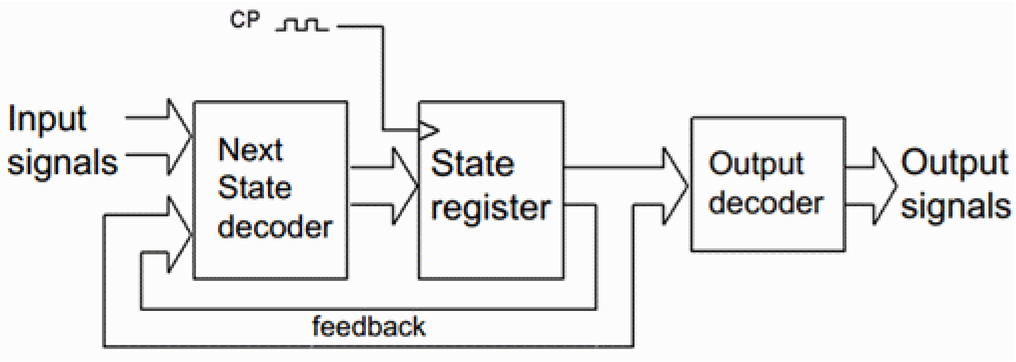
Laboratory VHDL introduction

Digital Design IE1204 (Note! *not* included for IE1205)

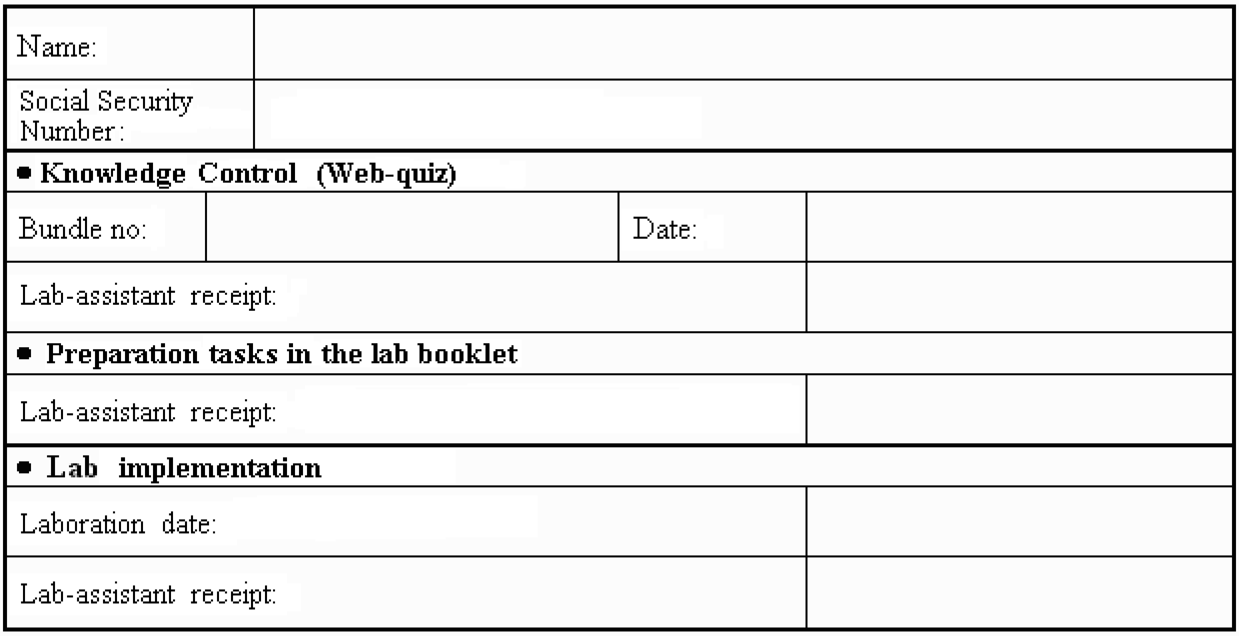


Attention! To access the laboratory experiment you must have:

- booked a lab time in the reservation system (Daisy).
- completed your personal knowledge control on the Web (Web-quiz).
- done all preparation tasks mentioned in the lab booklet.

During the lab you work in groups of two, but both students are responsible individually for their preparation and implementation.
Booth students should bring their lab booklets. This frontpage is used as your **receipt** that the lab is completed. Save the receipt until you have received the full course registered in the database (Ladok).

*Since this is your receipt you **must** fill in the table with ink.*



Name:			
Social Security Number:			
● **Knowledge Control (Web-quiz)**			
Bundle no:		Date:	
Lab-assistant receipt:			
● **Preparation tasks in the lab booklet**			
Lab-assistant receipt:			
● **Lab implementation**			
Laboration date:			
Lab-assistant receipt:			

Introduction

This lab is about how to design digital logic with VHDL language and modern CAD software. The idea is that you'll get a glimpse of how a "Digital" engineer work. VHDL language is a very complex programming language, and it is not reasonable to "learn" that this brief first Digital Design Course.
When you solve the lab assignments, you have therefore been given tutorials and template code on the course web.

The school has several good VHDL courses that can be chosen by those who want to know more, and who want to work with Digital Design in the future.

The best way to get to know a program is to install it on your own computer. Then you can in peace searche through menus and help pages, and can take the time it takes to sort out what you might have misunderstood.

If "computer hassles" threaten to consume too much time for you, you can also find the programs installed in the school's Windows computer rooms - as a backup solution.

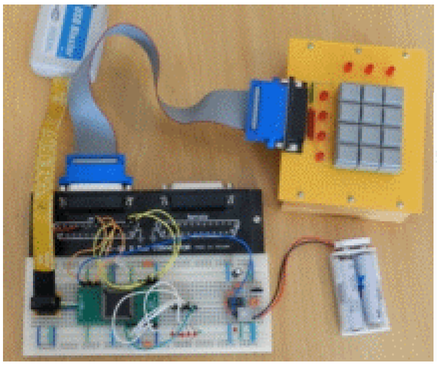
The goal of the lab

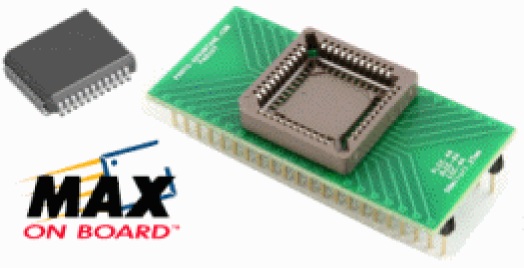
- Become familiar with modern CAD software, Quartus and ModelSim.
- Show how to simulate a digital design (a Moore-machine), how to generate input signals, stimuli, and how to observe the outputs and behavior (ModelSim-Wave).
- Orienting yourself on how a digital technician can write a VHDL "test bench" to ensure that the construction is completely correct.
- Practice the VHDL-construction of a state machine from a given state diagram (revise and expand a given template program).
- Practice how to tie together the design "signals" with target chip "pins".
- Show how you program the target chip (MAX3000), and trying the operation in reality.

Attention! Your lab time may be prior all course elements that may be needed for the lab has been lectured. You would then have to read the course material for yourself in advance - there are links to all slides for the lectures and exercises.

Attention! Lab equipment is completed. No wires should be changed, not added or removed.

A VHDL-code lock





Lab task is to construct a code lock that opens to a unique four-digit code, but we begin by studying a simpler template program, a code lock that opens to one key!

- **Preparation task 1 (done before the lab at home)**

Install the programs **Quartus II** and **ModelSim** on your own computer.

Follow the steps in the tutorial on the course web - Install the programs on your computer.

- **Laboratory task 1 (do at lab in school)**

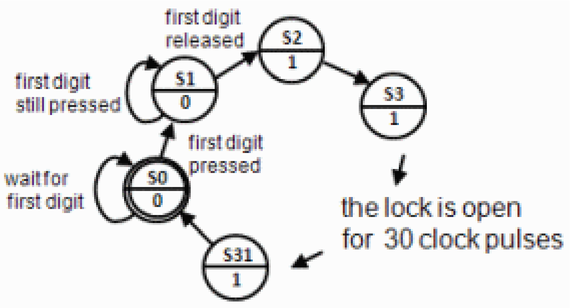
Log on to the lab computer. On the school centrally managed lab computers You do not have rights to install software. **Quartus II** and **ModelSim** are already installed. You may not access the folders under `C:\`. At the laboratory, you should therefore use your "server" folder `H:\`.

- Create a folder `H:\MAXWork\` for the files in this lab.

- Important operating system setup. Show file extensions should be set at all programming courses!
  Windows 7 show fileextensions

- **Preparation task 2 (done before the lab at home)**

Start Quartus and create a project `codelock`. Bring the content of the file lockmall.vhd as the project VHDL-file and then compile the code.
Follow the steps in the tutorial on the course web - VHDL-program with Quartus.



Read about the template program VHDL code in the description on the course web
- VHDL for a codelock.

Read on how to tie the signal names to specific pins of the target chip in Quartus.
- Pin-planning in Quartus.

Read about how to use the Quartus programming function with a JTAG USB Blaster.
- Chip-programing with Quartus.

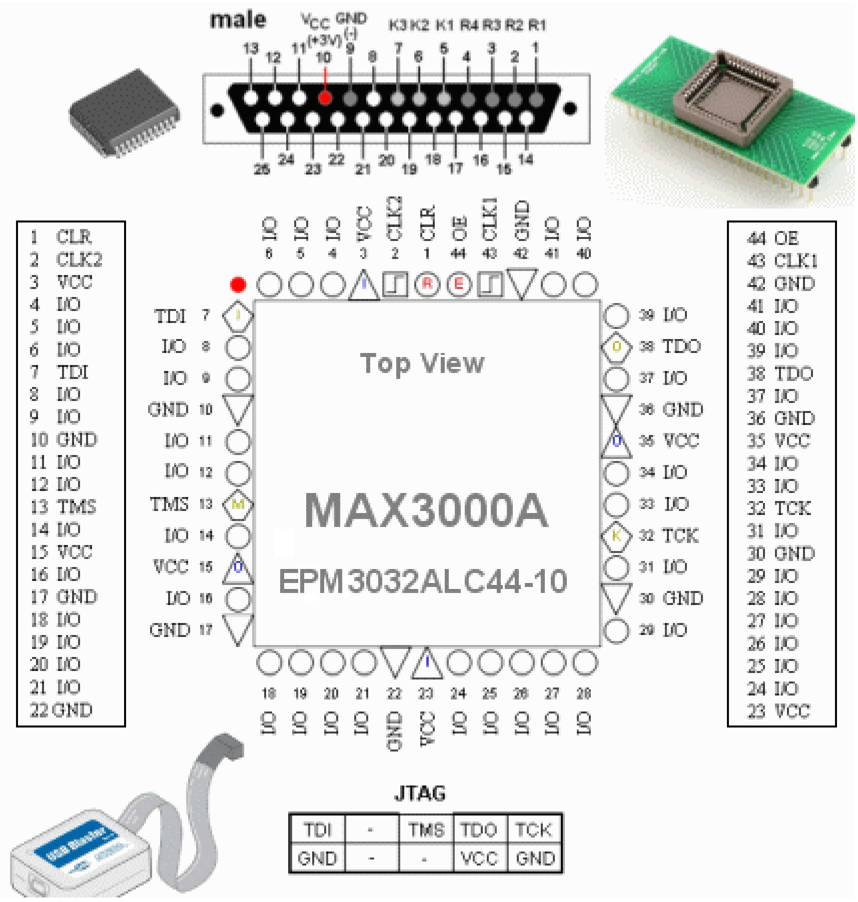
- **Laboratory task 2 (do at lab in school)**

- Start Quartus and create a project `codelock` in your server-folder `H:\MAXWork\`. Use the content of the file lockmall.vhd as the project VHDL-file and then compile the code.

- Lab equipments have different wiring! Examine your lab equipment and enter the Pin-planning table, and thereafter **Pin Planner** in Quartus.

Follow the steps in the tutorial on the course web - Pin-planning in Quartus

Node Name	Direction	Location
clk	Input	PIN_
K[1..3]	Input	
K[1]	Input	PIN_
K[2]	Input	PIN_
K[3]	Input	PIN_
q[4..0]	Output	
q[4]	Output	PIN_
q[3]	Output	PIN_
q[2]	Output	PIN_
q[1]	Output	PIN_
q[0]	Output	PIN_
R[1..4]	Input	
R[1]	Input	PIN_
R[2]	Input	PIN_
R[3]	Input	PIN_
R[4]	Input	PIN_
UNLOCK	Output	PIN_

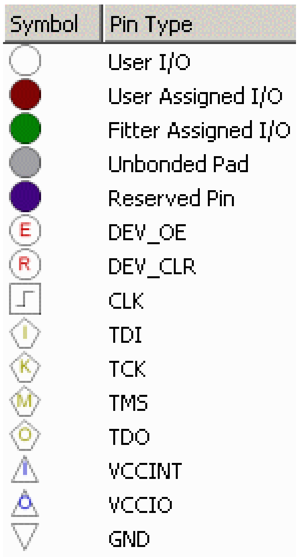


Description of the pin-symbols se figure.

- When you have completed the pin planner in Quartus so recompile the project.

- Program the device with the USB blaster.

- Check that the code lock opens when you press the "1" and then release the key.



Symbol	Pin Type
○	User I/O
🔴	User Assigned I/O
🟢	Fitter Assigned I/O
⚫	Unbonded Pad
🟣	Reserved Pin
Ⓔ	DEV_OE
Ⓡ	DEV_CLR
⎍	CLK
◇I	TDI
◇K	TCK
◇M	TMS
◇O	TDO
△I	VCCINT
△O	VCCIO
▽	GND

- **Preparation task 3 (done before the lab at home)**

At home, you have no hardware, no laboratory equipment. In such situations one usually simulate the code to see if it is correct.

The leading simulation software **ModelSim** is available in a version for Altera's chips. Start ModelSim and simulate the VHDL-code with the content from `lockmall.vhd` as VHDL-file.

Follow the steps in the tutorial on the course web - Simulate with ModelSim.

lockmall.vhd (lockmall.txt)

Laboratory task 3 (do at lab in school)

Even when having access to the hardware, it is common to mix simulations with hardware test.

Carry out the same simulation in school that you practiced at home, ie show in the **wave window** that the lock opens for "1". Show Your lab assistant your "simulation expertise."

Preparation task 3 (done before the lab at home)

Design of Digital hardware can often result in the production of an ASIC - An application specific Integrerated Circuit. It is then often several months of lead time and, manufacturing costs of the order of several million dollars.

Then you have to be sure that the design is absolutely correct!

(At the lab, we have a better starting point than the ASIC designer. If your design on a programmable CPLD chip is wrong, You get the chance to reprogram it - again and again.)

As you can see, it is the test engineer who is Digital Technology Hero!

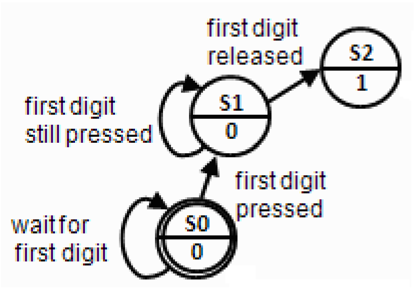
VHDL-language has various tools to enhance the ability to be able to write correct code.

- To reduce the risk of errors when transferring information from the data sheets, you can use index that runs up or down, to suit the method that was used in the data sheet.
- You has also the ability to create user-defined data types that fit the description of the construction. One can therefore often write VHDL code that is "obviously" correct!
- One can write a VHDL test bench. This is simulation code that can be used to test many/all signal combinations that the circuit may be exposed to.

Attention! A test bench is usually a more complicated program than the original design as it relates to test!

These lines from the template program is an example of code that *obviously* follow the given state diagram.

```
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;
                . . .
```



However, if one takes over the code from someone else, even if that person *promise* that it works, the situation is different.

```
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;
                . . .
```

Here are the conditions written in such a way that it is no longer obvious what the code does - And therefore we do not know if it is correct or if, despite all the promises, it is incorrect?

