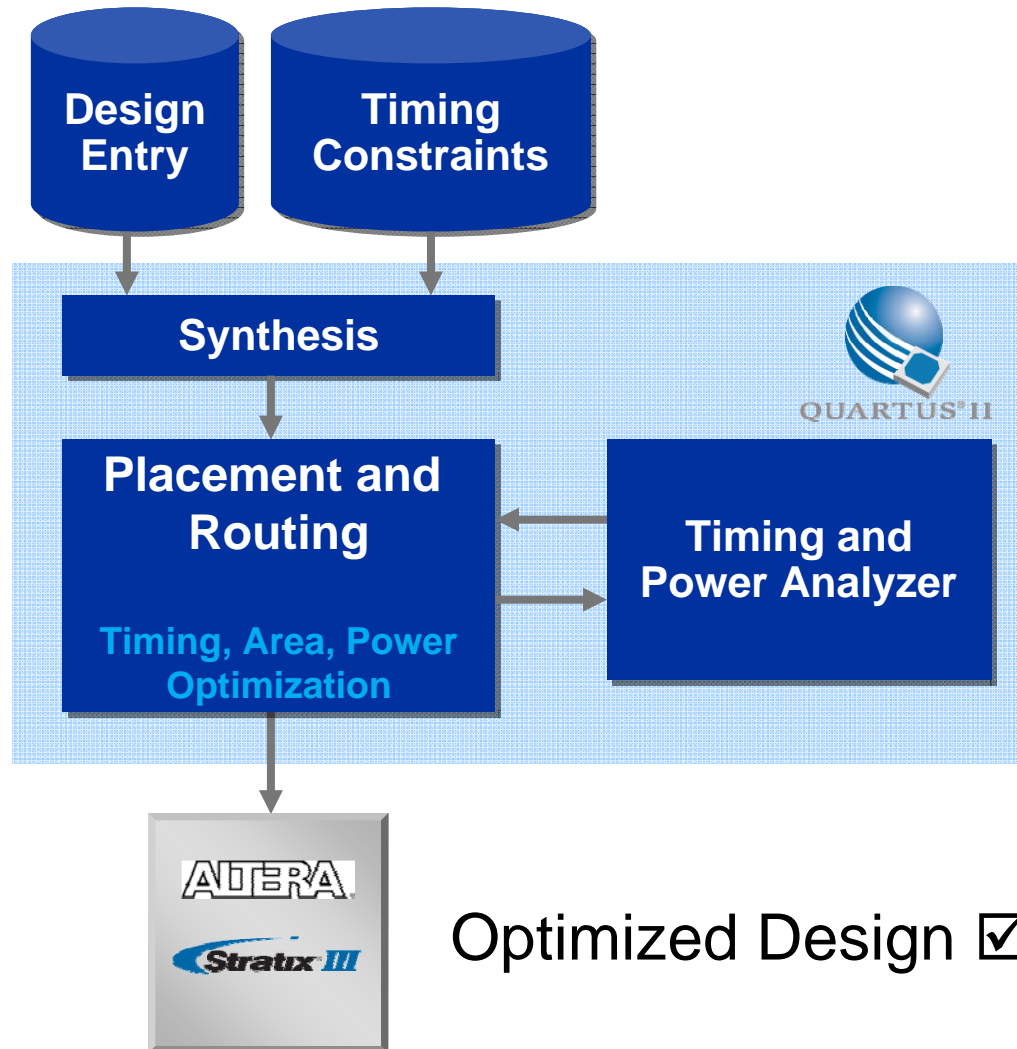


# QUARTUS II

# Quartus II Design Flow

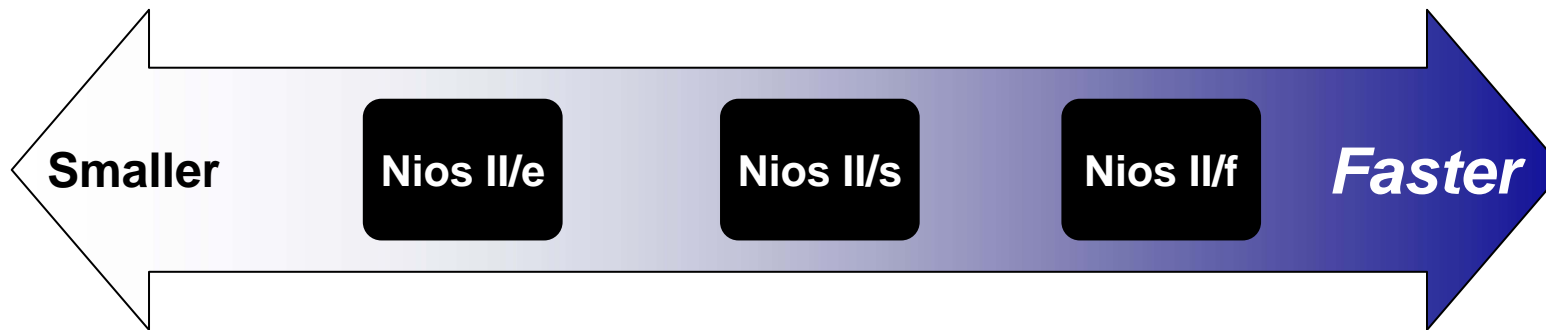


© 2010 Altera Corporation

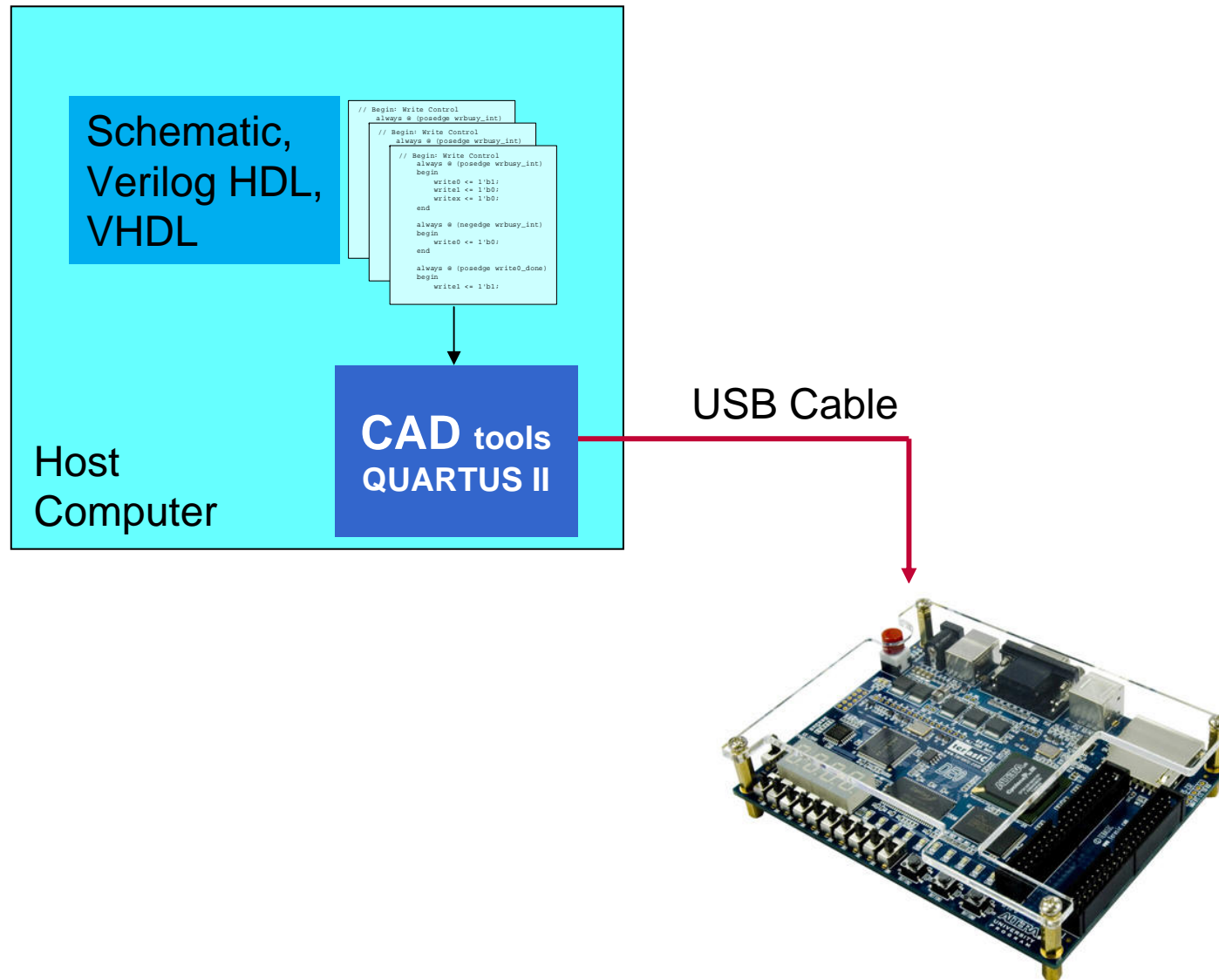
ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS & STRATIX are Reg. U.S. Pat. & Tm. Off. and Altera marks in and outside the U.S.

# Can I still use a Processor?

- YES!
- Three Altera Soft Processor Choices:
  - Nios II/f      Fast: Optimized for Performance
  - Nios II/s      Standard: Faster and Smaller than Nios
  - Nios II/e      Economy: Smallest FPGA Footprint
- Choose peripherals you want
- SOPC Builder software builds interfaces, arbitration etc.



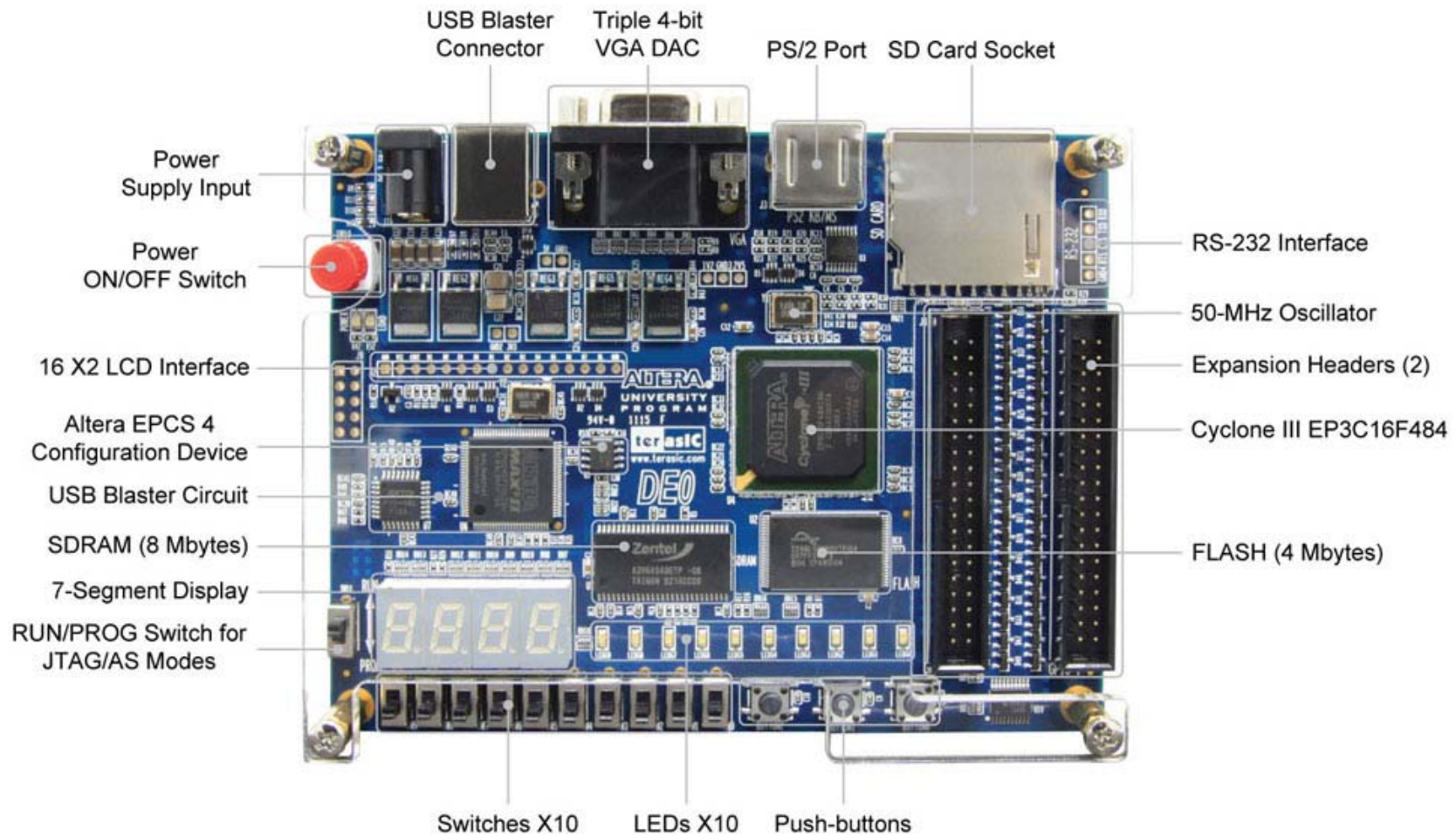
# Altera Quartus II CAD Tools



© 2010 Altera Corporation

ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS & STRATIX are Reg. U.S. Pat. & Tm. Off. and Altera marks in and outside the U.S.

# DE0

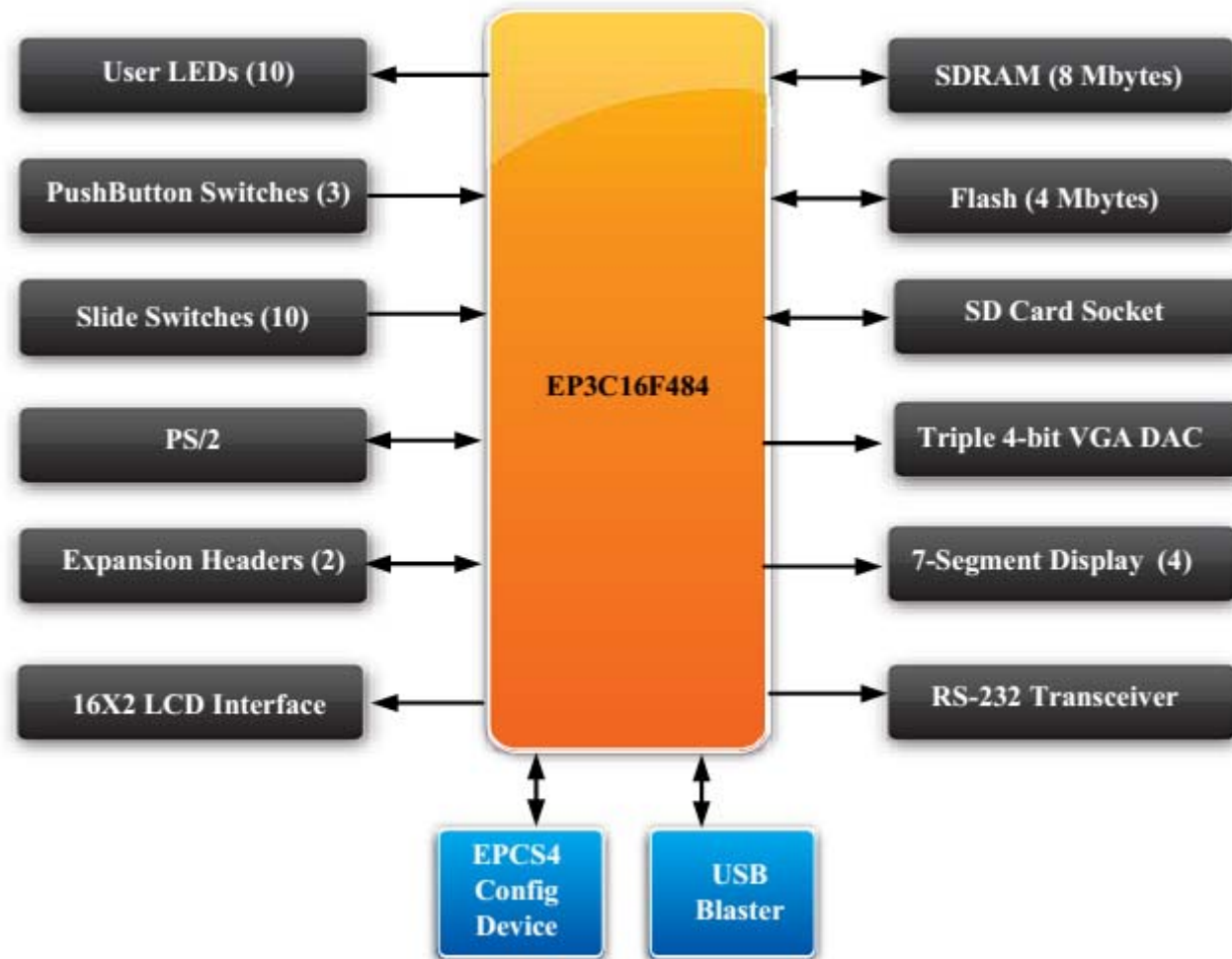


© 2010 Altera Corporation

ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS & STRATIX are Reg. U.S. Pat. & Tm. Off. and Altera marks in and outside the U.S.



