



EECE 353: Digital Systems Design

Lab 1: Combinational Logic

May – June 2015

Instructor: Scott Chin

scottc@ece.ubc.ca

Learning Objectives for Lab 1

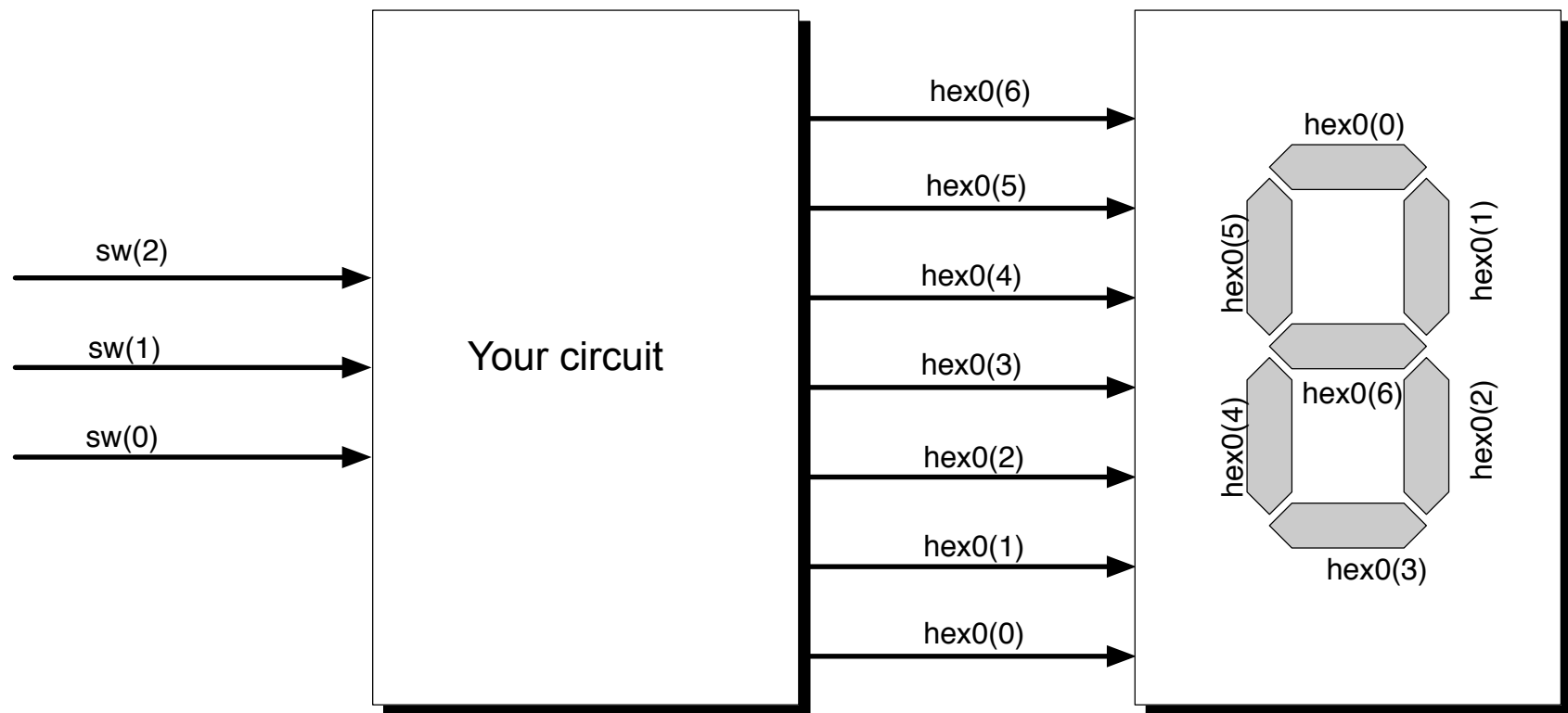
1. Become familiar with the Modelsim / Quartus II Design Flow
2. Become familiar with your board (DE2)
3. Become acquainted with VHDL
4. Start to think about debugging techniques

You do not need to know much VHDL to do this lab.

The handout teaches you (almost) everything you need to know.

The first circuit you will build

You will make a combinational
7 Segment Display LED Driver...



The Circuit

You will make a combinational
7 Segment Display LED Driver...