Try now the code correctness with a (pre-written) test bench

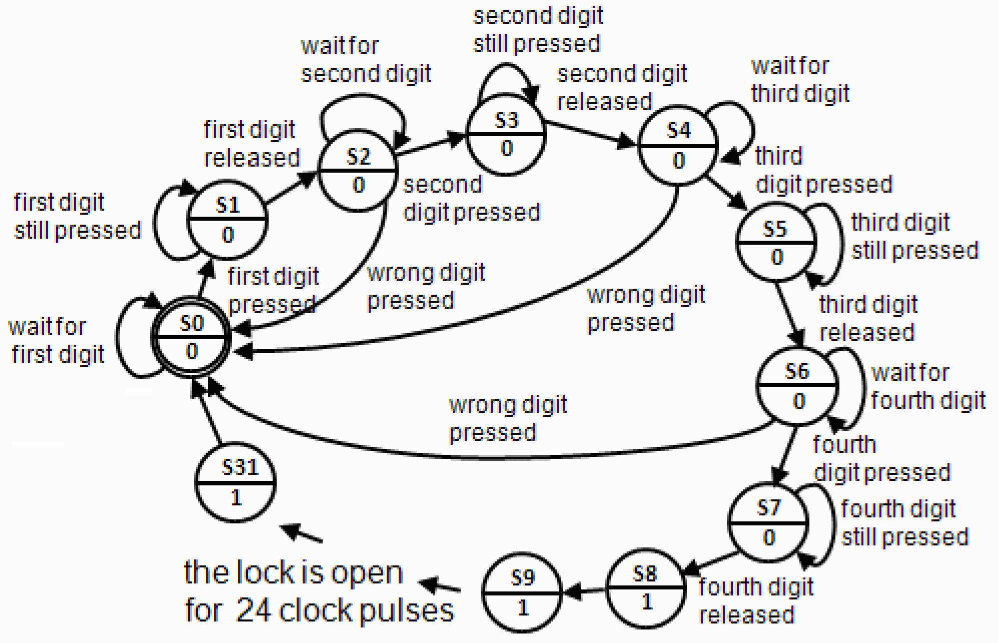
Follow the steps in the tutorial on the course web - Testbench in ModelSim.

lockmall.vhd (lockmall.txt)
lockmall_with_error.vhd (lockmall_with_error.txt)
tb_lockmall.vhd (tb_lockmall.txt)

- **Laboratory task 4 (do at lab in school)**

- Perform also the simulation of the test bench in the school, and show lab assistant your skills by simulating code with a test bench.
- Can you reveal anything wrong with the code? Show lab assistant.
- Close **ModelSim** and change to the program **Quartus II**. There you change the contents of the VHDL file from `lockmall.vhd` to `lockmall_with_error.vhd`. Compile and download the code to the MAX-chip.
- Check if the suspicious behavior from the simulation means that in practice You can open the code lock incorrectly?

- **Preparation task 5 (done before the lab at home)**



A code lock that opens to a single key press is of course ridiculously easy to force, normally you have a four digit code. Your task will be to develop the template program (`lockmall.vhd`) to such a lock with four digits - the last four digits of your civic number.

Prepare such a VHDL program at home. Check that it is possible to compile. Take with you the code to the school in any way, such as:

- Mail the text to yourself.
- Bring a USB flash drive with the code as a text file.
- Transfer the code to your server folder `H:\`.

Are you able to simulate the code at home, you'll increase the likelihood that you have the correct code from the start at school.
The time at the lab school will *not* be enough to write the program code from "scratch"!

- **Laboratory task 5 (do at lab in school)**

Make a code lock that opens to a four-digit code. Lab Assistant determines the digit combination so that - Two of the numbers taken from your preparation program, and the other two from your lab colleague preparation program.

- Show the working code lock for lab assistant.

Do you have time for more?

If you are well prepared for the lab, and if you are not suffering from intermittent connections or dead batteries, then you probably now have time for a "voluntary" task.

- Can you change the program so that it ***also opens*** for the previous *hidden* key combination?
  (so that there is a "hardware trojan" within the chip)

Good Luck!

When you are finished clean the lab desk.

Bill of materials

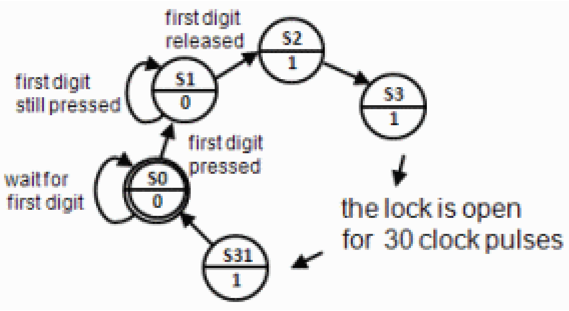
The "bill of material" for the lab equipment, could be helpful if you ever would need to use simple MAX-chips yourself.

```
Altera USB-blaster ELFA 73-898-90
ALTERA CPLD EPM3032ALC44-10
Proto Advantage PLCC-44 Socket to DIP-44 Adapter
Breadboard MB-85 ELFA 48-428-37
Contact breadboard to 25-pol D-sub ELFA 48-426-96
Pin contact (JTAG) ELFA 43-155-03
Lightdiode with series resistor 5V red  ELFA 75-012-59
Lightdiode with series resistor 5V yellow  ELFA 75-015-11
Electronic chip 555 ELFA 73-042-65
Trimming Potentiometer 500 kO with adjustment knob ELFA 64-635-25
```

William Sandqvist william@kth.se

VHDL for a code lock

Description of the code lock template

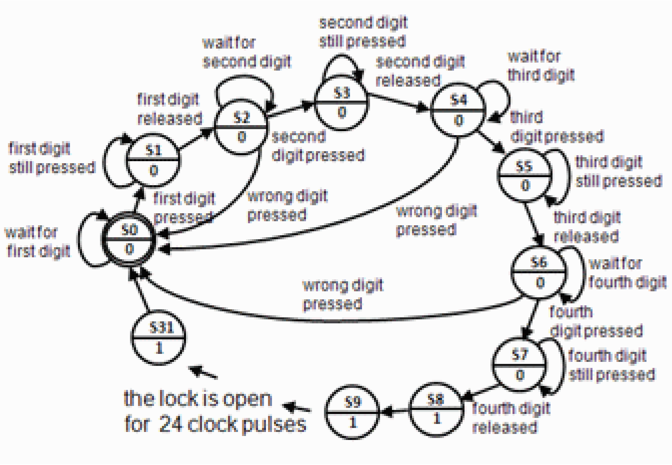


The Code Lock template applies to a simplified lock that opens when you press the key "1" and then release the key.

Almost all digital designs are now carried out using high-level languages like VHDL/Verilog. Our basic course in digital technology does not allow to teach VHDL language, however, you will be able to transform the "template code lock" into useful VHDL code at the lab.

If you think that the VHDL language seems interesting, then the school has several advanced digital technology courses

At lab you expand this code to create a four digit code lock!

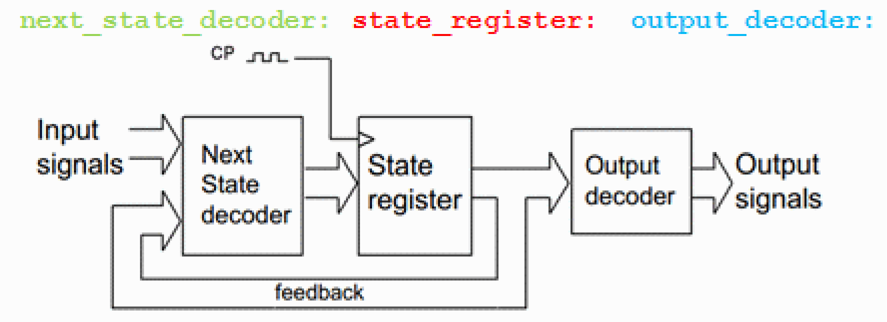


lockmall.vhd

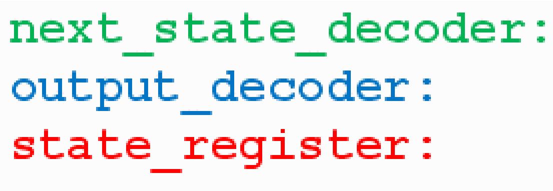
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

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
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;

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;
```

Moore machine

The code lock is The code lock is designed as a Moore machine.



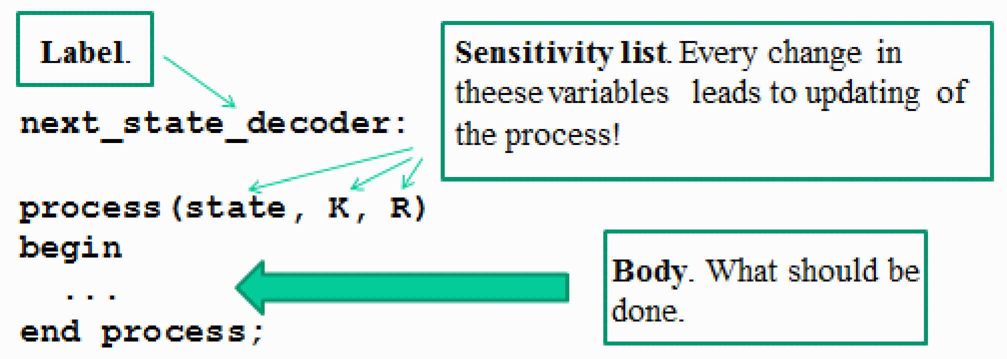
The different blocks in the code are identified with "labels".



```
next_state_decoder:
output_decoder:
state_register:
```

VHDL processes

With a "process" you can describe what should be performed in a block without having to go into the details of how this should be done.



Label.

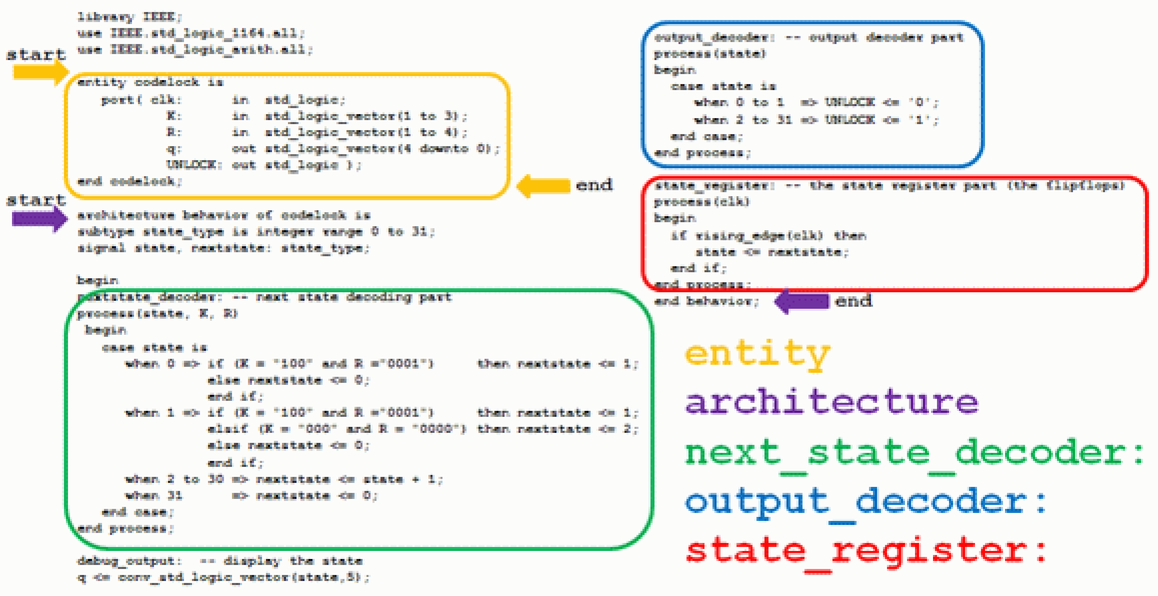
```
next_state_decoder:

process(state, K, R)
begin
   ...
end process;
```

Sensitivity list. Every change in theese variables leads to updating of the process!

Body. What should be done.

VHDL code is written as such processes.

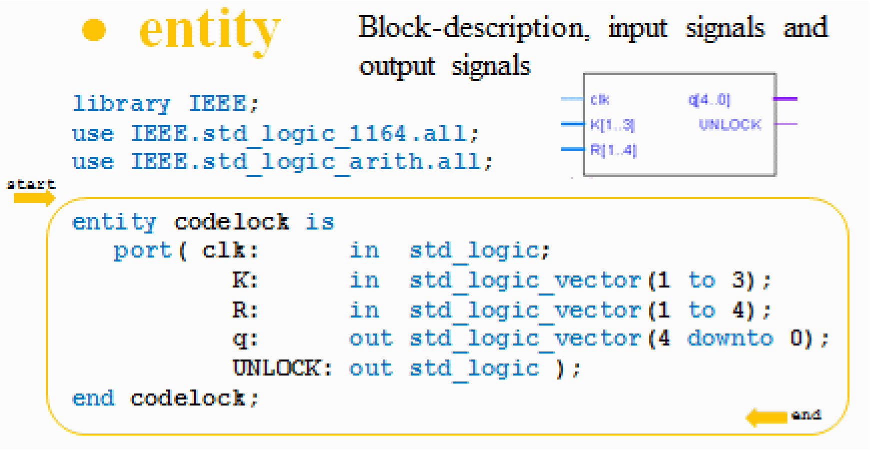
Parts of the program



```
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;
```

```
entity
architecture
      next_state_decoder:
      output_decoder:
      state_registers:
```

entity

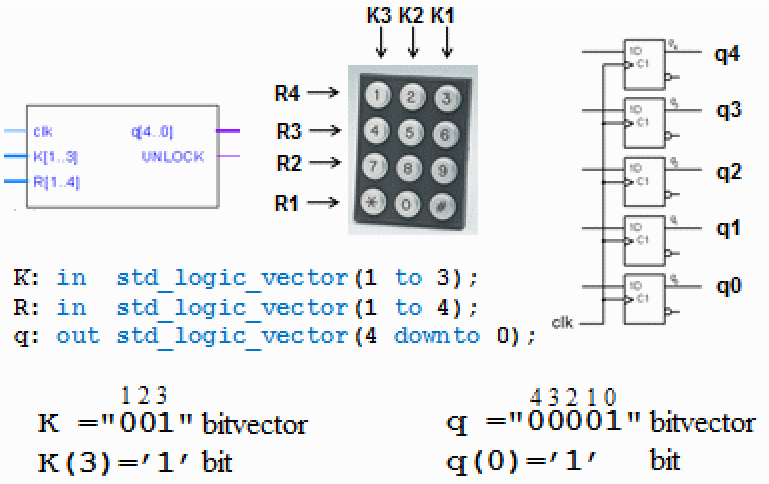


● entity Block-description, input signals and output signals

```
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;
```

The program `entity` is a description of the code lock as a "black box" with input and output signals.

Bits and Bitvectors



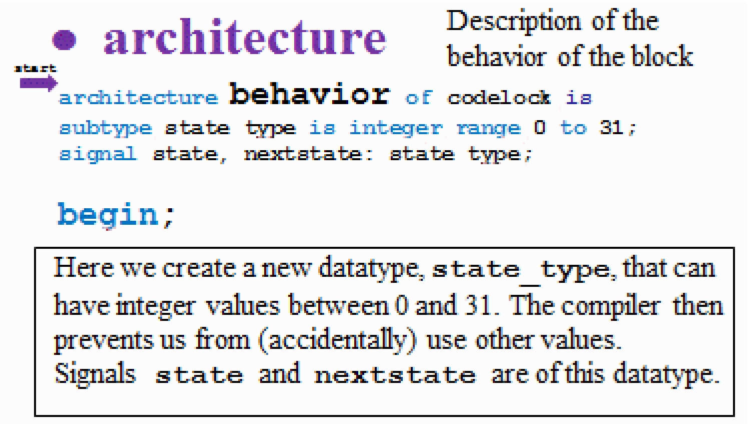
```
K: in  std_logic_vector(1 to 3);
R: in  std_logic_vector(1 to 4);
q: out std_logic_vector(4 downto 0);
```

```
      1 2 3                      4 3 2 1 0
K  ="001" bitvector          q  ="00001" bitvector
K(3)='1' bit                 q(0)='1'    bit
```

You can customize the indexing of variables so that it is consistent with the data sheets - less risk of mistakes!

Architecture - define suitable datatypes

The program part `architecture` contains the description of the behavior of the block.



- architecture

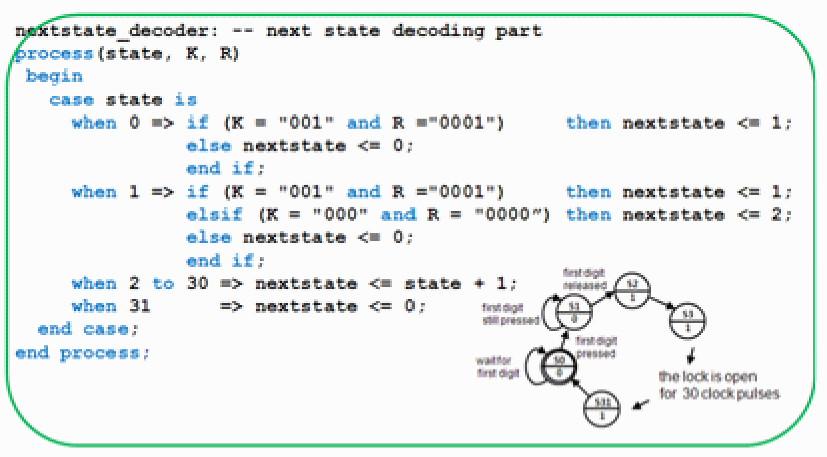
Description of the behavior of the block

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

begin;
```

Here we create a new datatype, `state_type`, that can have integer values between 0 and 31. The compiler then prevents us from (accidentally) use other values. Signals `state` and `nextstate` are of this datatype.