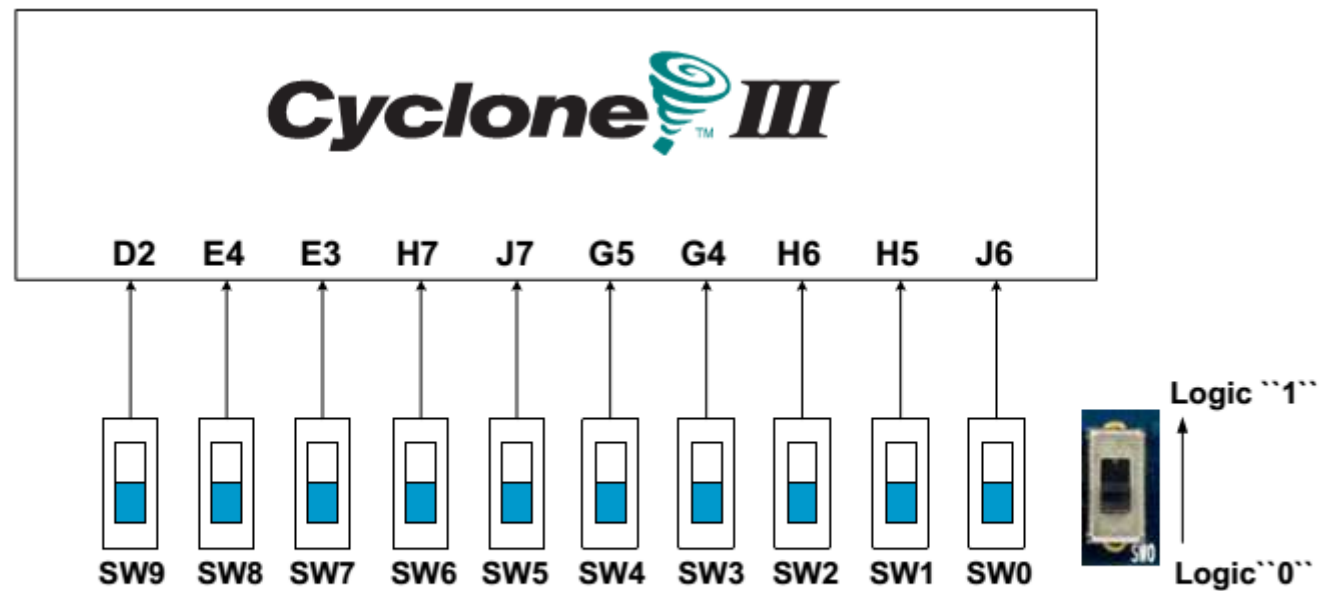
# DE0



© 2010 Altera Corporation

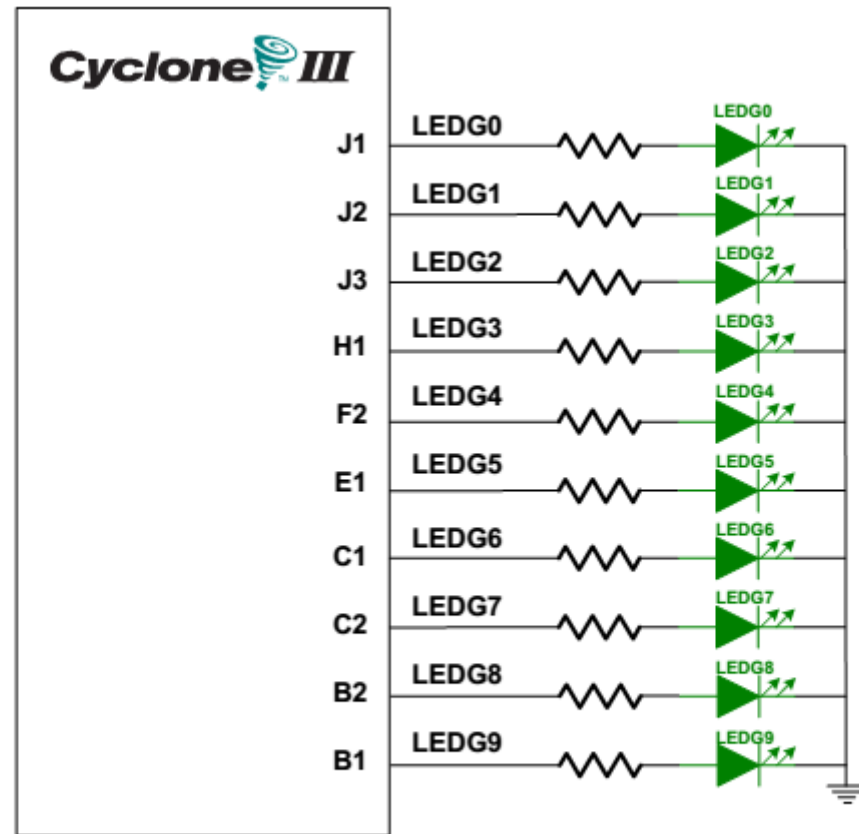
ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS & STRATIX are Reg. U.S. Pat. & Tm. Off. and Altera marks in and outside the U.S.

# DE0 – Switches



Connections between the toggle switches and Cyclone III FPGA

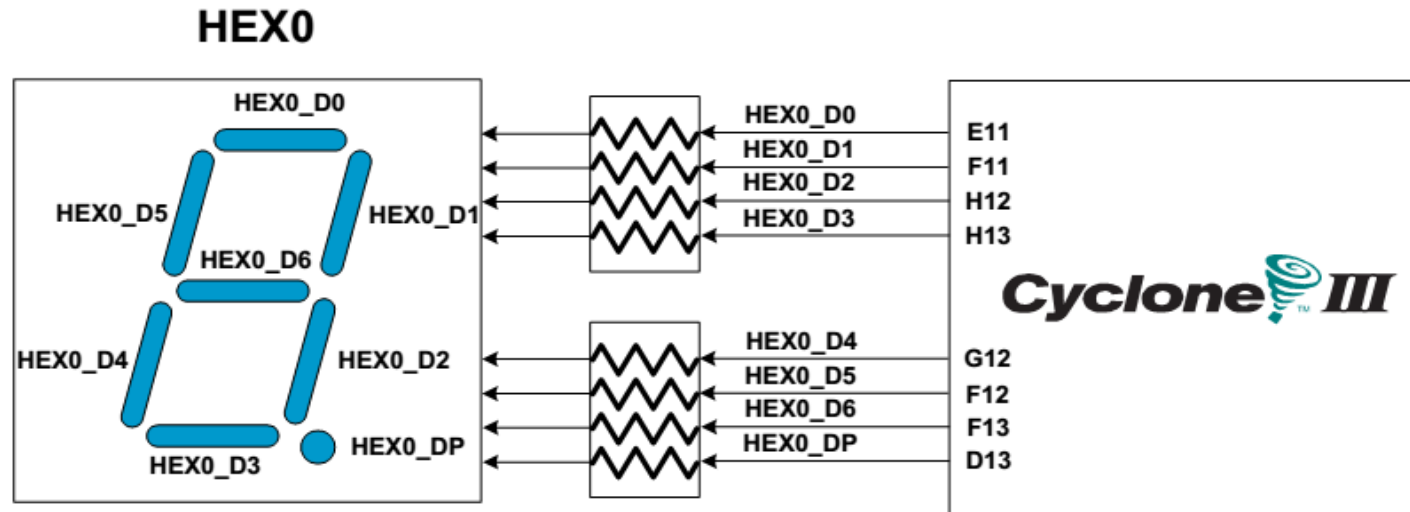
# DE0 – LEDs



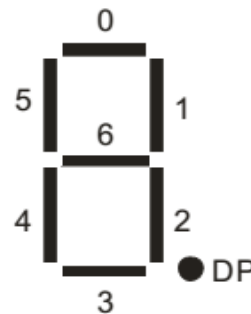
Connections between the LEDs and Cyclone III FPGA



# DE0 – 7-segment displays



Connections between the 7-segment displays and Cyclone III FPGA



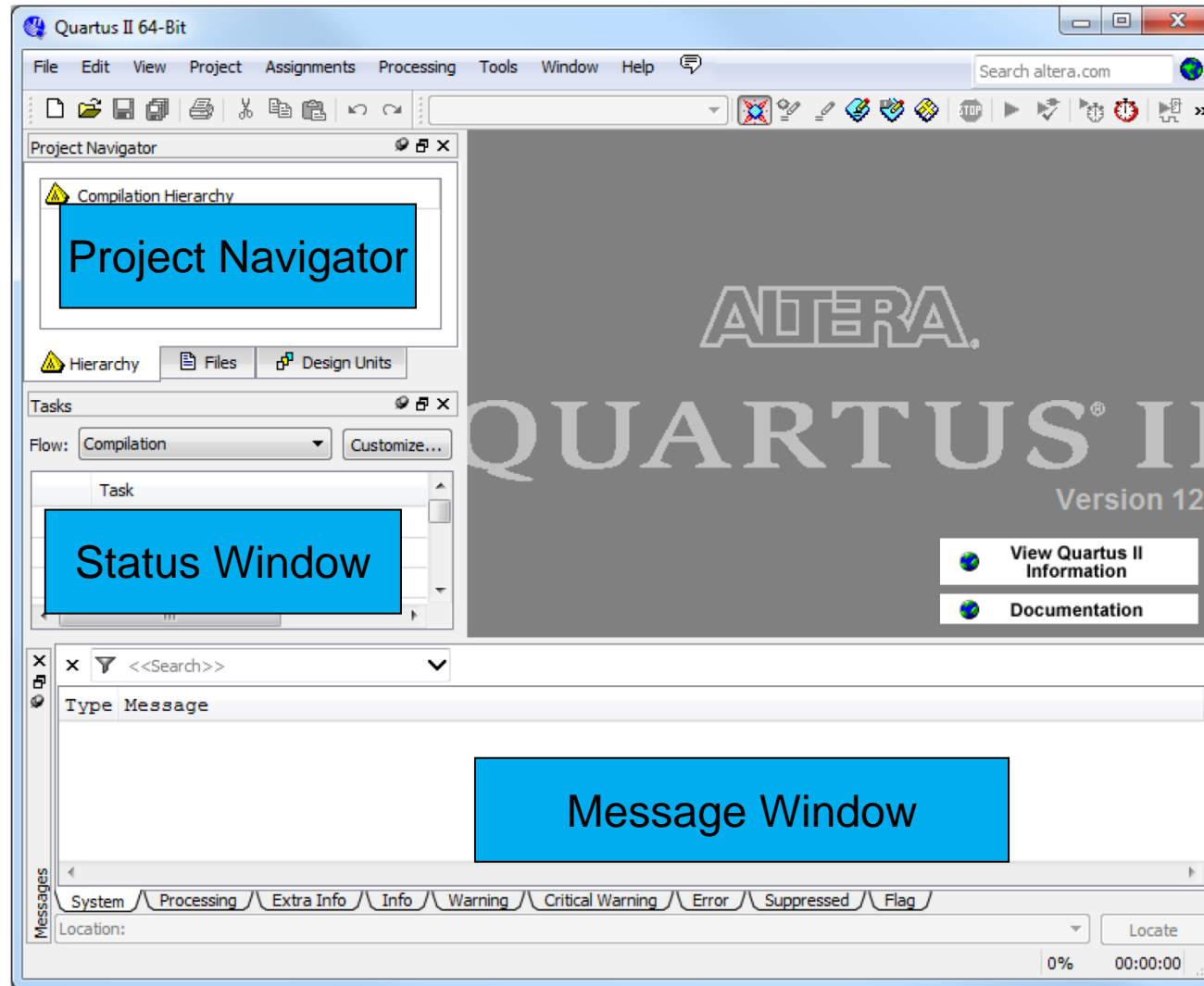
# Developing Digital Logic courses with Altera Technology

## Tutorial #1

# Outline

- Creating projects in Quartus II
- Targeting a project for a DE0 Board
- Downloading a circuit onto a DE0 board
- Compiling and debugging

# Step 1: Start Quartus II

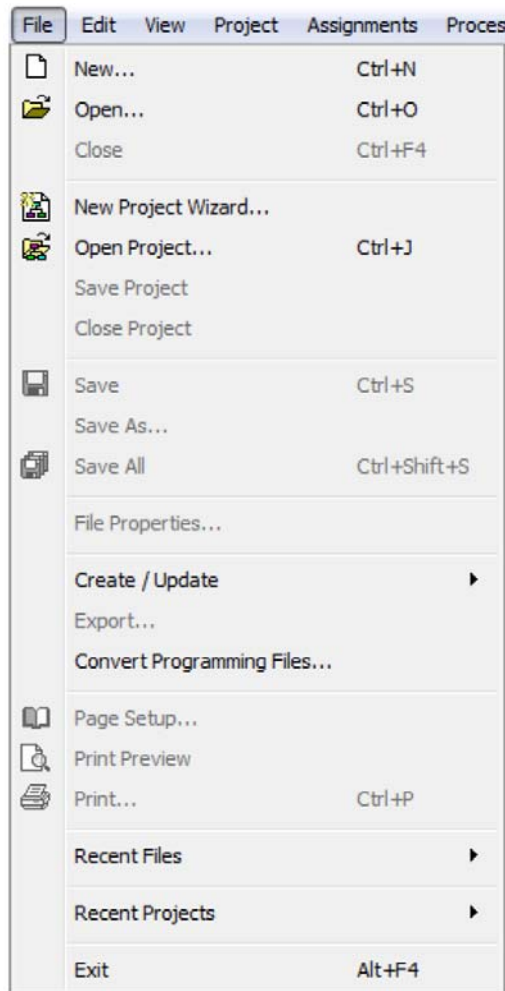


© 2010 Altera Corporation

ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS & STRATIX are Reg. U.S. Pat. & Tm. Off. and Altera marks in and outside the U.S.



# Step 2: Create a New Project



- Click **File** Menu
- Select **New Project Wizard**
- This will open a new window where project information can be specified

# Project Name and Directory

**New Project Wizard**

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

What is the working directory for this project?

D:\introtutorial

What is the name of this project?

light

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.