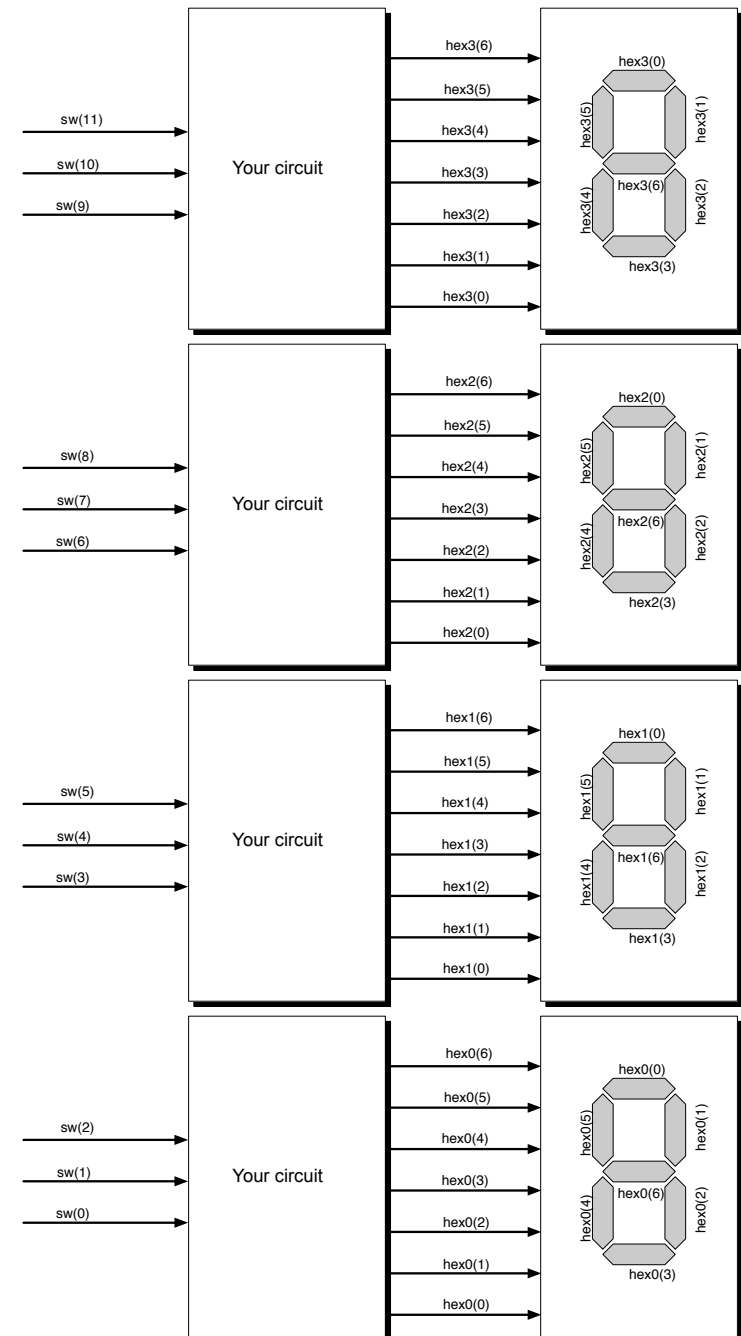
... and use it to drive 4 digits.

DO:

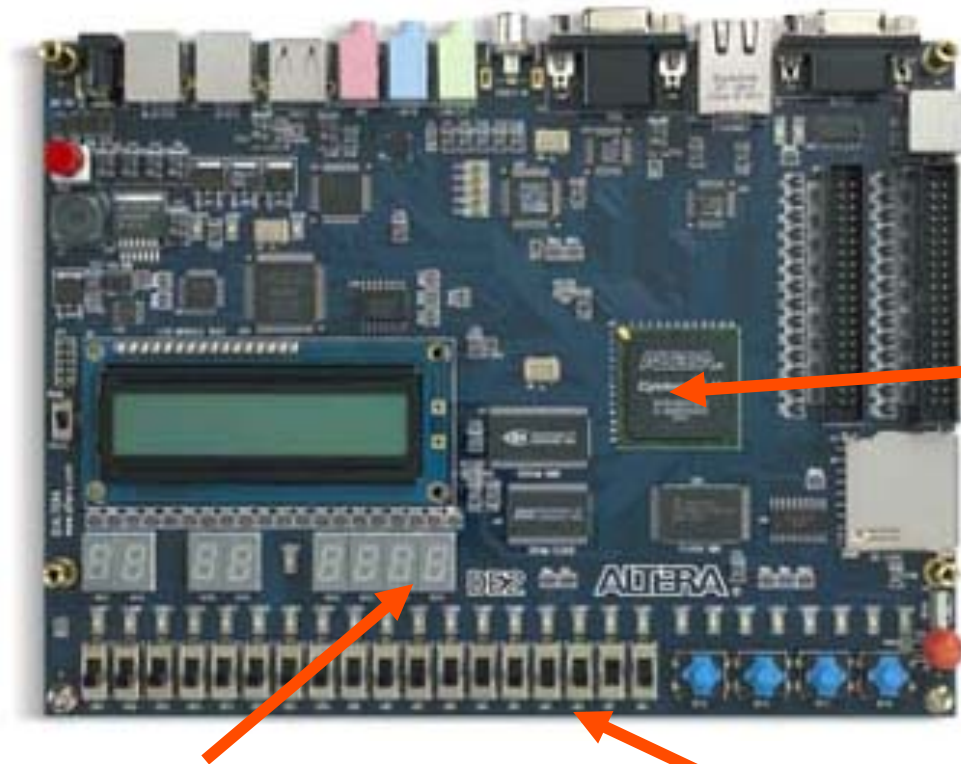
- learn to use structural hierarchy
- create 4 instances of converter