Here we create a new datatype, `state_type`, that can have integer values between 0 and 31. The compiler then prevents us from (accidentally) use other values. Signals state and nextstate are of this datatype.

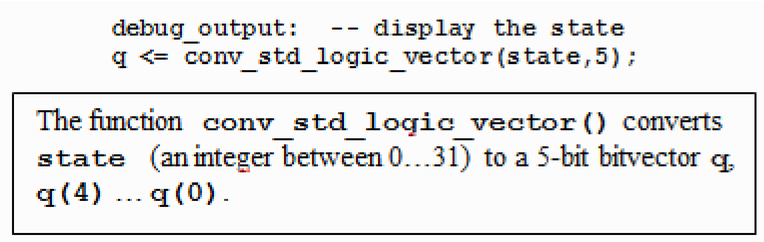
Architecture - next state decoder



```
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;
```

Next state decoder is the central part of the program. By using the `case`-statement you can write the code in such a way that it confirms to the state diagram.

Troubleshooting help - the state is shown with five LEDs

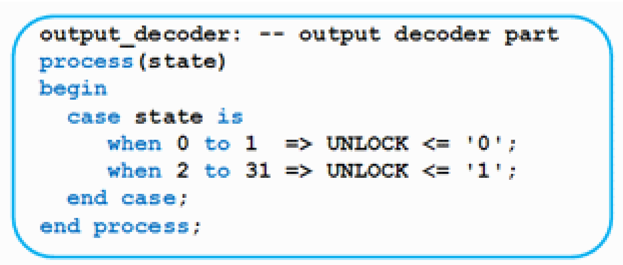


```
debug_output:  -- display the state
q <= conv_std_logic_vector(state,5);
```

The function `conv_std_logic_vector()` converts `state` (an integer between 0…31) to a 5-bit bitvector q, `q(4)` … `q(0)`.

To troubleshoot, we want to be able to follow which state the machine is in. The function `conv_std_logic_vector()` converts ,`state` (an integer between 0...31) to to a 5-bit bitvector `q`, `q(4)` ... `q(0)`. In order to use this conversion function one has to include the library `IEEE.std_logic_arith.all`.

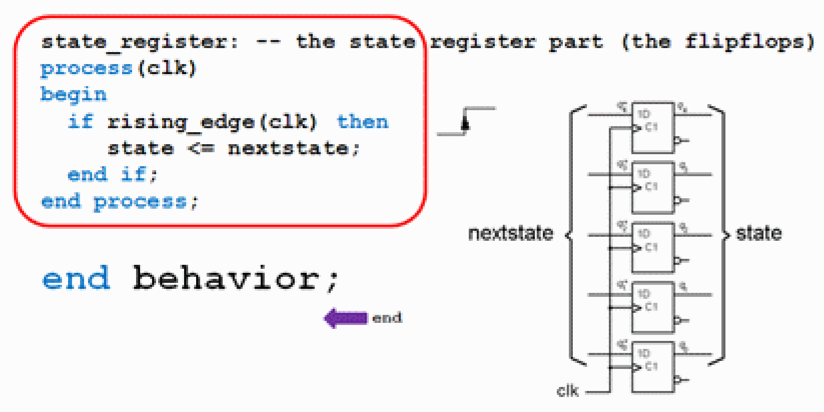
Output decoder



```
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;
```

The output decoder is written "straightforward" by a `case`-statement.

State register



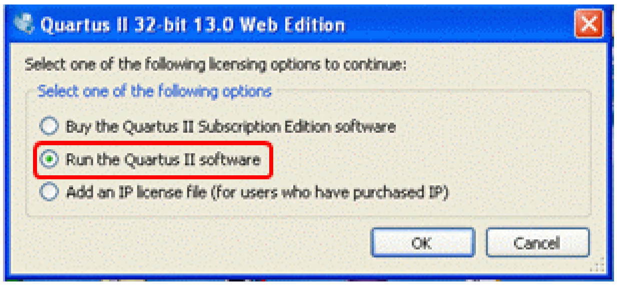
```
state_register: -- the state register part (the flipflops)
process(clk)
begin
  if rising_edge(clk) then
     state <= nextstate;
  end if;
end process;

end behavior;
```

By using the function `rising_edge(clk)` we let the compiler "understand" that we want to use the flip-flops inside the MAX-chip in order to build a register.

VHDL-program with Quartus

Choose the right program version from the school's start menu :

```
Altera 13.0.1.232 Web edition\
    Quartus II Web Edition 13.0.1.232\
         Quartus II 13.0sp1 (32bit)
```


Quartus II 32-bit 13.0 Web Edition

Select one of the following licensing options to continue:

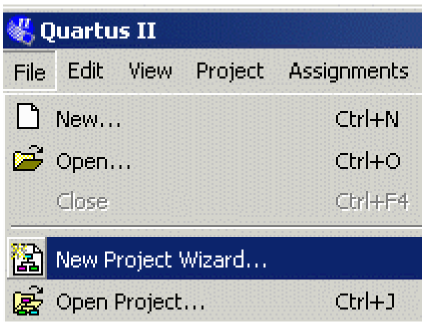
Select one of the following options

○ Buy the Quartus II Subscription Edition software

● Run the Quartus II software

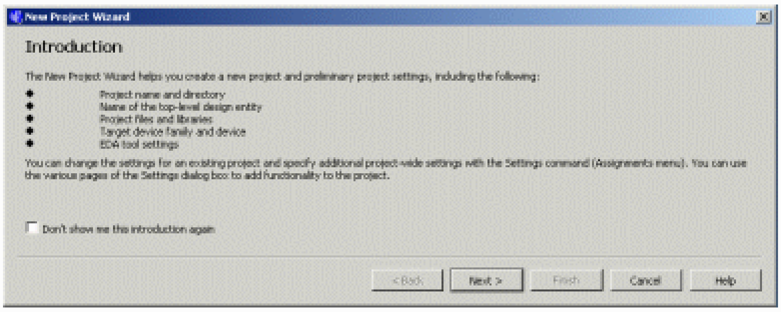
○ Add an IP license file (for users who have purchased IP)

Start Quartus. You need no license and you do not need not buy anything.
If not be directly offered to start **New Project Wizard**, You may also select this option from the **File** menu.



Introduction

Clic on **Next**.


New Project Wizard

Introduction

The New Project Wizard helps you create a new project and preliminary project settings, including the following:

- Project name and directory
- Name of the top-level design entity
- Project files and libraries
- Target device family and device
- EDA tool settings

You can change the settings for an existing project and specify additional project-wide settings with the Settings command (Assignments menu). You can use the various pages of the Settings dialog box to add functionality to the project.

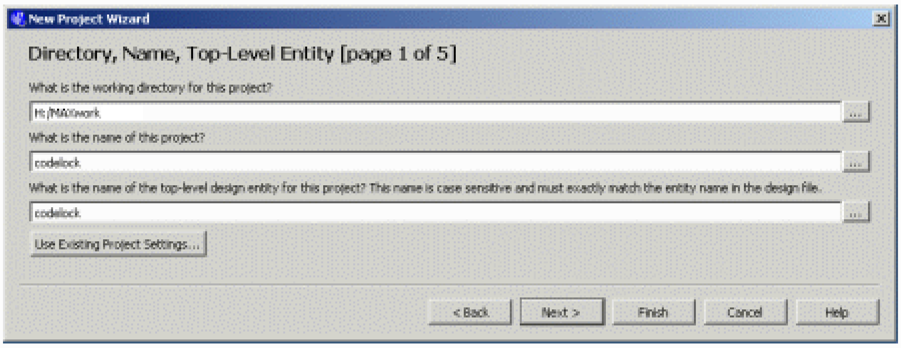
□ Don't show me this introduction again

Project Name and Directory

In school, the entire project must be on your `H:\`, eg. `H:\MAXwork` (at home on `C:\`, eg. `C:\MAXwork`)

Name: `codelock`
Top-Level Entity: `codelock`
(Note! the name `codelock` must "match" the name you later on specify as entity in your VHDL-file)

Proceed with **Next**.



New Project Wizard

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

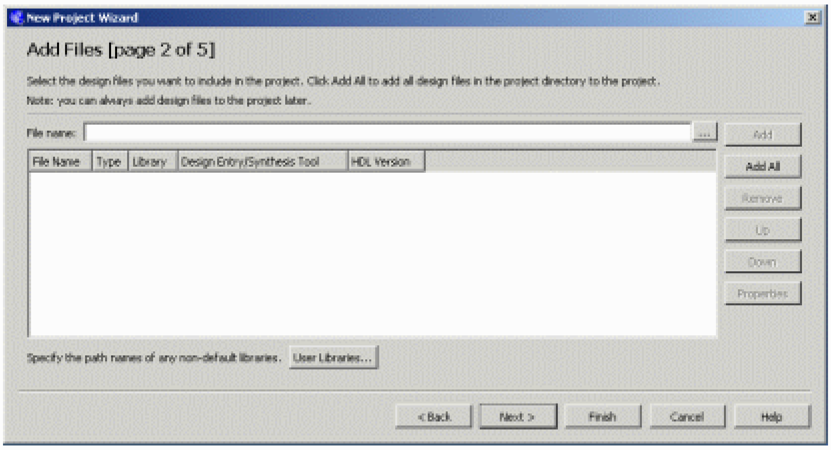
What is the working directory for this project?
H:/MAXwork

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

Add files

We have no files to add to the project, so we proceed with **Next**.



New Project Wizard

Add Files [page 2 of 5]

Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.
Note: you can always add design files to the project later.

File name:

File Name	Type	Library	Design Entry/Synthesis Tool	HDL Version

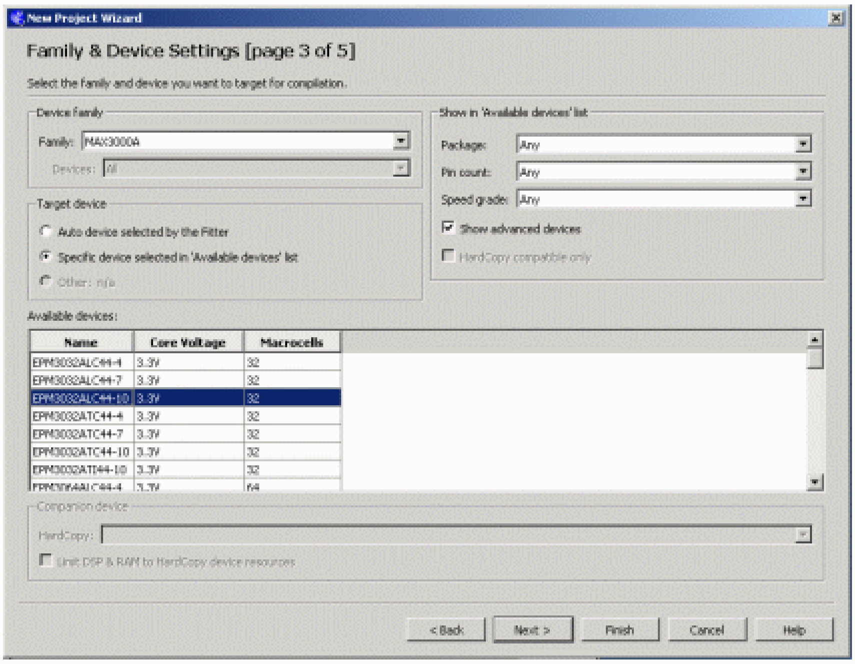
Specify the path names of any non-default libraries.

Family and Device Settings

Here we specify which chip we intend to use during the lab.

Family: `MAX3000A` Available devices: `EPM3032ALC44-10`

Proceed with **Next**.



Family & Device Settings [page 3 of 5]

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

Device family

Family: MAX3000A

Devices: All

Target device

○ Auto device selected by the Fitter

● Specific device selected in 'Available devices' list

○ Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: Any

Speed grade: Any

☑ 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
EPM3064ALC44-4	3.3V	64

Companion device

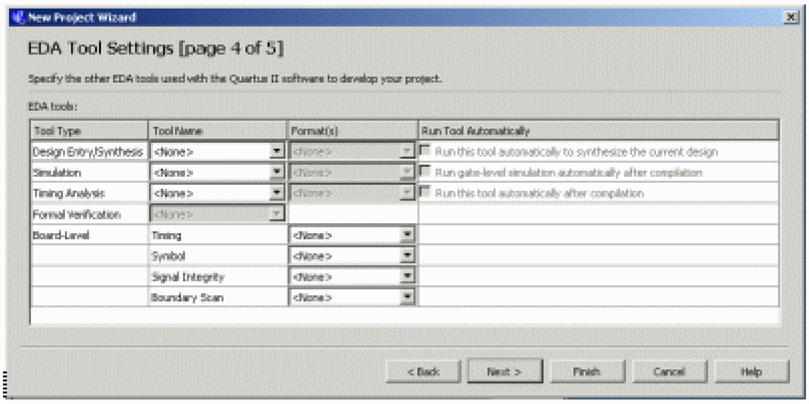
HardCopy:

□ Limit DSP & RAM to HardCopy device resources

EDA tools setting

Here you can create context with software tools from other vendors. We will simulate with the ModelSim program but that we need not enter.

Proceed with **Next**.



EDA Tool Settings [page 4 of 5]

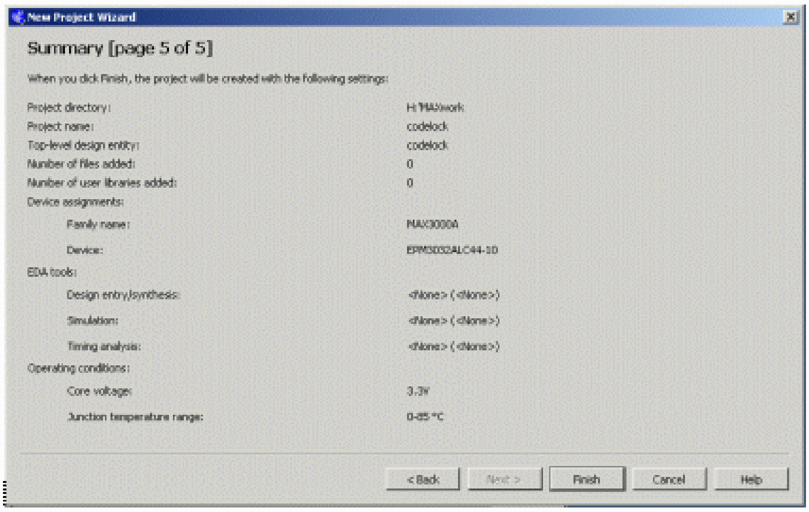
Specify the other EDA tools used with the Quartus II software to develop your project.

EDA tools:

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

Summary.

Here you can see a summary of your selections, exit the "Wizard" with **Finish**.