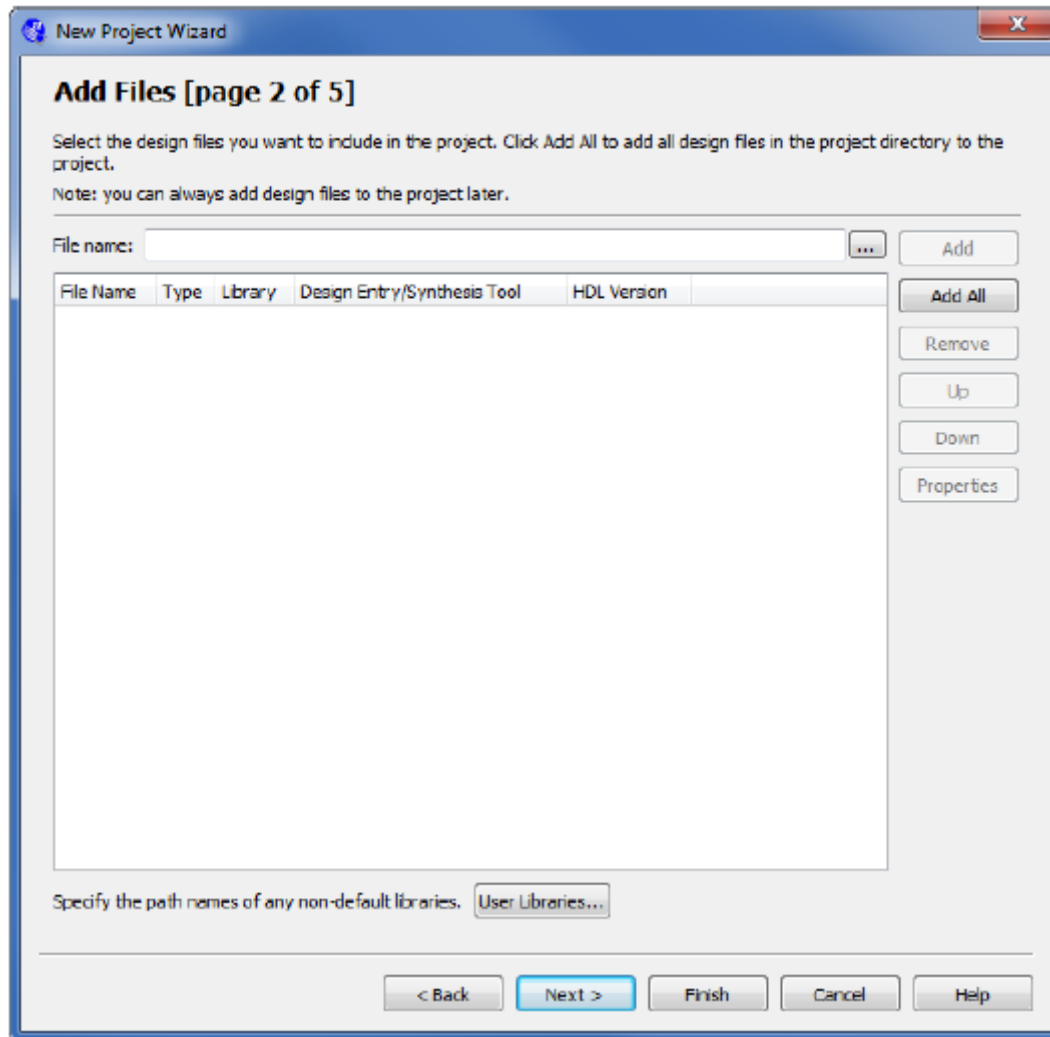
light

Use Existing Project Settings...

< Back Next > Finish Cancel Help

The project must have a name, which is usually the same as the top-level design entity that will be included in the project.

# Add Source Files to Project



The wizard makes it easy to specify which existing files (if any) should be included in the project. Assuming that we do not have any existing files, click Next



# Specify FPGA Device

## ■ Select the FPGA device on the board

- Cyclone III Family
  - For DE0 – EP3C16F484C6
- Cyclone II Family
  - For DE1 – EP2C20F484C7
  - For DE2 – EP2C35F672C6

Board	Device Name
DE0	Cyclone III EP3C16F484C6
DE0-Nano	Cyclone IVE EP4CE22F17C6
DE1	Cyclone II EP2C20F484C7
DE2	Cyclone II EP2C35F672C6
DE2-70	Cyclone II EP2C70F896C6
DE2-115	Cyclone IVE EP4CE115F29C7

**New Project Wizard**  
Family & Device Settings [page 3 of 5]

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

Device family  
Family: Cyclone II  
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  
 Name filter:   
☒ Show advanced devices ☐ HardCopy compatible only

Available devices:

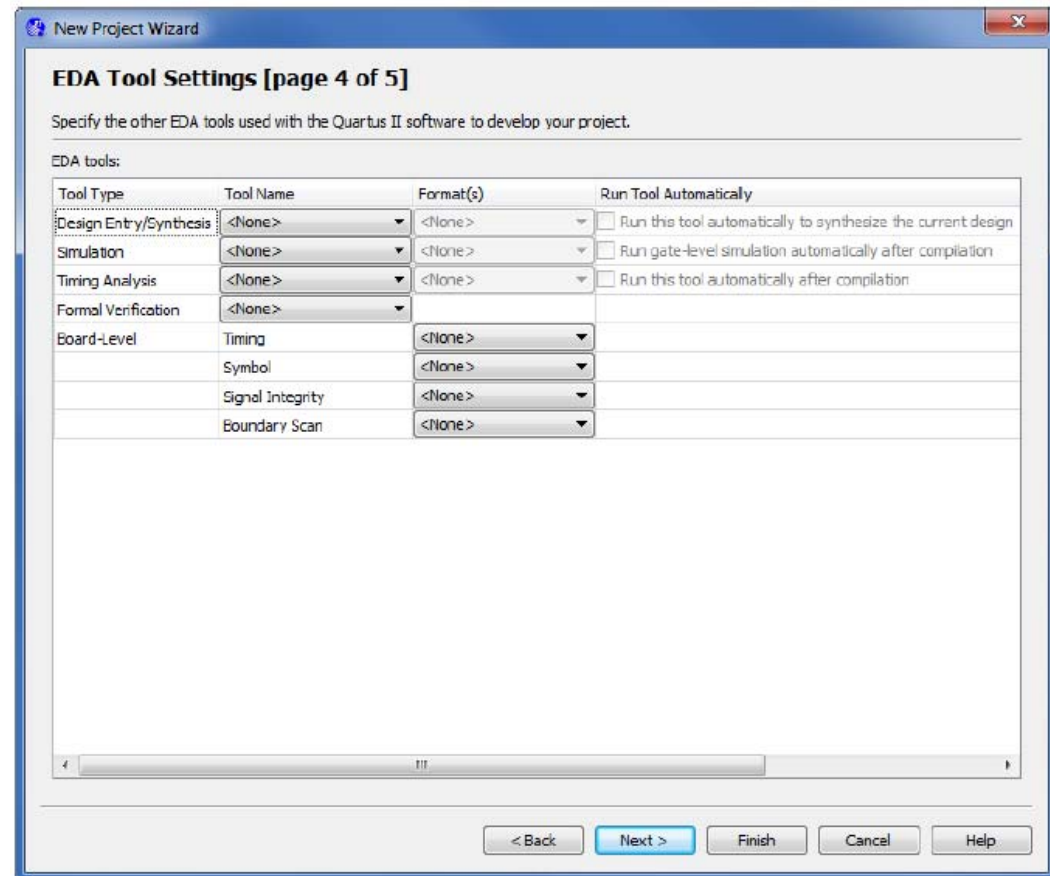
Name	Core Voltage	LEs	User I/Os	Memory Bits	Embedded multiplier 9
EP2C35F484C6	1.2V	33216	322	483840	70
EP2C35F484C7	1.2V	33216	322	483840	70
EP2C35F484C8	1.2V	33216	322	483840	70
EP2C35F484I8	1.2V	33216	322	483840	70
EP2C35F672C6	1.2V	33216	475	483840	70
EP2C35F672C7	1.2V	33216	475	483840	70
EP2C35F672C8	1.2V	33216	475	483840	70

Companion device  
HardCopy:   
☐ Limit DSP & RAM to HardCopy device resources

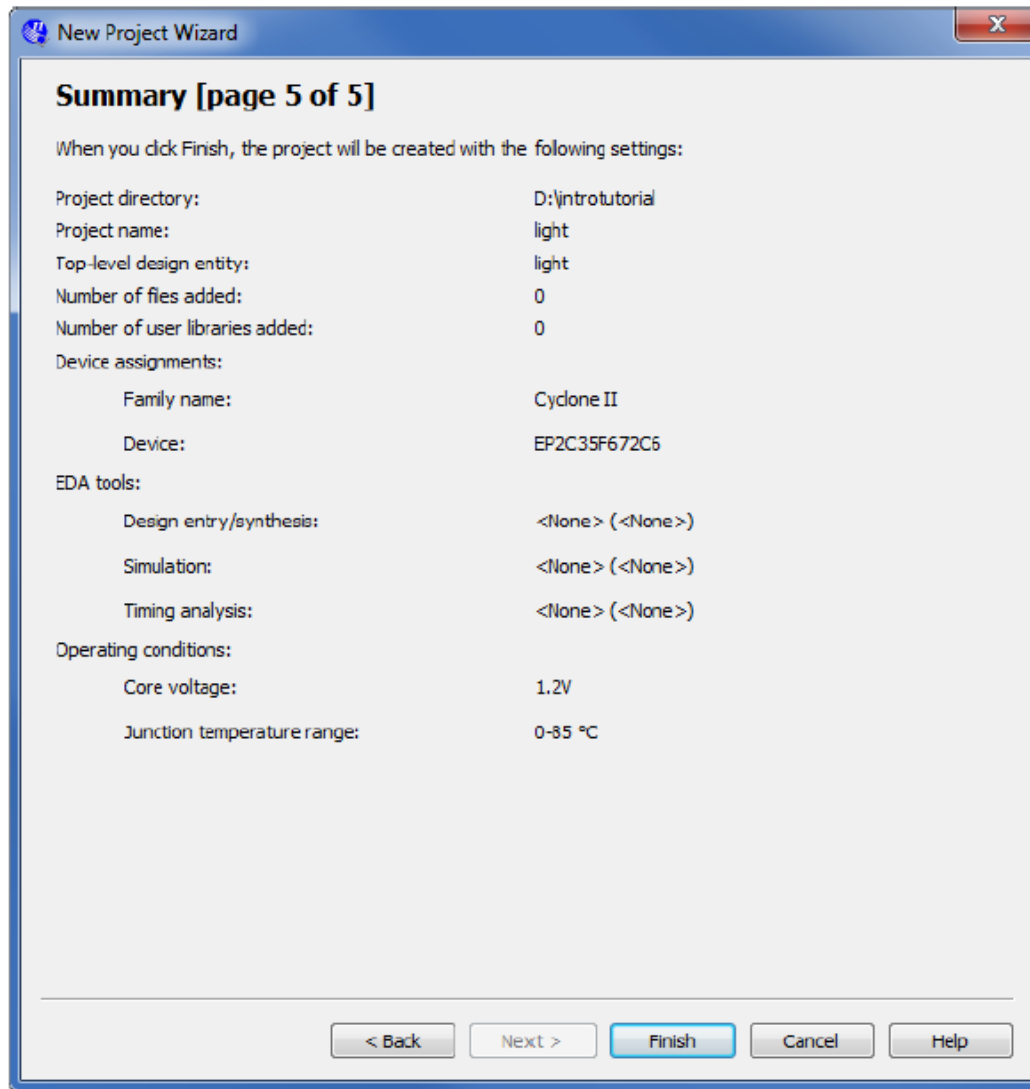
< Back Next > Finish Cancel Help

# Additional EDA Tools

- Specify Tools, in addition to Quartus II, that you will use
- These are unnecessary for small student designs
  - Leave all entries as <None>
  - Press Next

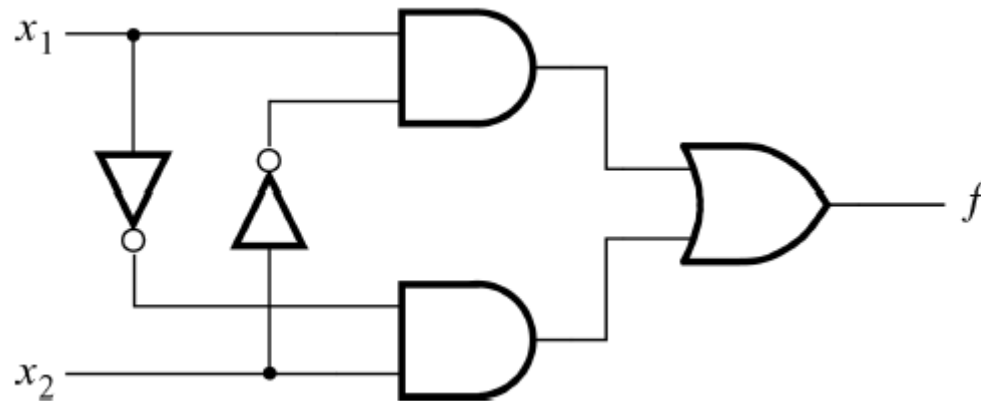


# New Project Summary



- A summary of the chosen settings appears in the screen shown in Figure.
- Press **Finish**, which returns to the main Quartus II window, but with *light* specified as the new project, in the display title bar

# Simple Project



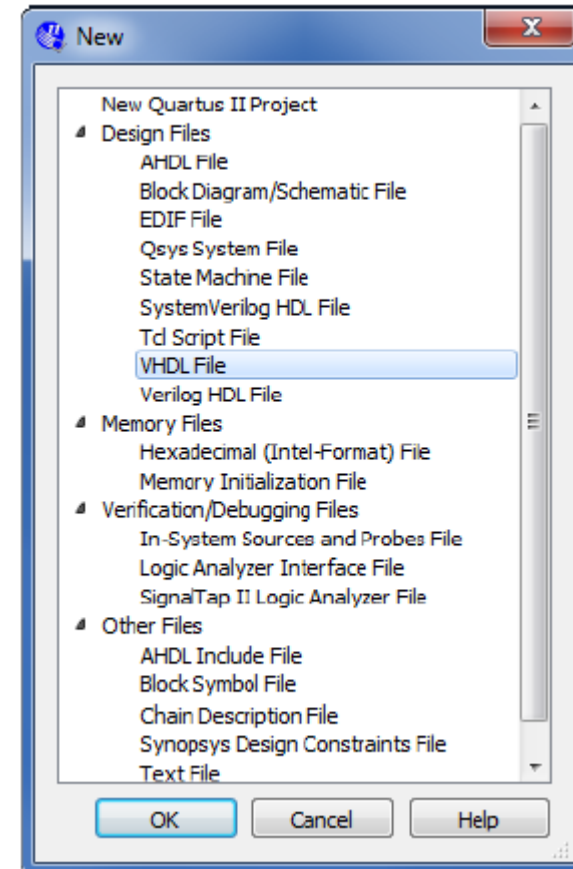
$x_1$	$x_2$	$f$
0	0	0
0	1	1
1	0	1
1	1	0

```

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY light IS
    PORT ( x1, x2 : IN    STD_LOGIC ;
          f      : OUT   STD_LOGIC ) ;
END light ;
ARCHITECTURE LogicFunction OF light IS
BEGIN
    f <= (x1 AND NOT x2) OR (NOT x1 AND x2);
END LogicFunction ;
    
```