DO NOT:

- copy CASE statement 4 times
- make 4 files: converter1.vhd,
converter2.vhd, converter3.vhd, ...



Board you will use:



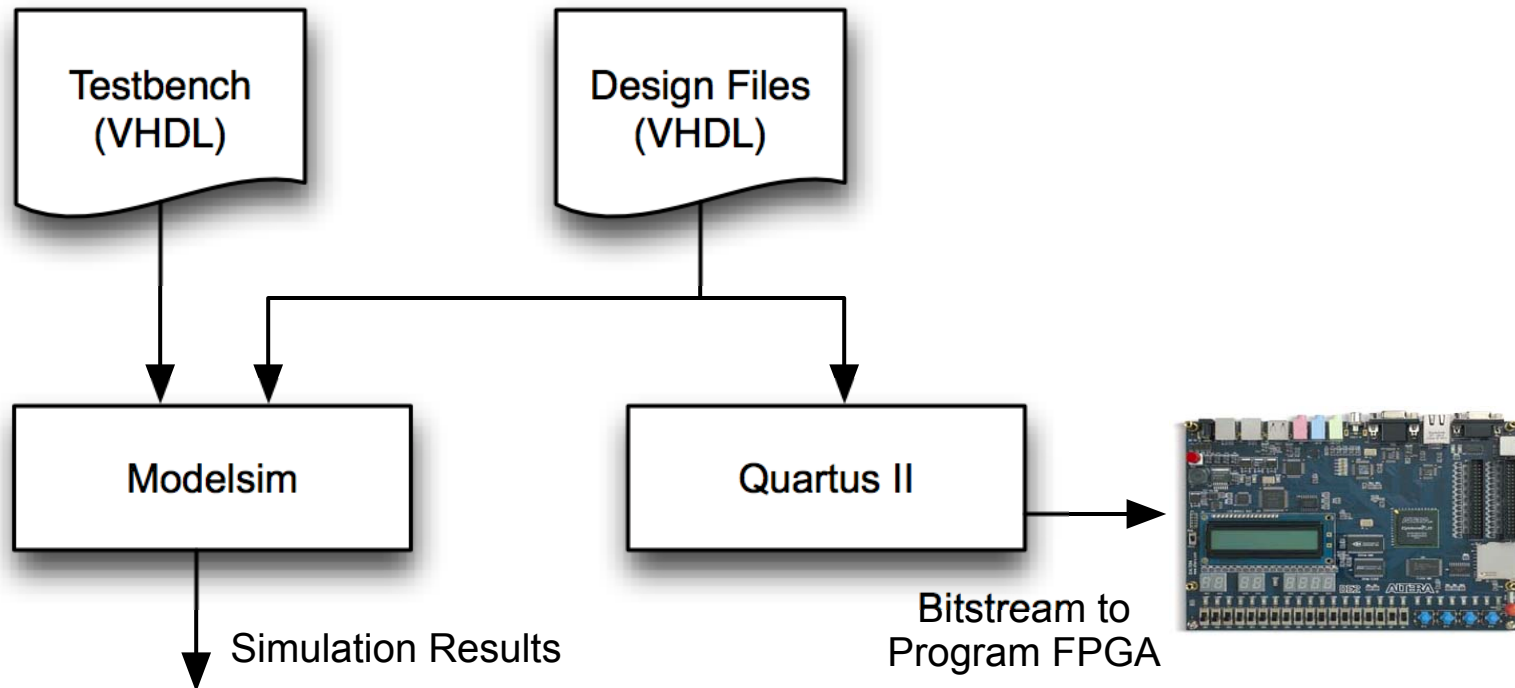
FPGA: This is where your circuit is implemented

Output: Seven-Segment Display

Input: Slider Switches

Display and switches are hard-wired to specific pins on the FPGA

Modelsim / Quartus II Design Flow



This flow provides **RTL simulation**. We will talk about other types of simulation later, and different ways to use Modelsim and Quartus II together

Steps to do this lab:

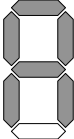
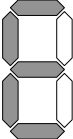
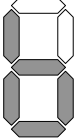
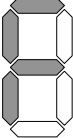
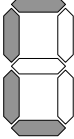
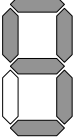

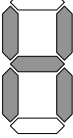
1. Download version 13.0sp1 of Quartus II to your own computer, or find a place in the department you can run the tools (most of the undergrad labs).