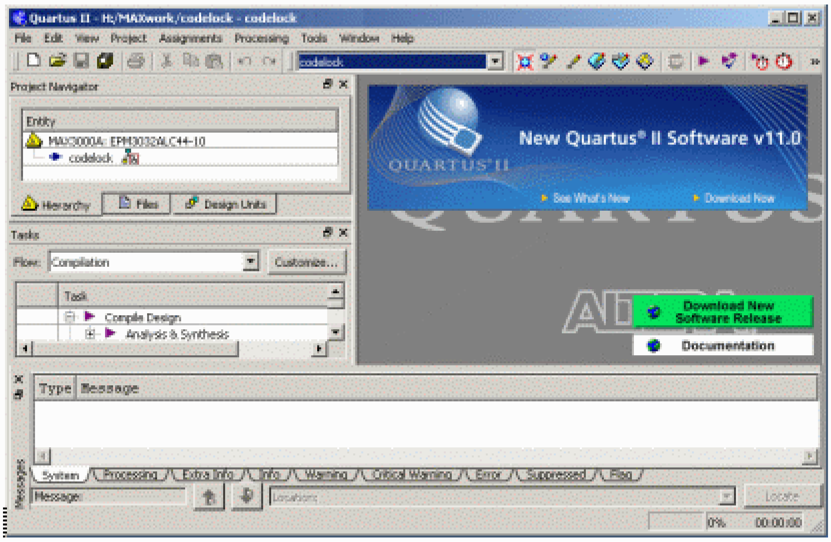


Summary [page 5 of 5]

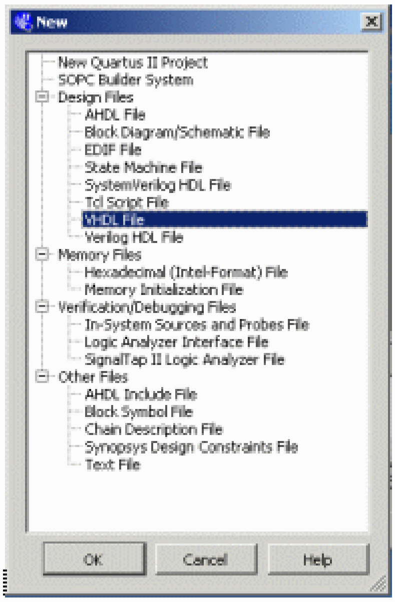
When you click Finish, the project will be created with the following settings:

Project directory:	H:\MAXwork
Project name:	codelock
Top-level design entity:	codelock
Number of files added:	0
Number of user libraries added:	0
Device assignments:	
Family name:	MAX3000A
Device:	EPM3032ALC44-10
EDA tools:	
Design entry/synthesis:	<None> (<None>)
Simulation:	<None> (<None>)
Timing analysis:	<None> (<None>)
Operating conditions:	
Core voltage:	3.3V
Junction temperature range:	0-85 °C

The project has been created



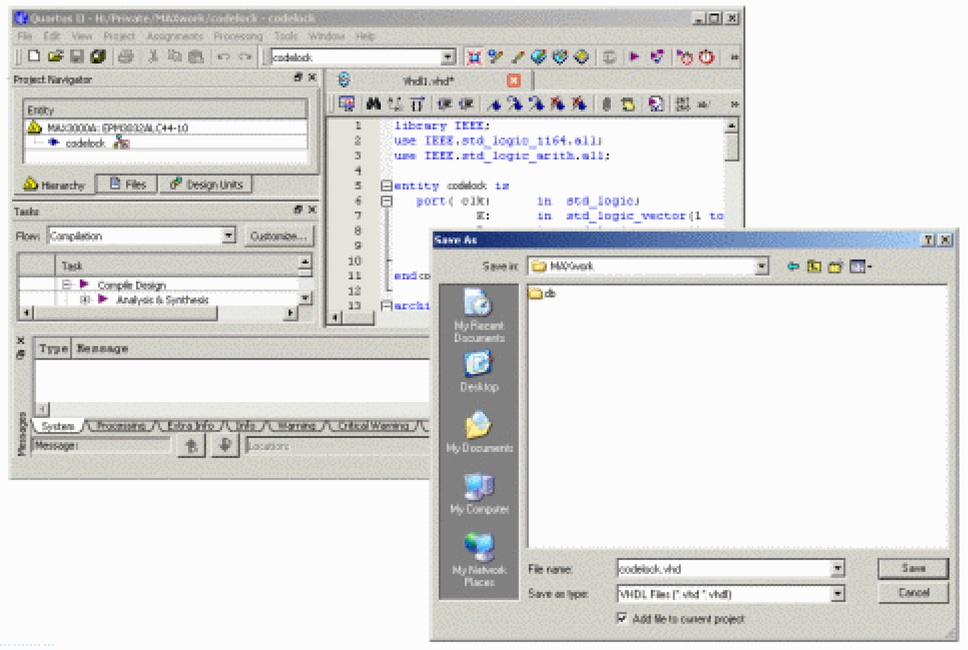
VHDL-code



Create a blank file for VHDL-code. **File**, **New**, **VHDL File**.

Copy the Template `lockmall.vhd` and paste it in Quartus text editor.

```
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 = "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;

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;
```



```
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

end co

archi
```

File name: codelock.vhd

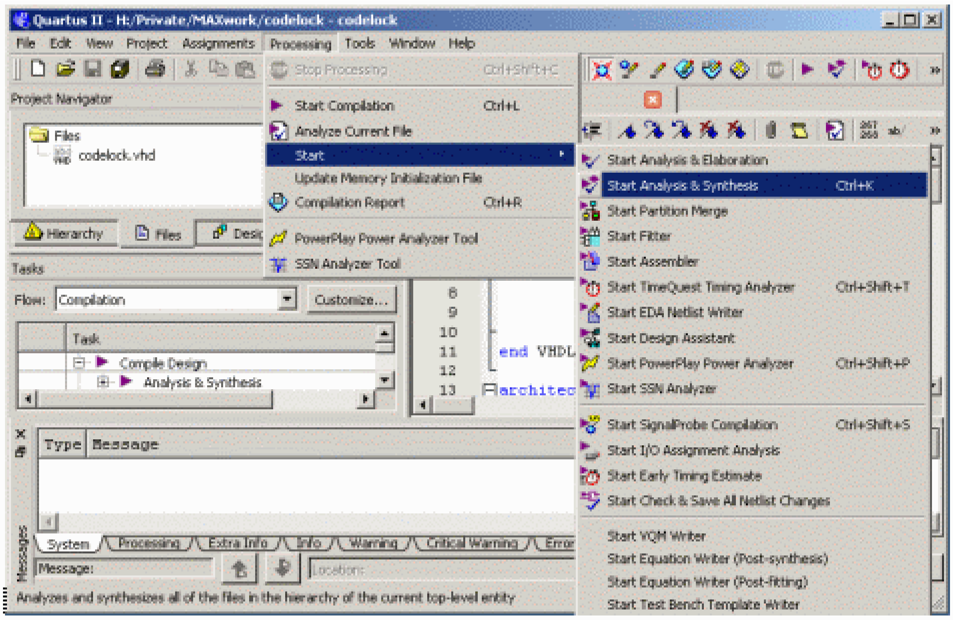
Save as type: VHDL Files (*.vhd *.vhdl)

☑ Add file to current project

Note that entity in the VHDL-file must "match" the project Top Level Entity!

Save the file: **File**, **Save As** and as VHDL-file. The name could be `codelock.vhd` (or another).
Add File to current project shall be checked!

Analysis and Synthesis

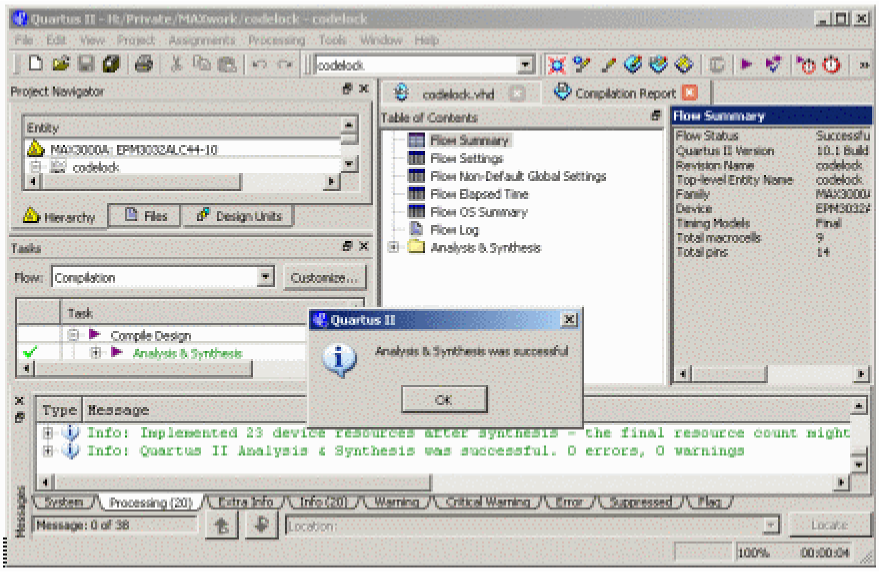


Quartus II - H:/Private/MAXwork/codelock - codelock
File Edit View Project Assignments Processing Tools Window Help
Project Navigator
Files
codelock.vhd
Hierarchy
Files
Desig
Tasks
Flow: Compilation
Customize...
Task
Compile Design
Analysis & Synthesis
Type Message
System Processing Extra Info Info Warning Critical Warning Error
Message:
Location:
Stop Processing Ctrl+Shift+C
Start Compilation Ctrl+L
Analyze Current File
Start
Update Memory Initialization File
Compilation Report Ctrl+R
PowerPlay Power Analyzer Tool
SSN Analyzer Tool
Start Analysis & Elaboration
Start Analysis & Synthesis Ctrl+K
Start Partition Merge
Start Fitter
Start Assembler
Start TimeQuest Timing Analyzer Ctrl+Shift+T
Start EDA Netlist Writer
Start Design Assistant
Start PowerPlay Power Analyzer Ctrl+Shift+P
Start SSN Analyzer
Start SignalProbe Compilation Ctrl+Shift+S
Start I/O Assignment Analysis
Start Early Timing Estimate
Start Check & Save All Netlist Changes
Start VQM Writer
Start Equation Writer (Post-synthesis)
Start Equation Writer (Post-fitting)
Start Test Bench Template Writer
8
9
10
11
12
13
end VHDL
architec

Analyzes and synthesizes all of the files in the hierarchy of the current top-level entity

When you have new code, it is unnecessary to run the entire tool chain - chances are that there are errors along the way ...

From the start you only do Analysis & Synthesis.



Flow: Compilation

Analysis & Synthesis was successful

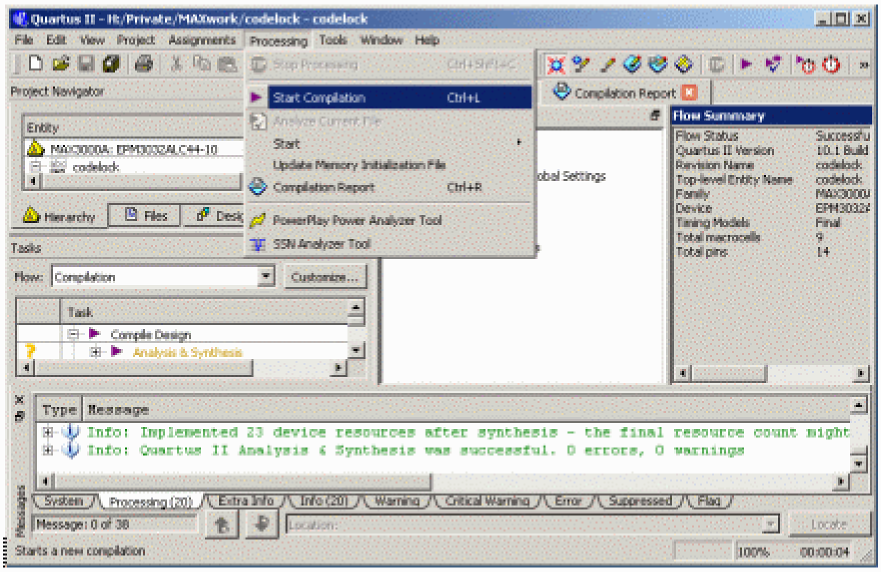
Flow Summary	
Flow Status	Successfu
Quartus II Version	10.1 Build
Revision Name	codelock
Top-level Entity Name	codelock
Family	MAX3000/
Device	EPM3032/
Timing Models	Final
Total macrocells	9
Total pins	14

```
Type Message
Info: Implemented 23 device resources after synthesis - the final resource count might
Info: Quartus II Analysis & Synthesis was successful. 0 errors, 0 warnings
```

Message: 0 of 38

Start Compilation

▶



Flow: Compilation

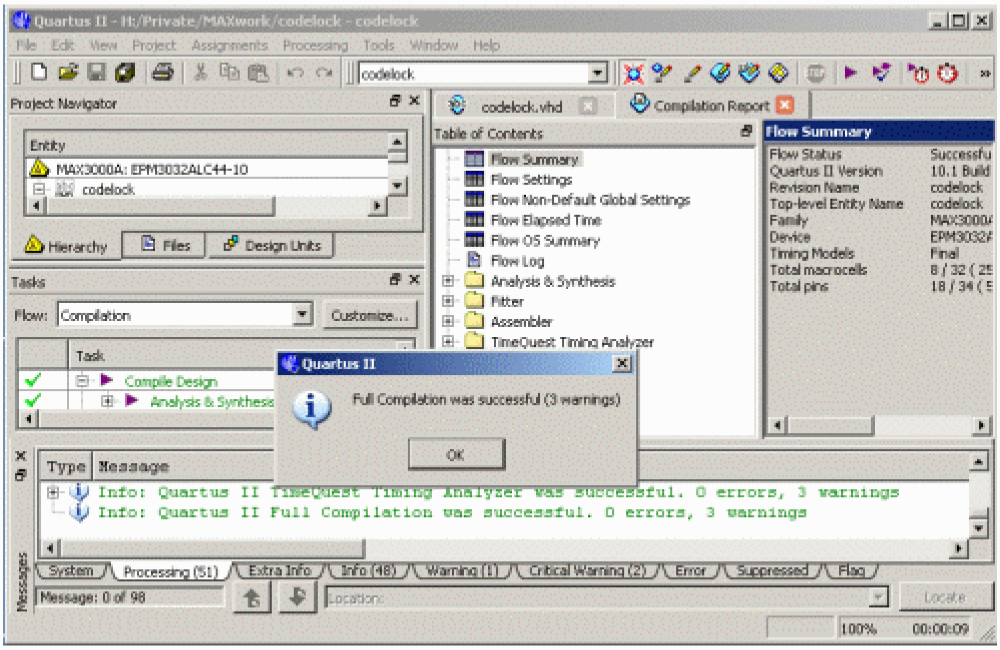
Flow Summary	
Flow Status	Successfu
Quartus II Version	10.1 Build
Revision Name	codelock
Top-level Entity Name	codelock
Family	MAX3000/
Device	EPM3032/
Timing Models	Final
Total macrocells	9
Total pins	14

```
Type Message
Info: Implemented 23 device resources after synthesis - the final resource count might
Info: Quartus II Analysis & Synthesis was successful. 0 errors, 0 warnings
```

Message: 0 of 38

Starts a new compilation

Start Compilation runs the full tool chain.



Flow Summary

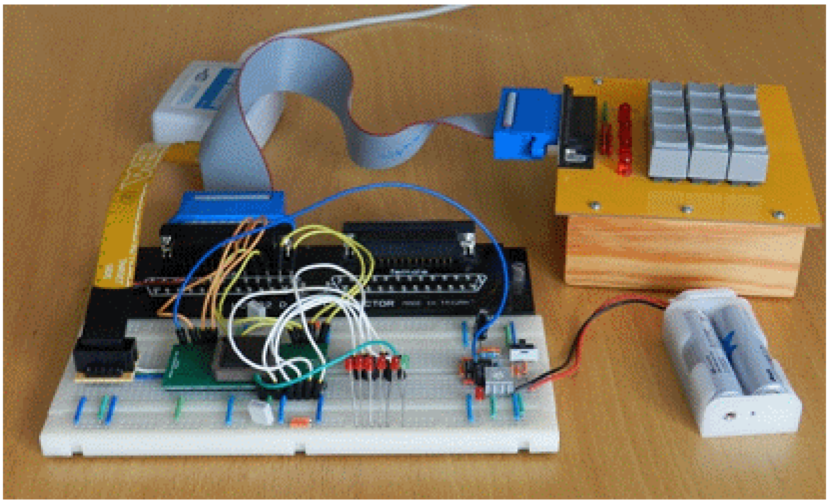
Flow Status	Successfu
Quartus II Version	10.1 Build
Revision Name	codelock
Top-level Entity Name	codelock
Family	MAX3000/
Device	EPM3032/
Timing Models	Final
Total macrocells	8 / 32 (25
Total pins	18 / 34 (5

Full Compilation was successful (3 warnings)

```
Info: Quartus II TimeQuest Timing Analyzer was successful. 0 errors, 3 warnings
Info: Quartus II Full Compilation was successful. 0 errors, 3 warnings
```

The 3 warnings (moore with other program versions) are about "software tools" that are missing in our program version but we don’t need them.

Pin-planning in Quartus

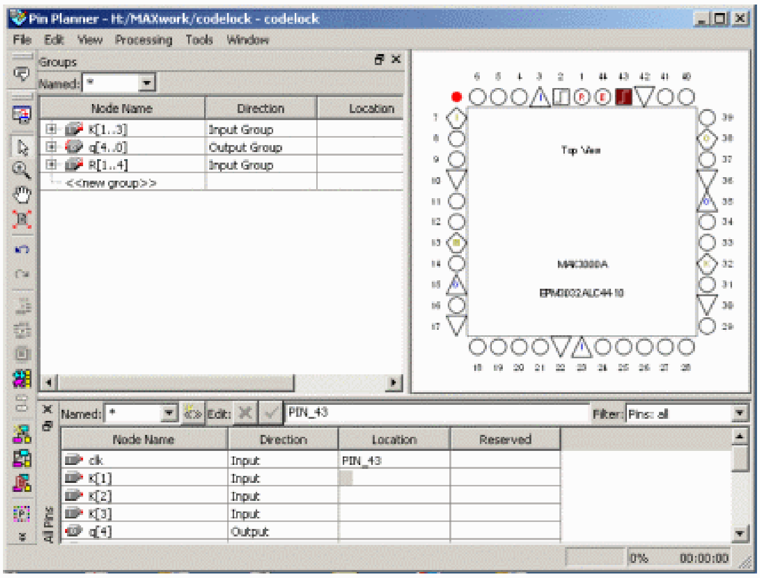


The lab code locks has different wiring. MAX chips must therefore be programmed differently to fit the equipment.