## Step 3a: Create Source File

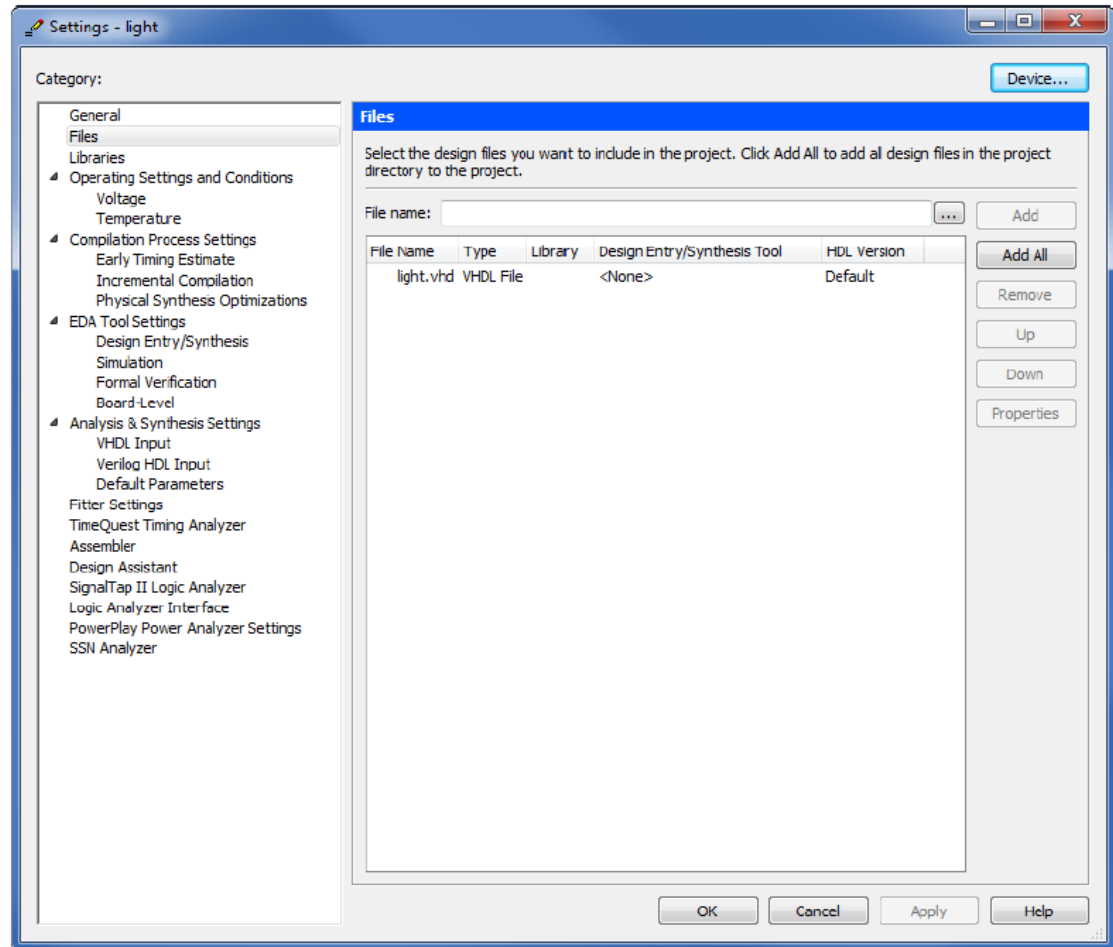
- Select File > New to get the window in Figure, choose VHDL File, and click OK. This opens the Text Editor window.
- Specify a name for the file that will be created and select File > Save to open a pop-up box and in the box labeled Save as type choose VHDL File



# Step 3b: Add Source File

Select:

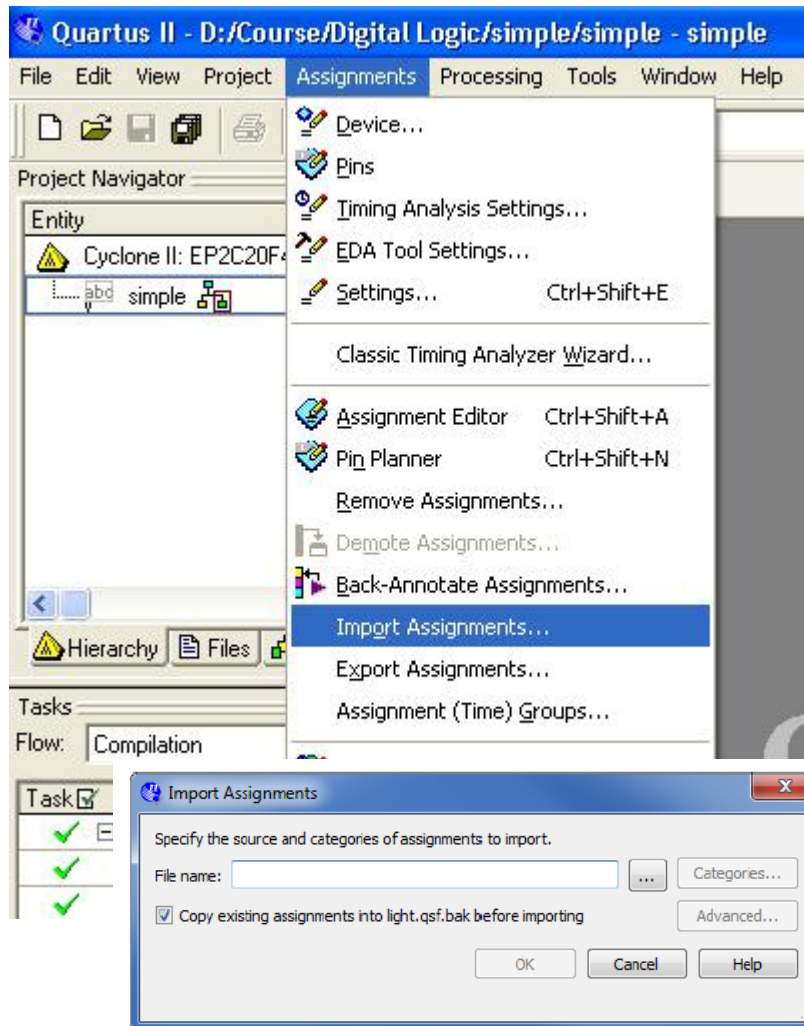
- Assignments > Settings and looks for File item or
- Project > Add/Remove Files in Project



## Step 4: Assign Pins to connect switches/lights to inputs and outputs of your circuit

- Click **Assignments**, then **Import Assignments...**
- Import file
  - DE0\_pin\_assignments.qsf
- Imports locations for predefined port names, such as SW, LEDG, KEY, and others
  - Can be done manually for custom port names

Component	DE0
$SW_0$	PIN_J6
$SW_1$	PIN_H5
$LEDG_0$	PIN_J1





## Step 4: Assign Pins to connect switches/lights to inputs and outputs of your circuit

# Pin & Location Assignments

# =====

```
set_location_assignment PIN_B1 -to LEDG[9]
set_location_assignment PIN_B2 -to LEDG[8]
set_location_assignment PIN_C2 -to LEDG[7]
set_location_assignment PIN_C1 -to LEDG[6]
set_location_assignment PIN_E1 -to LEDG[5]
set_location_assignment PIN_F2 -to LEDG[4]
set_location_assignment PIN_H1 -to LEDG[3]
set_location_assignment PIN_J3 -to LEDG[2]
set_location_assignment PIN_J2 -to LEDG[1]
set_location_assignment PIN_J1 -to LEDG[0]
set_location_assignment PIN_D2 -to SW[9]
set_location_assignment PIN_E4 -to SW[8]
set_location_assignment PIN_E3 -to SW[7]
set_location_assignment PIN_H7 -to SW[6]
set_location_assignment PIN_J7 -to SW[5]
set_location_assignment PIN_G5 -to SW[4]
set_location_assignment PIN_G4 -to SW[3]
set_location_assignment PIN_H6 -to SW[2]
set_location_assignment PIN_H5 -to SW[1]
set_location_assignment PIN_J6 -to SW[0]
set_location_assignment PIN_F1 -to KEY[2]
set_location_assignment PIN_G3 -to KEY[1]
set_location_assignment PIN_H2 -to KEY[0]
```

# Pin & Location Assignments

# =====

```
set_location_assignment PIN_E11 -to HEX0[0]
set_location_assignment PIN_F11 -to HEX0[1]
set_location_assignment PIN_H12 -to HEX0[2]
set_location_assignment PIN_H13 -to HEX0[3]
set_location_assignment PIN_G12 -to HEX0[4]
set_location_assignment PIN_F12 -to HEX0[5]
set_location_assignment PIN_F13 -to HEX0[6]
set_location_assignment PIN_D13 -to HEX0[7]
set_location_assignment PIN_A15 -to HEX1[6]
set_location_assignment PIN_E14 -to HEX1[5]
set_location_assignment PIN_B14 -to HEX1[4]
set_location_assignment PIN_A14 -to HEX1[3]
set_location_assignment PIN_C13 -to HEX1[2]
set_location_assignment PIN_B13 -to HEX1[1]
set_location_assignment PIN_A13 -to HEX1[0]
set_location_assignment PIN_B15 -to HEX1[7]
```

# Pin & Location Assignments

# =====


```
set_location_assignment PIN_F14 -to HEX2[6]
set_location_assignment PIN_B17 -to HEX2[5]
set_location_assignment PIN_A17 -to HEX2[4]
set_location_assignment PIN_E15 -to HEX2[3]
set_location_assignment PIN_B16 -to HEX2[2]
set_location_assignment PIN_A16 -to HEX2[1]
set_location_assignment PIN_D15 -to HEX2[0]
set_location_assignment PIN_A18 -to HEX2[7]
set_location_assignment PIN_G15 -to HEX3[6]
set_location_assignment PIN_D19 -to HEX3[5]
set_location_assignment PIN_C19 -to HEX3[4]
set_location_assignment PIN_B19 -to HEX3[3]
set_location_assignment PIN_A19 -to HEX3[2]
set_location_assignment PIN_F15 -to HEX3[1]
set_location_assignment PIN_B18 -to HEX3[0]
set_location_assignment PIN_G16 -to HEX3[7]
```

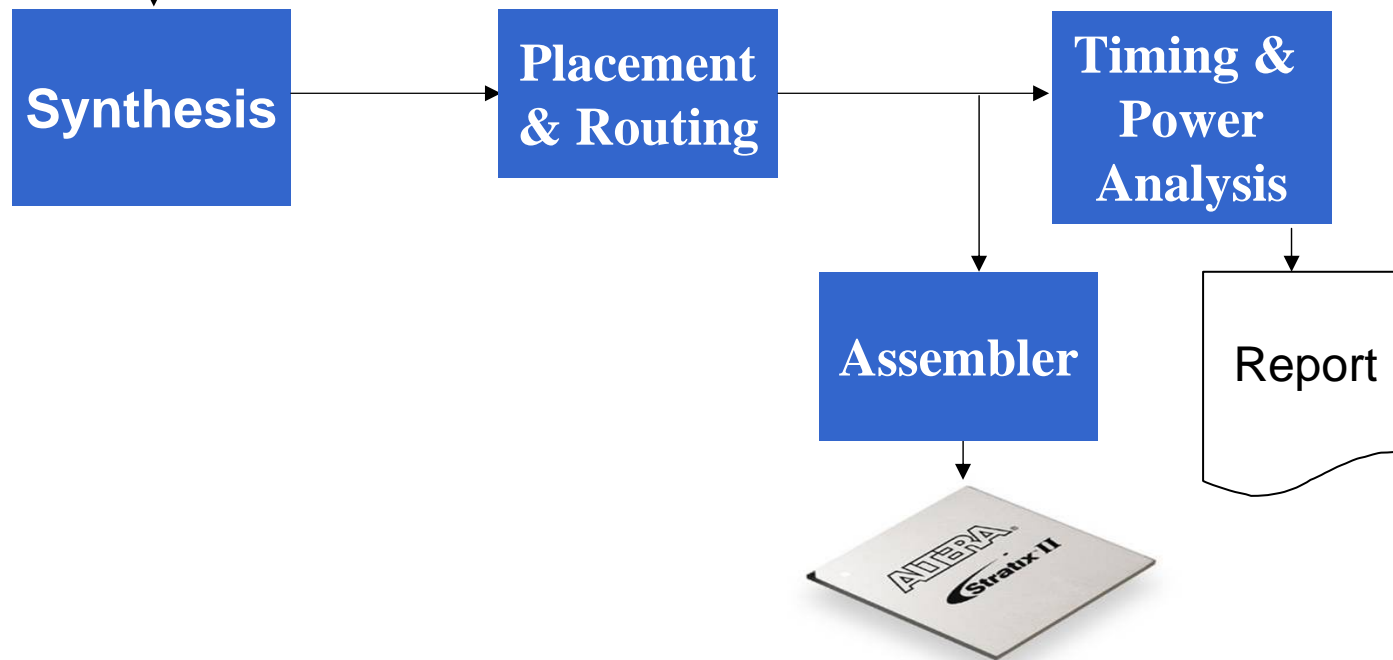
SW: IN STD\_LOGIC\_VECTOR(N DOWNT0 0);  
LEDG: OUT STD\_LOGIC\_VECTOR(N DOWNT0 0);  
HEX0: OUT STD\_LOGIC\_VECTOR(0 TO N)

x1, x2, f ??????

# Step 5: Compile Design



- Run the Compiler by selecting Processing > Start Compilation, or by clicking on the toolbar icon that looks like a purple triangle. 
- Successful (or unsuccessful) compilation is indicated in a pop-up box; acknowledge it by clicking OK, which leads to the report shown in next slide



# Step 6: Examine Compilation Report

The screenshot displays the Quartus II 64-Bit IDE interface. The main window shows the 'Compilation Report' for the project 'light'. The report is titled 'Flow Summary' and provides a detailed overview of the compilation process.

**Flow Summary**

Flow Status	Successful - Tue May 08 12:07:57 2012
Quartus II 64-Bit Version	12.0 Build 173 05/02/2012 SJ Full Version
Revision Name	light
Top-level Entity Name	light
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	1 / 33,216 (< 1 %)
Total combinational functions	1 / 33,216 (< 1 %)
Dedicated logic registers	0 / 33,216 (0 %)
Total registers	0
Total pins	3 / 475 (< 1 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

The 'Messages' window at the bottom shows the following information:

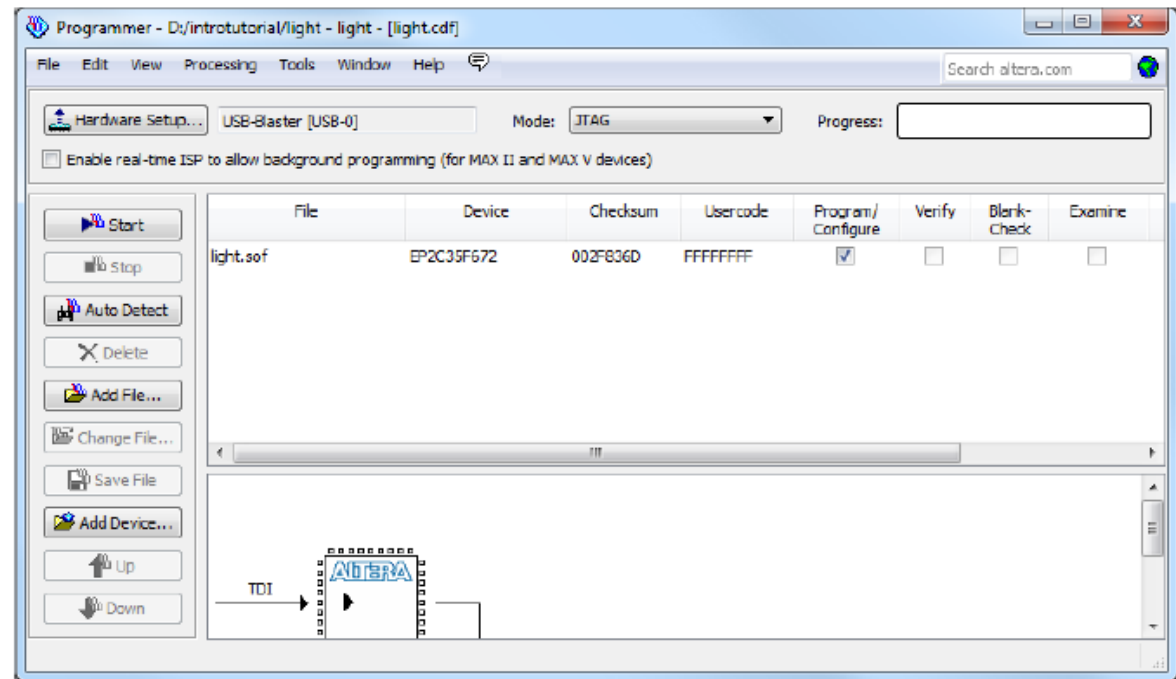
- Info (332102): Design is not fully constrained for hold requirements
- Info: Quartus II 64-Bit TimeQuest Timing Analyzer was successful. 0 errors, 3 warnings
- Info (293026): Skipped module PowerPlay Power Analyzer due to the assignment FLOW\_ENABLE\_POWER\_ANALYZER
- Info (293000): Quartus II Full Compilation was successful. 0 errors, 8 warnings

The status bar at the bottom indicates the compilation is complete with 100% progress and a time of 00:00:53.

