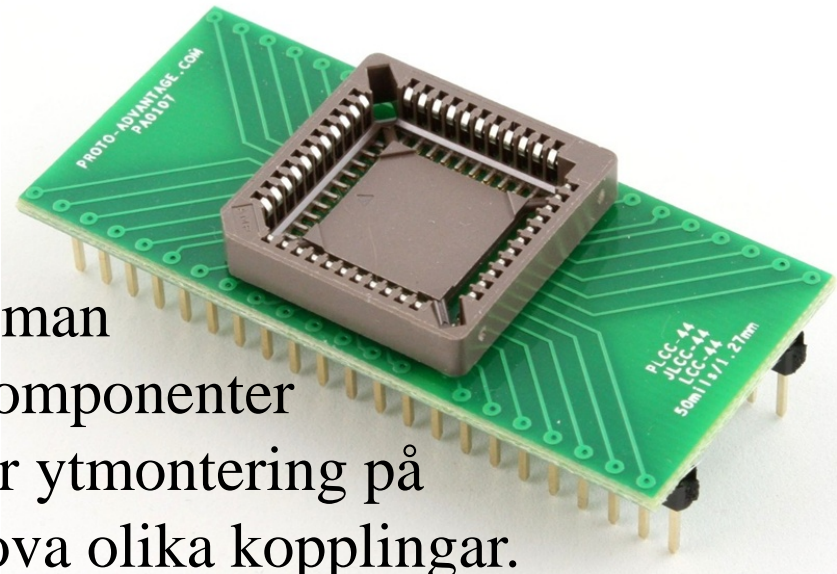


LAB VHDL-programmering



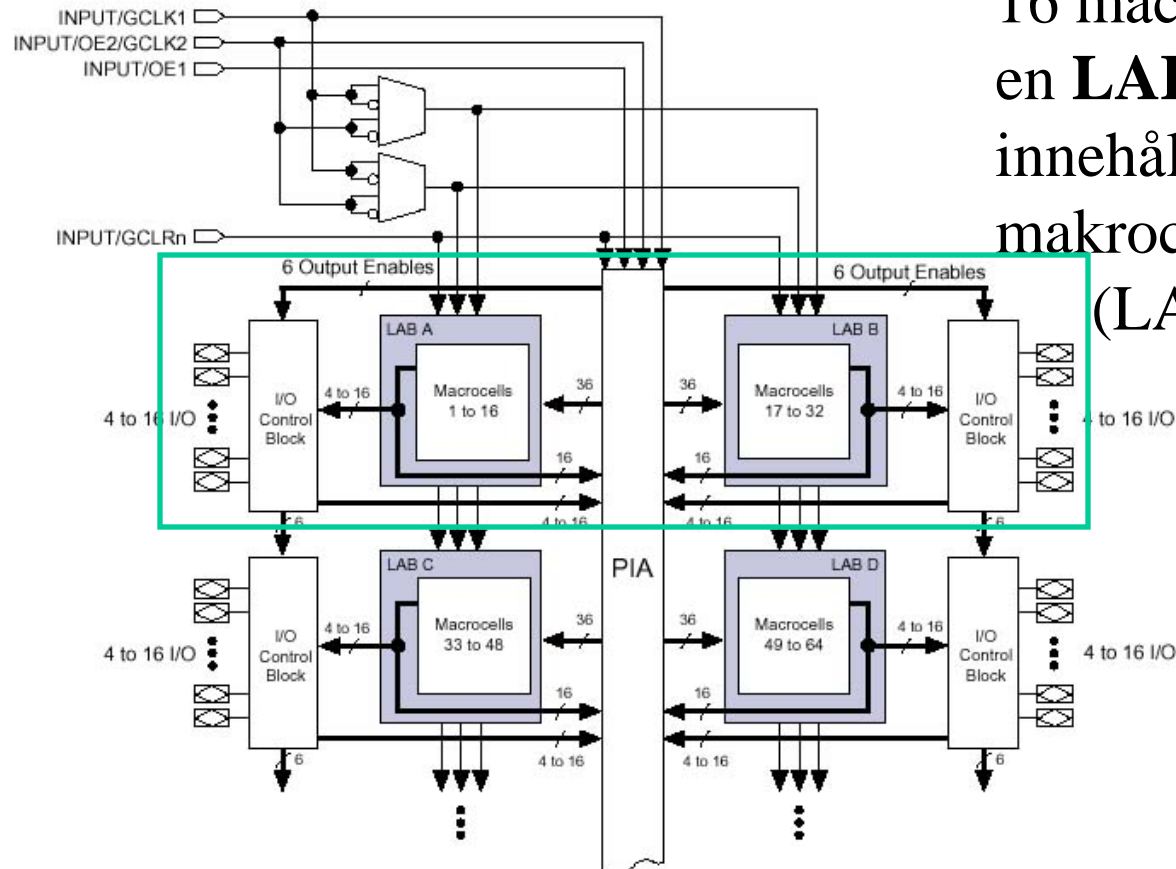
Med ett **breakoutboard** kan man använda kopplingsdäck till komponenter som egentligen är avsedda för ytmontering på kretskort. Man kan enkelt prova olika kopplingar.

På så sätt använder vi samma teknik som i föregående laboration – trots att vi nu går över till mer komplexa så kallade **CPLD**-kretsar och programmerar dem med VHDL-språket.

MAX-krets



Figure 1. MAX 3000A Device Block Diagram



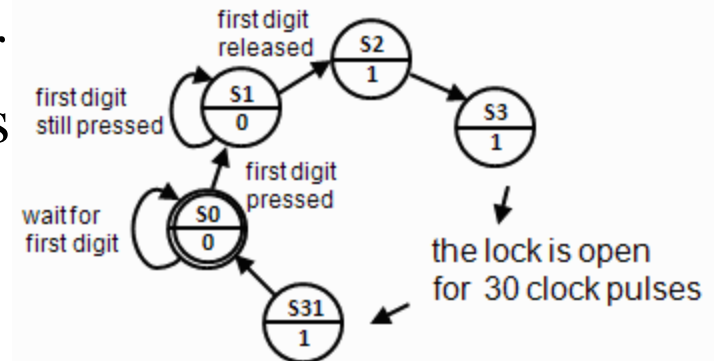
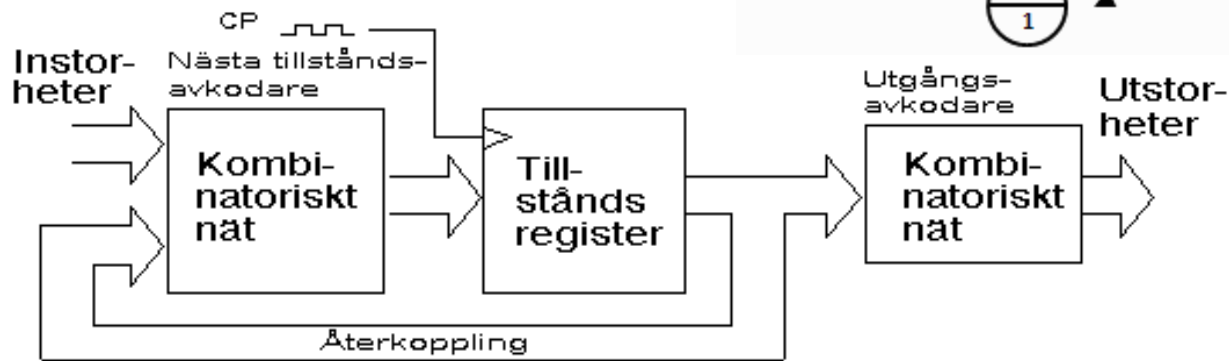
16 macroceller bildar
en **LAB**. Vår labkrets
innehåller 32 st
makroceller, 2 st LAB.
(LAB A LAB B)

Blocket **PIA**
används för att
sammanbinda LAB-
enheterna.

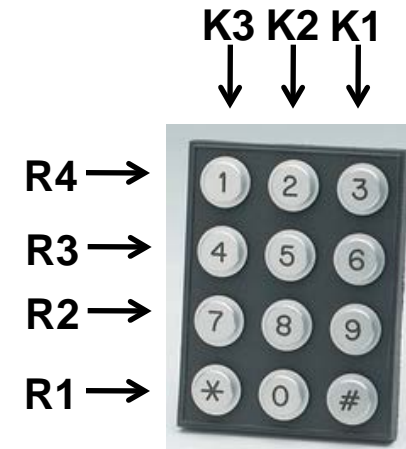
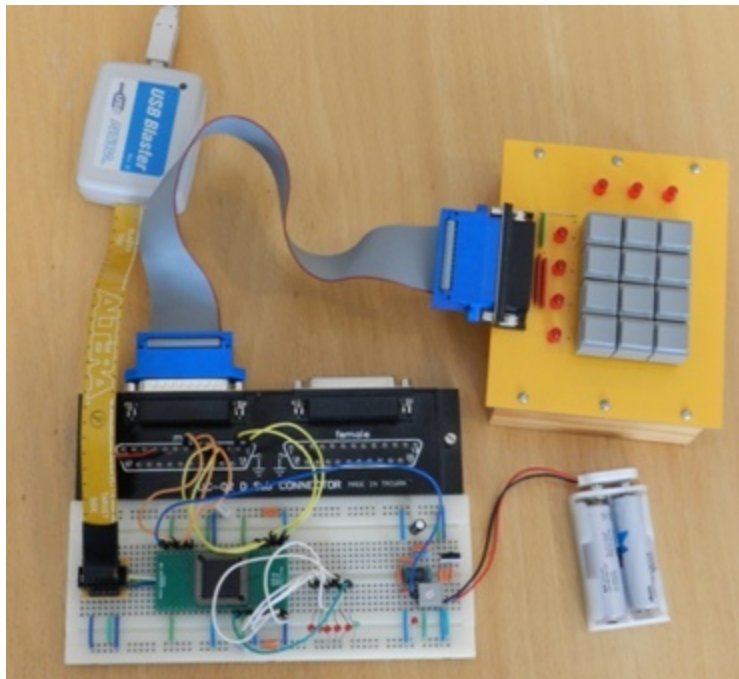
Laborationsuppgift - kodlås



- **Uppgift:** att skriva VHDL kod för ett kodlås som öppnas med koden ”de fyra sista siffrorna i ditt personnummer”.
- **Ledning:** en VHDL ”mall” för ett *förenklat* kodlås som öppnas med koden ”siffran ett”.



LAB utrustning med MAX-krets

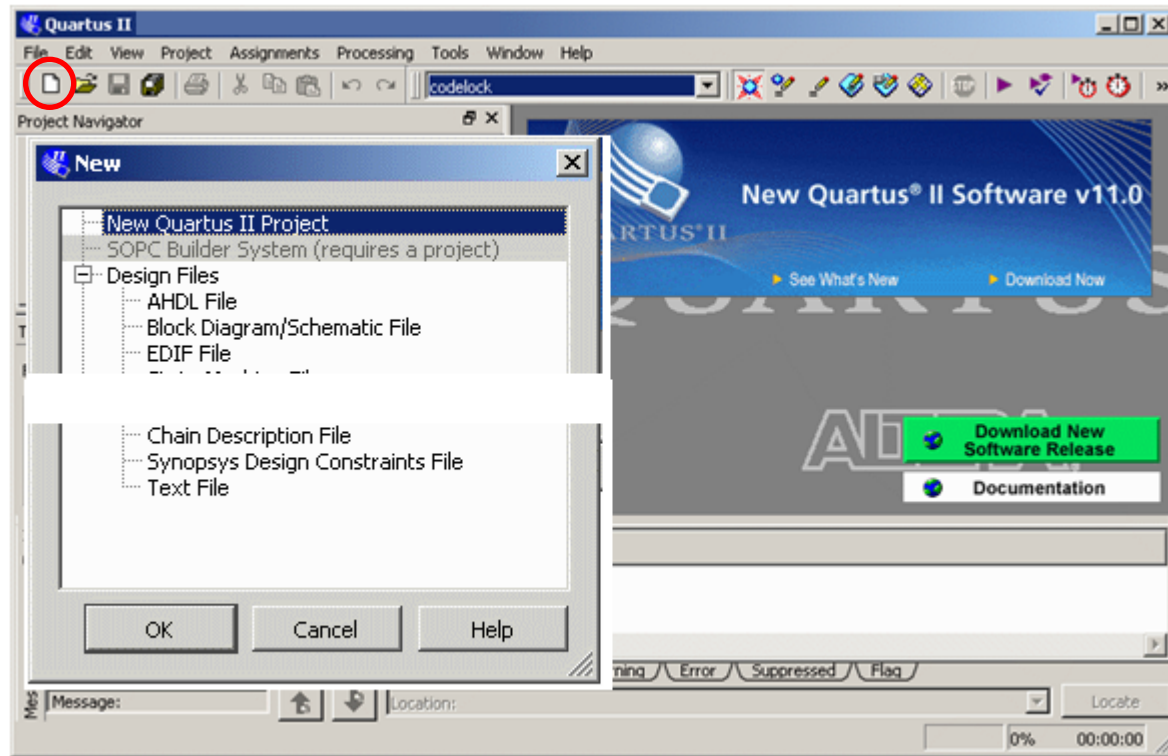


Tangenterna avkodas som
”en av tre” **K**olumner,
och ”en av fyra” **R**ader.

Quartus tutorial för MAX CPLD för skolans centralt administrerade datorer

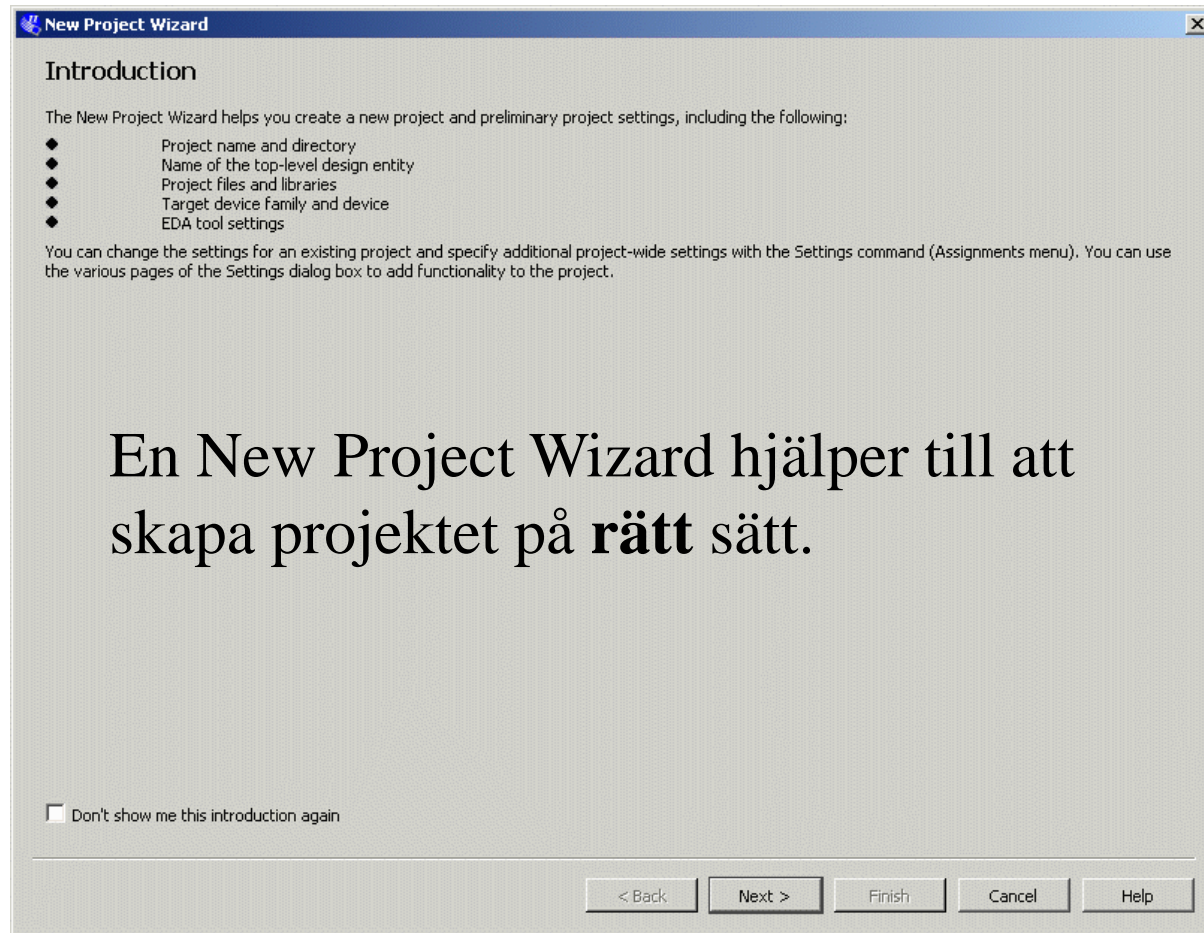


QuartusII



- Börja med att skapa ett projekt.
File, New, New Quartus II Project

New Project Wizard



En New Project Wizard hjälper till att skapa projektet på **rätt** sätt.