Quartus has a Pin-planning function. Either you can let the compiler choose pins for the various signals - And then adapt the wiring after this, or the wiring is determined in advance (which is the case here) and then you have to self pair signals with pins.

Start Pin Planner. Menu **Assignements**, choose **Pin Planner**, or by clicking on the icon for Pin Planner.

You write the pin-number you want to assign in the column **Location**. Double click in the box to get a list of possible pins. In the column **Fitter Location** shows which pin the autofitter function has selected - but it is usually possible to make other choices yourself.



Node Name	Direction	Location
K[1..3]	Input Group	
q[4..0]	Output Group	
R[1..4]	Input Group	
<<new group>>		

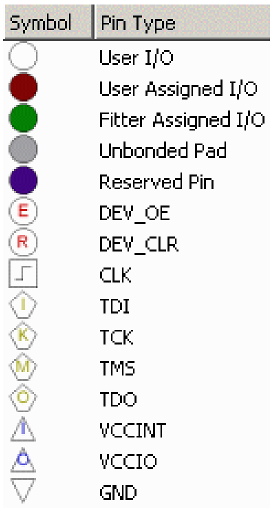
PIN_43

Node Name	Direction	Location	Reserved
clk	Input	PIN_43	
K[1]	Input		
K[2]	Input		
K[3]	Input		
q[4]	Output		

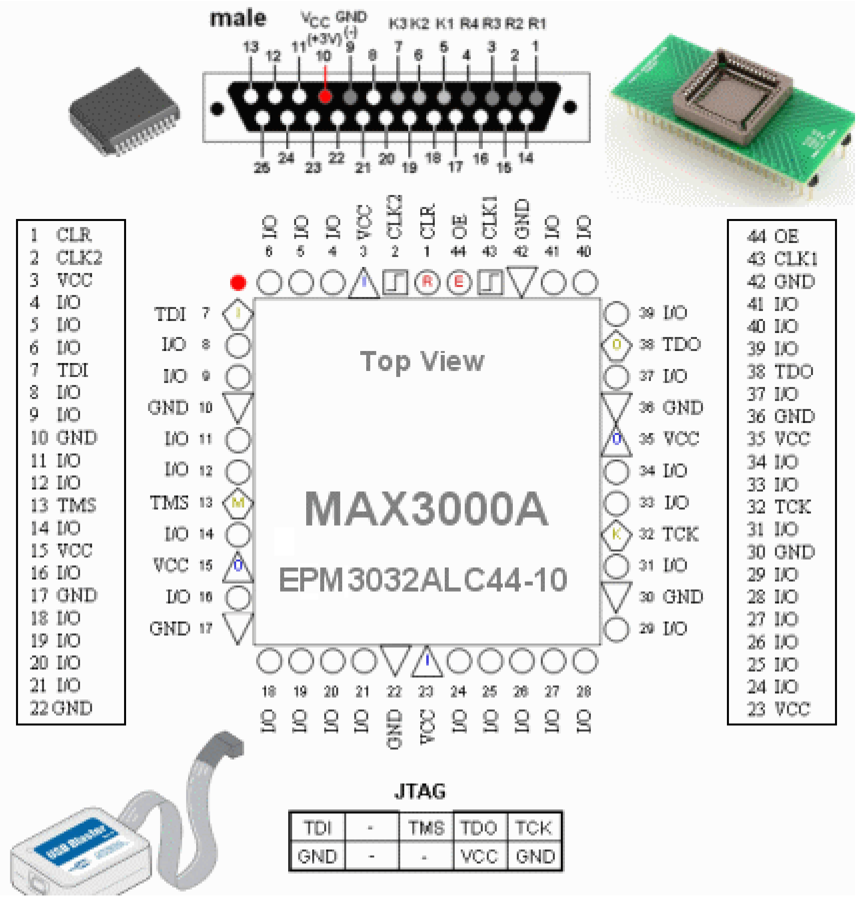
In the pin planner there is a picture of the pin layout for the selected chip, and a list of the variables/signals that has occurred in the VHDL code.
MAX3000-chip has 44-pin. Some of them are occupied for supply voltages. Often MAX circuits are used to connect digital equipment using different supply voltages, but since we do not have such needs, we have linked all those pins to the same supply voltage + 3V.
Other pins are used for chip programming, they are connected to the so-called JTAG connector (TDI TMS TCK TDO).
There are two clock inputs, one enable input and one reset input. Pin that are not used can be left unattended because the compiler makes sure they are disconnected.
Total remains 26 user defined pins, to freely use as inputs or outputs.

Connections to the lab breadboard

The keys on the keyboard are arranged as a matrix with four rows and three columns as a total of seven lines, connected to a DB-25 connector. The output signal is one yellow LED, and five red LEDs used to present the code lock internal state.



Symbol	Pin Type
○	User I/O
●	User Assigned I/O
●	Fitter Assigned I/O
●	Unbonded Pad
●	Reserved Pin
E	DEV_OE
R	DEV_CLR
⎍	CLK
I	TDI
K	TCK
M	TMS
O	TDO
△	VCCINT
△	VCCIO
▽	GND



1	CLR
2	CLK2
3	VCC
4	I/O
5	I/O
6	I/O
7	TDI
8	I/O
9	I/O
10	GND
11	I/O
12	I/O
13	TMS
14	I/O
15	VCC
16	I/O
17	GND
18	I/O
19	I/O
20	I/O
21	I/O
22	GND

44	OE
43	CLK1
42	GND
41	I/O
40	I/O
39	I/O
38	TDO
37	I/O
36	GND
35	VCC
34	I/O
33	I/O
32	TCK
31	I/O
30	GND
29	I/O
28	I/O
27	I/O
26	I/O
25	I/O
24	I/O
23	VCC

JTAG

TDI	-	TMS	TDO	TCK
GND	-	-	VCC	GND

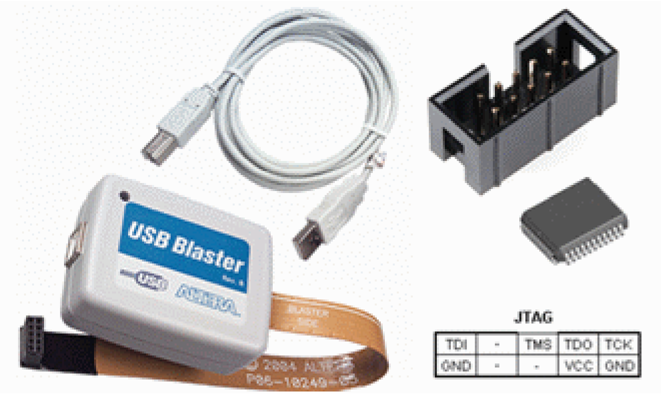
MAX-chip is contained in a TQFP-44 package. To use the MAX circuit on a breadboard, we have acquired an adapter, a breakout-board, which has pins positioned as a DIL-44 chip. The figure shows the pin placement on the MAX-chip and on the breadboard.

MAX-chip has many voltage connections. You can take them to help orient you on the breadboard.
(3 VCC, 10 GND, 15 VCC, 17 GND, 22 GND, 42 GND, 36 GND, 35 VCC, 30 GND, 23 VCC.)

Lab assistants has access to **`facit-files`** for the different lab equipment versions. With them, we can quickly rule out possible problems, eg. if you have entered the pin numbers incorrectly.

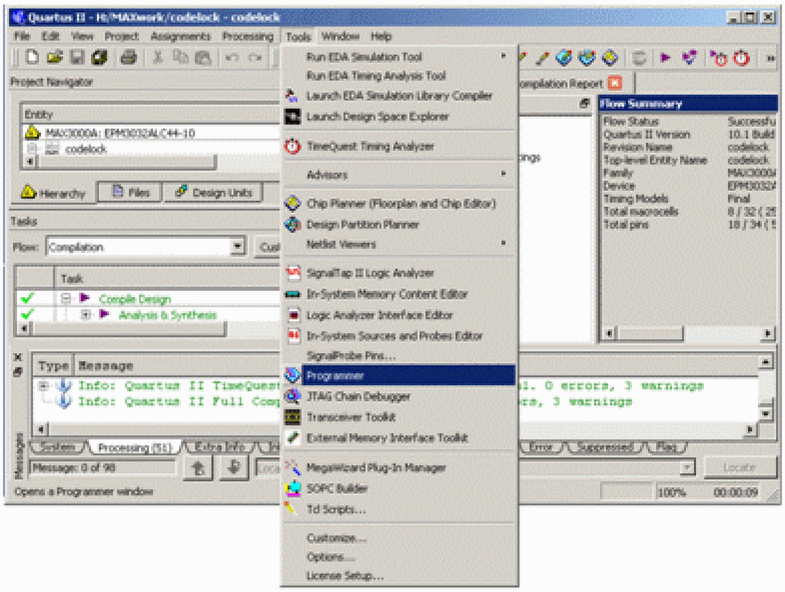
Chip-programing with Quartus

Choose chip programming equipment - USB-Blaster



Connect the USB blaster to your computer. Connect the USB Blastern JTAG connector to lab equipment. Note! The lab equipment's voltage shall be off when plugging in or unplugging the JTAG connector. Lab equipment's voltage must be turned on when you program the chip - USB power alone is not enough for this.

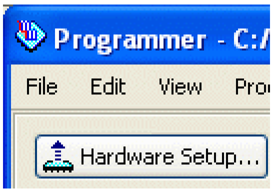
From within Quartus, select the menu item **Tools** and **Programmer**, or you clic the icon Programmer.


Quartus II - H:/MAXwork/codelock - codelock
File Edit View Project Assignments Processing Tools Window Help
Run EDA Simulation Tool
Run EDA Timing Analysis Tool
Launch EDA Simulation Library Compiler
Launch Design Space Explorer
TimeQuest Timing Analyzer
Advisors
Chip Planner (Floorplan and Chip Editor)
Design Partition Planner
Netlist Viewers
SignalTap II Logic Analyzer
In-System Memory Content Editor
Logic Analyzer Interface Editor
In-System Sources and Probes Editor
SignalProbe Pins...
Programmer
JTAG Chain Debugger
Transceiver Toolkit
External Memory Interface Toolkit
MegaWizard Plug-In Manager
SOPC Builder
Tcl Scripts...
Customize...
Options...
License Setup...
Project Navigator
Entity
MAX3000A: EPM3032ALC44-10
codelock
Hierarchy
Files
Design Units
Tasks
Flow: Compilation
Task
Compile Design
Analysis & Synthesis
Compilation Report
Flow Summary
Type
Message
System
Processing (51)
Extra Info
Error
Suppressed
Flag
Message: 0 of 98
Locate
Opens a Programmer window
100%
00:00:09

Flow Summary	
Flow Status	Successfu
Quartus II Version	10.1 Build
Revision Name	codelock
Top-level Entity Name	codelock
Family	MAX3000A
Device	EPM3032A
Timing Models	Final
Total macrocells	8 / 32 (25
Total pins	18 / 34 (5

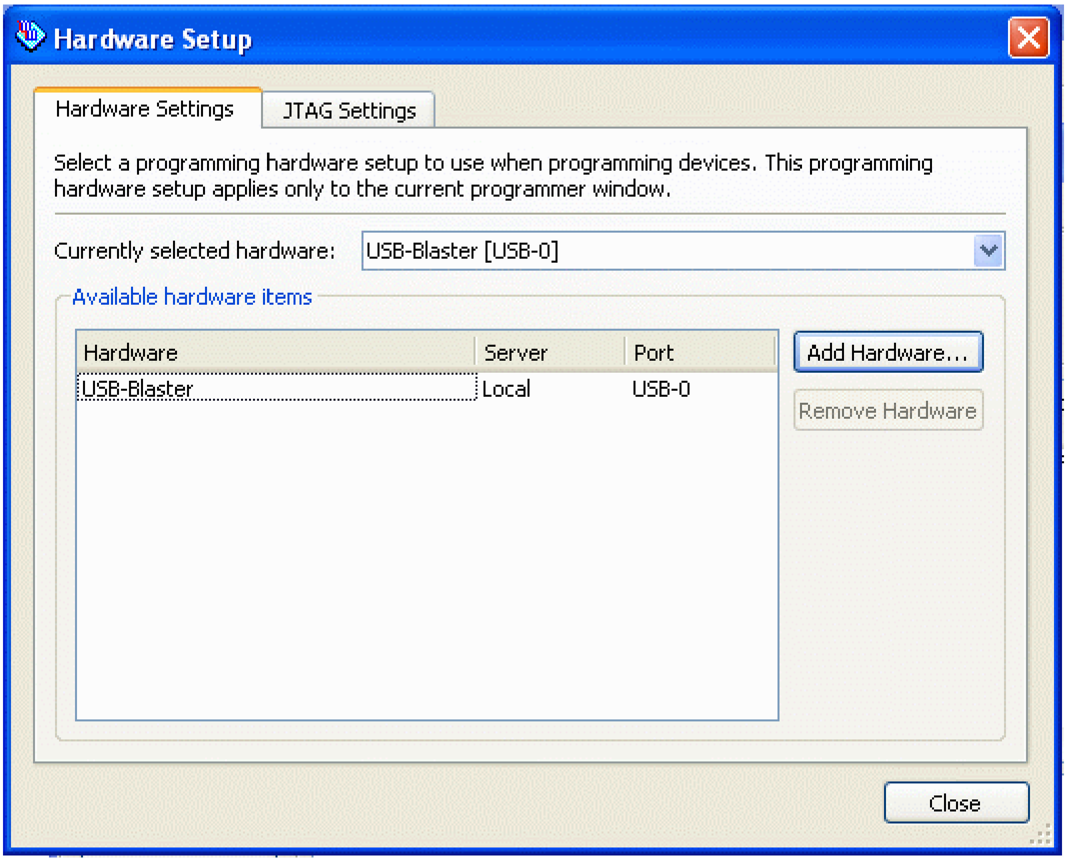
```
Info: Quartus II TimeQuest ... l. 0 errors, 3 warnings
Info: Quartus II Full Comp ... rs, 3 warnings
```

In the window **Programmer** you clic on **Hardware Setup** in order to select the USB blaster as the programming equipment. (The computer then remembers this setting)



In the window **Hardware Setup** there is a list of "Available hardware items". There stands USB blaster. (If not then you may have forgotten to plug it?) Select USB-blaster and clic on **Add Hardware**.

Now the USB-blast should be your "Currently selected hardware". click on **Close**.
Quartus will remember your choice, so probably you do not need to repeat this one more time.



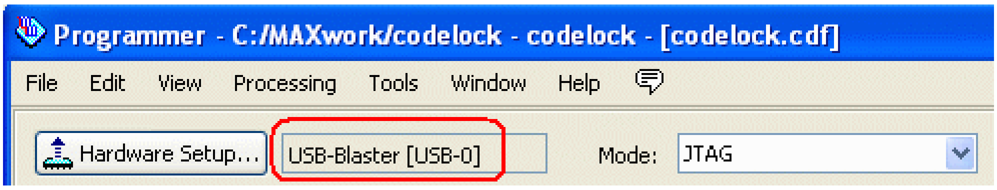
Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: USB-Blaster [USB-0]

Available hardware items

Hardware	Server	Port
USB-Blaster	Local	USB-0

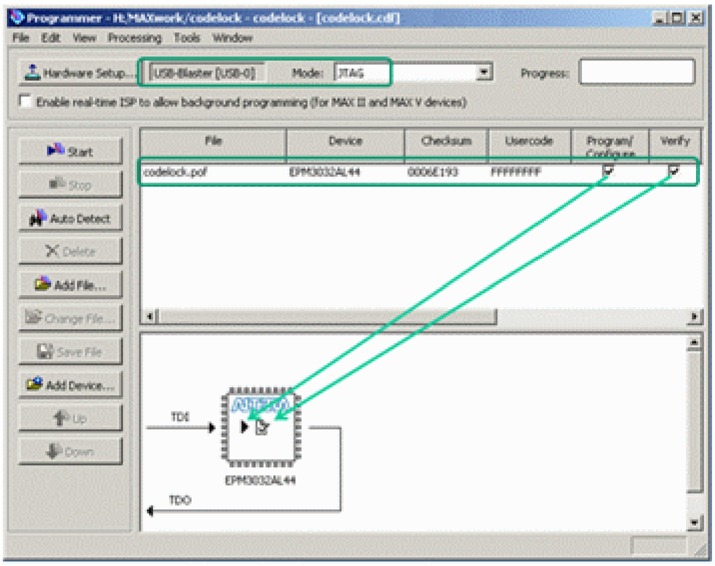
In the window **Programmer** you can see what hardvare (chip) that is selected.