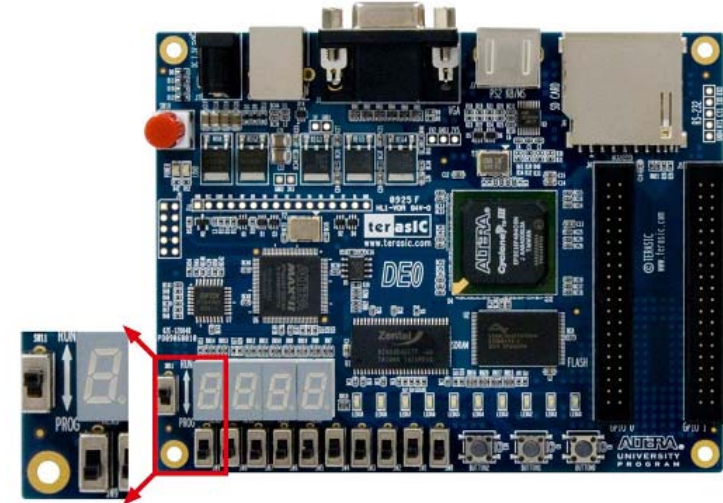
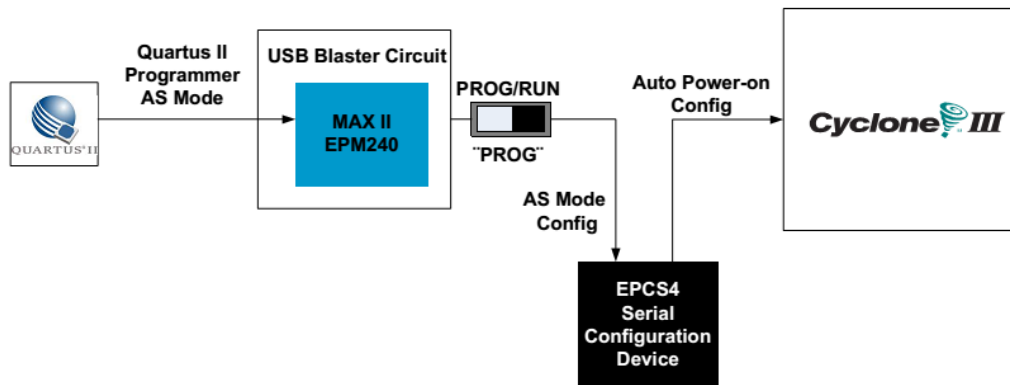
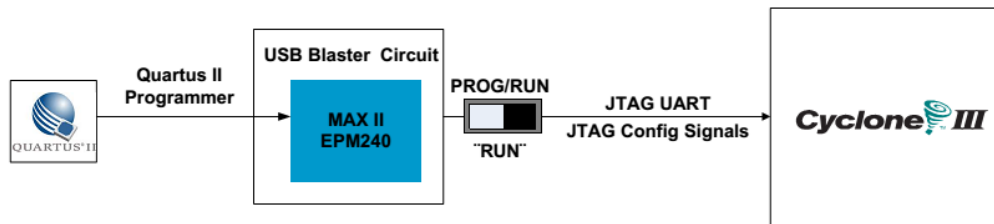
# Step 7a: Program the DE0 Board

- **FPGA JTAG mode:** it will retain its configuration as long as the power remains turned on.
- Active Serial (AS) mode: a configuration device that includes some flash memory is used to store the configuration data
- FPGA JTAG mode: flip the RUN/PROG switch into the RUN position and select Tools > Programmer

- Observe that the configuration file light.sof is listed in the programmer window. If the file is not already listed, then click Add File and select it



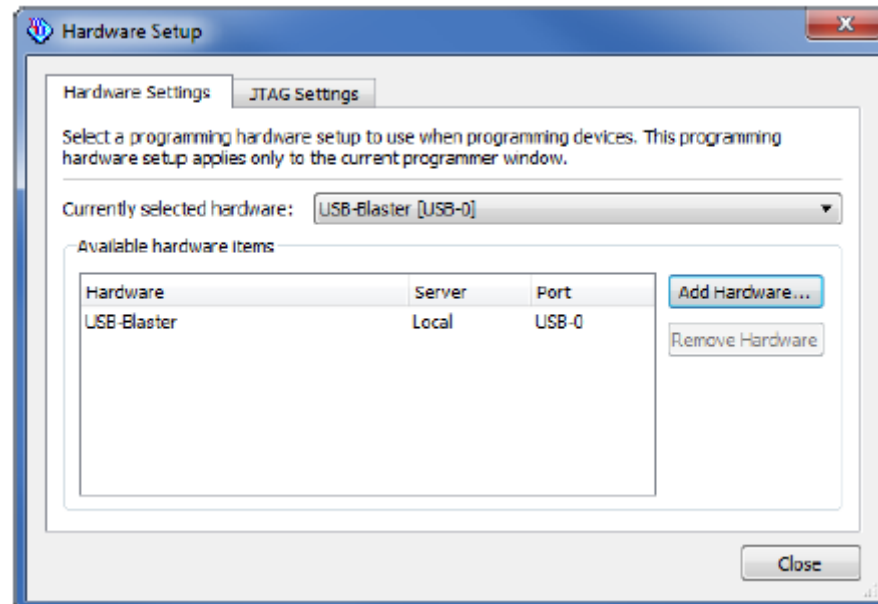
# Step 7a: Program the DE0 Board



The RUN/PROG switch (SW2) is set in JTAG mode

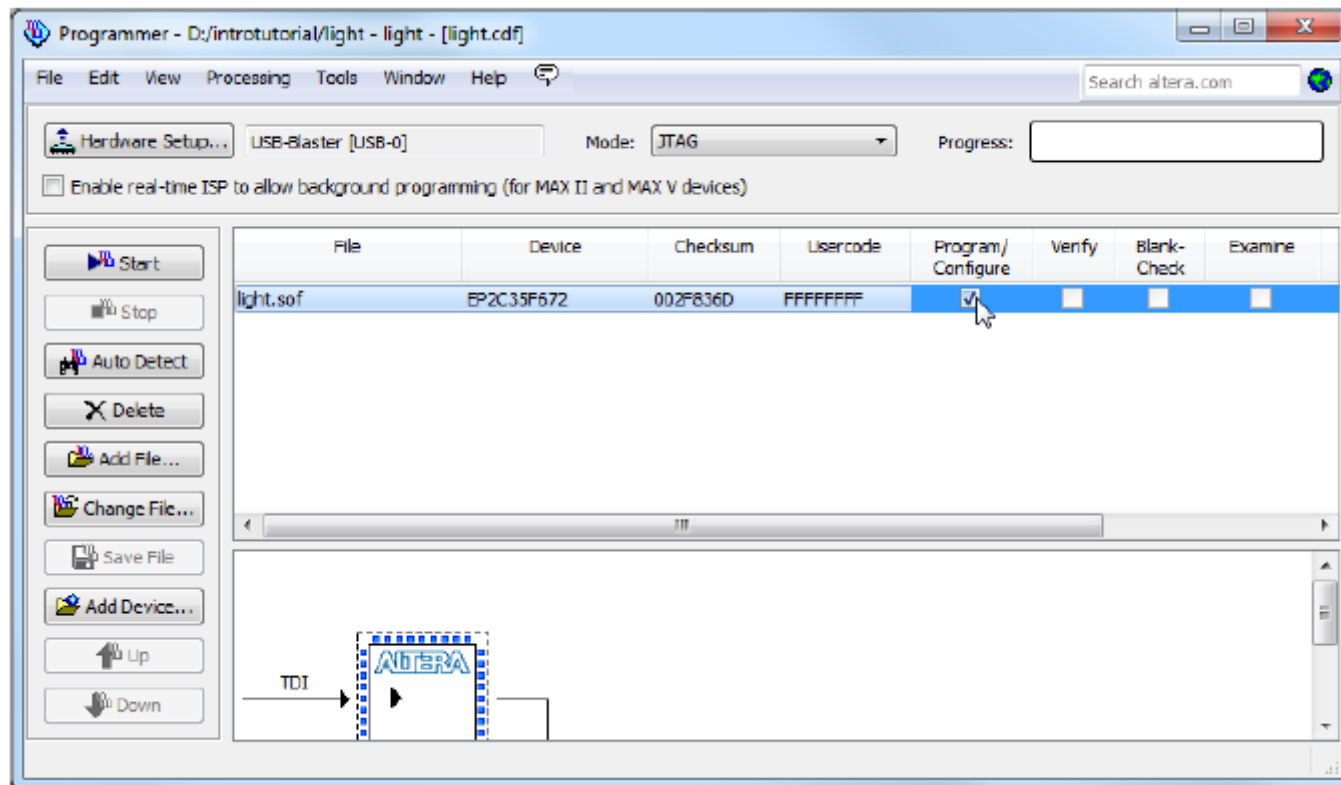
## Step 7b: Program the DE0 Board

- Here it is also necessary to specify the programming hardware and the mode that should be used. If not already chosen by default, select JTAG in the Mode box.
- Also, if the USB-Blaster is not chosen by default, press the Hardware Setup... button and select the USB-Blaster in the window that pops up



# Step 7c: Program the DE0 Board

- Click on the Program/Configure check box
- Press Start in the Programmer window; an LED on the board will light up when the configuration data has been downloaded successfully



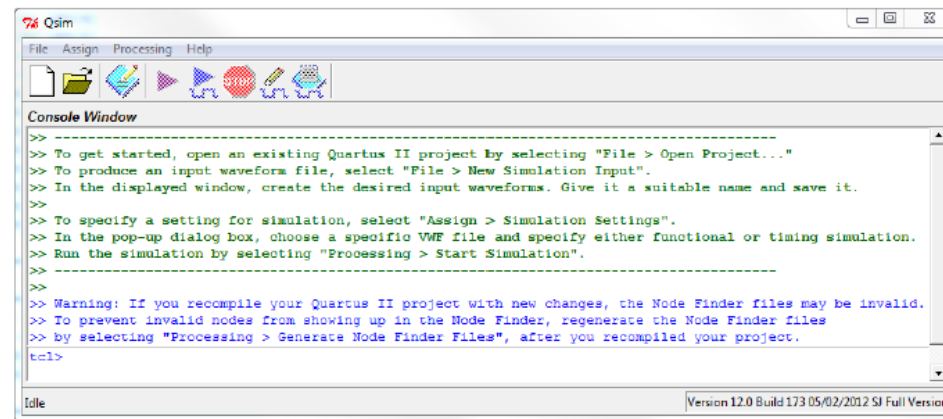


## Step 8: See your design work on the board

- Having downloaded the configuration data into the FPGA device, you can now test the implemented circuit.
- Flip the RUN/PROG switch to RUN position.
- Try all four valuations of the input variables x1 and x2, by setting the corresponding states of the switches SW1 and SW0.

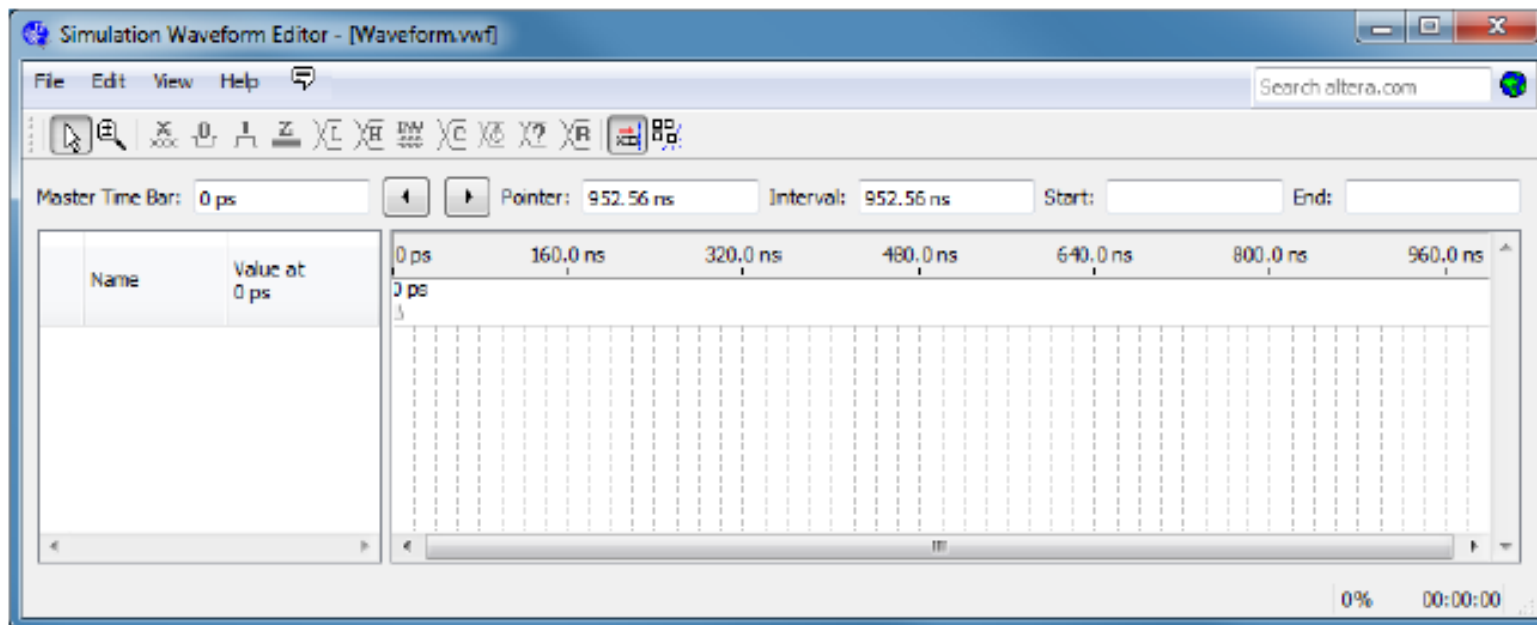
## Step 9a: Simulate your design

- Select Start > All Programs > Altera > University Program > Simulation Tools > QSim to open the Qsim tools.
- Select File > Open Project to display a popup window in which you can browse your directories and choose a project file (.qpf file).
- Select the project you wish to simulate and click OK.
- Generate the node finder files by selecting Processing > Generate Node Finder Files
- From QSim, open the Waveform Editor window by selecting File > New Simulation Input File.