Project Name and Directory

New Project Wizard

Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?

H:/MAXwork

What is the name of this project?

codelock

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

codelock

Use Existing Project Settings...

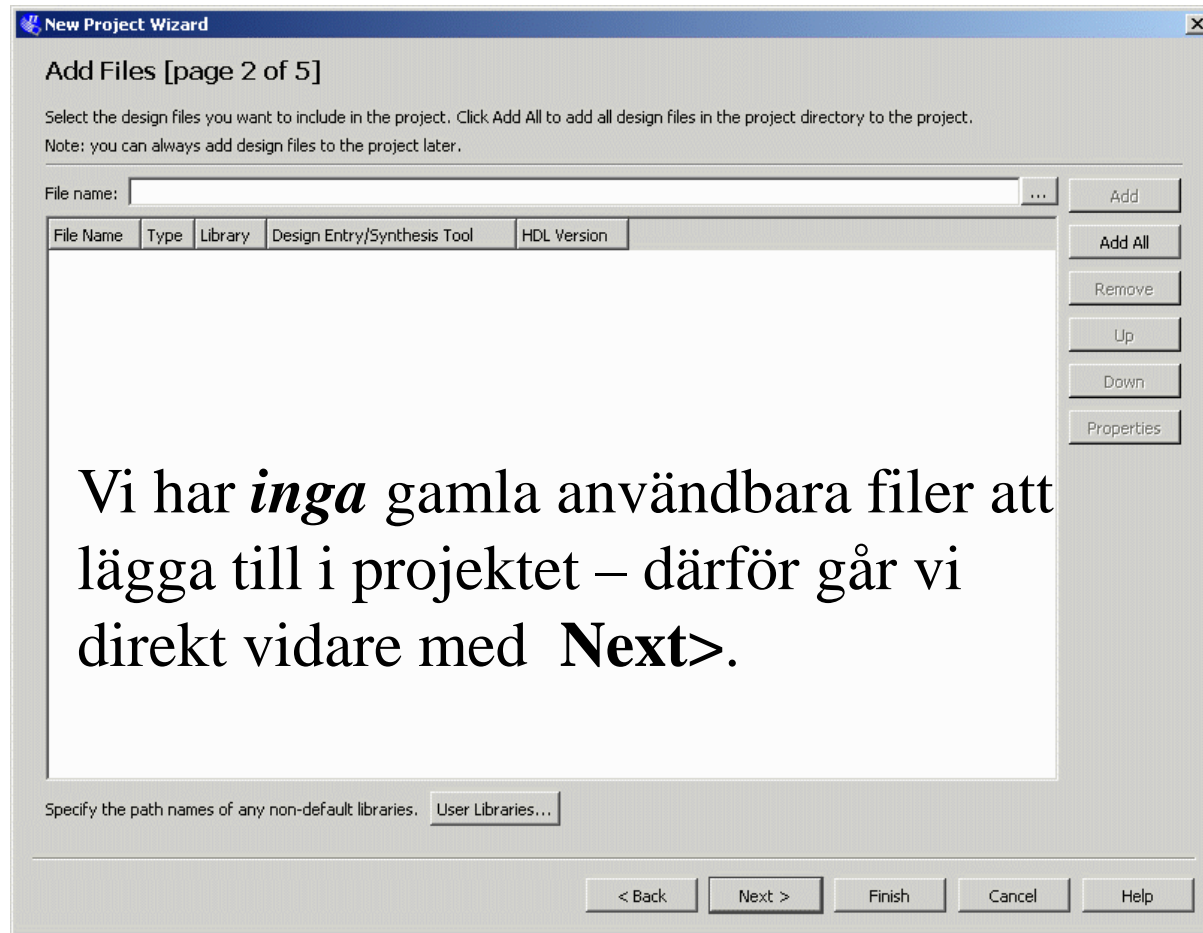
< Back Next > Finish Cancel Help

I skolan måste hela projektet ligga på ditt **H: **

- **Name: codelock**
- **Top-Level Entity: codelock**

(OBS **codelock** måste "matcha" det namn
Du senare anger som **entity** i din VHDL-fil)

Add Files



Family and Device Settings



New Project Wizard

Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.

Device family

Family:

Devices:

Target device

☐ Auto device selected by the Filter

☒ Specific device selected in 'Available devices' list

☐ Other: n/a

Show in 'Available devices' list

Package:

Pin count:

Speed grade:

☒ Show advanced devices

☐ HardCopy compatible only

Available devices:

Name	Core Voltage	Macrocells
EPM3032ALC44-4	3.3V	32
EPM3032ALC44-7	3.3V	32
EPM3032ALC44-10	3.3V	32
EPM3032ATC44-4	3.3V	32
EPM3032ATC44-7	3.3V	32
EPM3032ATC44-10	3.3V	32
EPM3032ATI44-10	3.3V	32
FPM3064AI C44-4	3.3V	64

Companion device

HardCopy:

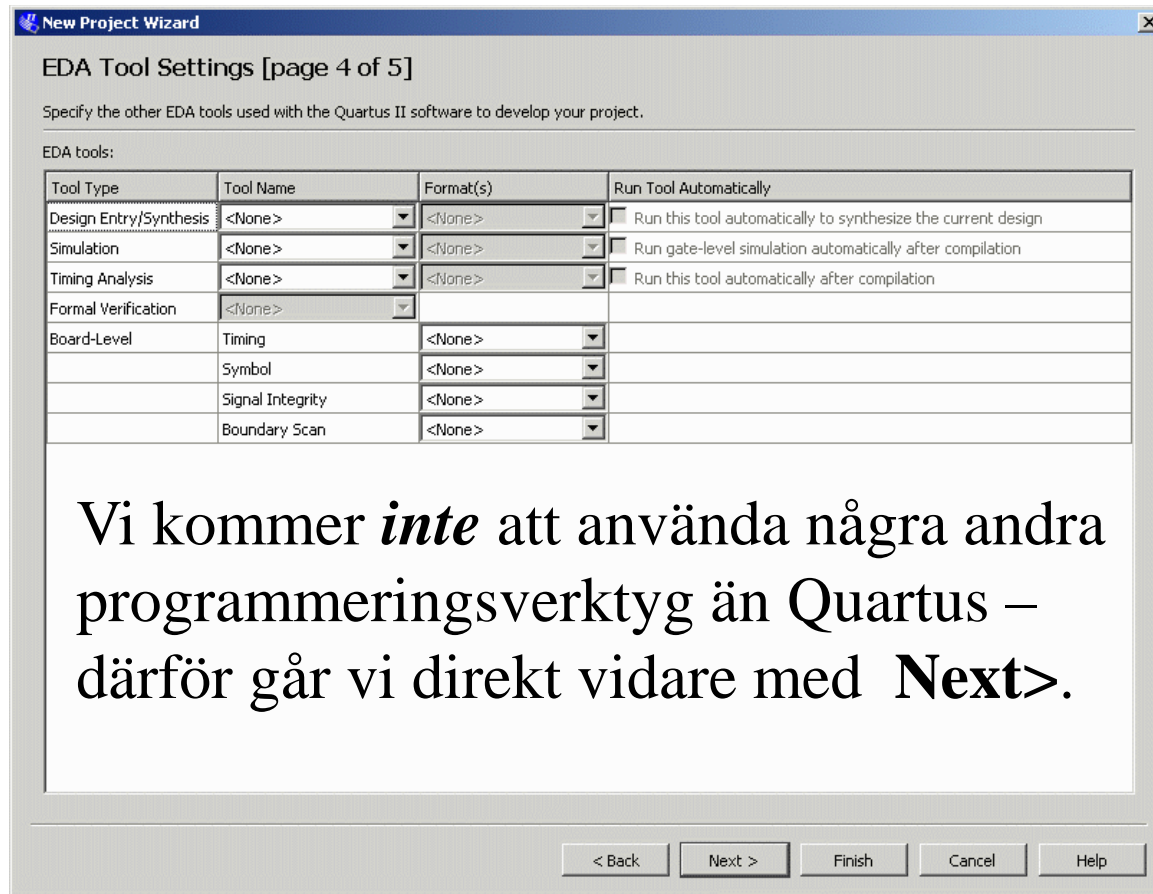
☐ Limit DSP & RAM to HardCopy device resources

< Back Next > Finish Cancel Help

Family: MAX3000A Available devices: EPM3032ALC44-10

William Sandqvist william@kth.se

EDA Tool Settings



New Project Wizard

EDA Tool Settings [page 4 of 5]

Specify the other EDA tools used with the Quartus II software to develop your project.

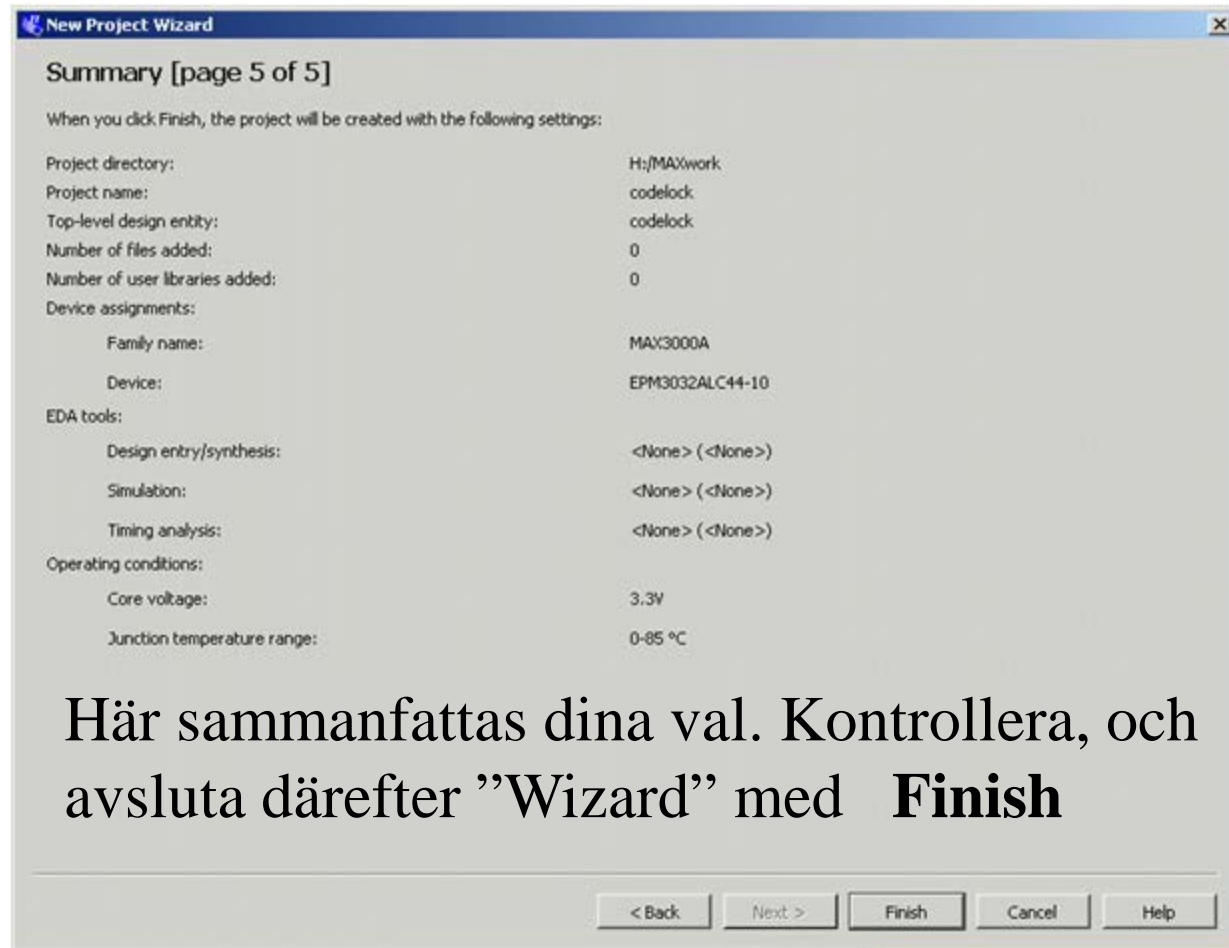
EDA tools:

Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/Synthesis	<None>	<None>	<input checked="" type="checkbox"/> Run this tool automatically to synthesize the current design
Simulation	<None>	<None>	<input checked="" type="checkbox"/> Run gate-level simulation automatically after compilation
Timing Analysis	<None>	<None>	<input checked="" type="checkbox"/> Run this tool automatically after compilation
Formal Verification	<None>		<input type="checkbox"/>
Board-Level	Timing	<None>	
	Symbol	<None>	
	Signal Integrity	<None>	
	Boundary Scan	<None>	

Vi kommer *inte* att använda några andra programmeringsverktyg än Quartus – därför går vi direkt vidare med **Next>**.

