vector(3 downto 0);
begin
  process(clk50)
  begin
    if rising_edge(clk50) then
      if reset='1' then
        counter <= (others => '0');
      else
        counter <= counter + 1;
      end if;
    end if;
  end process;
  squarewave <= counter(3);
end architecture;
```

reset-signalen behöver inte ingå i sensitivitetslistan

reset-signalen kollas inuti den klockade delen av processen

lägre värde för att underlätta demon

Demo? OK!

Räknare med andra frekvenser

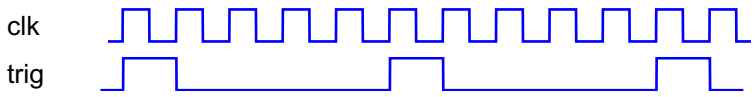
Problem: Föregående VHDL-kod kan endast dela ned frekvenser med en tvåpotens

Lösning: En mer flexibel räknare fås genom att nollställa `counter` när den uppnått ett visst värde

Undvik olikhetsjämförelser, som kostar mycket hårdvara.

Digitala triggsignaler

(kallas även ”aktiveringssignal” eller ”enable”)



1. Skapa ”trig”:

```
process(clk, reset)
begin
  if reset='1' then
    counter <= (others => '0');
  elsif rising_edge(clk) then
    if counter=4 then
      trig <= '1';
      counter <= (others => '0');
    else
      trig <= '0';
      counter <= counter+1;
    end if;
  end if;
end process;
```

2. Använd ”trig”:

```
process(...)
begin
  if rising_edge(clk) then
    if trig='1' then
      ...
    end if;
  end if;
end process;
```

Vilken periodtid?

Svar: 5

Var rädd om klockan

- Synkron elektronik förutsätter att kretsarna klockas exakt samtidigt
- Syntesverktygen implementerar därför klockan separat från andra signaler
- Stör inte klockans funktion!

Exempel på olämplig klockning

- *Misstag 1:* Kombinera inte `rising_edge` med andra villkor.

```
if rising_edge(clk) and (...) then
```

kan införa små fördröjningar som hindrar komponenterna från att klockas samtidigt.

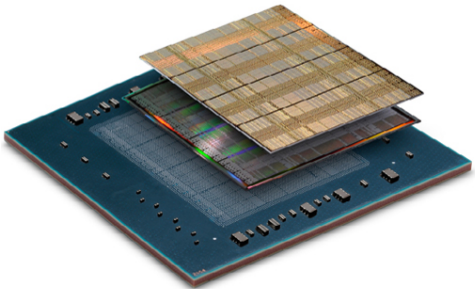
- *Misstag 2:* Använd inte `rising_edge` på användar-definierade signaler.