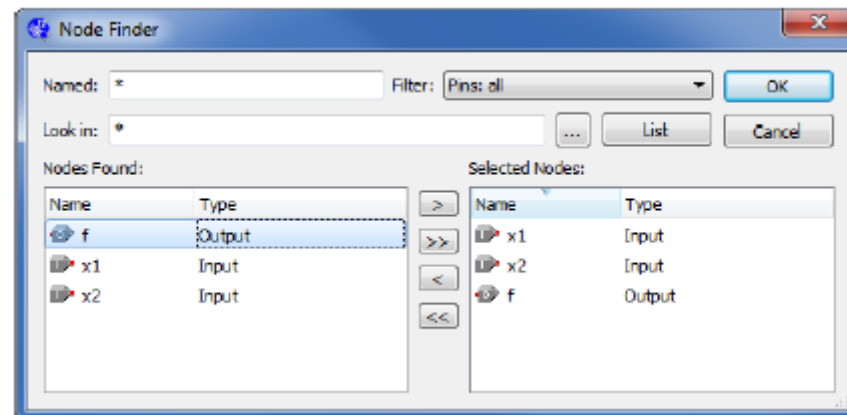
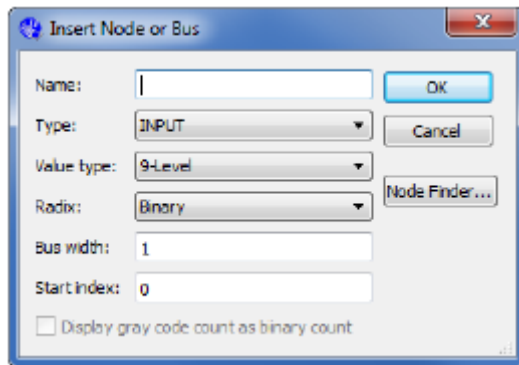
## Step 9b: Simulate your design

- Save the file under the name light.vwf; note that this changes the name in the displayed window.
- Set the desired simulation to run from 0 to 200 ns by selecting Edit > Set End Time and entering 200 ns in the dialog box that pops up.
- Select View > Fit in Window



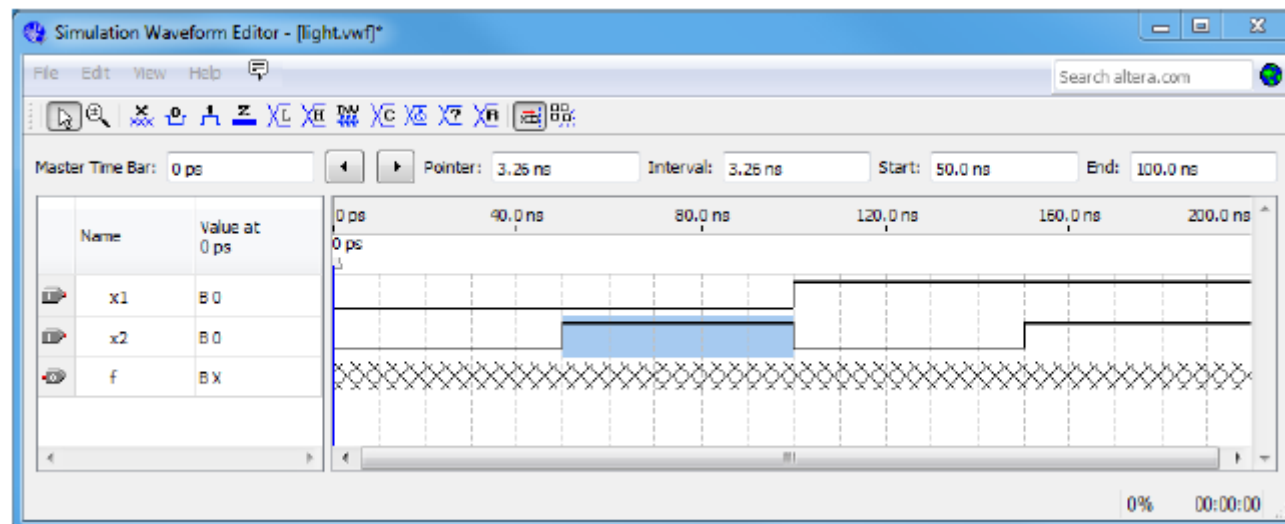
## Step 9c: Simulate your design

- Include the input and output nodes of the circuit to be simulated.
- Click Edit > Insert > Insert Node or Bus: it is possible to type the name of a signal (pin) into the Name box, or use the Node Finder to search your project for the signals.



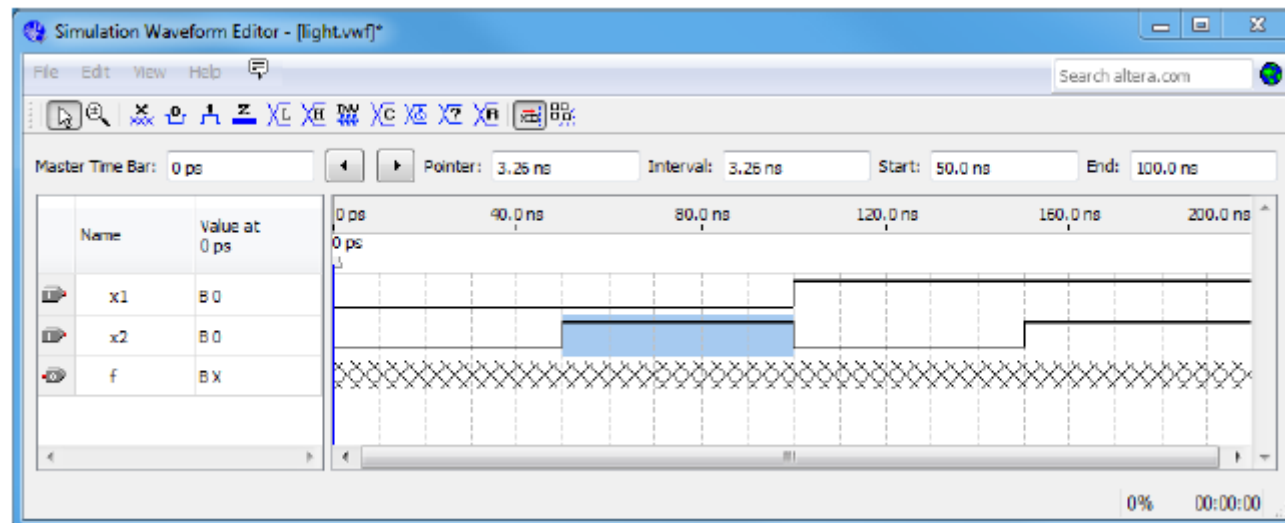
## Step 9d: Simulate your design

- We will now specify the logic values to be used for the input signals x1 and x2 during simulation
- Click on the waveform for the x1 node and using the Edit > Value command, or via the toolbar, draw the desired waveforms.
- Commands are available for setting a selected signal to 0, 1, unknown (X), high impedance (Z), weak low (L), weak high (H), a count value (C), an arbitrary value, a random value (R), inverting its existing value (INV), or defining a clock waveform.



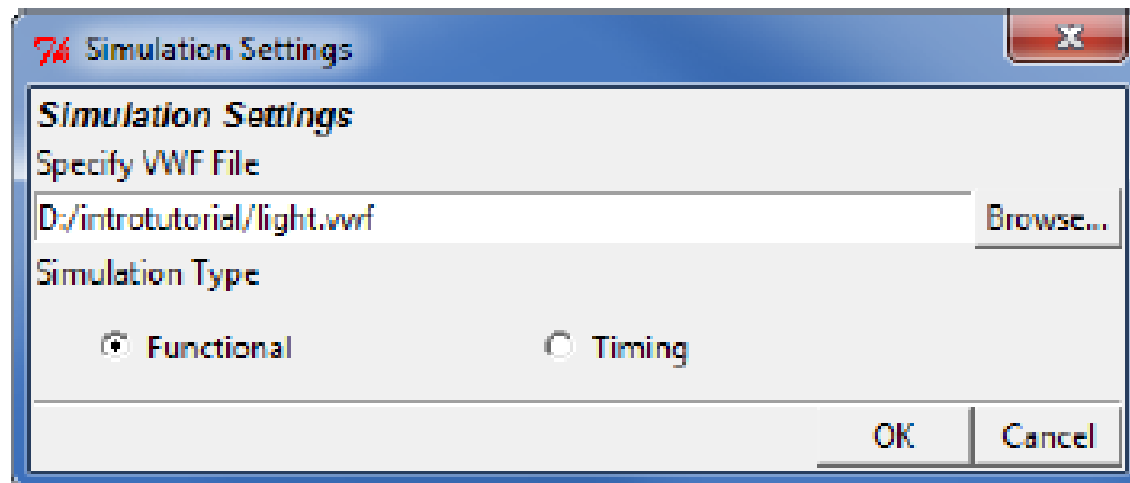
## Step 9e: Simulate your design

- E.g. set x2 to 1 in the time interval 50 to 100 ns . Do this by pressing the mouse at the start of the interval and dragging it to its end, which highlights the selected interval, and choosing the logic value 1 in the toolbar.
- Observe that the output f is displayed as having an unknown value at this time, which is indicated by a hashed pattern; its value will be determined during simulation.
- Save the file.



## Step 9f: Simulate your design

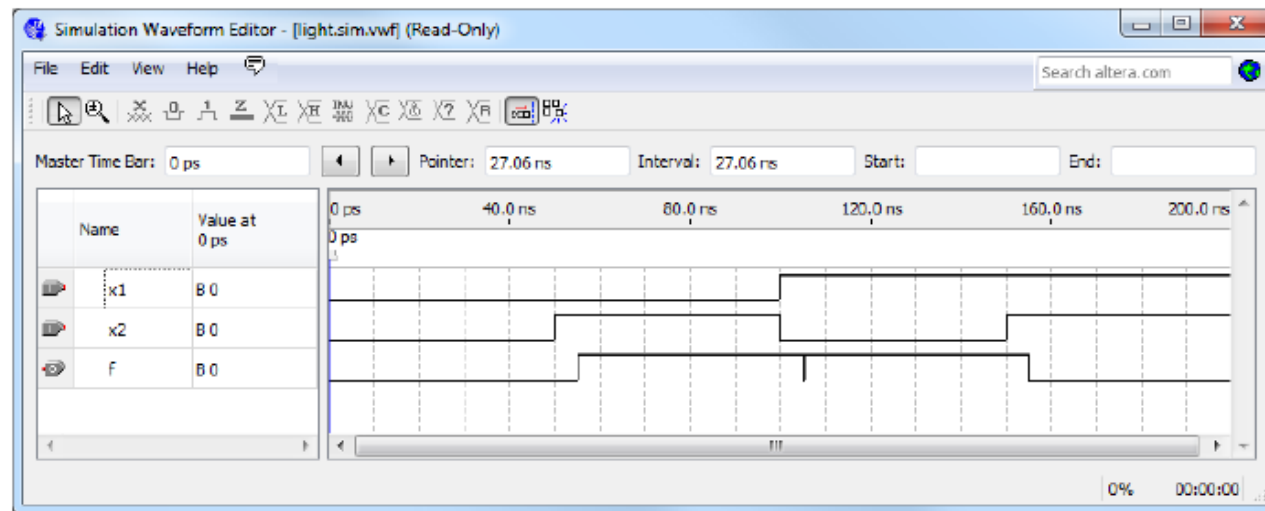
- To perform the functional simulation, return to the QSim Window and select Assign > Simulation Settings...
- Click the Browse button and select the light.vwf file you created.
- Choose Functional as the simulation type, and click OK.
- Select Processing > Generate Simulation Netlist.
- A simulation run is started by Processing > Start Simulation,





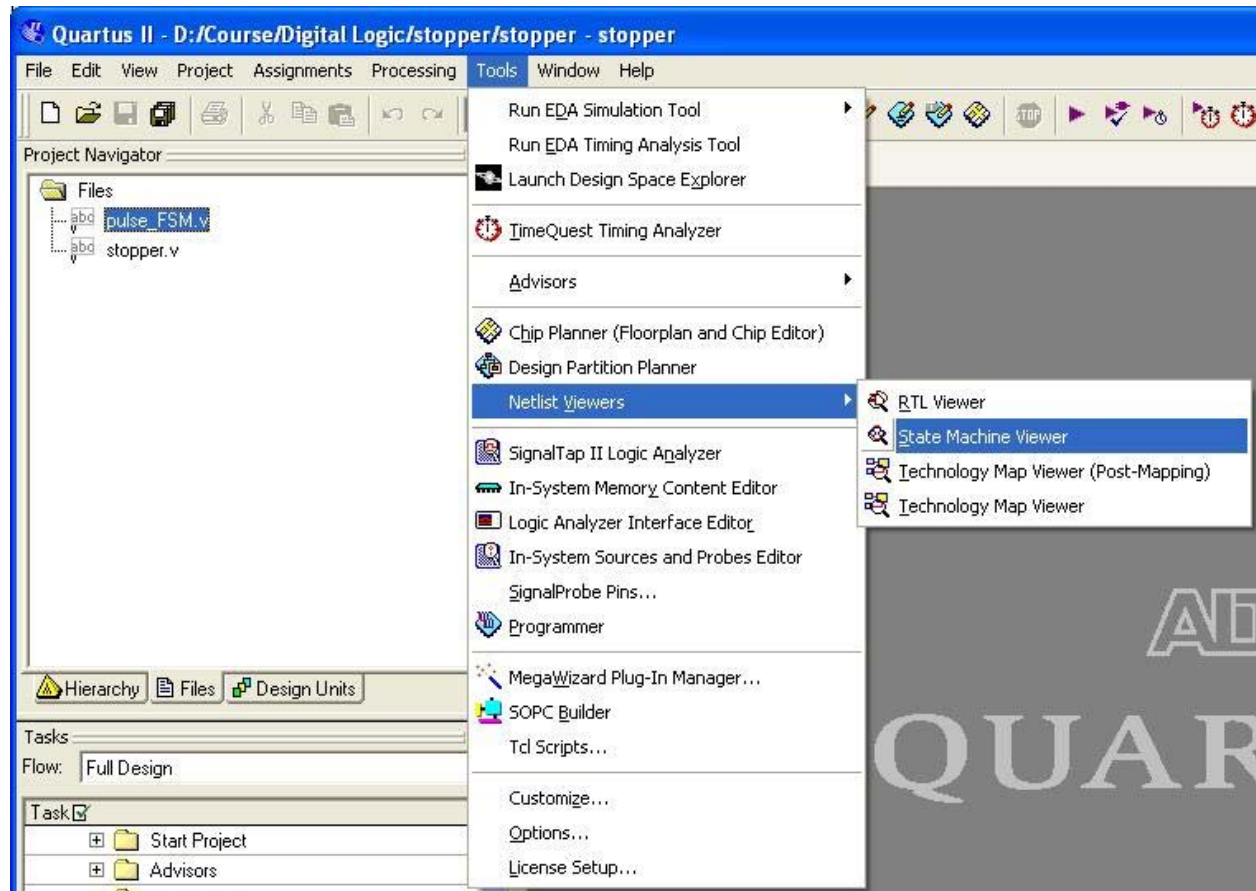
## Step 9g: Simulate your design

- To perform timing simulation choose Select Assign > Simulation Settings...
- Choose Timing as the simulation type, and click OK.
- Run the simulator
- Observe that there is a delay in producing a change in the signal f from the time when the input signals, x1 and x2, change their values.
- This delay is due to the propagation delays in the logic element and the wires in the FPGA device



# FSM Viewer

- Open the FSM Viewer
  - Click **Tools**
  - Expand **Netlist Viewers**
  - Click **State Machine Viewer**



# Examine State Machine

Quartus II - D:/Course/Digital Logic/stopper/stopper - stopper - [State Machine Viewer | pulse\_FSM:FSM | y\_Q]

File Edit View Project Assignments Processing Tools Window Help

Project Navigator

Files

- abd pulse\_FSM.v
- abd stopper.v

stopper

stopper.v pulse\_FSM.v Compilation Report - Flow Sum... State Machine Viewer | ...