< Back Next > Finish Cancel Help

Summary - sammanfattning

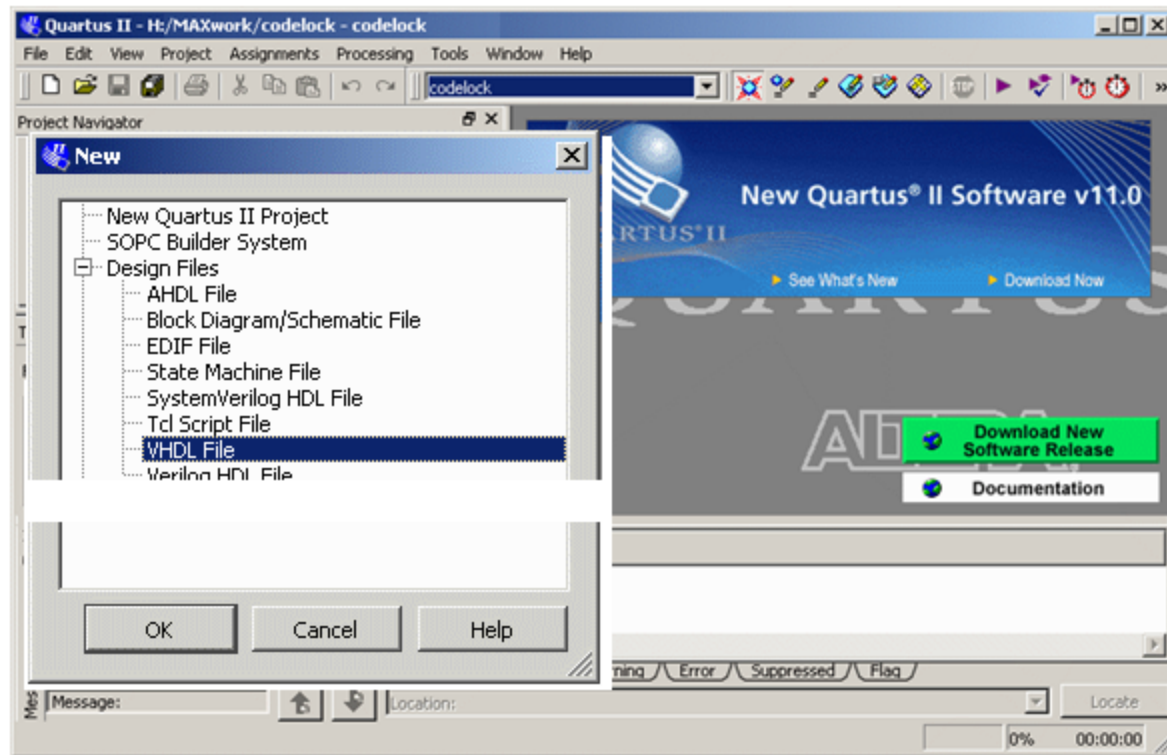


Här sammanfattas dina val. Kontrollera, och avsluta därefter "Wizard" med **Finish**

Projektet har skapats

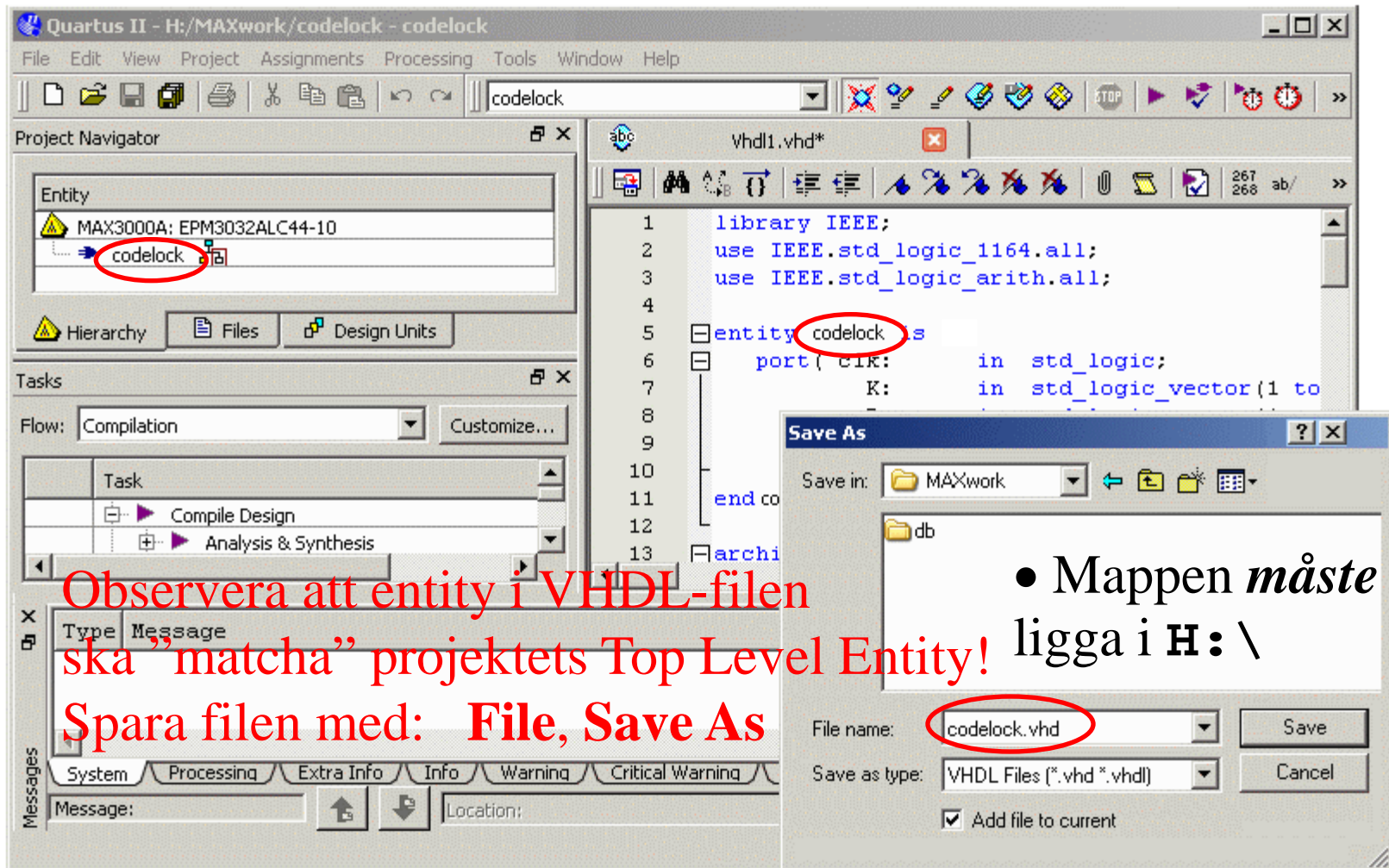


VHDL koden



- Skapa en blank fil för VHDL-koden. **File, New, VHDL File**
- Mallprogrammet är komplett (men det är till ett förenklat kodlås).

Klistra in VHDL koden



The screenshot shows the Quartus II IDE with the following components:

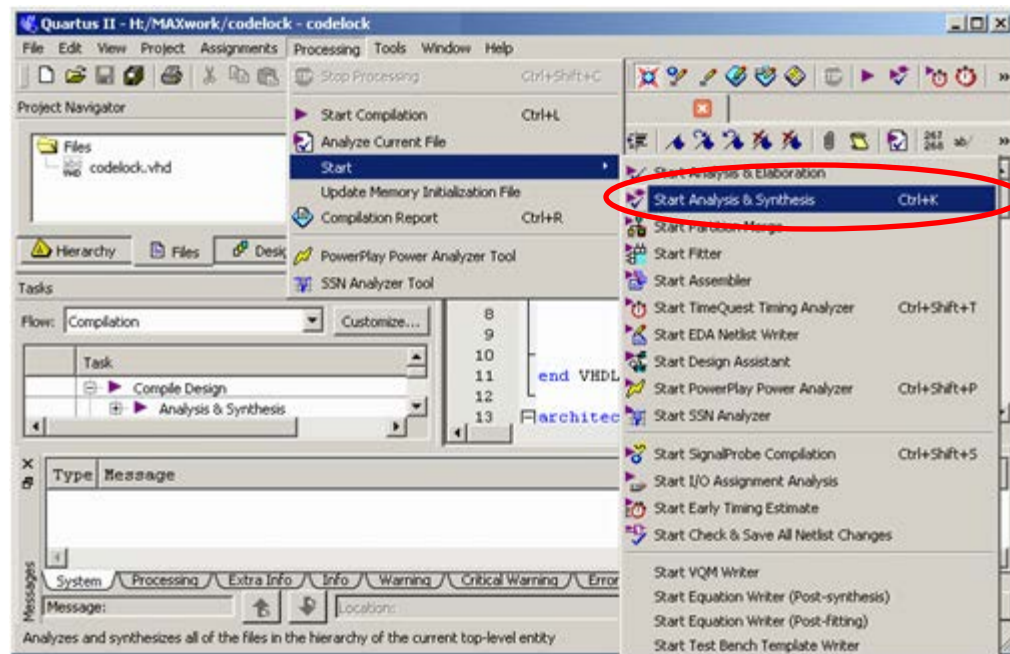
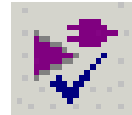
- Project Navigator:** Shows the entity hierarchy with "MAX3000A: EPM3032ALC44-10" and "codelock" (circled in red).
- Tasks:** Shows the "Compile Design" and "Analysis & Synthesis" tasks.
- VHDL Editor:** Displays the code for "entity codelock" (circled in red). The code includes:

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3 use IEEE.std_logic_arith.all;
4
5 entity codelock is
6     port ( clk: in std_logic;
7           K: in std_logic_vector(1 to
8
9
10
11 end codelock;
12
13 architecture
```
- Save As Dialog:** Shows the file name "codelock.vhd" (circled in red) and the save location "MAXwork\db".

Observera att entity i VHDL-filen ska "matcha" projektets Top Level Entity! Spara filen med: **File, Save As**

**• Mappen *måste* ligga i H: **

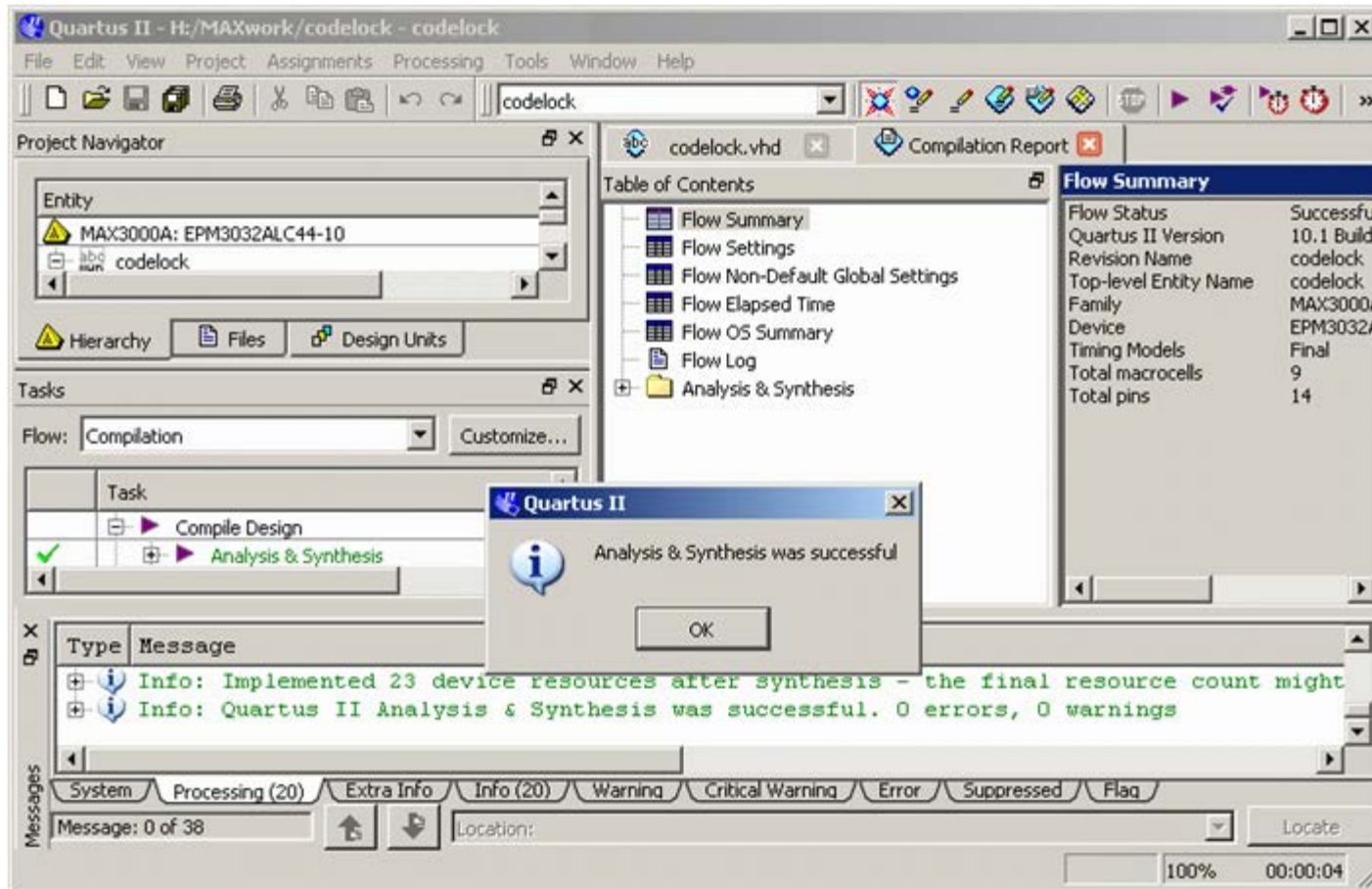
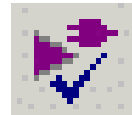
Analysis and Synthesis



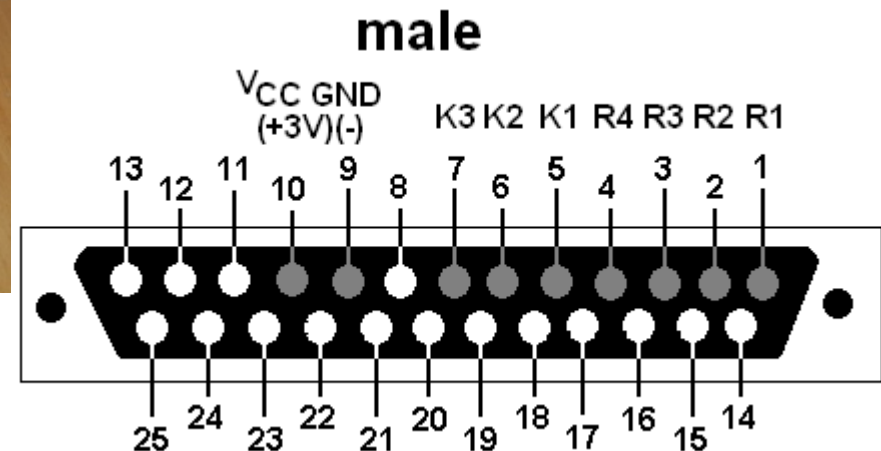
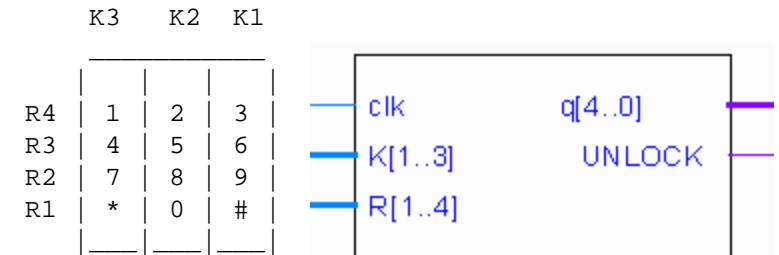
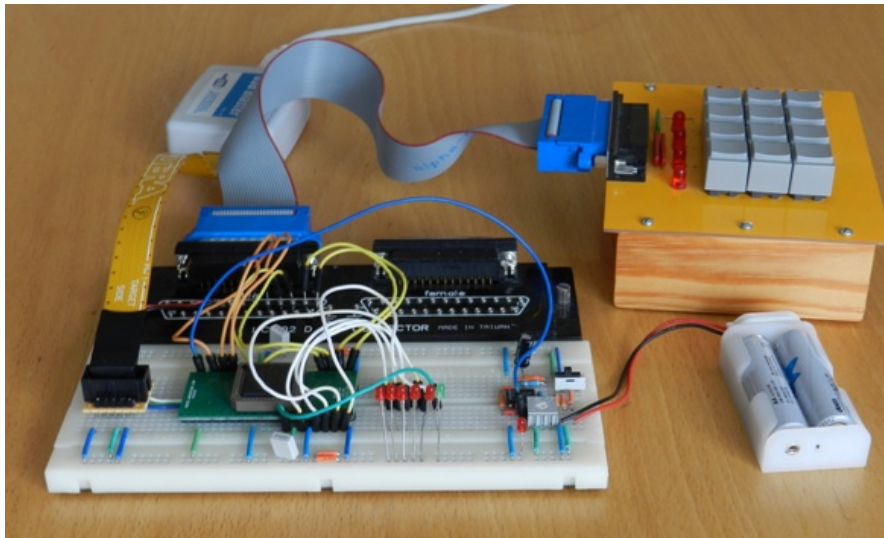
När man har nyskriven kod är det onödigt att köra hela verktygskedjan – risken är stor att det finns felaktigheter längs vägen ...

- Från början kör man bara **Analysis & Synthesis**.

Analysis and Synthesis

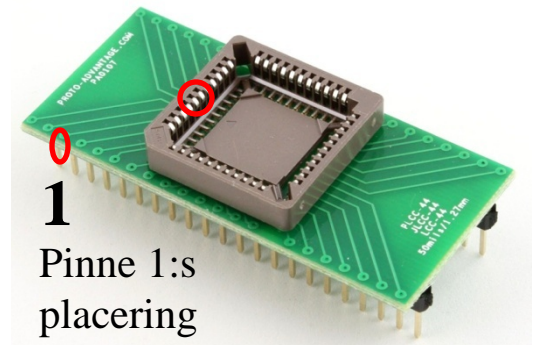


Labutrustningarna har *olika* ledningsdragnings!



Det är DB25 kontaktens stift nr 1... 10 som används av tangenterna.

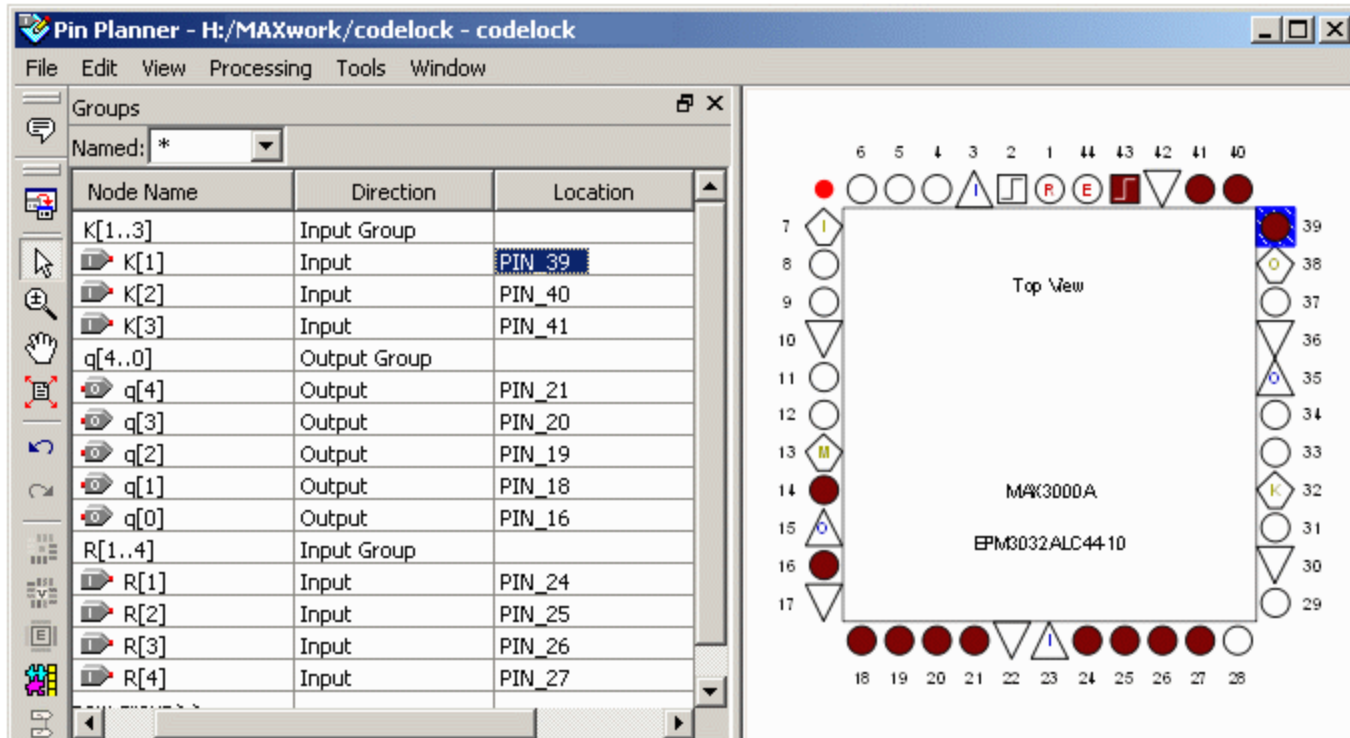
Vad gäller för just *din* utrustning?



*Så här identifierar
Du pinne nr 1.*

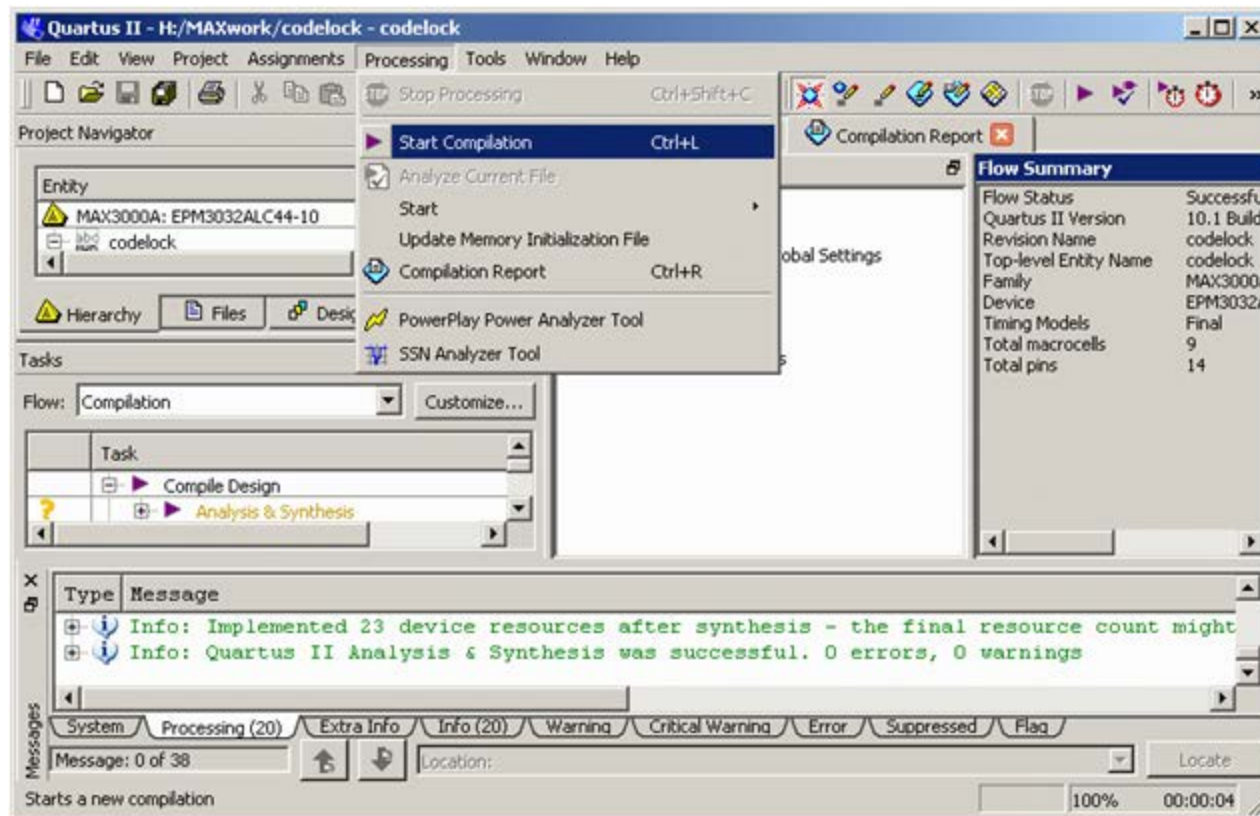
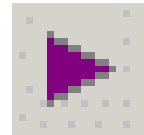


Pin Planner



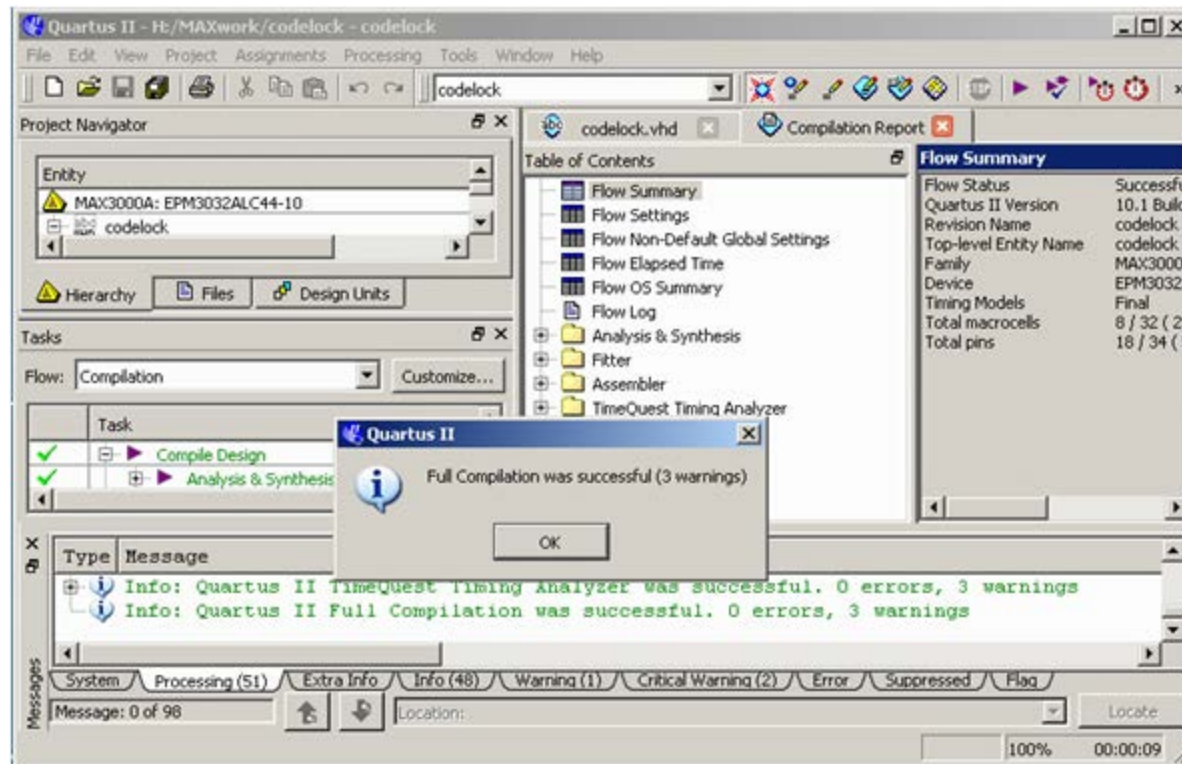
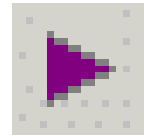
Vid lab är alla utrustningarna kopplade på olika sätt, så Du måste göra en egen pin-planering för din labutrustning. Bildens pinnplan kan bara ses som ett exempel.

Start Compilation



- **Start Compilation** kör *hela* verktygskedjan.

Compilation successful



De 3 varningarna (fler med annan programversion) handlar om ”verktyg” som *saknas* i vår programversion men som vi *inte* behöver.

William Sandqvist william@kth.se

Chip-programmering



JTAG

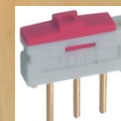
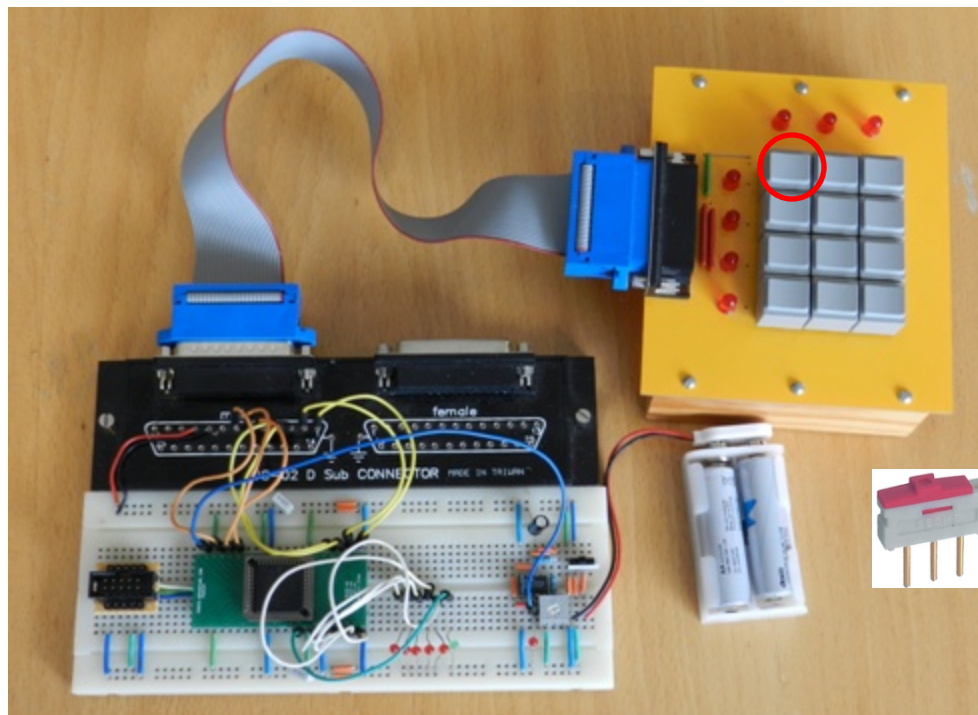
TDI	-	TMS	TDO	TCK
GND	-	-	VCC	GND



En **JTAG** kontakt är ansluten till MAX-chippet för ”in circuit programming”.

Chip-programmering sker med en ***USB-blastar***.

Prova funktionen!

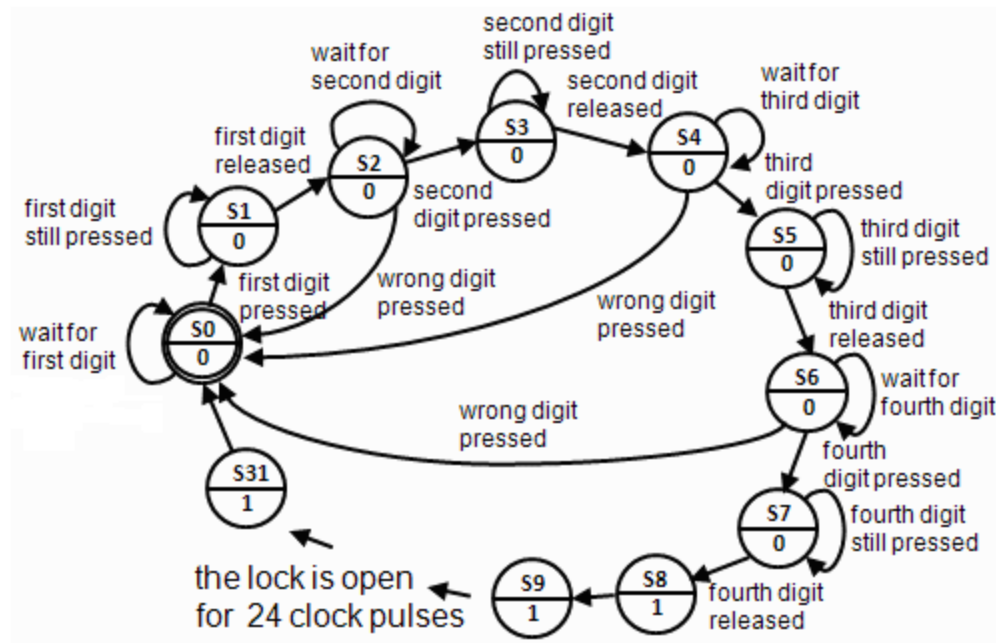


Power
On/Off

Mall-programmet gäller ett förenklat kodlås som öppnar för tangenten "1", lite väl enkelt kan nog tyckas ... !

Öppna låset med ditt personnummer!

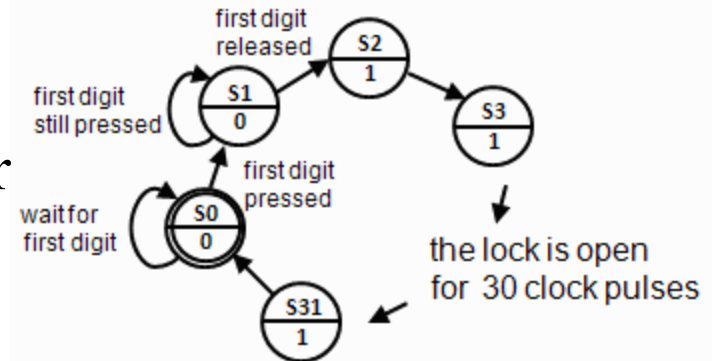
- Nu är det dags att skriva om VHDL-koden så att låset öppnar för de fyra sista siffrorna i ditt personnummer!*



(Om Du förbereder koden för ditt personnummer, så kan två i en laborationsgrupp bidra med hälften av koden var vid laborationen).

Beskrivning av kodlåsmallen

Kodlåsmallen gäller för ett förenklat lås som öppnar direkt när man trycker på tangenten för ”1”.

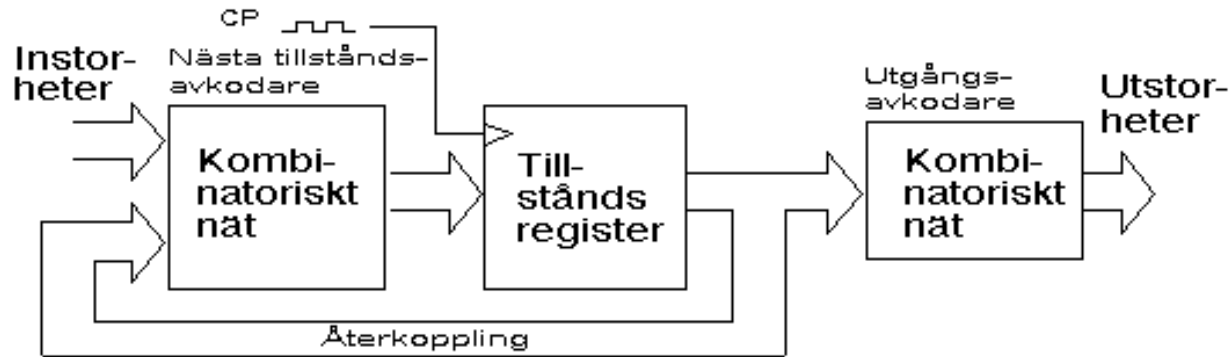


Så gott som all digital design sker numera med hjälp av högnivåspråk som VHDL/VERILOG. Vår grundkurs i digital-teknik ger *inte* utrymme att lära ut VHDL-språket, däremot kommer Du att kunna omforma ”kodlåsmallen” till användbar VHDL-kod inför laborationen.

Tycker Du att VHDL-språket verkar intressant, så har skolan flera digitaltekniska fortsättningskurser ...

Moore automat

next_state_decoder: state_register: output_decoder:



De olika blocken identifieras i koden med etiketter, "label"

next_state_decoder:

output_decoder:

state_register:

VHDL process

Med en "process" kan man beskriva ***vad*** ett block ska utföra utan att behöva gå in på detaljer om ***hur*** detta skall gå till.

Label. Etikett.

`next_state_decoder:`

`process(state, K, R)`

`begin`

`...`

`end process;`

Sensitivity list. Varje *förändring* av dessa variabler leder till att processen uppdateras!

Body. Vad som skall utföras.

lockmall.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

start → entity codelock is
    port( clk:      in  std_logic;
          K:        in  std_logic_vector(1 to 3);
          R:        in  std_logic_vector(1 to 4);
          q:        out std_logic_vector(4 downto 0);
          UNLOCK:   out std_logic );
end codelock;

start → architecture behavior of codelock is
    subtype state_type is integer range 0 to 31;
    signal state, nextstate: state_type;

    begin
        nextstate_decoder: -- next state decoding part
        process(state, K, R)
        begin
            case state is
                when 0 => if (K = "100" and R ="0001")    then nextstate <= 1;
                           else nextstate <= 0;
                           end if;
                when 1 => if (K = "100" and R ="0001")    then nextstate <= 1;
                           elsif (K = "000" and R = "0000") then nextstate <= 2;
                           else nextstate <= 0;
                           end if;
                when 2 to 30 => nextstate <= state + 1;
                when 31      => nextstate <= 0;
            end case;
        end process;

        debug_output: -- display the state
        q <= conv_std_logic_vector(state,5);

        output_decoder: -- output decoder part
        process(state)
        begin
            case state is
                when 0 to 1  => UNLOCK <= '0';
                when 2 to 31 => UNLOCK <= '1';
            end case;
        end process;

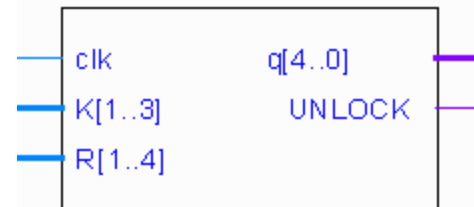
        state_register: -- the state register part (the flipflops)
        process(clk)
        begin
            if rising_edge(clk) then
                state <= nextstate;
            end if;
        end process;
    end behavior;
end
```