Programmer - C:/MAXwork/codelock - codelock - [codelock.cdf]
File
Edit
View
Processing
Tools
Window
Help
Hardware Setup...

USB-Blaster [USB-0]

Mode: JTAG

Download the code to the chip



Programmer - H:/MAXwork/codelock - codelock - [codelock.cdf]

Hardware Setup... USB-Blaster [USB-0] Mode: JTAG

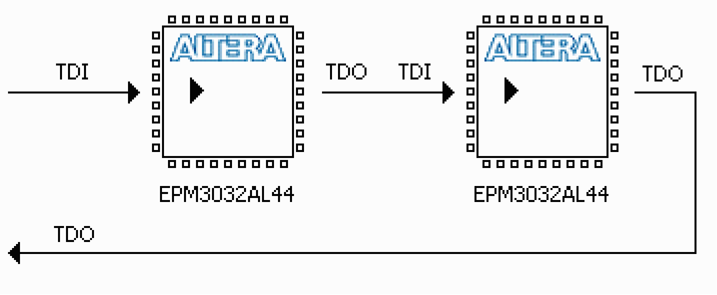
☐ Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify
codelock.pof	EPM3032AL44	0006E193	FFFFFFFF	☑	☑

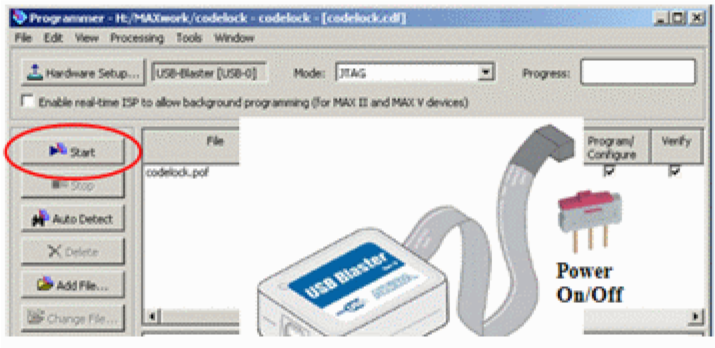
The compiled program is in a `*.pof`-file. Browse to it with **Add File ...** (it is probably in a subfolder `output_files`) and choose the file.
The compiled project name is now in the box **File**.

Check **Program/Configure** and **Verify**. In the picture with the chip there is now emerging symbols of your choices.

It is possible to program many chips in the same equipment. The picture then shows a daisy chain of the chips that can be programmed, one at a time with different code.



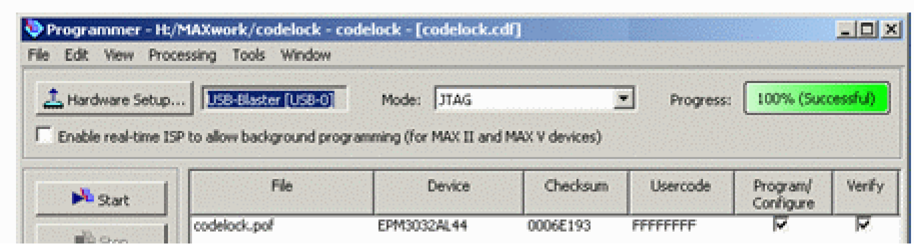
We only have one chip, and you start programming by clicking on the **Start** button.
Attention! Do not forget that the power to the lab card should be on during programming!


Programmer - H:/MAXwork/codelock - codelock - [codelock.cdf]
File Edit View Processing Tools Window
Hardware Setup...
Start
Stop
Auto Detect
Delete
Add File...
Change File...
USB Blaster
Power On/Off

USB-Blaster [USB-0] Mode: JTAG Progress:

☐ Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Program/ Configure	Verify
codelock.pof	☑	☑


Programmer - H:/MAXwork/codelock - codelock - [codelock.cdf]
File Edit View Processing Tools Window
Hardware Setup...
Start
Stop

USB-Blaster [USB-0] Mode: JTAG Progress: 100% (Successful)

☐ Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify
codelock.pof	EPM3032AL44	0006E193	FFFFFFFF	☑	☑

Try twice if it does not work - then contact the lab assistant.
The most common problem is a bad battery.

Simulate with ModelSim

ModelSim - simulation software

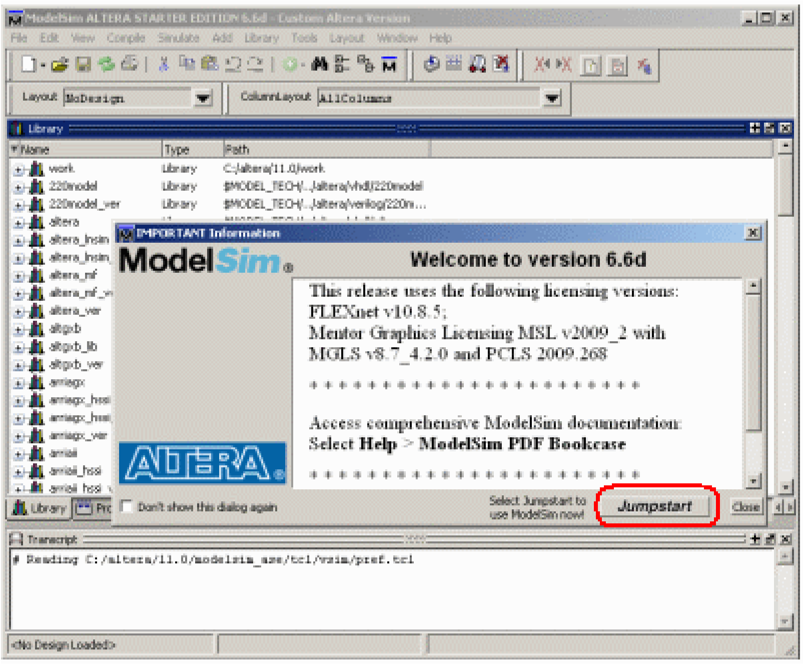
ModelSim can be used to simulate VHDL-code, to determine whether it is "right" thinking. The Altera version of **ModelSim** is also integrated with a "database" with facts about Altera-chips, eg. MAX-chips, so one can also do simulations that take into account the "time delay" and other phenomena within the intended target circuit. (As long as the target circuit is of Altera's brand ...).

ModelSim ALTERA STARTER EDITION 6.6d - Custom Altera Version

Select the correct software version - in school there are several versions installed in the Start menu!

```
Altera 13.0.1.232 Web edition\
    ModelSim-Altera Starter Edition 13.0.1.232\
          ModelSim-Altera 10.1d(Quartus II 13.0sp1)
```

Start **ModelSim**.


ModelSim ALTERA STARTER EDITION 6.6d - Custom Altera Version

IMPORTANT Information

Welcome to version 6.6d

This release uses the following licensing versions:
FLEXnet v10.8.5;
Mentor Graphics Licensing MSL v2009_2 with
MGLS v8.7_4.2.0 and PCLS 2009.268

* *

Access comprehensive ModelSim documentation:
Select **Help > ModelSim PDF Bookcase**

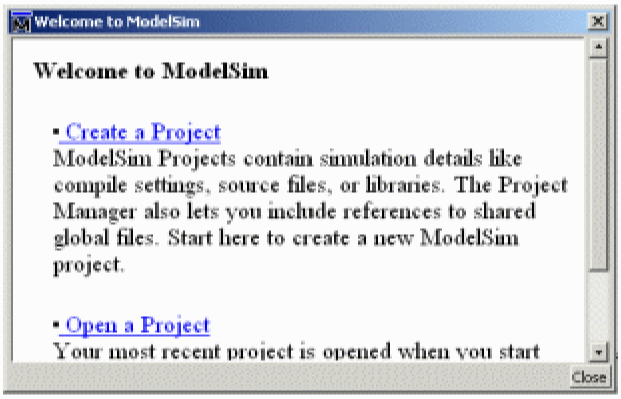
* *

☐ Don't show this dialog again

Select Jumpstart to use ModelSim now!

```
# Reading C:/altera/11.0/modelsim_ase/tcl/vsim/pref.tcl
```

In the window "important information" you click on **Jumpstart** to get help with setting up a project.



Welcome to ModelSim

Welcome to ModelSim

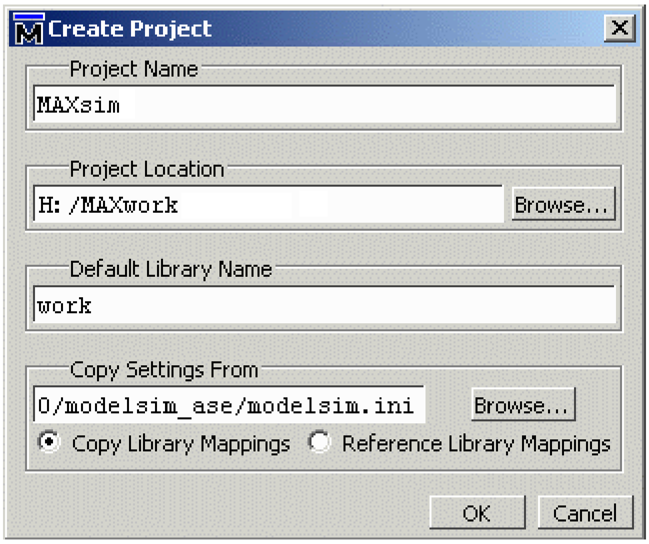
- Create a Project

ModelSim Projects contain simulation details like compile settings, source files, or libraries. The Project Manager also lets you include references to shared global files. Start here to create a new ModelSim project.

- Open a Project

Your most recent project is opened when you start

Then you click on "Create a Project" in the welcome window.



Create Project

Project Name: MAXsim

Project Location: H: /MAXwork

Default Library Name: work

Copy Settings From: 0/modelsim_ase/modelsim.ini

● Copy Library Mappings ○ Reference Library Mappings

Create a project.

Project Name

`MAXsim` can be a suitable name

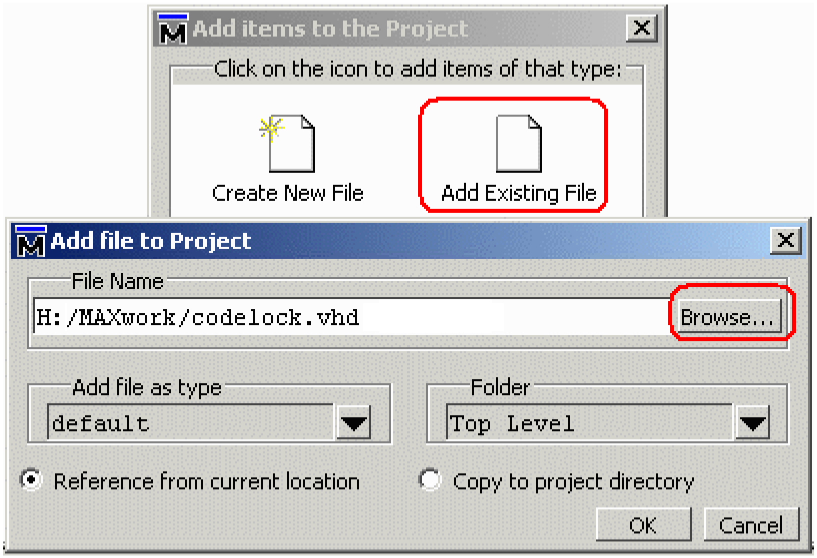
Project location

`H:/MAXwork` browse to the same folder you used for Quartus.

Default Library Name

`work` keep the suggested name, it's the standard at VHDL-simulation

Klicka på **OK**.



Add file to Project

File Name: H:/MAXwork/codelock.vhd

Add file as type: default

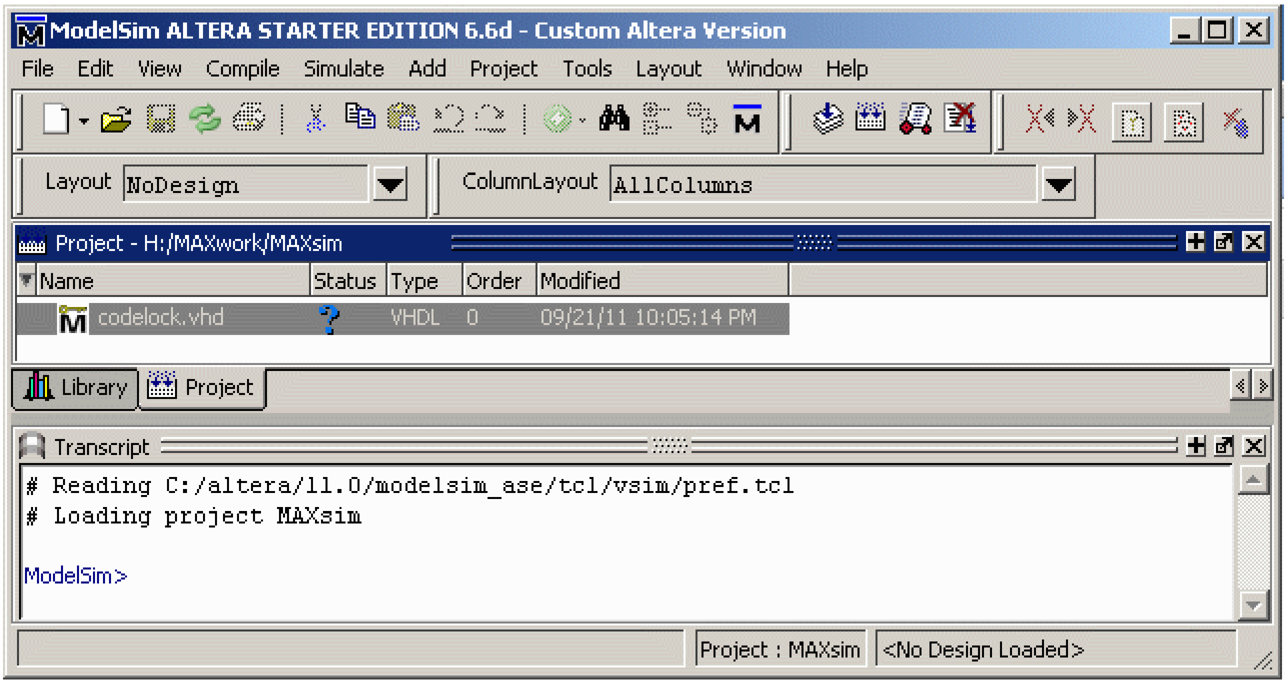
Folder: Top Level

● Reference from current location ○ Copy to project directory

We choose "Add Existing File" to add a VHDL-file to the project.
"Browse" to the file `codelock.vhd` that we created earlier with **Quartus**.

Clic on **OK**. Then click on **Close**.

Codelock code in ModelSim



ModelSim ALTERA STARTER EDITION 6.6d - Custom Altera Version
File
Edit
View
Compile
Simulate
Add
Project
Tools
Layout
Window
Help

Layout: NoDesign ColumnLayout: AllColumns

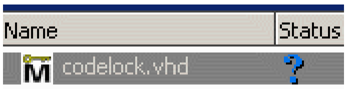
Project - H:/MAXwork/MAXsim

Name	Status	Type	Order	Modified
codelock.vhd	?	VHDL	0	09/21/11 10:05:14 PM

```
# Reading C:/altera/11.0/modelsim_ase/tcl/vsim/pref.tcl
# Loading project MAXsim

ModelSim>
```

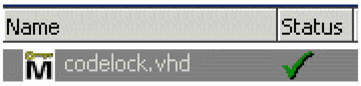
Project : MAXsim <No Design Loaded>

ModelSim has a *own* compiler to produce the code for simulation. Though we have compiled the VHDL code in **Quartus** we must now compile it again for **ModelSim**.



Name	Status
codelock.vhd	?

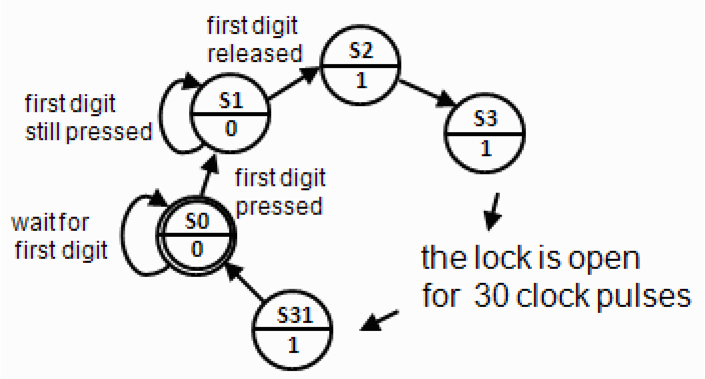
Choose **Compile** menu, alternative **Compile All**.



Name	Status
codelock.vhd	✓

Now the VHDL-code is also compiled for **Modelsim**.
Status symbol changes from a blue question mark to a green check!