The software is free, and does not require a licence file. The tutorial document (on the Connect site) describes where to get the software.

2. Become familiar with Quartus II
 - We have provided a tutorial document on the Connect site with some example files

... continued

3. Create a VHDL code converter:

sw(2 downto 0)	Output	sw(2 downto 0)	Output
000		100	
001		101	
010		110	
011		111	

We've provided a skeleton file on the Connect site and the handout explains how to modify it to implement the converter.

Hint: In the skeleton file given to you, you will see this:

case SW is

when "000" => HEX0 <= "0001000";

when "001" => HEX0 <= "0000011";

when "010" => HEX0 <= "insert a string of 7 bits here";

when "011" => HEX0 <= "insert a string of 7 bits here";

when "100" => HEX0 <= "insert a string of 7 bits here";

when "101" => HEX0 <= "insert a string of 7 bits here";

when "110" => HEX0 <= "insert a string of 7 bits here";

when others => HEX0 <= "insert a string of 7 bits here";

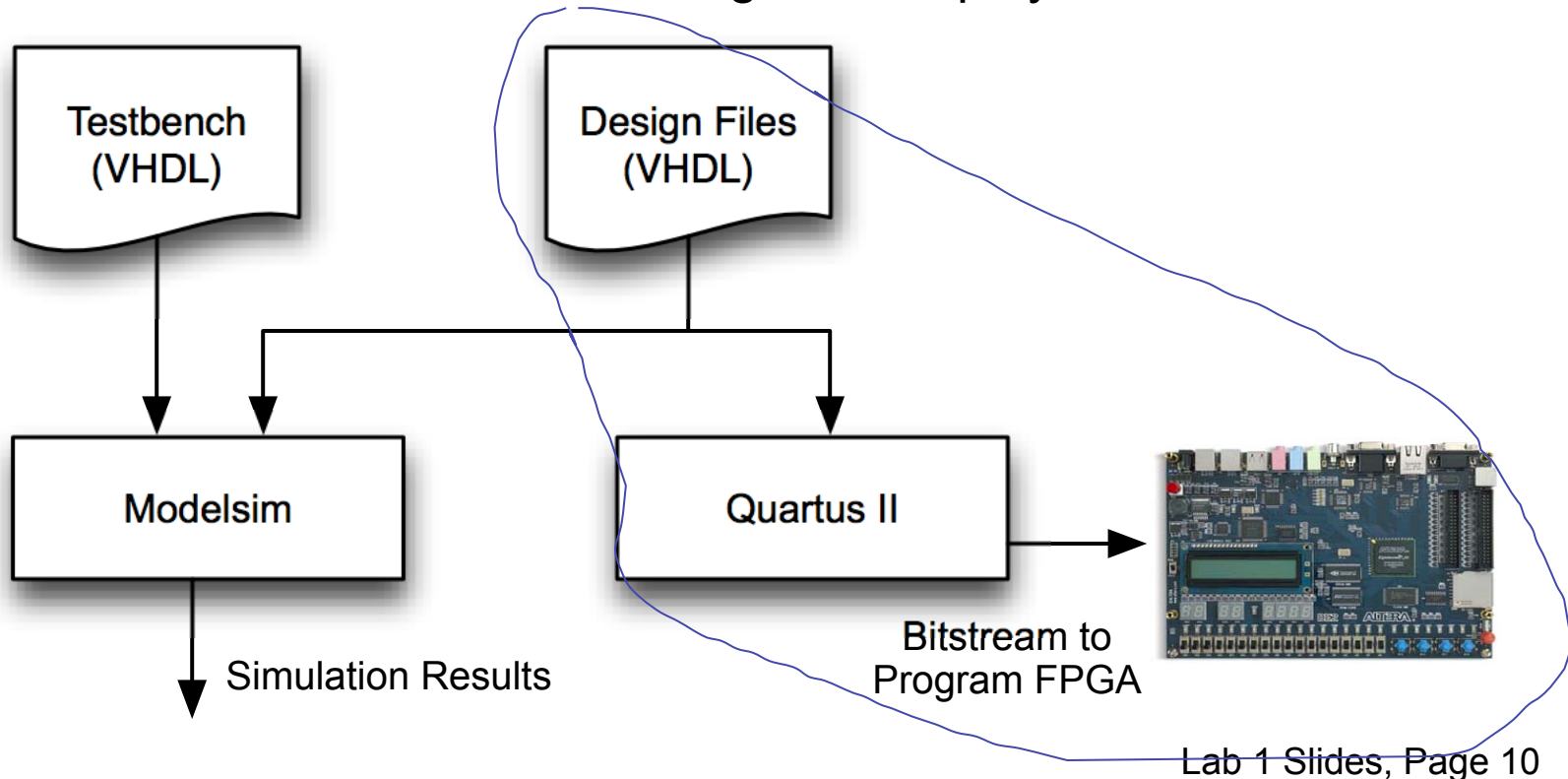
end case;

Outputs

Inputs

The handout explains, in more detail, how to modify this.