State Machine: |stopper|pulse\_FSM:FSM|y\_Q

```

graph LR
    S0((S0)) -- "i_pulse" --> S0
    S0 -- "!i_pulse" --> S0
    S0 -- "i_pulse" --> S1((S1))
    S1 -- "Resetn" --> S0
    S1 -- "Resetn" --> S2((S2))
    S2 -- "i_pulse" --> S0
    S2 -- "i_pulse" --> S2
  
```

	Source State	Destination State	Condition
1	S0	S0	(i_pulse) + (!i_pulse).(!Resetn)
2	S0	S1	(i_pulse).(Resetn)
3	S1	S0	(!Resetn)
4	S1	S2	(Resetn)
5	S2	S0	(i_pulse) + (!i_pulse).(!Resetn)
6	S2	S2	(i_pulse).(Resetn)

Task

Flow: Full Design

Task

- Start Project
- Advisors
- Create Design
- Assign Constraints
- Compile Design
- Analysis & Synthesis
- Filter (Place & Route)
- Assembler (Generate programming fi
- Classic Timing Analysis
- EDA Netlist Writer
- Program Device (Open Programmer)
- Verify Design
- Simulate Design

Transitions / Encoding /

Message: 0 of 614 Location: Locate

For Help, press F1

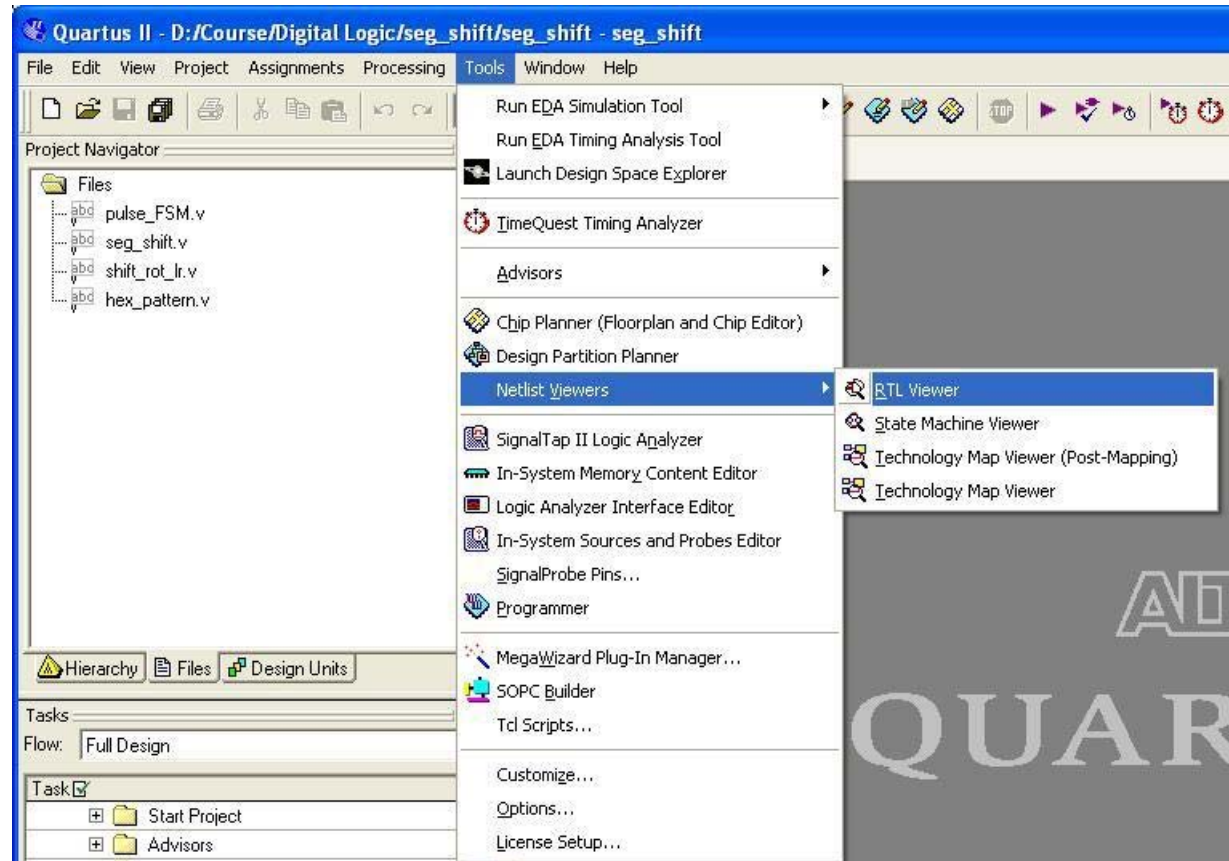
Idle NUM

© 2010 Altera Corporation

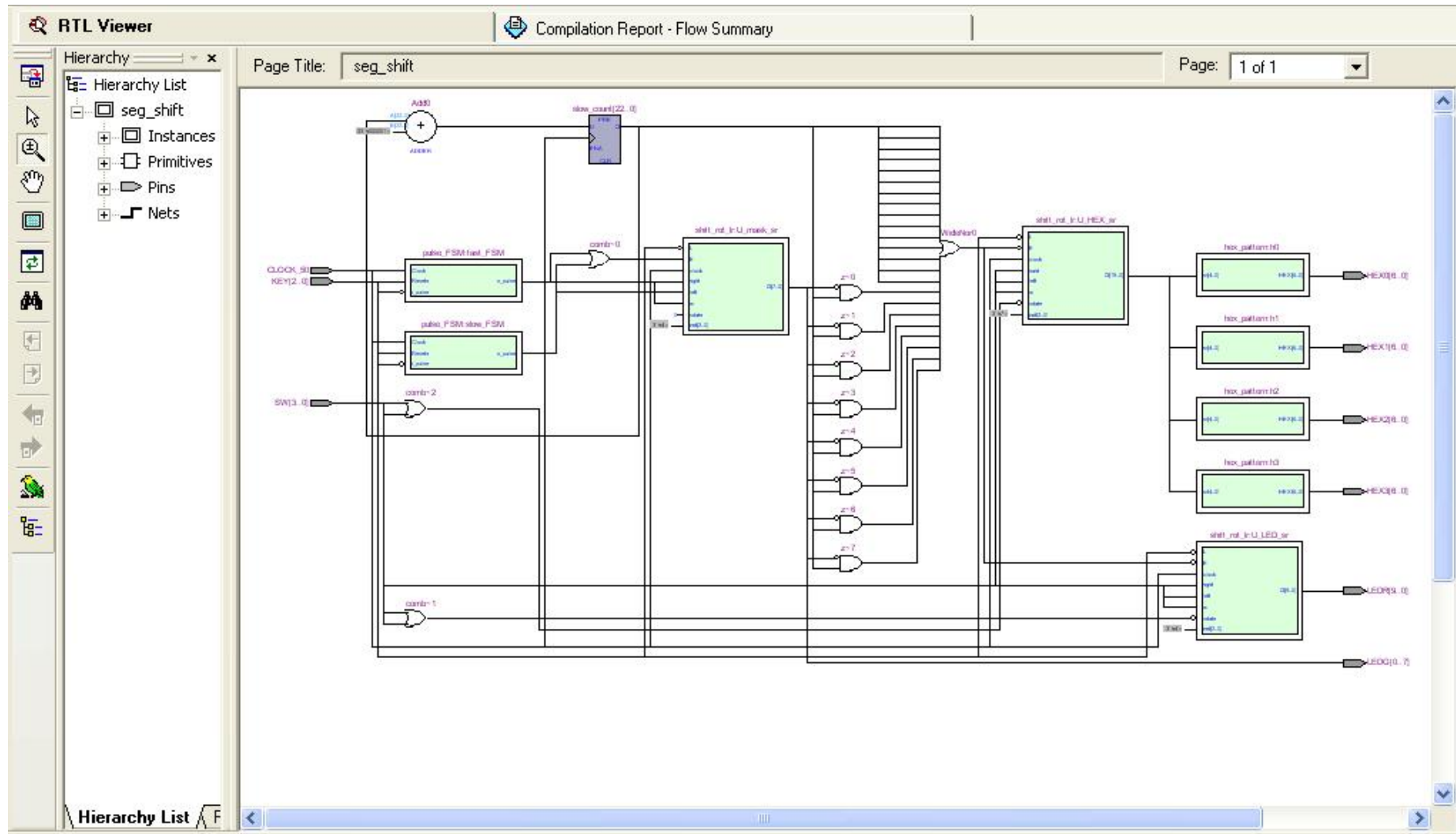
ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS & STRATIX are Reg. U.S. Pat. & Tm. Off. and Altera marks in and outside the U.S.

# See the Circuit in RTL Viewer

- Start the **RTL Viewer**
  - Click **Tools**
  - Expand the **Netlist Viewers** list
  - Click **RTL Viewer**



# Examine the Circuit

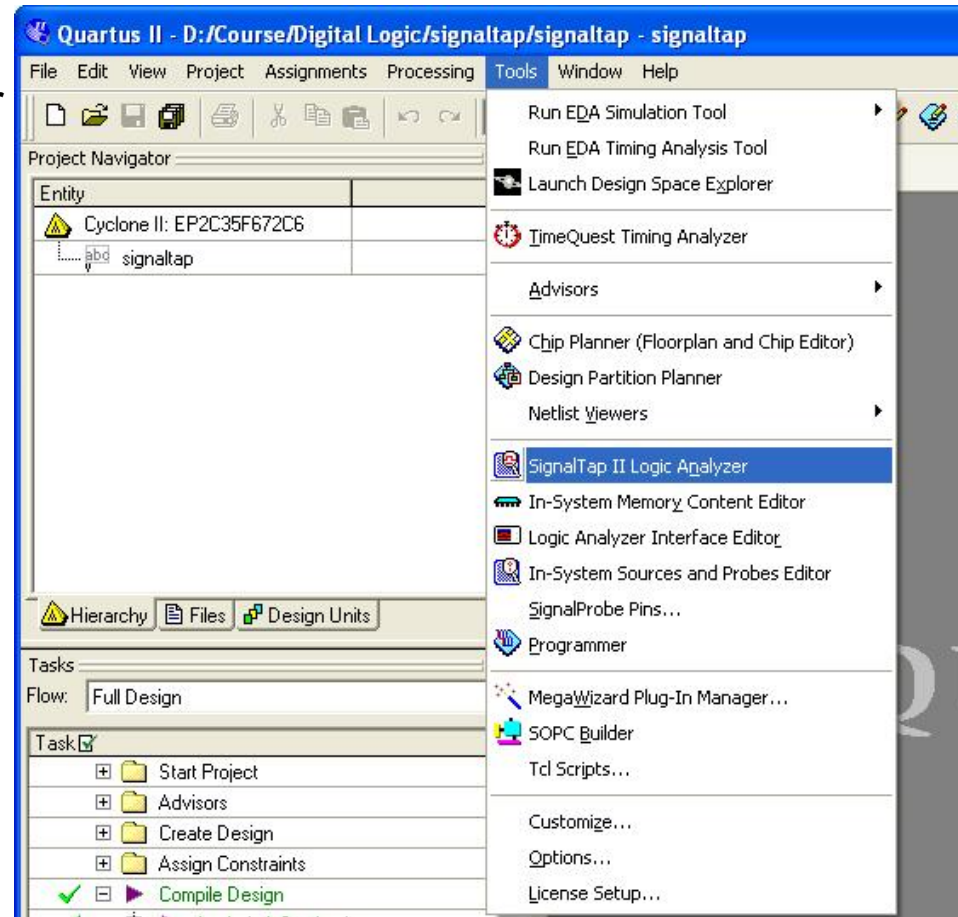


© 2010 Altera Corporation

ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS & STRATIX are Reg. U.S. Pat. & Tm. Off. and Altera marks in and outside the U.S.

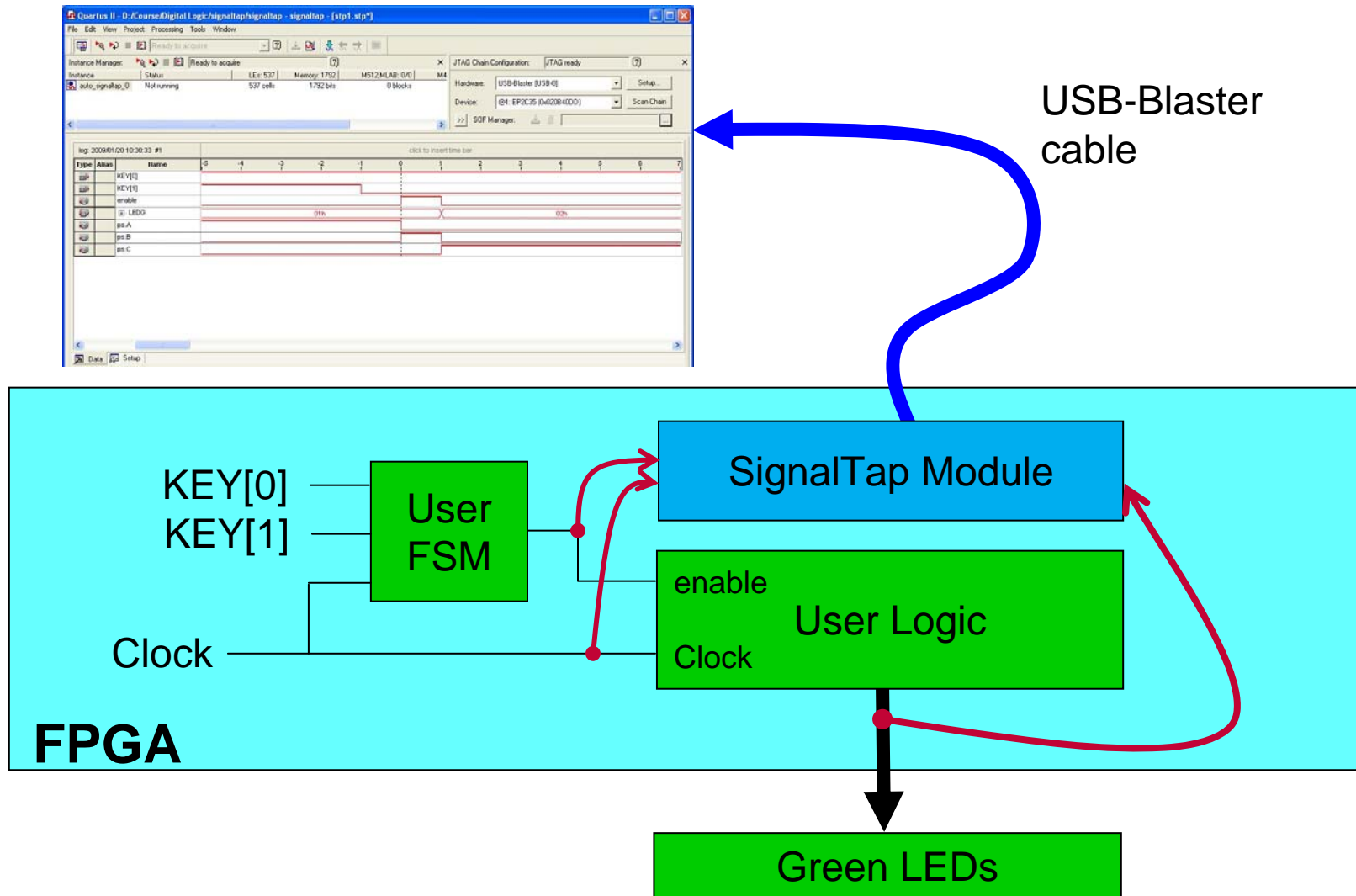
# SignalTap II Embedded Logic Analyzer

- A soft logic analyzer
  - Instantiate as a module in your design
- Connects to the board on which a design is running
- Collects data when a trigger event occurs
- Displays data on your computer
- How does it work?