entity
architecture
next_state_decoder:
output_decoder:
state_register:

Kodlås VHDL

- **entity** Block-beskrivning, insignaler och utsignaler

```
library IEEE;  
use IEEE.std_logic_1164.all;  
use IEEE.std_logic_arith.all;
```



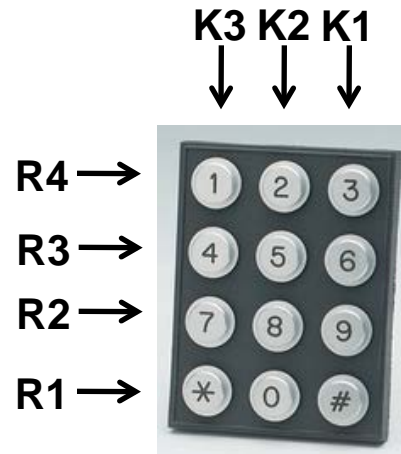
start

```
entity codelock is  
  port( clk:      in  std_logic;  
        K:        in  std_logic_vector(1 to 3);  
        R:        in  std_logic_vector(1 to 4);  
        q:        out std_logic_vector(4 downto 0);  
        UNLOCK:   out std_logic );  
end codelock;
```

end

Bitvektorer och bitar

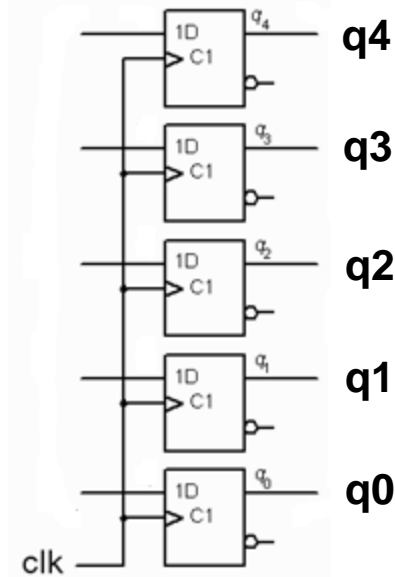
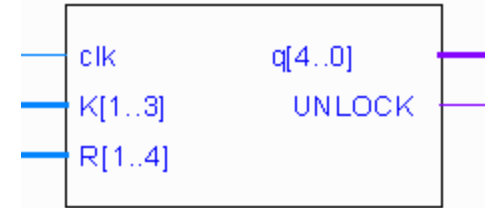
Man kan anpassa indexeringen av variabler så att den överensstämmer med databladen – mindre risk för misstag!



```
K: in std_logic_vector(1 to 3);  
R: in std_logic_vector(1 to 4);  
q: out std_logic_vector(4 downto 0);
```

1 2 3
K = "001" bitvektor
K(3) = '1' bit

4 3 2 1 0
q = "00001" bitvektor
q(0) = '1' bit



Kodlås VHDL ...

- **architecture**

Beskrivning av
blockets beteende



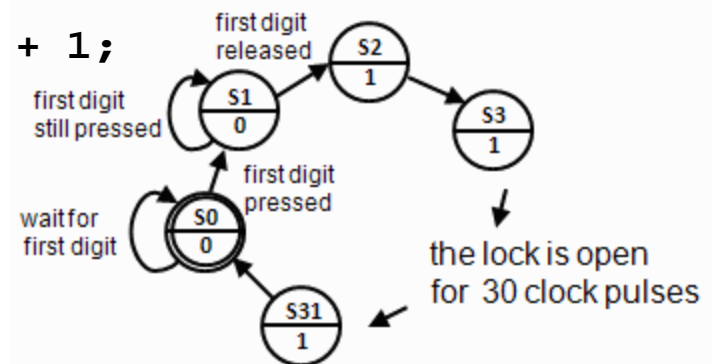
```
architecture behavior of codelock is
  subtype state_type is integer range 0 to 31;
  signal state, nextstate: state_type;

begin;
```

Här skapar vi en ny datatyp, **state_type**, som kan ha heltalsvärden mellan 0 och 31. Kompilatorn förhindrar oss då från att (av misstag) använda andra värden. Signalerna **state** och **nextstate** är av denna typ.

Kodlås VHDL ...

```
nextstate_decoder: -- next state decoding part
process(state, K, R)
begin
  case state is
    when 0 => if (K = "001" and R = "0001") then nextstate <= 1;
               else nextstate <= 0;
               end if;
    when 1 => if (K = "001" and R = "0001") then nextstate <= 1;
               elsif (K = "000" and R = "0000") then nextstate <= 2;
               else nextstate <= 0;
               end if;
    when 2 to 30 => nextstate <= state + 1;
    when 31 => nextstate <= 0;
  end case;
end process;
```



Kodlås VHDL ...

För att kunna felsöka vill vi kunna följa vilket tillstånd automaten befinner sig i ...

```
debug_output:  -- display the state  
q <= conv_std_logic_vector(state,5);
```

Funktionen **conv_std_logic_vector()** omvandlar **state** (ett heltal mellan 0...31) till en 5-bitars bitvektor **q**, **q(4)** ... **q(0)**.

Kodlås VHDL ...

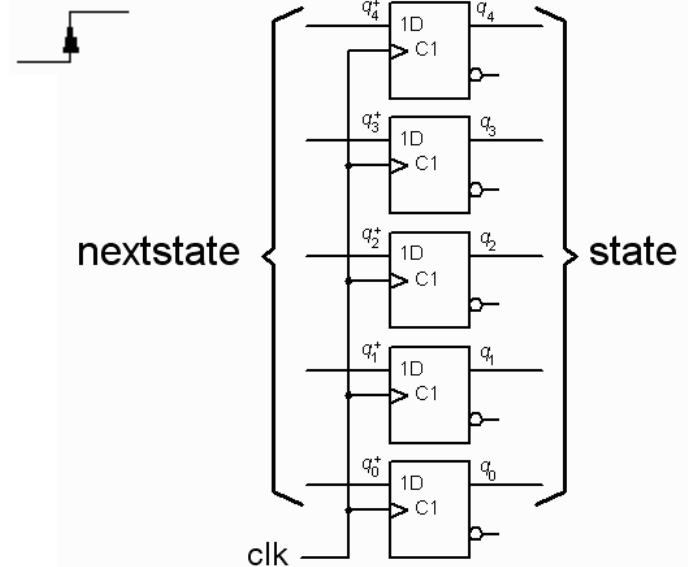
```
output_decoder: -- output decoder part
process(state)
begin
    case state is
        when 0 to 1  => UNLOCK <= '0';
        when 2 to 31 => UNLOCK <= '1';
    end case;
end process;
```

Kodlås VHDL ...

```
state_register: -- the state register part (the flipflops)
process(clk)
begin
    if rising_edge(clk) then
        state <= nextstate;
    end if;
end process;
```

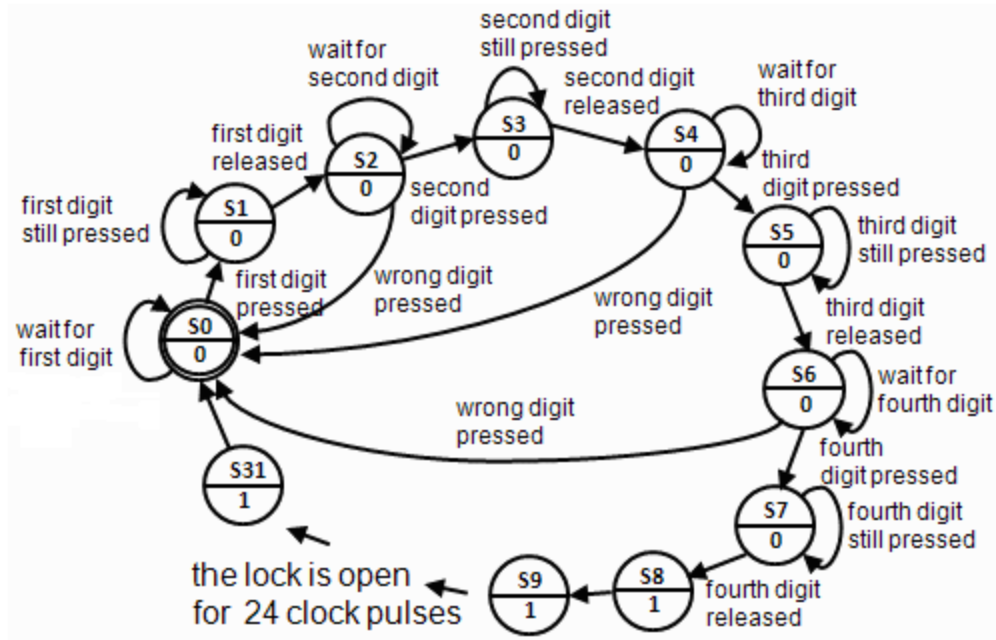
```
end behavior;
```

← end



Öppna låset med ditt personnummer!