4. Compile your design using Quartus II and download it to the board.
Demonstrate your working design to the TA.
6. Create two versions: 1 seven-segment display, 3 switches
4 seven-segment displays, 12 switches



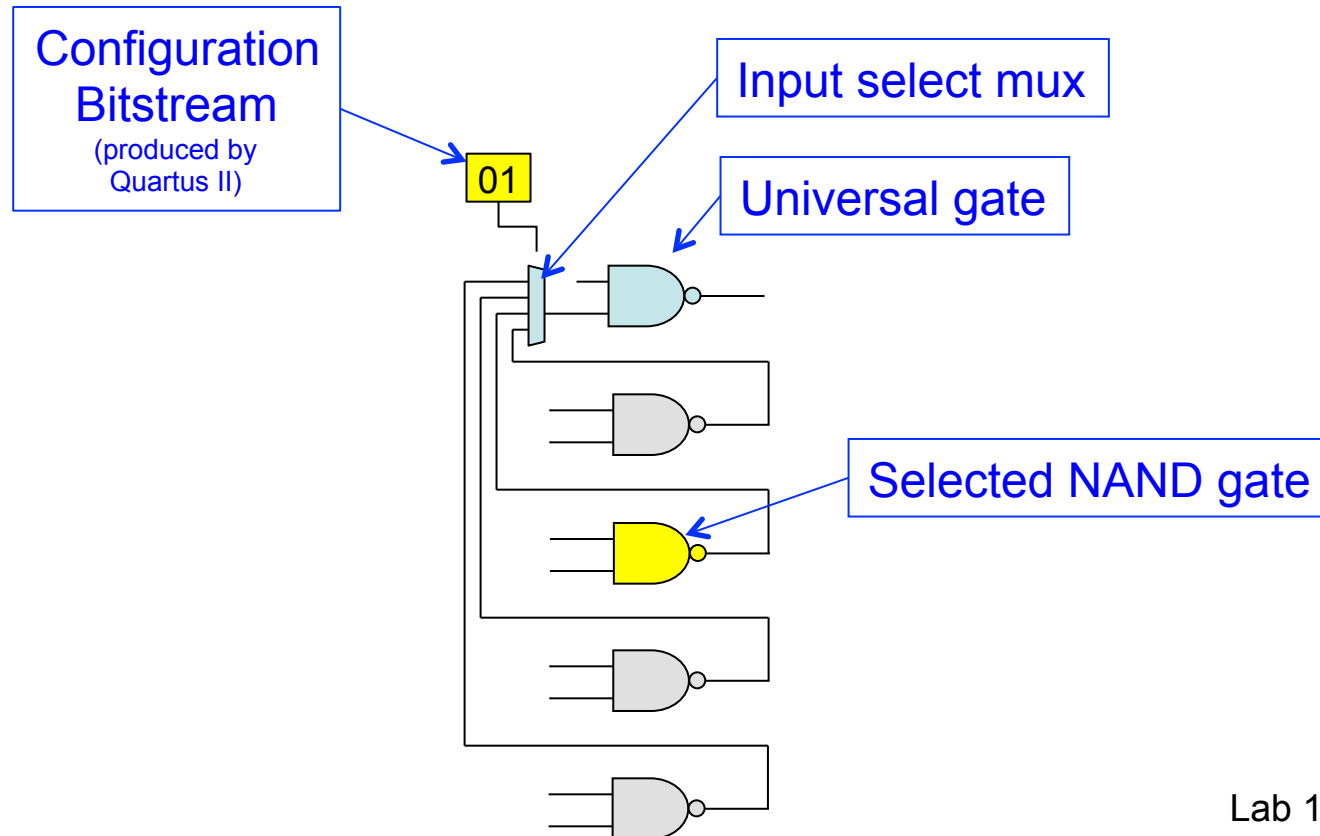
Important Thing to Remember:
The FPGA does not “execute” the VHDL.
It implements the “gates”.