```
if rising_edge(trig) then
```

där `trig` definierades i bild 24, hindrar syntesverktyget från att använda FPGAns specialiserade klocknät.

3. Labkortet DE1

Altera DE1 Board



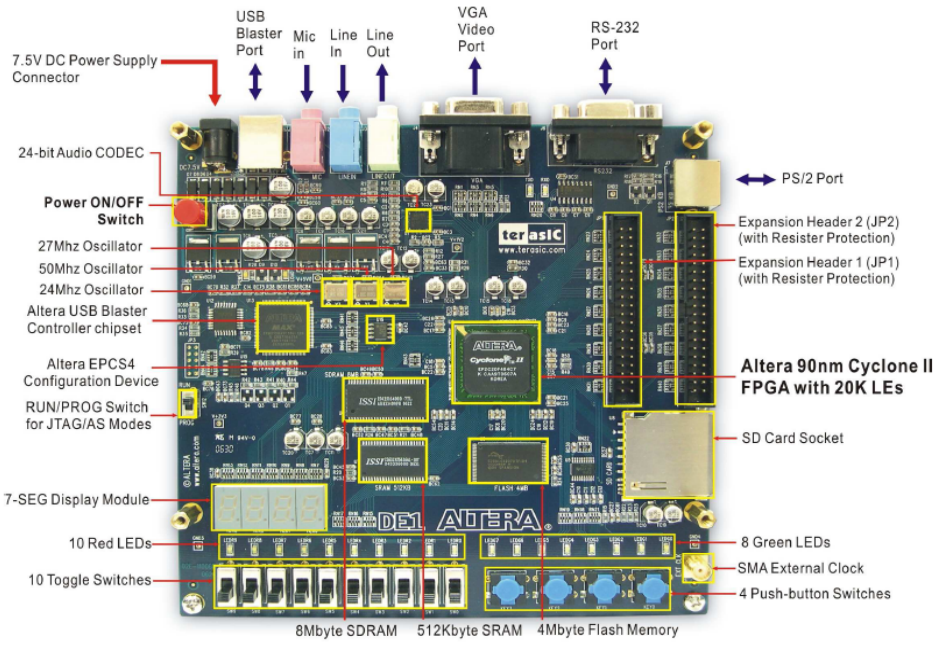
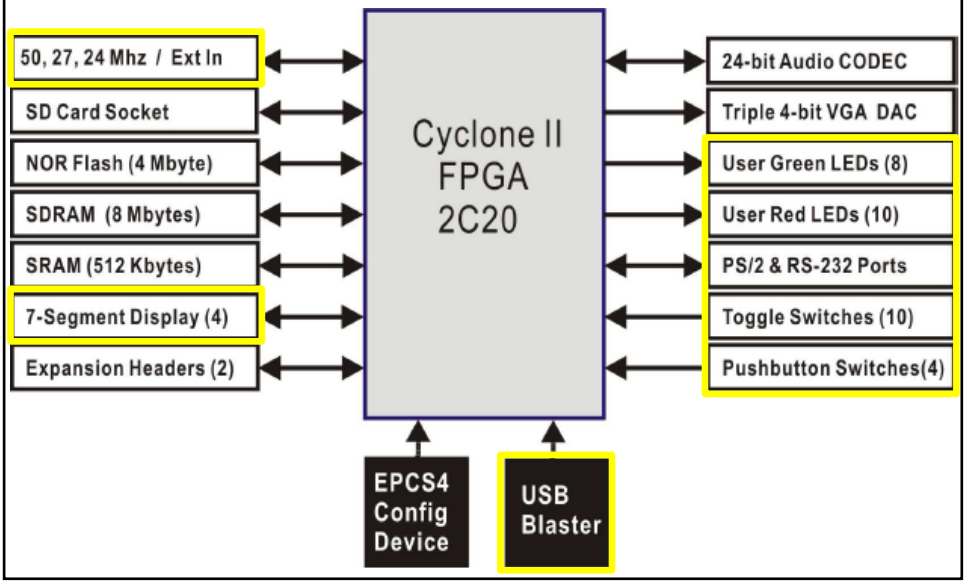


Figure 2.1. The DE1 board.

Input/output



Pushbutton switches

- 4 pushbutton switches
- Debounced by a Schmitt trigger circuit
- Normally high; generates one active-low pulse when the switch is pressed

Toggle switches

- 10 toggle switches for user inputs
- A switch causes logic 0 when in the DOWN (closest to the edge of the DE1 board) position and logic 1 when in the UP position

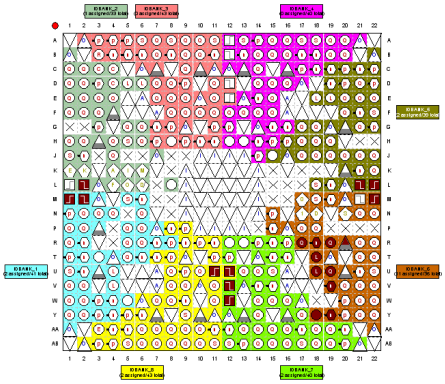
Clock inputs

- 50-MHz oscillator
- 27-MHz oscillator
- 24-MHz oscillator
- SMA external clock input

Pinn-konfigurering

De flesta av FPGA-kretsens pinnar sitter ihop med DE1-kortets kringutrustning:

- omkopplare
- tryckknappar
- LED-arrayer
- 7-segmentsdisplay
- klocksignaler
- …



Vid FPGA-programmering måste man definiera vilka pinnar på kretsen som hör till vilka signaler i FPGA-koden. Det kallas ”pinn-konfigurering”.

Exempel på pinnar

Signal Name	FPGA Pin No.	Description
KEY[0]	PIN_R22	Pushbutton[0]
KEY[1]	PIN_R21	Pushbutton[1]
KEY[2]	PIN_T22	Pushbutton[2]
KEY[3]	PIN_T21	Pushbutton[3]
LEDG[0]	PIN_U22	LED Green[0]
LEDG[1]	PIN_U21	LED Green[1]
LEDG[2]	PIN_V22	LED Green[2]
LEDG[3]	PIN_V21	LED Green[3]
LEDG[4]	PIN_W22	LED Green[4]
LEDG[5]	PIN_W21	LED Green[5]
LEDG[6]	PIN_Y22	LED Green[6]
LEDG[7]	PIN_Y21	LED Green[7]
CLOCK_50	PIN_L1	50 MHz clock input

(ur DE1 User Manual, kap 4)

Egna pinn-tabeller

- För återkommande projekt gör man gärna en pinn-tabell i en separat fil