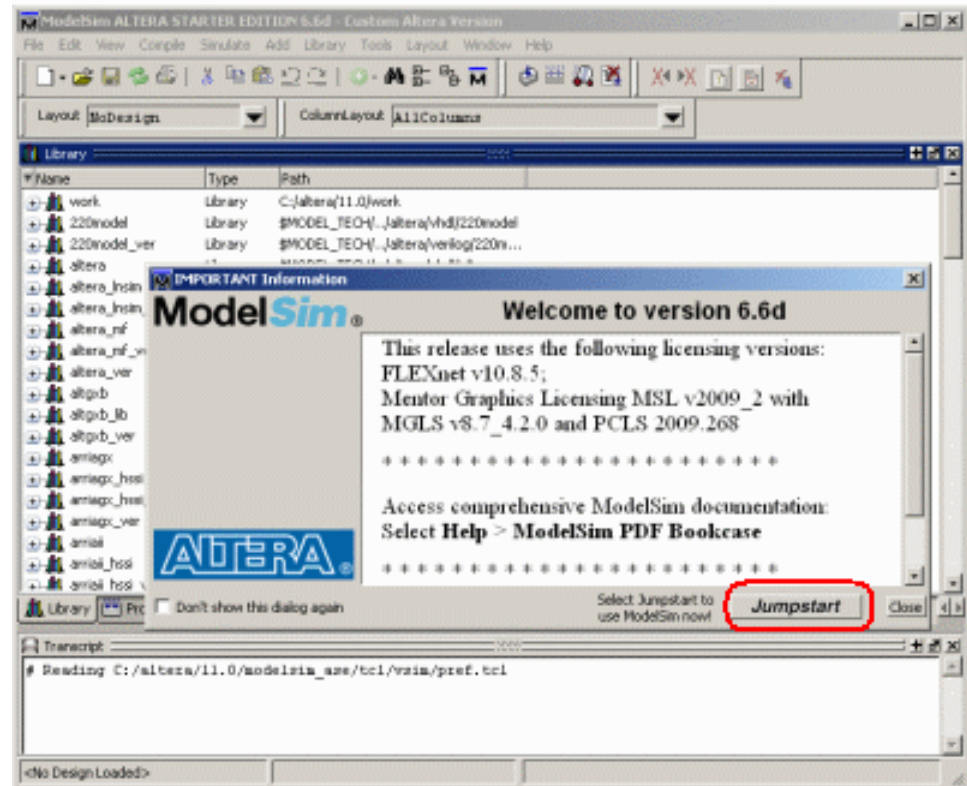
- Nu är det dags att skriva om VHDL-koden så att låset öppnar för de fyra sista siffrorna i ditt personnummer!*



Simulera med ModelSim

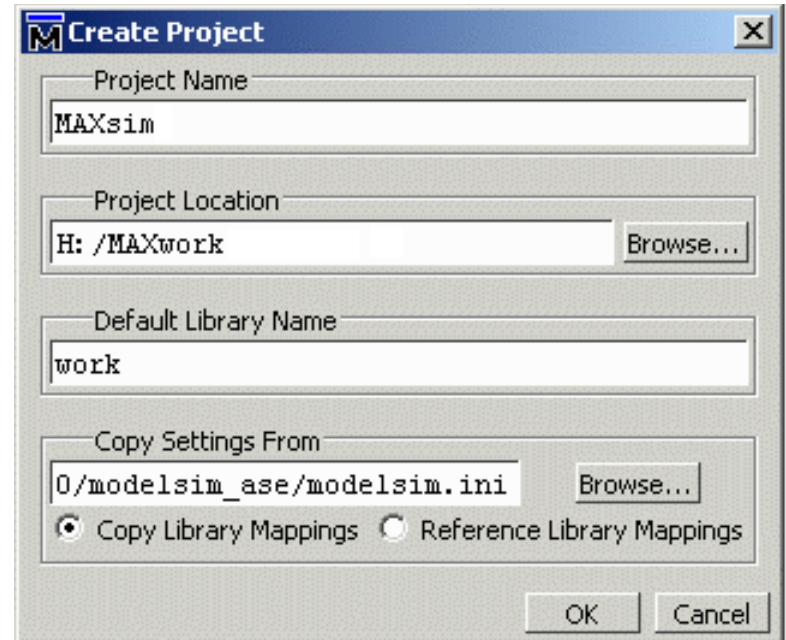
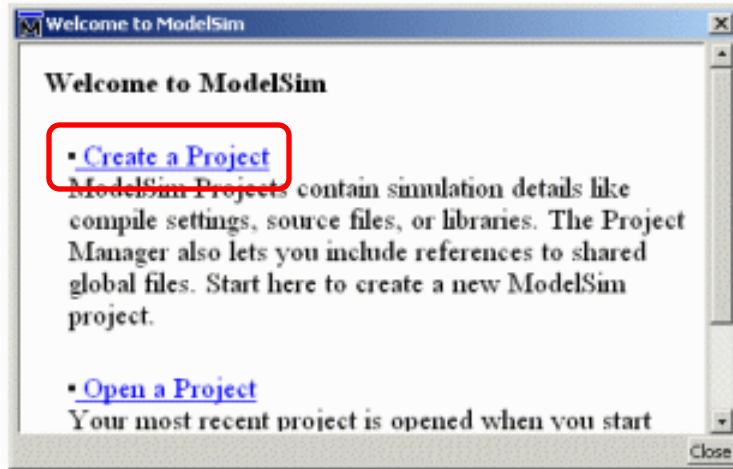
ModelSim kan användas till att simulera VHDL-kod, för att avgöra om den är "rätt" tänkt.

Man kan göra simuleringar som tar hänsyn till "tidsfördröjningar" och andra fenomen inuti den tänkta målkretsen.



Starta **ModelSim**. Klicka på **Jumpstart** för att få hjälp med att sätta upp ett projekt.

Skapa projekt



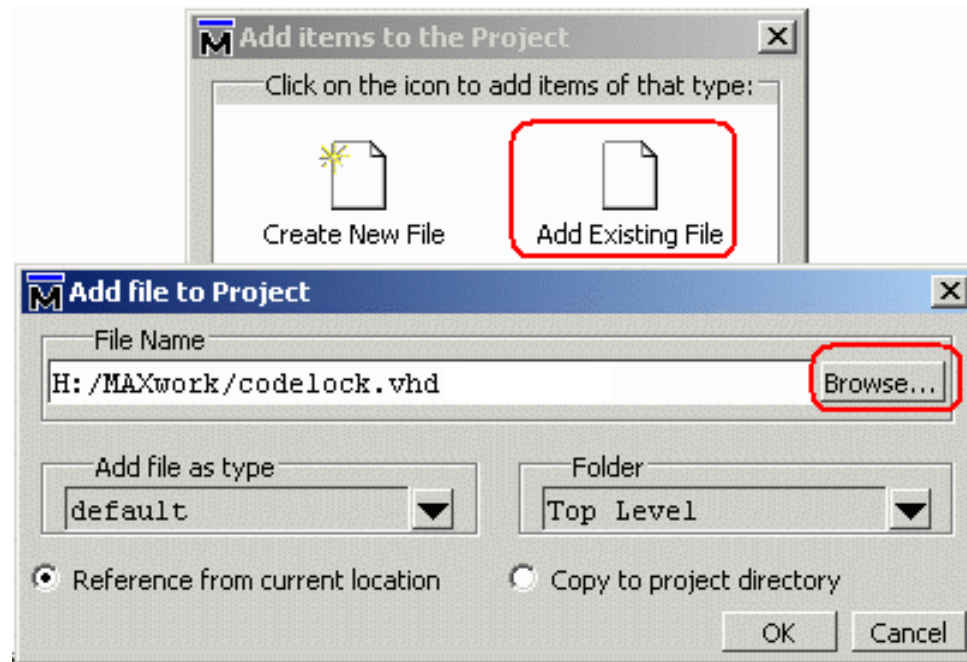
Project Name

MAXsim kan vara ett lämpligt namn

Project location

H: /MAXwork bläddra dig fram till *samma arbetsmapp* som Du använde för Quartus

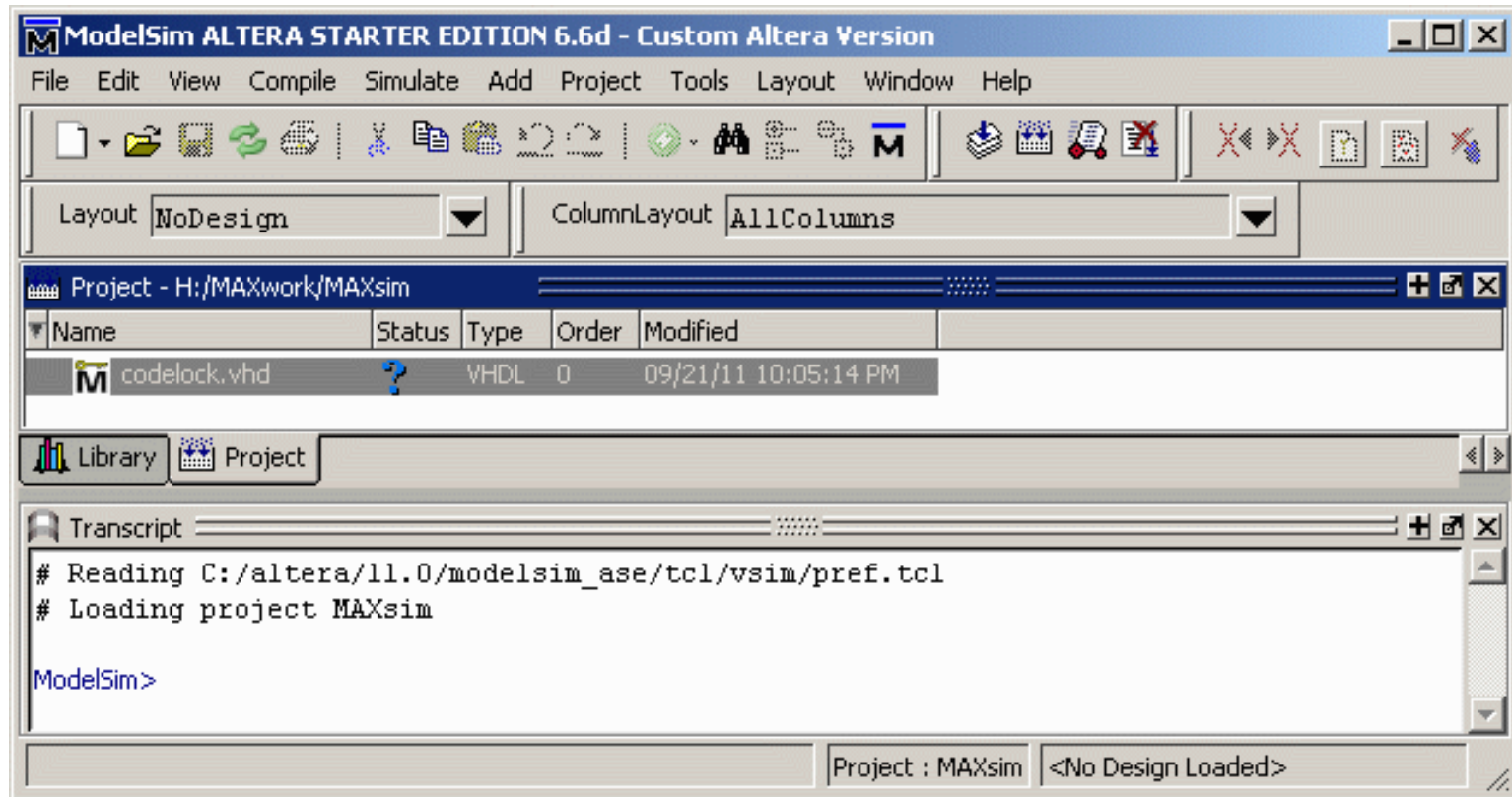
Lägg till VHDL-filen



Vi väljer "Add Existing File" för att lägga till en VHDL-fil till projektet.



"Bläddra" fram till filen **codelock.vhd** som vi tidigare skapade med **Quartus**.

Kodlåskoden i ModelSim





Kompilera för simulering

ModelSim har en *egen* kompilator för att ta fram simuleringen ur VHDL-koden. Fast vi har kompilerat VHDL-koden i **Quartus** måste vi trots det kompilera den igen för **ModelSim**.

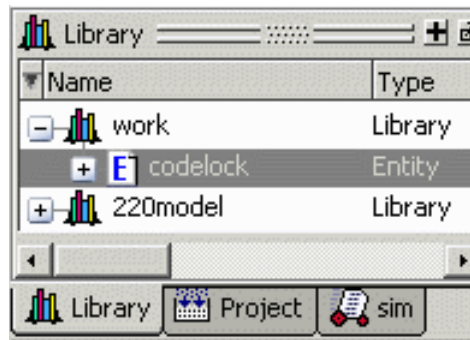
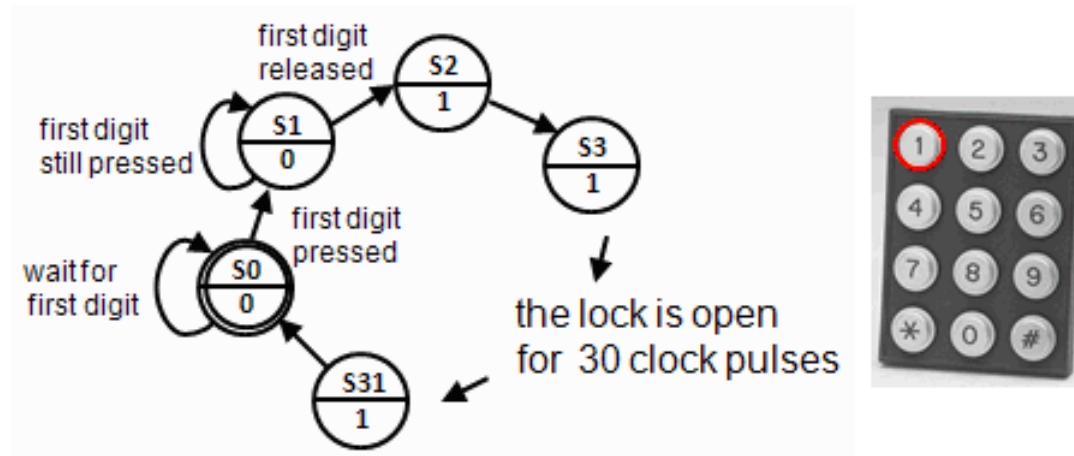
Name	Status
 codelock.vhd	

Välj **Compile** menyn, alternativet **Compile All**.