ADDITIONAL INFO

How an FPGA Works Internally

Greatly simplified view on how an FPGA works internally:

- every gate input has a “select mux”
- which can choose from among dozens of several gate outputs
- “configuration bitstream”, computed by Quartus, controls the “select muxes” for thousands of gates



Pin Assignments

In your VHDL description, you will use ports with names such as **SW** (to refer to the switches on the board) or **HEX0** (to refer to one of the 7 segment displays).

```
entity converter is
    port( SW : in std_logic_vector(2 downto 0);
          HEX0 : out std_logic_vector(6 downto 0));
end converter ;
```

How does the compiler know that these names map to the various hardware on the DE2 board outside of the FPGA?

You tell the tool this mapping through the **PIN ASSIGNMENTS** file

Pin Assignments

If you open the supplied DE2_pin_assignments_UBC.csv file:

```
SW[ 0 ], PIN_N25 ,  
SW[ 1 ], PIN_N26 ,  
SW[ 2 ], PIN_P25 ,  
...  
HEX0[ 0 ], PIN_AF10 ,  
HEX0[ 1 ], PIN_AB12 ,  
HEX0[ 2 ], PIN_AC12 ,  
...
```

This file tells the the compiler which pin on the FPGA chip you are referring to when you use names such as **SW** or **HEX** in your top level port list.

The supplied DE2_pin_assignments_UBC.csv specifies the mapping of the various components on the DE2 board to the FPGA pins.

If you forget to setup pin assignments in your Quartus project, your inputs/outputs will be assigned to random pins

Your inputs will not be driven by the switches and your outputs will not drive the seven-segment display. You will go crazy trying to debug the problem with your design until you realize you forgot this step.

You need to do the pin assignments **before** compilation

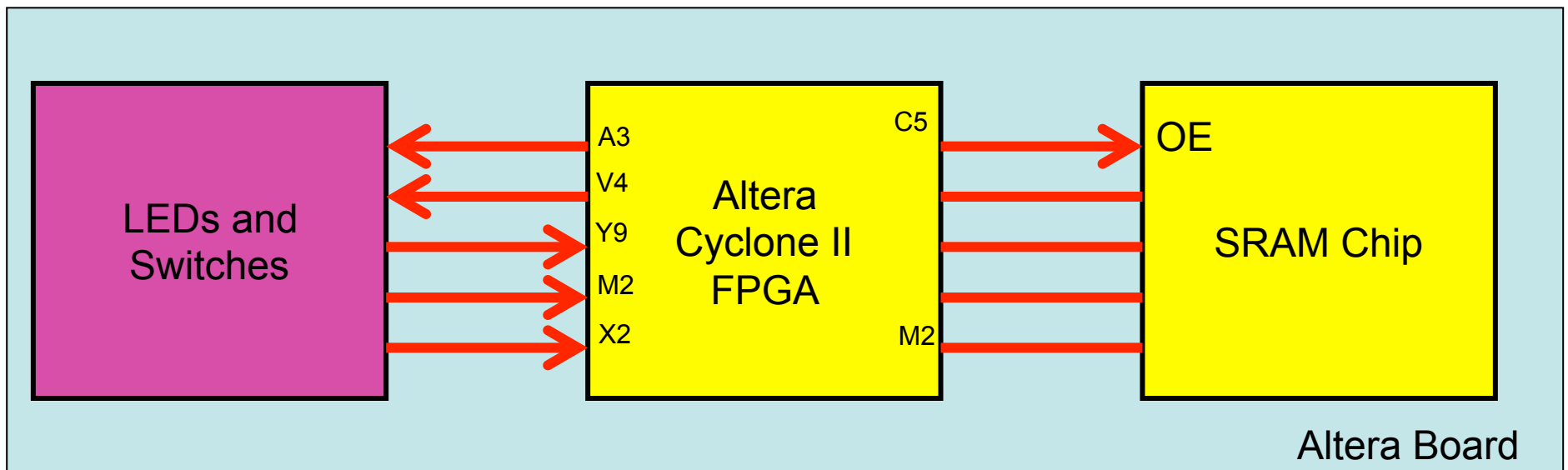
- If you forget, then do pin assignments and re-compile.

Warning: If the pin assignment is not done correctly, you can actually damage the board!!!