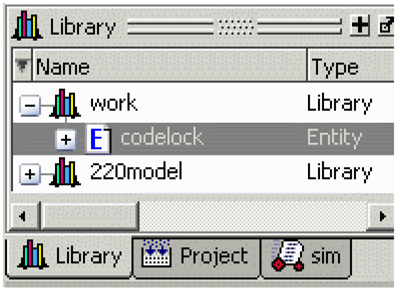
Simulate codelock-template!



We simulate by giving different commands in the **Transcript**-window, and then follow some selected signals in the window **Wave**.

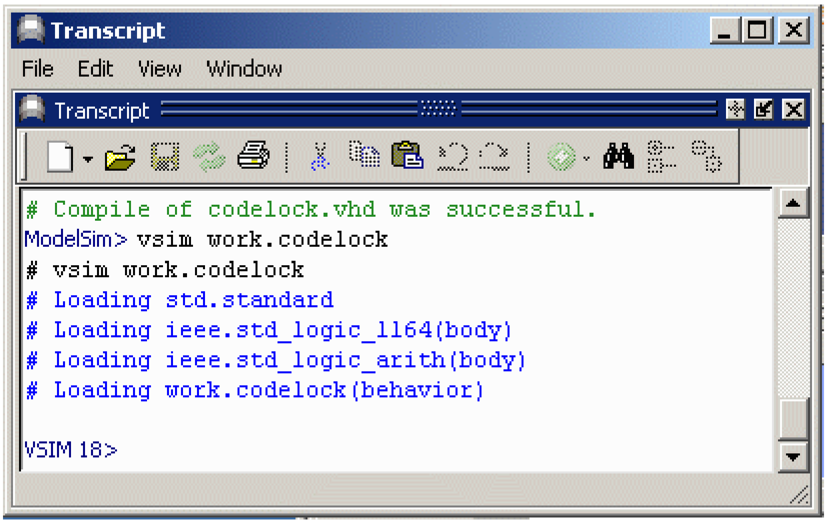
Transcript-window is a terminal window where you enter commands, but you can also give most commands by menu selection, or by clicking on buttons. Commands are always written in the**Transcript**-window, regardless of how they are given.

Load the Design to simulator..
Choose the tab **Library**, and open the folder **work**. Doubleclick on "Entity" `codelock`. A series of commands are now executed resulting in that the design is loaded into the simulator.



Name	Type
work	Library
codelock	Entity
220model	Library

In the **Transcript**-window you can follow the commands executed.



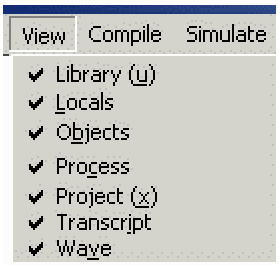
```
# Compile of codelock.vhd was successful.
ModelSim> vsim work.codelock
# vsim work.codelock
# Loading std.standard
# Loading ieee.std_logic_1164(body)
# Loading ieee.std_logic_arith(body)
# Loading work.codelock(behavior)

VSIM 18>
```

Prepare simulation

We need to have a number of windows open in order to follow the simulation.

Give thees commands in **Transcript**-window or check in the **View**-menu.

```
VSIM> view objects
VSIM> view locals
VSIM> view source
VSIM> view wave -undock
```



Modelsim consists of "windows". It can be hard to see everything at the same time. With the button **Zoom/Unzoom** you can enlarge the window. With the button **Dock/Undock** the window can be moved to any location, it is that alternative we choose for **Wave**-window. With the button **Close** those windows not needed for the time can be closed.

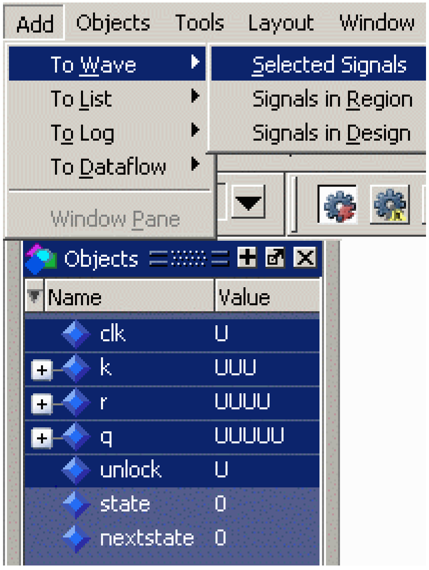
Signals in Wave window

If you have many signals, it is a good idea to select the signals you are interested to follow in **Wave**-window, but this time we choose to follow them all:

```
add wave *
```

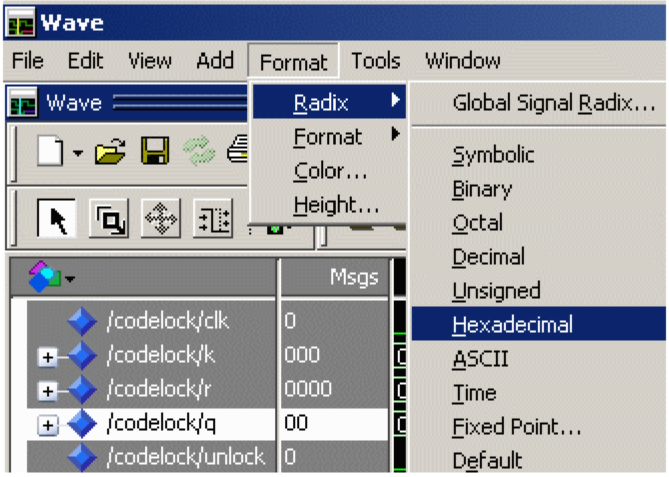
There are several ways to add signals to the **Wave**-window:

- Select signals in **Object**-window (Shift+Left Button) and "drag and drop" the selection to **Wave**-window.
- Right-click in the **Object**-window and choose **Add to Wave**.
- A **Add to Wave** dialoge-window is reachable from the menu, **Add**.



Name	Value
clk	U
k	UUU
r	UUUU
q	UUUUU
unlock	U
state	0
nextstate	0

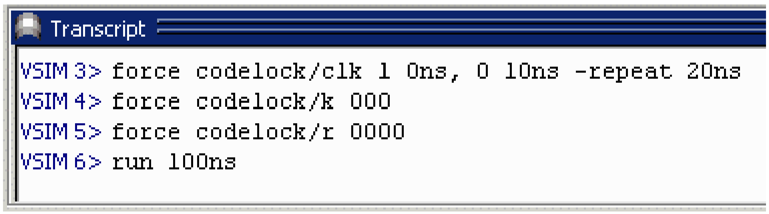
Format, Radix, Hexadecimal



	Msgs
/codelock/clk	0
/codelock/k	000
/codelock/r	0000
/codelock/q	00
/codelock/unlock	0

The state variable q has 32 different states, such a variable is easier to follow if it is presented as a hexa-decimal number, `00 ... 1F` instead of a binary number. We therefore suggest that you check the variable and change the presentation to hexadecimal. UUUUU is exchanged to XX in the **Wave**-window. Other variables are best suited to be presented as binary numbers.

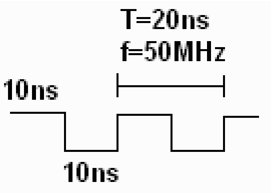
Create stimuli



```
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
```

The default time resolution in **Wave** is nanoseconds, `ns`. A suitable clock frequency for a code lock may however be as low as 5 Hz, or a period time of 0.2 sec.
The easiest way, of not having to make extensive adjustments of the program, is to "scale"

our problem to a higher clock speed with a period of 20 ns. We then has to imagine that there are fast fingers that press the keys!



Stimuli, Inputsignals as clock pulses or key-presses, are created with the command `force` in the **Transcript**-window.

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

Generates clockpulses for ever.

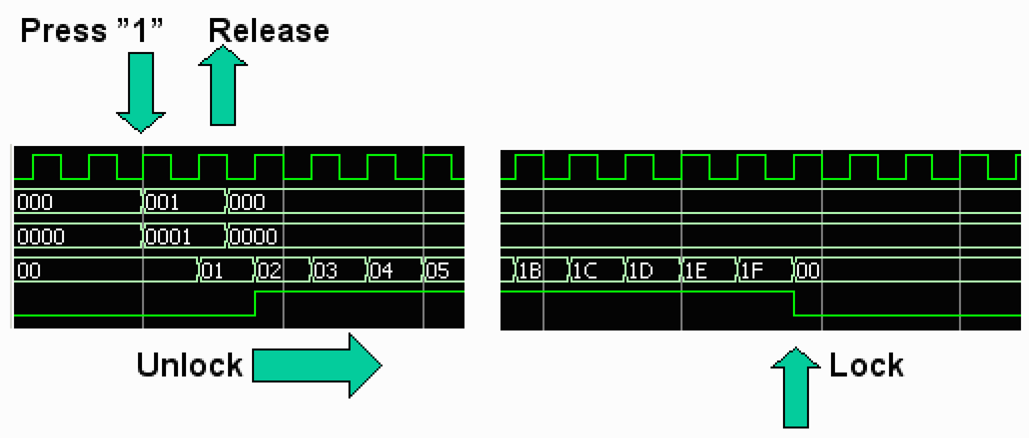
```
force codelock/k 000
force codelock/r 0000
```

Initiates variables `r` and `k`.

```
run 100ns
```

Runs the simulation in 100 ns, which is five clock cycles.

Simulate keypresses



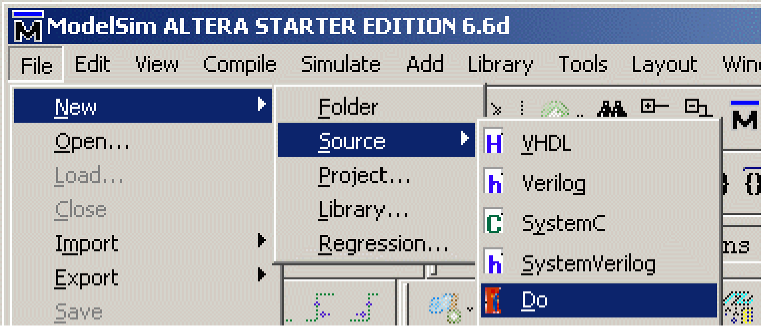
```
force codelock/k 001
force codelock/r 0001
run 30ns
force codelock/k 000
force codelock/r 0000
run 800ns
```

30 ns (20+10) means that the keypress is certain be in the gap between the clock edges. The full simulationtime 100 ns +30ns + 800 ns = 930 ns corresponds to 46.5 clock pulse periods. This is enough to show the lock's entire opening sequence.

Do-file

Transcript-window, you can run many commands in sequence from a Do-file.
Alternatively, you can copy the text in Windows (Ctrl-C) and paste it (Ctrl-V) in **Transcript**.

```
delete wave *
add wave codelock/clk
add wave codelock/k
add wave codelock/r
add wave codelock/q
add wave codelock/unlock
force codelock/clk 1 0ns, 0 10ns -repeat 20ns
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
```

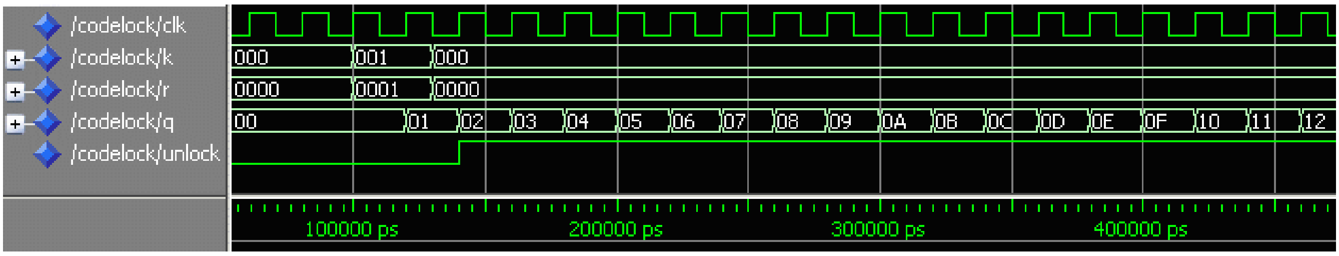


This is how to create a Do-file. Paste text commands above in the file. Then save it among the other files (in MAXwork) with extension `.do`.

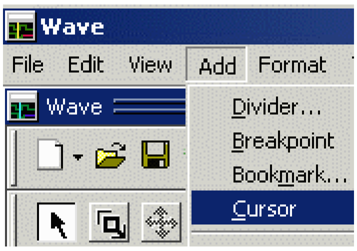
You run a Do-file with these commands (in **Transcript**):

```
restart -f
do lock.do
```

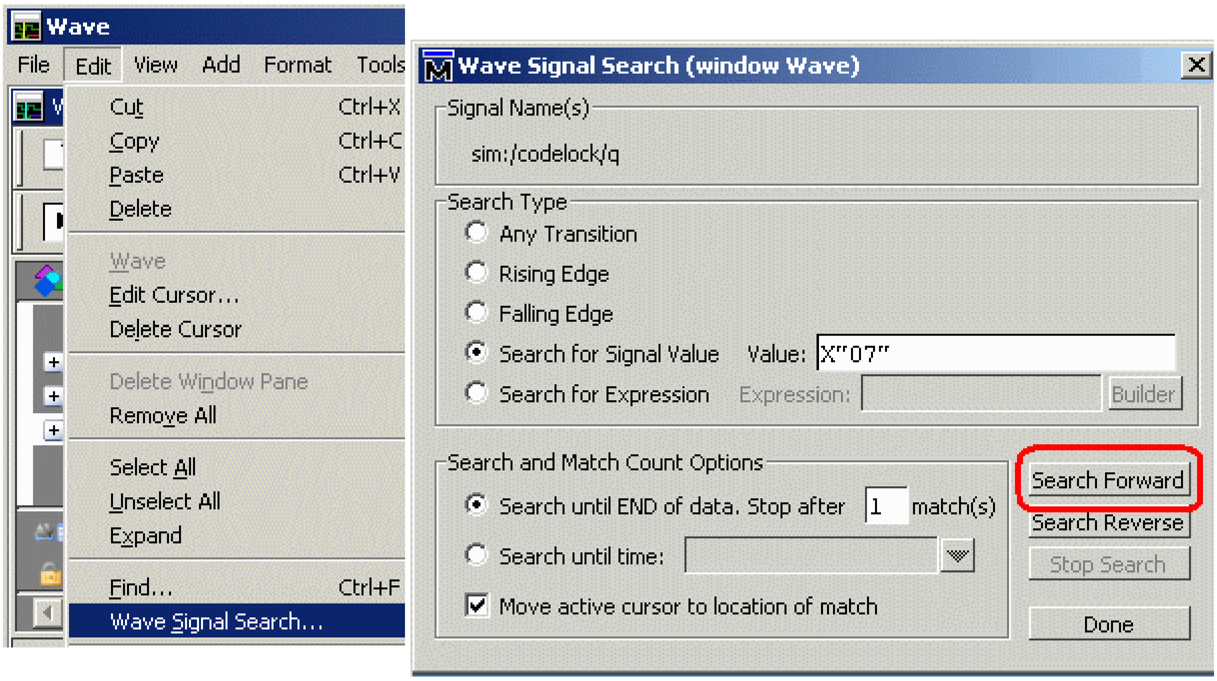
Find in the Wave window



It can be difficult to find what you are looking for in the Wave window. Therefore, there is a whole series of tools like Zoom, Expanded time, Cursors ...
Add, **Cursor**.



A Cursor can be used together with the function **Edit**, **Wave Signal Search**.



Signal Name(s)

sim:/codelock/q

Search Type

○ Any Transition

○ Rising Edge

○ Falling Edge

● Search for Signal Value Value: X"07"

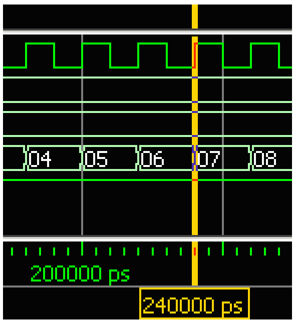
○ Search for Expression Expression:

Search and Match Count Options

● Search until END of data. Stop after 1 match(s)

○ Search until time:

☑ Move active cursor to location of match



Now the Cursor points what happens (this time nothing special!) when q has the state 07.

Spend a little time now to try different tools available for orientation in Wave Window!