# SignalTap II Operation



# Altera U. P. Teaching Materials for Digital Logic

- Available for download: <http://university.altera.com>
- Tutorials
  - Teach students how to use Quartus II, etc.
- Lab Exercises
  - 10 digital logic exercises
  - Complete with documentation and solutions (both in Verilog and VHDL)
- IP Cores for DE boards
  - VGA, Audio I/O, and others
  - Great for projects



# Altera U.P. Web Site



## University Program

[→Edit Page](#) [→Downloads](#)

[→Educational Materials](#) [→Announcements](#) [→Innovate Contests](#) [→Members](#) [→Support](#)

[Altera Home](#) > [University Program](#)

Altera® University Program provides complete support for introducing students to digital technology. The support includes hardware, software, and teaching materials. Hardware support is in the form of Development and Education boards (DE1, DE2, and DE3), specially designed for use in teaching and research laboratories. Software support consists of the Quartus® II Computer Aided-Design software, Nios II soft processor, and simulation tools. Teaching materials comprise tutorials and ready-to-teach laboratory exercises for use in digital logic and computer organization courses.

The DE boards provide a high-quality, yet low-cost platform. To mitigate the cost of equipping teaching and research laboratories, Altera University Program boards are provided at a price which is at, or below, the manufacturing cost. Altera also offers a hardware donation program for qualified schools. Professors and lecturers can request support under this program through the [Members](#) section of this University Program website.

Also, Altera sponsors design contests for students around the world. These contests have been highly popular and have resulted in many innovative designs.

For more information about the Altera University Program and its offerings, click on the links below or read the [Altera University Program Overview](#) document.

- [Educational Materials](#)
- [University Program Members](#)
- [Innovate Design Contests](#)
- [Support](#)

☒ [Rate This Page](#)

[Products](#) | [End Markets](#) | [Technology](#) | [Training](#) | [Support](#) | [About Altera](#) | [Buy Online](#)  
[Jobs](#) | [Investor Relations](#) | [Contact Us](#) | [Site Map](#) | [Privacy](#) | [Legal Notice](#)

Copyright © 1995-2009 Altera Corporation. All Rights Reserved.



RSS Feeds



Email Updates

© 2010 Altera Corporation

ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS & STRATIX are Reg. U.S. Pat. & Tm. Off. and Altera marks in and outside the U.S.

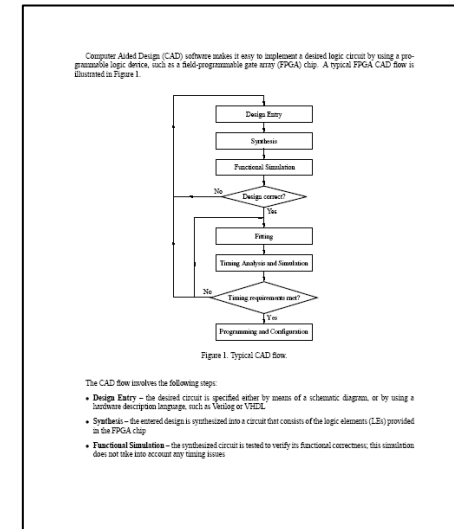


# Tutorials

## ■ Tutorials

- Getting Started with Altera's DE0 Lab Board
- Introduction to Quartus II
  - With Verilog, or VHDL, or Schematic
- Using library modules (LPMs)
  - With Verilog or VHDL
- Quartus II Simulation
  - With Verilog or VHDL
- Timing Considerations
  - With Verilog or VHDL
- Signal Tap II
- Hardware Debugging
- Debugging with ModelSim

## Introduction to Quartus II



Acrobat Document

# Digital Logic Lab Exercises

- Ten lab exercises released
  - From basic logic gates to simple processors:
  - **Lab 1: *Switches, Lights, and Multiplexers***
  - Lab 2: *Numbers and Displays*
  - Lab 3: *Latches, Flip-flops, and Registers*
  - Lab 4: *Counters*
  - Lab 5: *Real-time Clock and Timers*
  - Lab 6: *Adders, Subtractors, and Multipliers*
  - **Lab 7: *Finite State Machines***
  - Lab 8: *Memory Blocks*
  - Lab 9: *A Simple Processor*
  - Lab 10: *An Enhanced Processor*

Lab Number	Lab Title	Lab Objectives
1	Switches, Lights, and Multiplexers	Implement a 4-to-1 multiplexer using logic gates and switches.
2	Numbers and Displays	Implement a 4-bit binary counter and a 7-segment display driver.
3	Latches, Flip-flops, and Registers	Implement a D flip-flop and a 4-bit register.
4	Counters	Implement a 4-bit binary counter and a decade counter.
5	Real-time Clock and Timers	Implement a real-time clock and a timer.
6	Adders, Subtractors, and Multipliers	Implement a 4-bit adder, subtractor, and multiplier.
7	Finite State Machines	Implement a finite state machine for a sequence detector.
8	Memory Blocks	Implement a 4-bit memory block.
9	A Simple Processor	Implement a simple processor with ALU, registers, and control logic.
10	An Enhanced Processor	Implement an enhanced processor with ALU, registers, and control logic.

# Lab 1: Switches and Lights

## Part 1

**/\* connect switches to lights through FPGA \*/**

LEDG(9) <= SW(9);

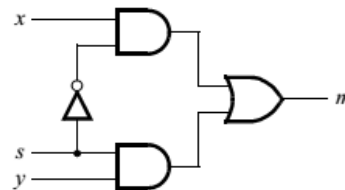
LEDG(8) <= SW(8);

...

LEDG <= SW;

## Part 2

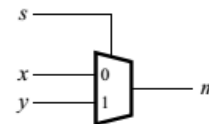
**/\* build a 2-to-1 multiplexer \*/**



a) Circuit

s	m
0	x
1	y

b) Truth table



c) Symbol

# ... Lab 1

Part 2

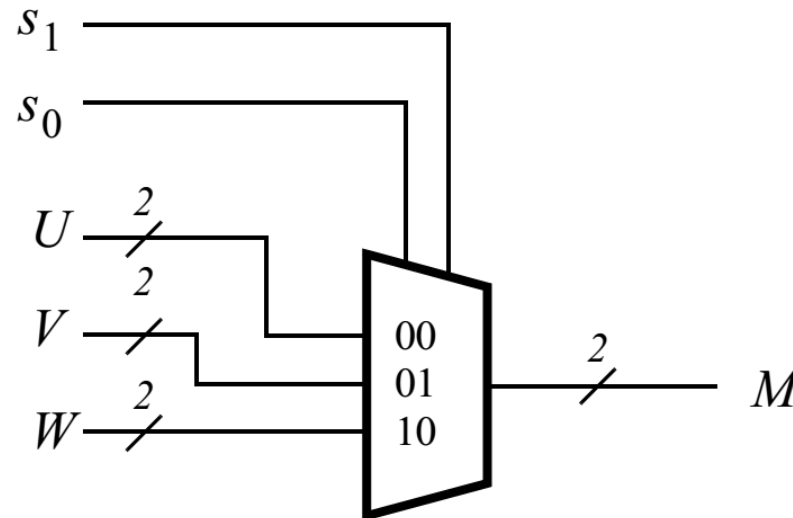
$m \leq (\text{NOT}(s) \text{ AND } x) \text{ OR } (s \text{ AND } y);$

...

Make 3 copies for 3-bit wide 2-to-1 multiplexers,  
then connect to 3 LEDs

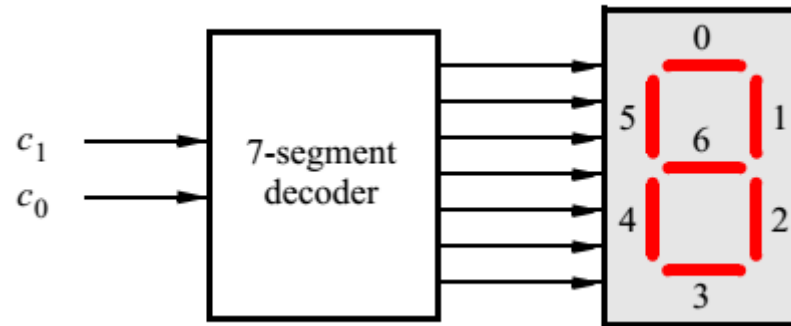
Part 3

Build a more complex multiplexer:  
2-bit wide 3-to-1 multiplexer



# ... Lab 1

## Part 4

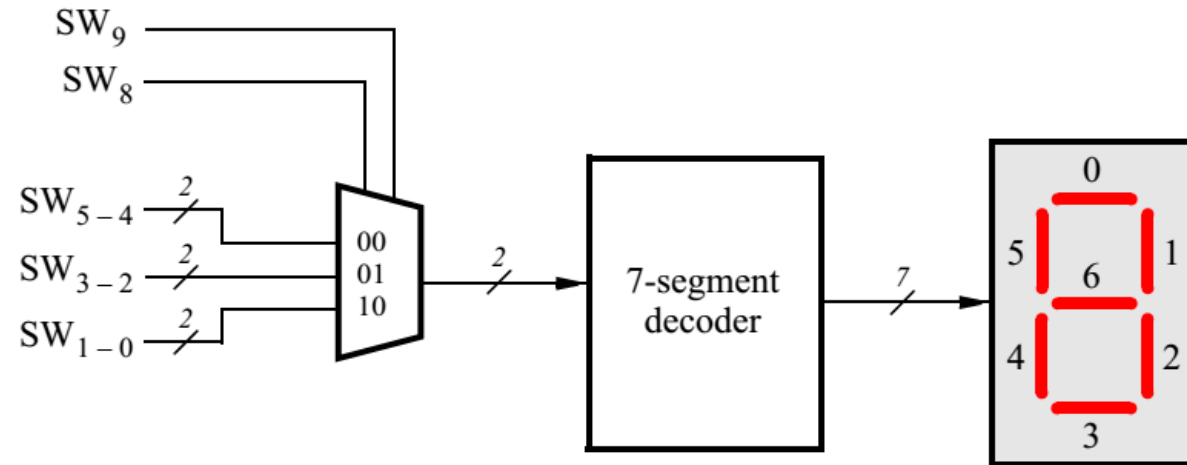


$c_1 c_0$	Character
00	d
01	E
10	0
11	

# ... Lab 1

## Part 5

Select and display one of three characters



Rotate the word dE0 on three displays

## Part 5

$SW_9$	$SW_8$	Character pattern		
00		d	E	0
01		E	0	d
10		0	d	E

## ... Lab 1

### Part 6

Extend your design from Part V so that it uses all four 7-segment displays on the DE0 board

$SW_9$ $SW_8$	Character pattern			
00		d	E	0
01	d	E	0	
10	E	0		d
11	0		d	E

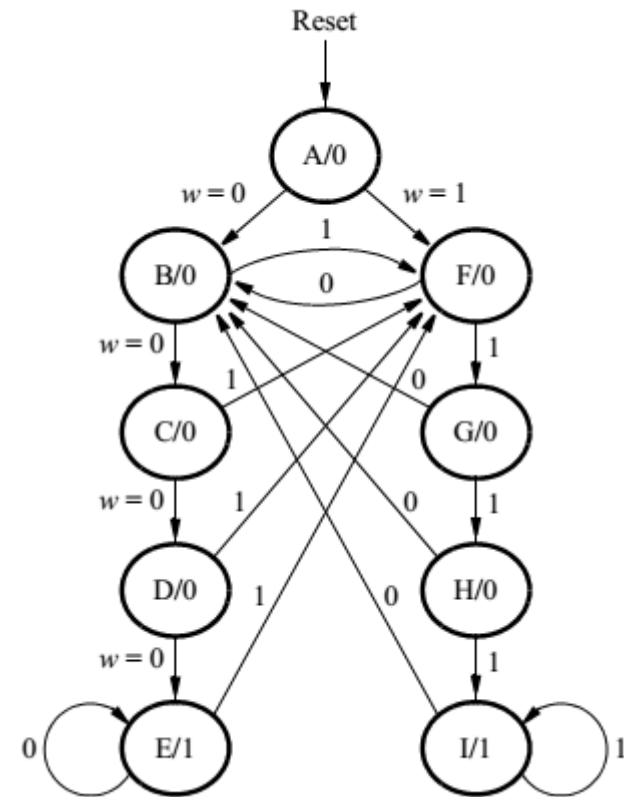


# Lab 7

## ■ FSM – Sequence recognizer

- There is an input  $w$  and an output  $z$ .
- Whenever  $w = 1$  or  $w = 0$  for four consecutive clock pulses the value of  $z$  has to be 1; otherwise,  $z = 0$ .
- Overlapping sequences are allowed, so that if  $w = 1$  for five consecutive clock pulses the output  $z$  will be equal to 1 after the fourth and fifth pulses

Part 1

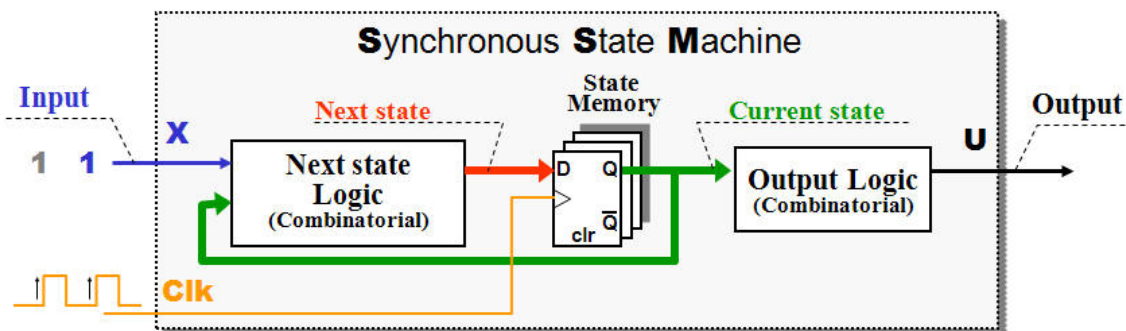


# Lab 7

## Part 1

Current State	W	Next state	Z
A: 000000001	0	B: 000000010	0
B: 000000010	0	C: 000000100	0
C: 000000100	0	D: 000001000	0
D: 000001000	0	E: 000010000	0
E: 000010000	0	E: 000010000	1
F: 000100000	0	B: 000000010	0
G: 001000000	0	B: 000000010	0
H: 010000000	0	B: 000000010	0
I: 100000000	0	B: 000000010	1

Current State	W	Next state	Z
A: 000000001	1	F: 000100000	0
B: 000000010	1	F: 000100000	0
C: 000000100	1	F: 000100000	0
D: 000001000	1	F: 000100000	0
E: 000010000	1	F: 000100000	1
F: 000100000	1	G: 001000000	0
G: 001000000	1	H: 010000000	0
H: 010000000	1	I: 100000000	0
I: 100000000	1	I: 100000000	1



$Y\_D(1) \leq (y\_Q(0) \text{ OR } y\_Q(5) \text{ OR } y\_Q(6) \text{ OR } y\_Q(7) \text{ OR } y\_Q(8)) \text{ AND NOT } (w);$

# Lab 7

## ■ FSM – Sequence recognizer

- Describe the state table for the FSM by using a VHDL CASE statement in a PROCESS block, and use another PROCESS block to instantiate the state flip-flops.

```
ARCHITECTURE Behavior OF part2 IS
  SIGNAL Clock, Reseth, w, z : STD_LOGIC;
  TYPE State_type IS (A, B, C, D, E, F, G, H, I);
  SIGNAL y_Q, Y_D : State_type;
  ...
```

Part 2