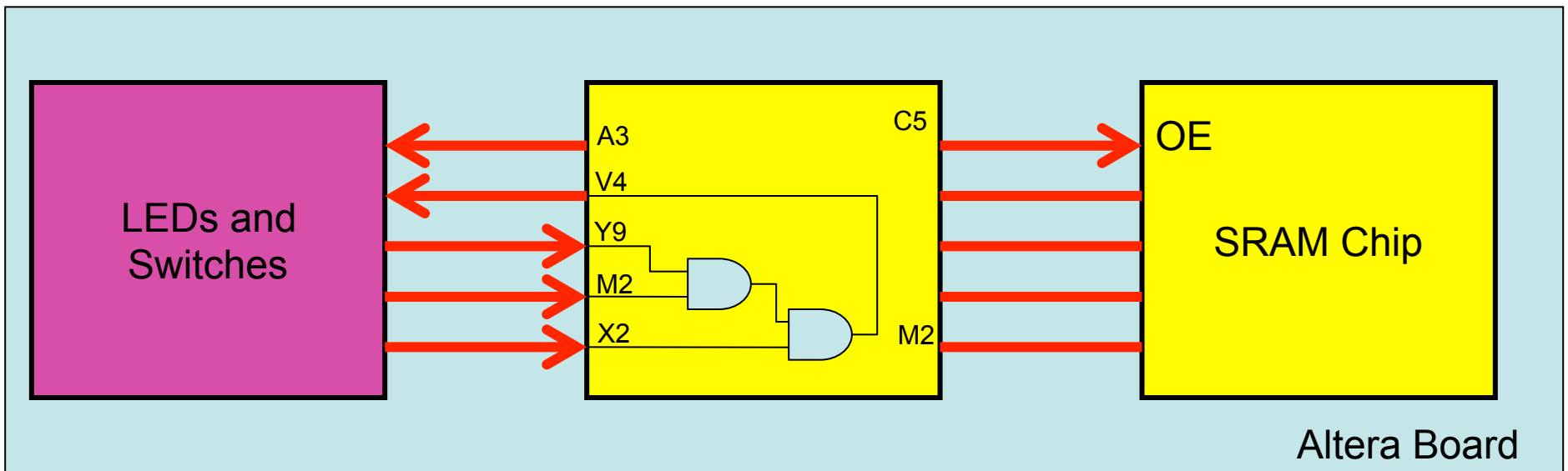
This can happen accidentally if you let the software perform random pin assignments!!