Name	Status
 codelock.vhd	

Nu är VHDL-koden också kompilerad för **Modelsim**. Statussymbolen ändras från ett blått frågetecken till en grön bock!

Simulera kodlås-mallen



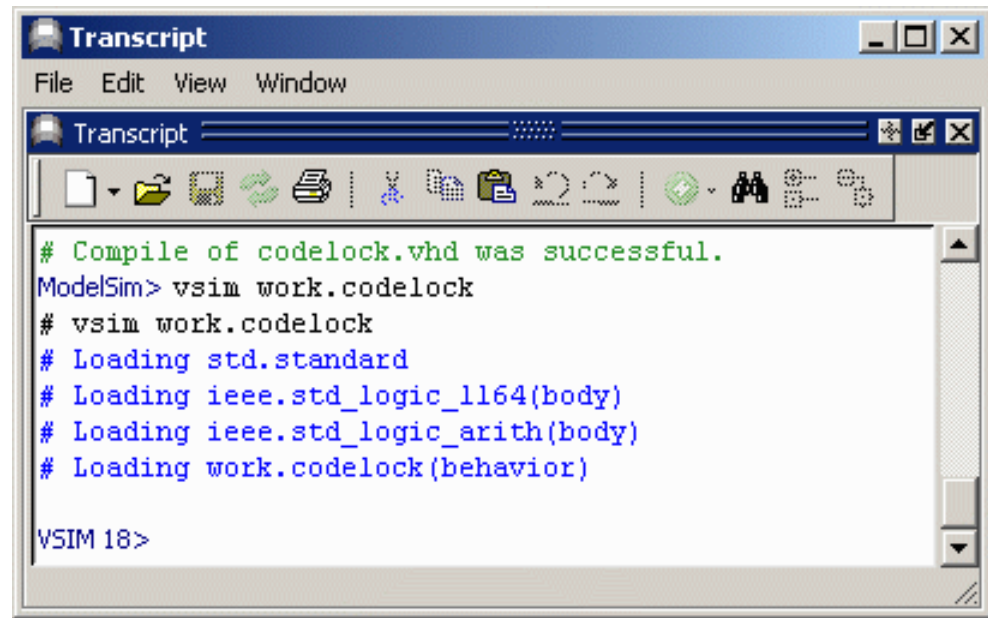
Ladda Designen till simulatorn.
Välj fliken **Library**, och öppna mappen **work**. Dubbelklicka på "Entity" codelock.

Transcript-fönstret

En serie kommandon utförs nu som resulterar i att designen laddats in till simulatoren.

I **Transcript**-fönstret kan man följa vilka kommandon det är som utförts.

Transcript-fönstret är ett terminalfönster där man ger kommandon, men man kan även ge de flesta kommandon genom menyval, eller genom att klicka på knappar.



```
Transcript
File Edit View Window

# Compile of code.lock.vhd was successful.
ModelSim> vsim work.code.lock
# vsim work.code.lock
# Loading std.standard
# Loading ieee.std_logic_1164(body)
# Loading ieee.std_logic_arith(body)
# Loading work.code.lock(behavior)

VSIM 18>
```

Kommandon skrivs dock ut i ut **Transcript**-fönstret, oavsett hur dom givits.

Förbered simulering



Vi behöver ha ett antal fönster öppna för att kunna följa simuleringen.

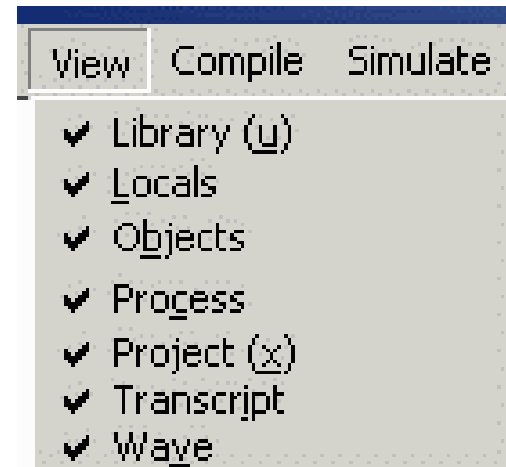
Ge kommandon i **Transcript**-fönstret eller klicka för i **View**-menyn.

```
VSIM> view objects
```

```
VSIM> view locals
```

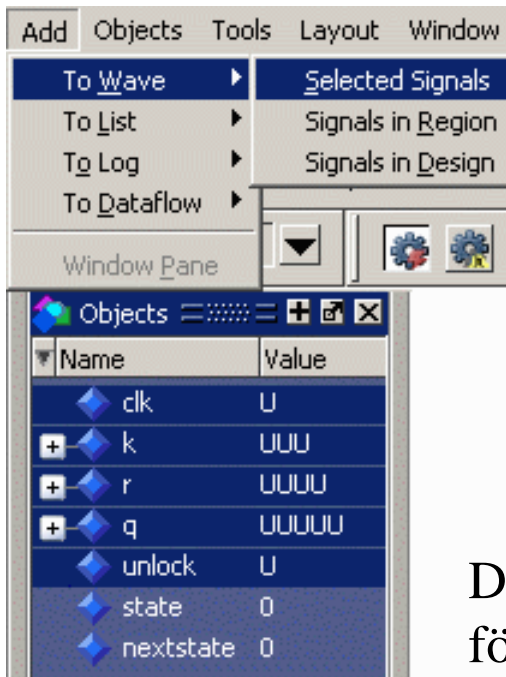
```
VSIM> view source
```

```
VSIM> view wave -undock
```



Modelsim består av "fönster". Det kan vara svårt att se allt på en gång. Med knappen **Zoom/Unzoom** förstörar man fönstret. Med knappen **Dock/Undock** kan fönstret flyttas till valfri plats. Med knappen **Close** stänger man fönster.

Signaler i Wave-fönstret



Signaler i Wave

Har man många signaler är det en bra idé att välja ut de signaler man är intresserad av att följa i **Wave**-fönstret, men här väljer vi att följa alla:

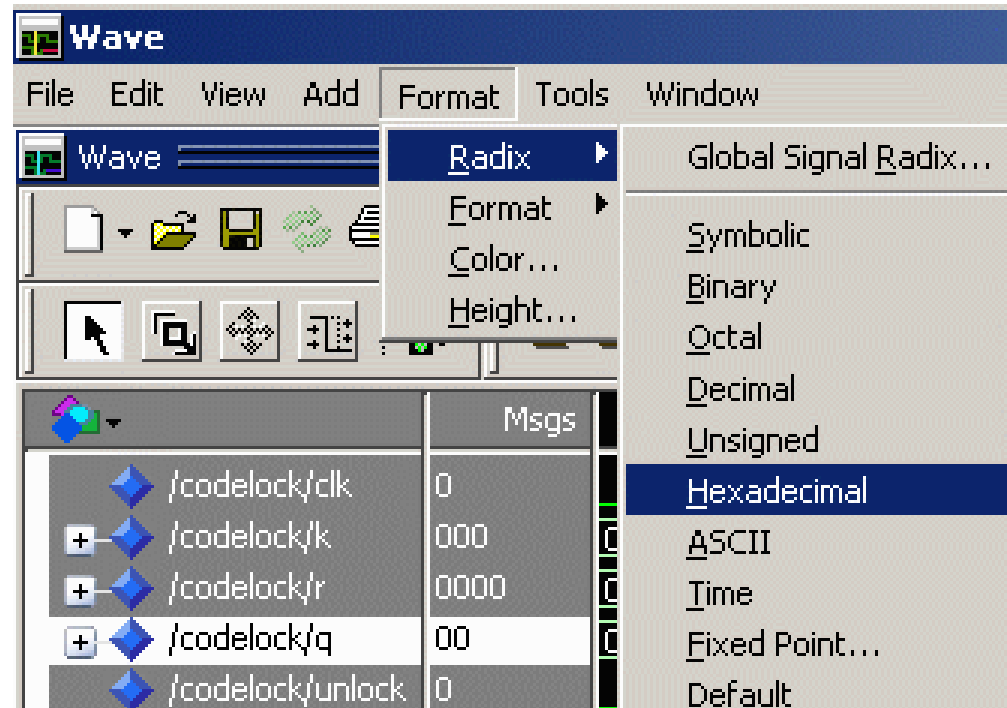
add wave *

Det finns flera sätt att lägga till signaler till **Wave**-fönstret:

- Välj signaler i **Object**-fönstret och "dra och släpp" urvalet till **Wave**-fönstret.
- Högerklicka i **Object**-fönstret och välj **Add to Wave**.

Format, Radix, Hexadecimal

Tillståndsvariabeln **q** har 32 olika tillstånd, en sådan variabel är lättare att följa om den anges som en hexadecimal siffra, 00 ... 1F i stället för som ett femsiffrigt binärtal.



UUUUU byts mot **xx** i **Wave**-fönstret. Övriga variabler passar bäst som binärtal.

Skapa stimuli

Stimuli. Insignaler som klockpulser eller knapptryckningar, skapas med kommandot `force` i **Transcript**-fönstret.

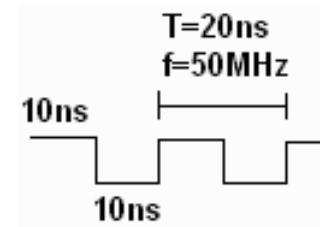
```
Transcript
VSIM 3> force codelock/clk 1 0ns, 0 10ns -repeat 20ns
VSIM 4> force codelock/k 000
VSIM 5> force codelock/r 0000
VSIM 6> run 100ns
```

Den förinställda tidsupplösningen i **Wave** är nanosekunder, **ns**. En lämplig klockfrekvens för ett kodlås kan däremot vara så låg som 5 Hz, dvs. en periodtid om 0,2 sek.

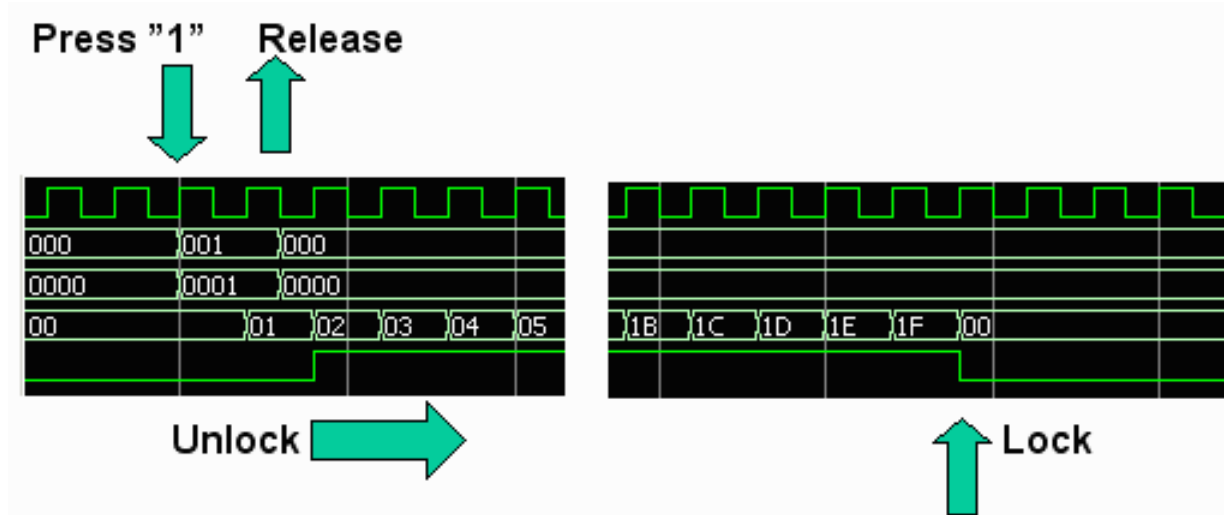
Vi skalar därför om till hög klockfrekvens med periodtiden 20 ns

`force codelock/clk 1 0ns, 0 10ns -repeat 20ns`

Genererar klockpulser för evigt.



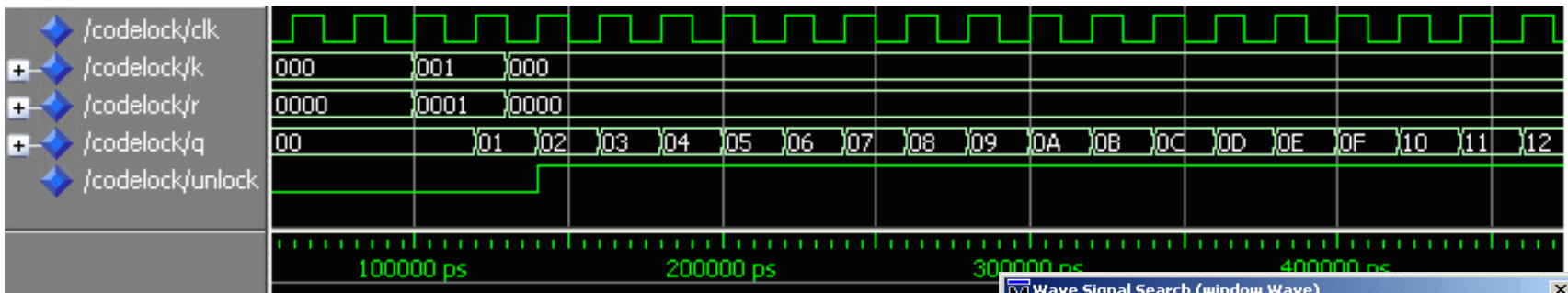
Simulera knapptryckningen



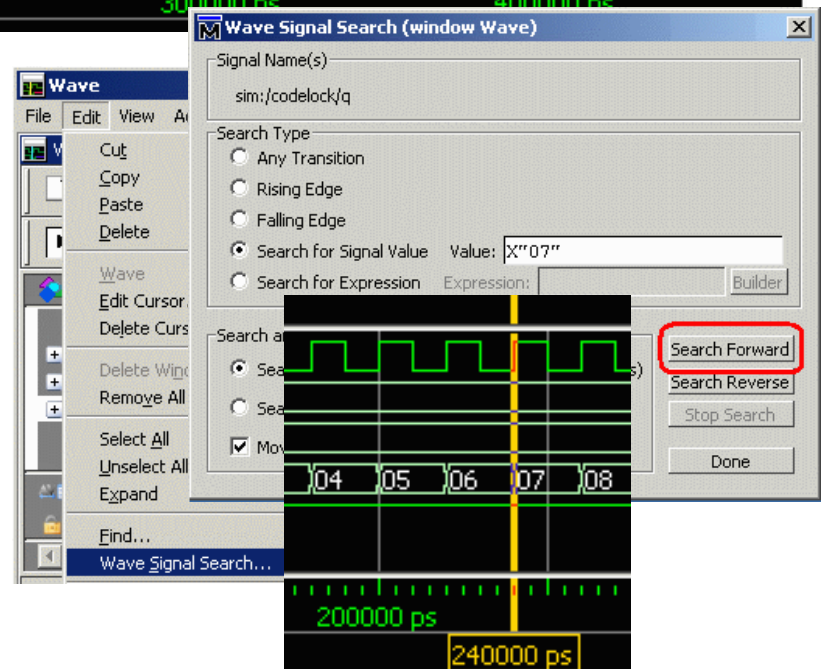
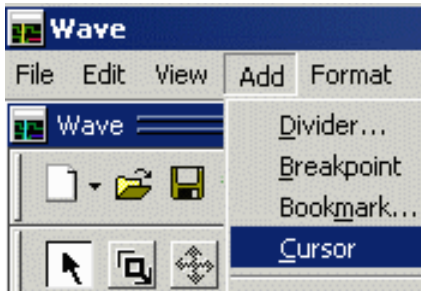
```
force codelock/k 000
force codelock/r 0000
run 100ns
force codelock/k 001
force codelock/r 0001
run 30ns
```

```
force codelock/k 000
force codelock/r 0000
run 800ns
```