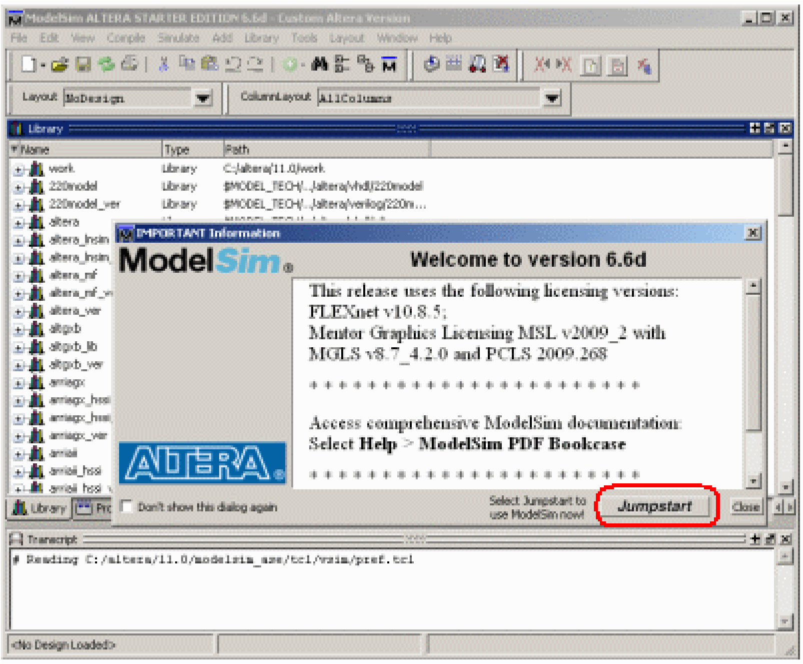
The simulation has shown that the lock opens to the intended key-press, but this is not enough - There is need for more "testing" before one can trust the construction!

Testbench in ModelSim

Select the correct software version - in school there are several versions installed in the Start menu!

```
Altera 13.0.1.232 Web edition\
    ModelSim-Altera Starter Edition 13.0.1.232\
          ModelSim-Altera 10.1d(Quartus II 13.0sp1)
```

Start **ModelSim**.



ModelSim ALTERA STARTER EDITION 6.6d - Custom Altera Version

Name	Type	Path
work	Library	C:/altera/11.0/work
220model	Library	$MODEL_TECH/../altera/vhdl/220model
220model_ver	Library	$MODEL_TECH/../altera/verilog/220m...

IMPORTANT Information

Welcome to version 6.6d

This release uses the following licensing versions:
FLEXnet v10.8.5;
Mentor Graphics Licensing MSL v2009_2 with
MGLS v8.7_4.2.0 and PCLS 2009.268

* *

Access comprehensive ModelSim documentation:
Select **Help > ModelSim PDF Bookcase**

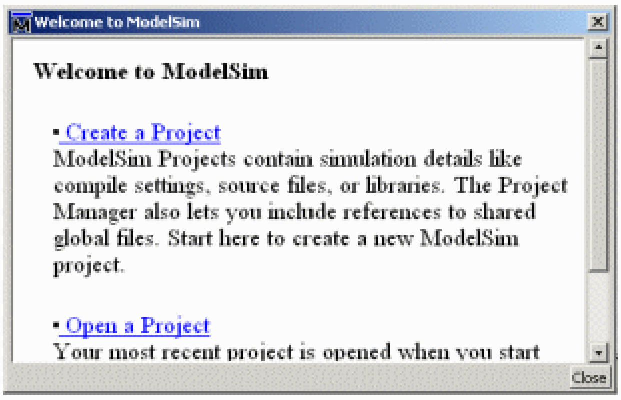
* *

□ Don't show this dialog again

Select Jumpstart to use ModelSim now!

```
# Reading C:/altera/11.0/modelsim_ase/tcl/vsim/pref.tcl
```

Klicka på **Jumpstart**. This time we choose "Open Project" for continue with our previous MAXsim-project.



Welcome to ModelSim

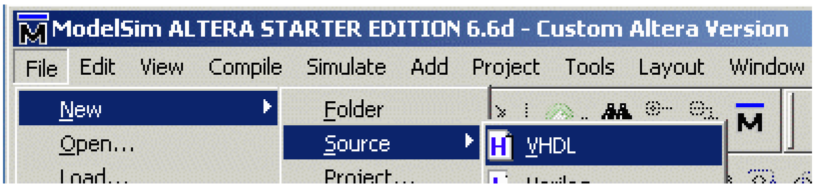
Welcome to ModelSim

- Create a Project
  ModelSim Projects contain simulation details like compile settings, source files, or libraries. The Project Manager also lets you include references to shared global files. Start here to create a new ModelSim project.

- Open a Project
  Your most recent project is opened when you start

Testbench

In addition to the VHDL code for the lock, we now need another VHDL file for the test bench code.

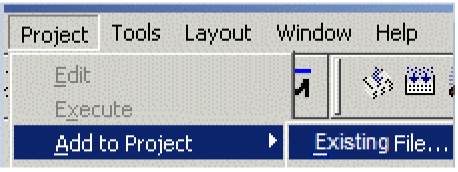


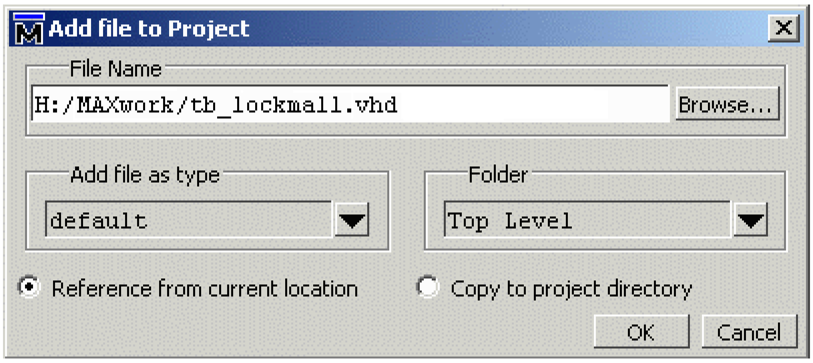
Create a new empty VHDL-file. Copy and paste the content of the file `tb_lockmall.vhd` and then save the file using the same name, `tb_lockmall.vhd`, among the other files in the project.

Alternatively, you can copy the file `tb_lockmall.vhd` to the folder with the other files of the project.

tb_lockmall.vhd (tb_lockmall.txt)

Add the VHDL-file to the project





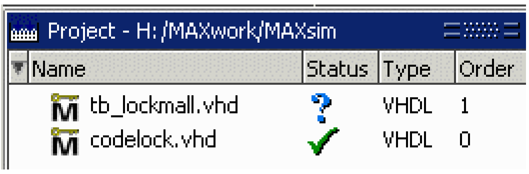
File Name: H:/MAXwork/tb_lockmall.vhd

Add file as type: default

Folder: Top Level

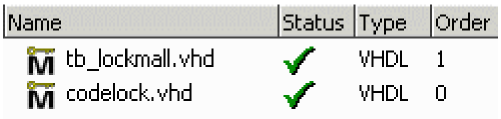
● Reference from current location ○ Copy to project directory

The project now has two files. The file `tb_lockmall.vhd` is not compiled yet, which can be seen on the blue question mark.



Name	Status	Type	Order
tb_lockmall.vhd	?	VHDL	1
codelock.vhd	✓	VHDL	0

Choose the menu **Compile** and the alternative **Compile All**. Now the file is compiled `tb_lockmall.vhd`. By the content of the file we can see that it controles the other file `codelock.vhd`, so that's why it gets the highest order.



Name	Status	Type	Order
tb_lockmall.vhd	✓	VHDL	1
codelock.vhd	✓	VHDL	0

Load the Design to the simulator.

We want to test the bench messages to appear (as arrows) at the top of the **Wave**-window. Therefore, we write in the **Transcript** window:

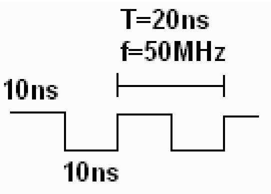
```
vsim -msgmode both -displaymsgmode both tb_codelock
```

Then add the signals in the **Wave**-window.

```
add wave *
```

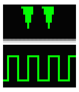
Start simulation

In the file `tb_codelock.vhd` you find `clk <= not clk after 10 ns;`. It creates the clock pulses with a period of 20 ns, as we used before.

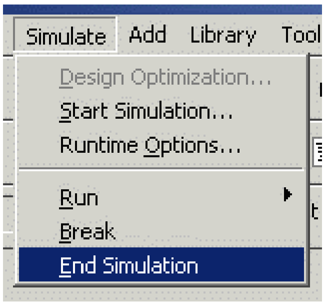


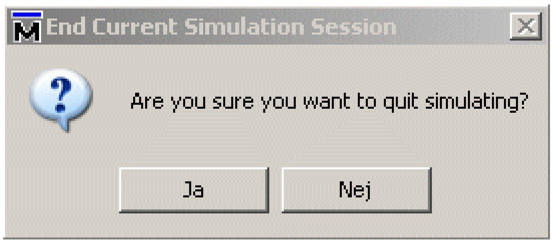
We can immediately start the simulation with the command `run 3us` in the **Transcript**-window. The time 3 µs sufficient to try all possible key-press combinations.

In the **Transcript**-window we can read the encouraging message `Note: Lock tries to open for the right sequence!`.
In **Wave**-window choose **View**, **Zoom**, and **Zoom Full** and then you can see the entire process.



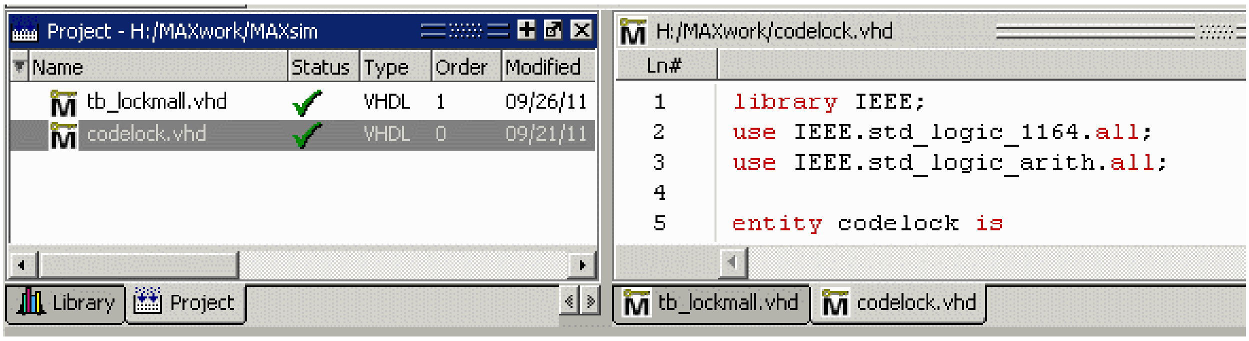
The green arrows show the *when* the desired event happens!
Now exit the simulation.





Are you sure you want to quit simulating?

Reveal the wrong code!



Name	Status	Type	Order	Modified
tb_lockmall.vhd	✓	VHDL	1	09/26/11
codelock.vhd	✓	VHDL	0	09/21/11

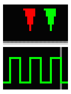
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity codelock is
```

Now we need to in **Project** window, double-click the file `codelock.vhd` so that it appears in the text editor where we can change it. Copy and paste the content of the file `lockmall_with_error.vhd`. Then save the file under the same name as before and recompile everything.

lockmall_with_error.vhd (lockmall_with_error.txt)

Simulate in the same way as before. In **Transcript**-window you can read the message from the test bench when event happens!

The red arrow in the **Wave**-window shows *when/where* the undesired event happens!



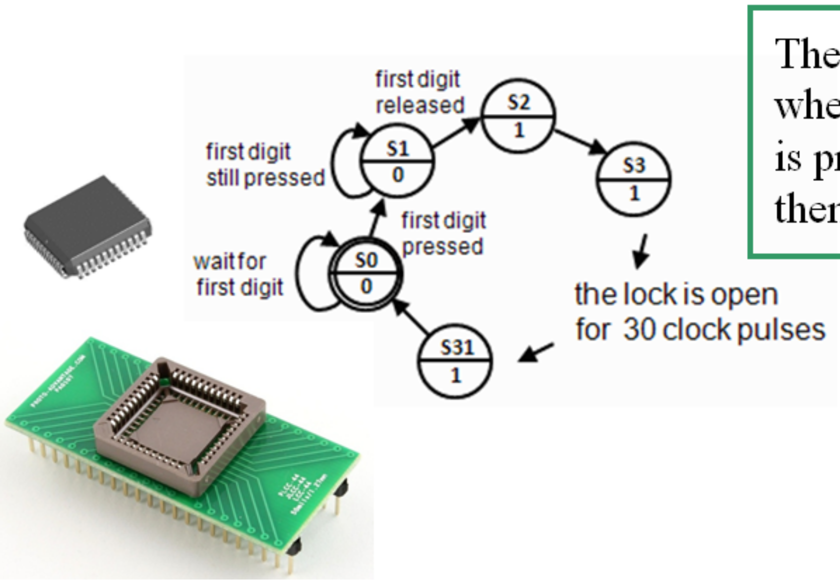
VHDL-test bench file

You do not of course be able to write a VHDL Test Bench file after a short first course on Digital Design. Still take the opportunity to go through the file and see if you can understand how it is intended!

Se **VHDL testbench file**

VHDL Testbench for ModelSim

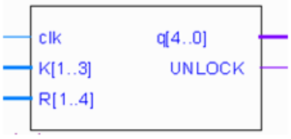
Template -program action

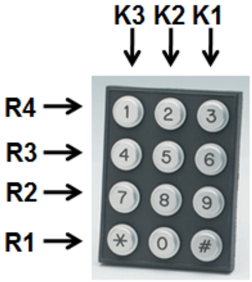


The lock opens when the key "1" is pressed and then released.

Keypad and Statecounter

Good choice of data types makes the code self-explanatory!





```
K: in  std_logic_vector(1 to 3);
R: in  std_logic_vector(1 to 4);
```

```
    1 2 3                      1 2 3 4
K ="001" bitvector          R ="0001" bitvector
K(3)='1' bit                R(4)='1'  bit
```

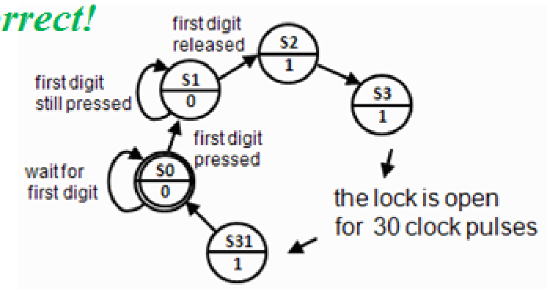
```
                  4 3 2 1 0
Statecounter:  q ="00001" bitvector
               q(0)='1'   bit
```

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;
```

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

Does both expressions mean the same?

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

Is this really the same thing?

```
(((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'))))))
```

Someone "promises" that the code is correct - but how can you know that this is absolutely true?

tb_lockmall.vhd

We need to write a VHDL-testbench

A test bench program can test all the possible key combinations and report if there is a problem ...

It can automatically loop through all possible key-presses and report on whether the lock trying to open or not.

There are 2^7 = 128 possible key combinations and we'd be totally exhausted if we tried to test them all by hand.

entity – a testbench has no ports

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

Some internal signals are needed

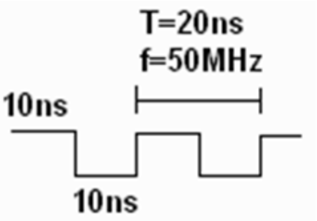
```
signal          clk : std_logic := '0';
signal       K_test : std_logic_vector(1 to 3);
signal       R_test : std_logic_vector(1 to 4);
signal  prev_K_test : std_logic_vector(1 to 3);
signal  prev_R_test : std_logic_vector(1 to 4);
signal            q : std_logic_vector(4 downto 0);
signal      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

Our codelock is used as a 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;
```

Generate a simulation clock

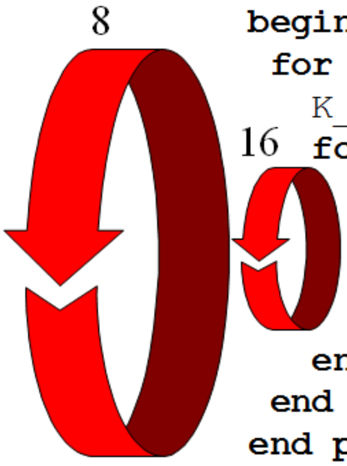
```
-- generate a simulation clock
clk <= not clk after 10 ns;
```



Instantiation and 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 );
```

Two nested loops creates keystrokes

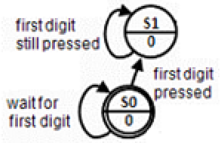


```
process
begin
 for k in 0 to 7 loop
  K_test <= conv_std_logic_vector(k,3);
  for r in 0 to 15 loop
   prev_K_test <= K_test;
   prev_R_test <= R_test;
   R_test <= conv_std_logic_vector(r,4);
   wait until CLK='1';
  end loop;
 end loop;
end process;
```

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;
```

Simulate and find the error!

What else besides pressing the "1" key could open the lock?

?