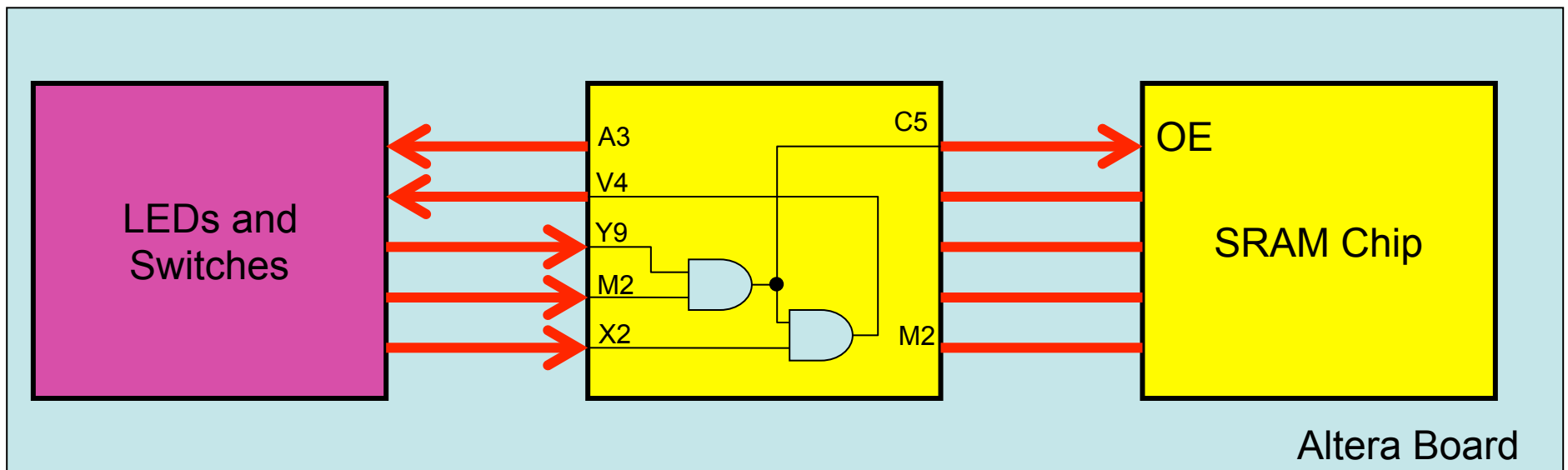
FPGA Pins



Good Pin Assignment

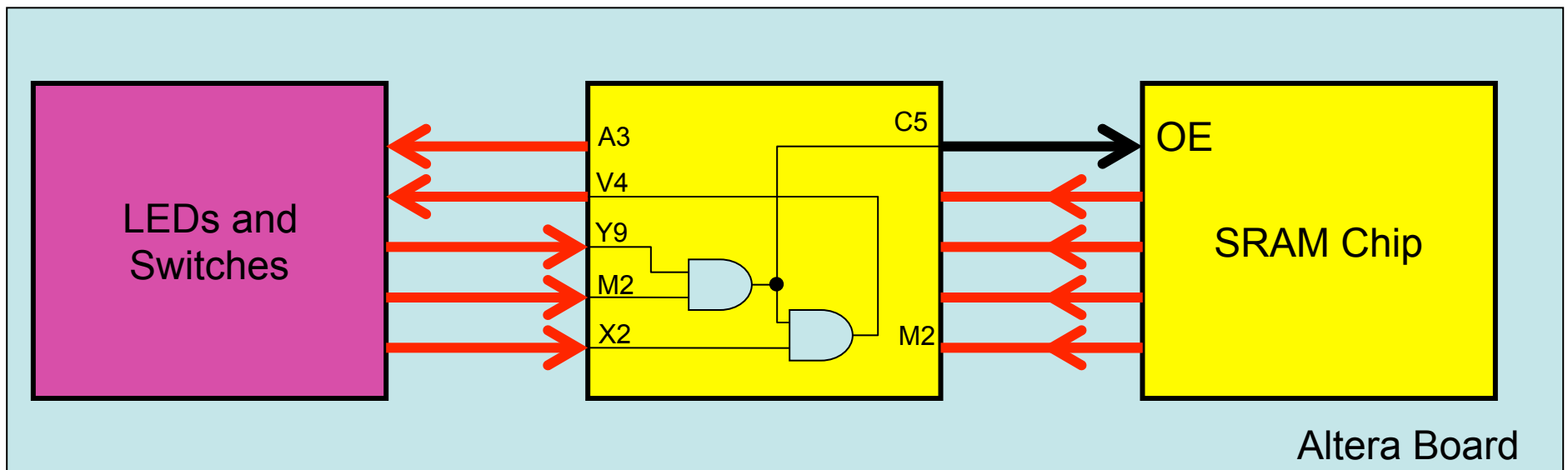


Bad Pin Assignment



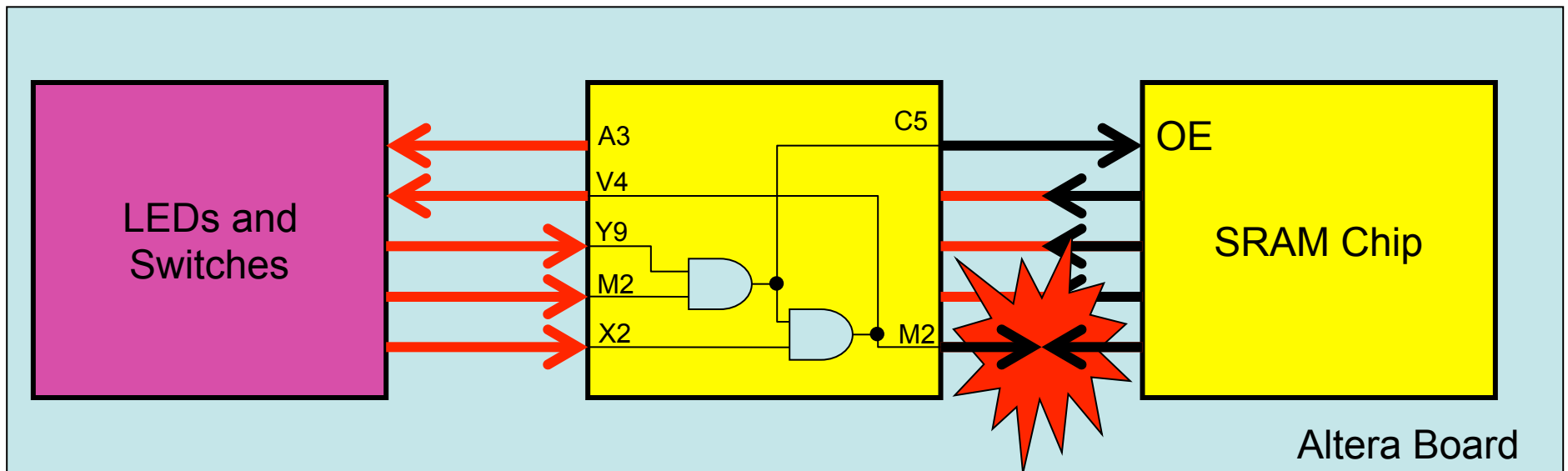
Why is this bad?

Why Is This Bad?



Because OE enables SRAM...
And SRAM drives data pins...

Why Is This Bad?



Altera chip may drive data pin too!
Especially with multiple bad pin assignments!

Isn't there board-level protection for this sort of thing?
Yes, but don't rely on it being enough.

Moral: Assign pins as described in the **Tutorial document**.