Hitta i Wave-fönstret



*Lägg till en Cursor.
Sök efter "Signal Value".*



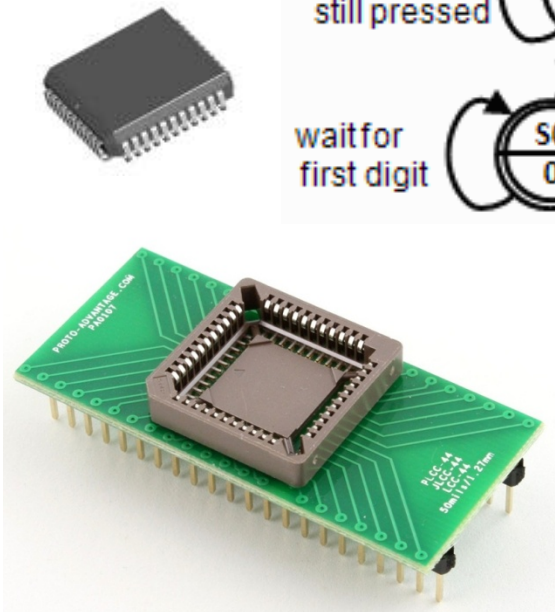
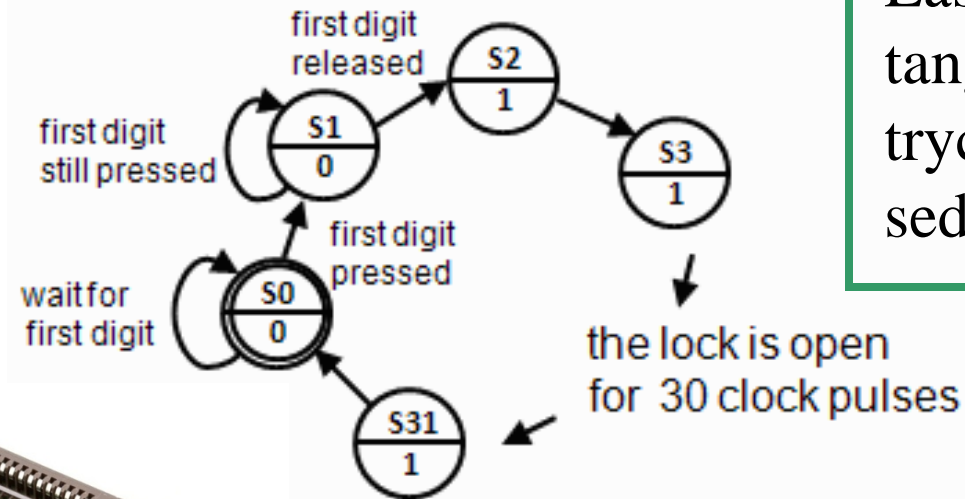
Öva hemma inför laborationen.

VHDL testbänk

Mall-programmets funktion

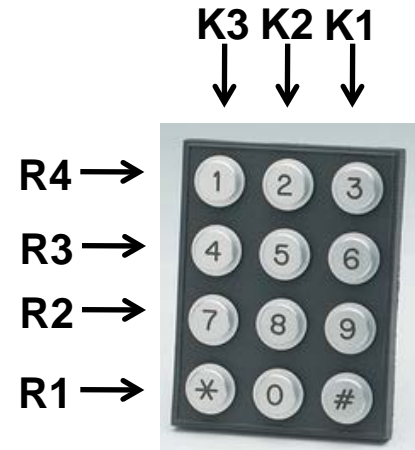
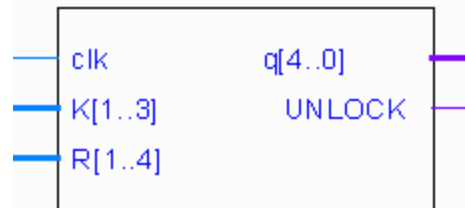


Låset öppnas när tangenten "1" trycks ned och sedan släpps.



Keypad och Statecounter

*Bra val av
datatyper gör
koden själv-
förklarande!*



```
K: in std_logic_vector(1 to 3);
```

```
R: in std_logic_vector(1 to 4);
```

1 2 3
K = "001" bitvector

K(3) = '1' bit

1 2 3 4
R = "0001" bitvector

R(4) = '1' bit

4 3 2 1 0
Statecounter: q = "00001" bitvektor
q(0) = '1' bit

lockmall.vhd

This code is given



```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity codelock is
    port( clk:      in  std_logic;
          K:        in  std_logic_vector(1 to 3);
          R:        in  std_logic_vector(1 to 4);
          q:        out std_logic_vector(4 downto 0);
          UNLOCK: out std_logic );
end codelock;
```

```
architecture behavior of codelock is
    subtype state_type is integer range 0 to 31;
    signal state, nextstate: state_type;
```

```
begin
    nextstate_decoder: -- next state decoding part
    process(state, K, R)
    begin
```

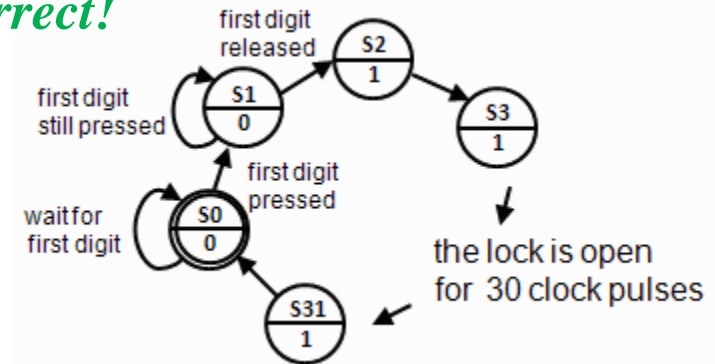
```
        case state is
            when 0 => if (K = "100" and R = "0001") then nextstate <= 1;
                       else nextstate <= 0;
                       end if;
            when 1 => if (K = "100" and R = "0001") then nextstate <= 1;
                       elsif (K = "000" and R = "0000") then nextstate <= 2;
                       else nextstate <= 0;
                       end if;
            when 2 to 30 => nextstate <= state + 1;
            when 31 => nextstate <= 0;
        end case;
    end process;
```

```
    debug_output: -- display the state
    q <= conv_std_logic_vector(state,5);
```

```
output_decoder: -- output decoder part
process(state)
begin
    case state is
        when 0 to 1 => UNLOCK <= '0';
        when 2 to 31 => UNLOCK <= '1';
    end case;
end process;
```

```
state_register: -- the state register part (the flipflops)
process(clk)
begin
    if rising_edge(clk) then
        state <= nextstate;
    end if;
end process;
end behavior;
```

It's easy to see that this is correct!



lockmall_with_error.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity codelock is
    port( clk:        in  std_logic;
          K:          in  std_logic_vector(1 to 3);
          R:          in  std_logic_vector(1 to 4);
          q:          out std_logic_vector(4 downto 0);
          UNLOCK: out std_logic );
end codelock;

architecture behavior of codelock is
    subtype state_type is integer range 0 to 31;
    signal state, nextstate: state_type;

begin
    nextstate_decoder: -- next state decoding part
    begin
        nextstate_decoder: -- next state decoding part
        process(state, K, R)
        begin
            case state is
                when 0 => if(((R(2)='0') and (R(3)='0') and (K(2)='0') and (K(3)='1')) and
                    ( not (( not ((K(1)='0') and (R(1)='0') and (R(4)='1')) and
                    ( not ((K(1)='1') and (R(1)='1') and (R(4)='0'))))))))
                    then nextstate <= 1;
                    else nextstate <= 0;
                    end if;
                when 1 => if(((R(2)='0') and (R(3)='0') and (K(2)='0') and (K(3)='1')) and
                    ( not (( not ((K(1)='0') and (R(1)='0') and (R(4)='1')) and
                    ( not ((K(1)='1') and (R(1)='1') and (R(4)='0'))))))))
                    then nextstate <= 1;
                    elsif (K = "000" and R = "0000") then nextstate <= 2;
                    else nextstate <= 0;
                    end if;
                when 2 to 30 => nextstate <= state + 1;
                when 31      => nextstate <= 0;
            end case;
        end process;

        debug_output: -- display the state
        q <= conv_std_logic_vector(state,5);

        output_decoder: -- output decoder part
        process(state)
        begin
            case state is
                when 0 to 1  => UNLOCK <= '0';
                when 2 to 31 => UNLOCK <= '1';
            end case;
        end process;

        state_register: -- the state register part (the flipflops)
        process(clk)
        begin
            if rising_edge(clk) then
                state <= nextstate;
            end if;
        end process;
    end behavior;
end codelock;
```

Now it's hard to see if this is correct or not?

lockmall_with_error.vhd

Betyder båda uttrycken samma sak?

```
( K = "100" and R = "0001" )
```

Är verkligen detta samma sak?

```
(( (R(2)='0') and (R(3)='0') and (K(2)='0') and (K(3)='1')) and  
( not (( not ((K(1)='0') and (R(1)='0') and (R(4)='1')) and  
( not ((K(1)='1') and (R(1)='1') and (R(4)='0'))))))
```

Någon "lovar" att koden är korrekt – men hur kan man veta att detta är absolut sant?

Testbench

thank's to: *Francesco Robino*

tb_lockmall.vhd

`tb_lockmall.vhd`

Vi behöver skriva en VHDL-testbench

Ett testbänksprogram kan testa alla möjliga tangentkombinationer och rapportera om det uppstår något problem ...

Det kan automatiskt loopa igenom all möjliga tangenttryckningar och rapportera om om låset försöker att öppna.

Det finns $2^7 = 128$ möjliga tangentkombinationer och vi skulle bli helt uttröttade om vi försökte att prova dem alla för hand.

entity – en testbänk har inga portar

```
entity tb_codelock is  
  -- entity tb_codelock has no ports  
  -- because it's for simulation only  
end tb_codelock;
```

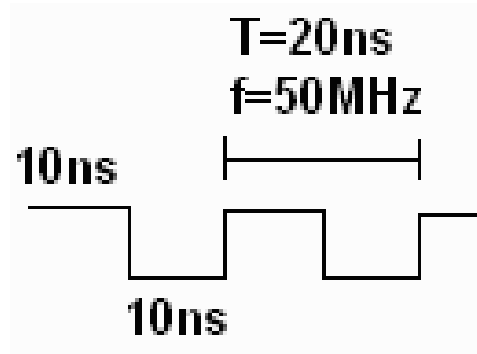
Några interna signaler behövs

```
signal          clk : std_logic := '0';
signal          K_test : std_logic_vector(1 to 3);
signal          R_test : std_logic_vector(1 to 4);
signal prev_K_test : std_logic_vector(1 to 3);
signal prev_R_test : std_logic_vector(1 to 4);
signal          q : std_logic_vector(4 downto 0);
signal          unlock : std_logic;
```

Vårt codelock används som component

```
-- we use our codelock as a component
component codelock
  port( clk : in std_logic;
        K : in std_logic_vector(1 to 3);
        R : in std_logic_vector(1 to 4);
        q : out std_logic_vector(4 downto 0);
        UNLOCK : out std_logic );
end component;
```

Genera en simuleringsklocka

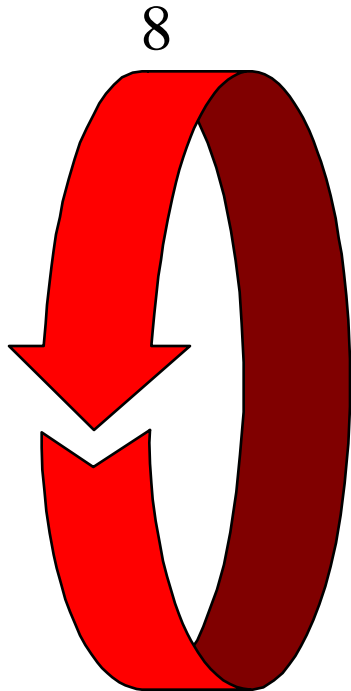


```
-- generate a simulation clock  
clk <= not clk after 10 ns;
```

Instantiatiering och signal mapping

```
-- instantiation of the device under test,  
-- mapping of signals  
inst_codelock:  
  codelock  
  port map (  
      clk => clk,  
      K  => K_test,  
      R  => R_test,  
      q  => q,  
      UNLOCK => unlock );
```

En nästlad slinga skapar tangentryckningarna



```
process
begin
  for k in 0 to 7 loop
    K_test <= conv_std_logic_vector(k,3);
16  for r in 0 to 15 loop
    prev_K_test <= K_test;
    prev_R_test <= R_test;
    R_test <= conv_std_logic_vector(r,4);
    wait until CLK='1';
  end loop;
end loop;
end process;
```

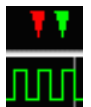
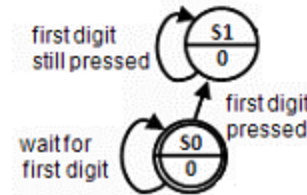
8·16=128 turns

report, severity **note**, severity **error**

Tests if state $q = "00001"$ will be reached by any combination.

check:

```
process(q)
begin if ((q = "00001") and
          (prev_K_test = conv_std_logic_vector(1,3)) and
          (prev_R_test = conv_std_logic_vector(1,4)))
  then assert false report
    "Lock tries to open for the right sequence!"
    severity note;
  else if ((q = "00001"))
  then
    assert false report
    "Lock tries to open with the wrong sequence!"
    severity error;
  else report "Lock closed!" severity note;
    end if;
  end if;
end process check;
```



Simulera och hitta felet!

Vad annat än att trycka på ”1” tangenten skulle kunna öppna låset?

?