- Hemsidans tabell ”Fixed_DE1_pin_assignments” är en bra utgångspunkt

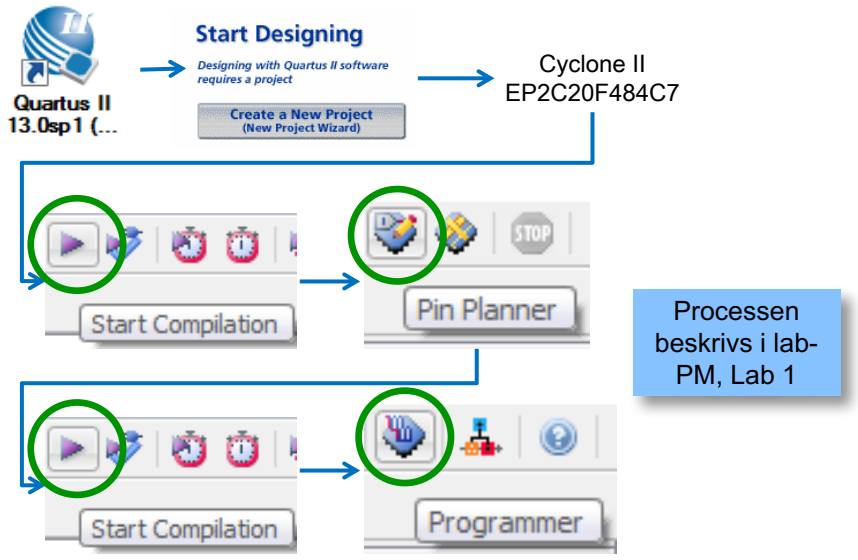
	B	D	E	H
1	Name	Location	Enabled	I/O Standard
2	CLOCK_50	PIN_L1	Yes	LVTTL
3				
4	SW[0]	PIN_L22	Yes	LVTTL
5	SW[1]	PIN_L21	Yes	LVTTL
6	SW[2]	PIN_M22	Yes	LVTTL
7	SW[3]	PIN_V12	Yes	LVTTL
8	SW[4]	PIN_W12	Yes	LVTTL
9	SW[5]	PIN_U12	Yes	LVTTL
10	SW[6]	PIN_U11	Yes	LVTTL
11	SW[7]	PIN_M2	Yes	LVTTL
12	SW[8]	PIN_M1	Yes	LVTTL
13	SW[9]	PIN_L2	Yes	LVTTL
14				
15	KEY[0]	PIN_R22	Yes	LVTTL
16	KEY[1]	PIN_R21	Yes	LVTTL
17	KEY[2]	PIN_T22	Yes	LVTTL
18	KEY[3]	PIN_T21	Yes	LVTTL
19				
20	LEDR[0]	PIN_R20	Yes	LVTTL
21	LEDR[1]	PIN_R19	Yes	LVTTL
22	LEDR[2]	PIN_U19	Yes	LVTTL
23	LEDR[3]	PIN_Y19	Yes	LVTTL
24	LEDR[4]	PIN_T18	Yes	LVTTL
25	LEDR[5]	PIN_V19	Yes	LVTTL
26	LEDR[6]	PIN_Y18	Yes	LVTTL
27	LEDR[7]	PIN_U18	Yes	LVTTL

4. Mjukvaran Quartus

Används för syntes av VHDL, d.v.s.

- kompilera VHDL
- pinn-konfigurering
- tanka ned kod till FPGAn

Arbetsgång



Processen beskrivs i lab-PM, Lab 1

Quartus-varningar

Quartus genererar många varningar vid kompilering!

- 15–20 varningar är normalt även för små, felfria program
- De flesta är ofarliga, t.ex. om timing och kapacitanser
- Några är viktiga, t.ex. om pinntilldelning som saknas och om ”latchar”

Latch = minneselement, D-vippa

Exempel på en ”latch”

Någon ville ha en krets som kopplar omkopplaren SW till lysdioden LED varje gång man trycker på knappen KEY. Vad blev fel?

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

entity switch is
port (
  KEY: in std_logic;
  SW: in std_logic;
  LED: out std_logic);
end entity;

architecture arch of switch is
begin
  process(SW, KEY)
  begin
    if KEY='1' then
      LED <= SW;
    end if;
  end process;
end architecture;
```


Vad Quartus säger…

```
Warning (10631): VHDL Process Statement warning at
switch.vhd(12): inferring latch(es) for signal or
variable "LED", which holds its previous value in
one or more paths through the process
```

…och vad Quartus menar:

Du har otilldelade utsignaler.

Lösningar

Först måste man bestämma sig för vad som skall hända när knappen *inte* är nedtryckt.

Metod 1 (else):

```
process(SW, KEY)
begin
  if KEY='1' then
    LED <= SW;
  else
    LED <= '0';
  end if;
end process;
```

Metod 2 (default-tilldelning):

```
process(SW, KEY)
begin
  LED <= '0'
  if KEY='1' then
    LED <= SW;
  end if;
end process;
```


Implicit minne

Ibland kan man vilja skapa en ”latch” avsiktligt i *synkron VHDL* genom att *inte* täcka alla alternativ. Det kallas då implicit minne.

```
-- T-flipflop: change output
-- value if t=1
library ieee;
use ieee.std_logic_1164.all;

entity Tvippa is port (
  t: in std_logic;
  clk, reset: in std_logic;
  q: out std_logic);
end entity;
```

Om en utsignal i synkron VHDL inte tilldelas ett värde, behålls värdet från förra klockcykeln

```
architecture arch of Tvippa is
  signal s: std_logic;
begin
  process(clk,reset)
  begin
    if reset='1' then
      s <= '0';
    elsif rising_edge(clk) then
      if t='1' then
        s <= not s;
      end if;
    end if;
  end process;
  q <= s;
end architecture;
```

Veckosammanfattning

Efter den här veckan behärskar ni…

- VHDLs grunder
  - bibliotek, entitet och arkitektur
  - in- och ut-signaler
  - signaltilldelning med **<=**
  - vektorer
  - heltalsaddition
- Parallell VHDL (utanför process)
  - grindnivå (**and**, **or**, **not**)
  - interna signaler
  - **with**

- Sekventiell VHDL (innanför process)
  - sensitivitetslista
  - variabler
  - **if** och **case**
  - asynkron VHDL (utanför **if rising_edge( )**)
  - synkron VHDL (innanför **if rising_edge( )**)
  - reset-signaler (asynkron och synkron)
- Simulering med ModelSim

Labinformation

- Anmäl er till labgrupper (de som inte redan gjort det)
- De som anmält sig ensamma får gärna bilda par
- Lab-PM för lab 1 finns på hemsidan
- Obligatoriska förberedelseuppgifter före varje lab
- Man måste ha löst alla förberedelseuppgifterna före labben, men det måste inte vara 100% rätt.
- Båda personerna i gruppen skall kunna redovisa förberedelseuppgifterna.

Läs & lös

Läs:

- Kompendiet ”Kretskonstruktion med VHDL”, avsnitt 4–5 och 10
- Lab-PM för lab 1

Lös:

- Förberedelseuppgifter till lab 1 (måste vara gjorda före labben)