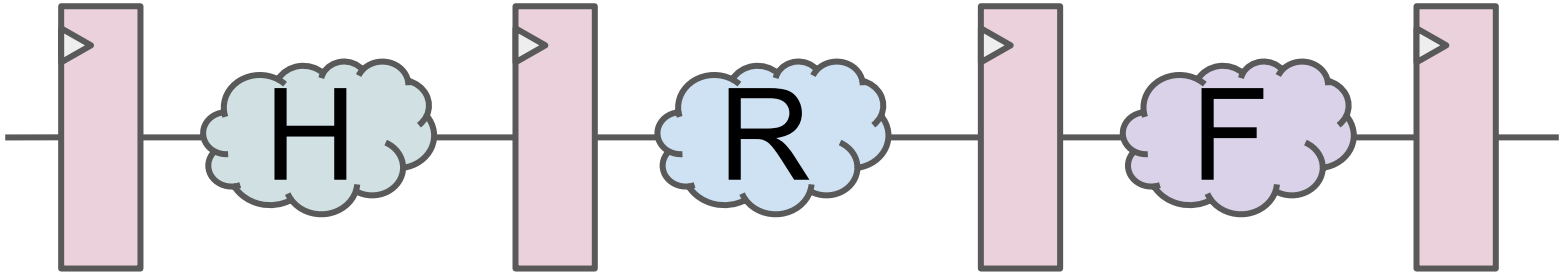


HDL, RTL and FPGA: Lab 1

Your first step in designing digital hardware

Yuri Panchul, Senior Hardware Design Engineer, MIPS
Lecture for Innopolis University - 2018-01-25

MIPS

The informal explanation of acronyms

- HDL - Hardware Description Language
  - A language to design, simulate and verify circuits. We will use Verilog-2001.
- RTL - Register Transfer Level
  - A methodology to describe a circuit using HDL in a way that allows semi-automated conversion of the code into a blueprint for a physical chip manufactured on a foundry
- ASIC - Application-Specific Integrated Circuit
  - An example of ASIC is a chip that runs Android on your smartphone
- SoC - System on Chip
  - An ASIC that has one or several processor cores, memories and other components that form the whole computer system on a single chip
- FPGA - Field-Programmable Gate Array
  - A reconfigurable chip we will use as a substitution for manufacturing ASIC

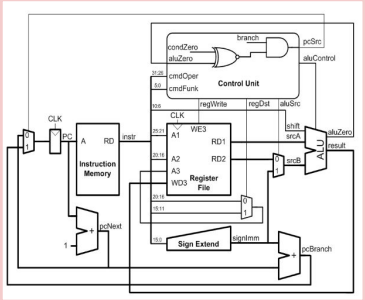
Hardware/software dualism - an informal example

Microcontroller (embedded chip, ASIC, SoC)

CPU

Designed
in Verilog

Executes
instructions



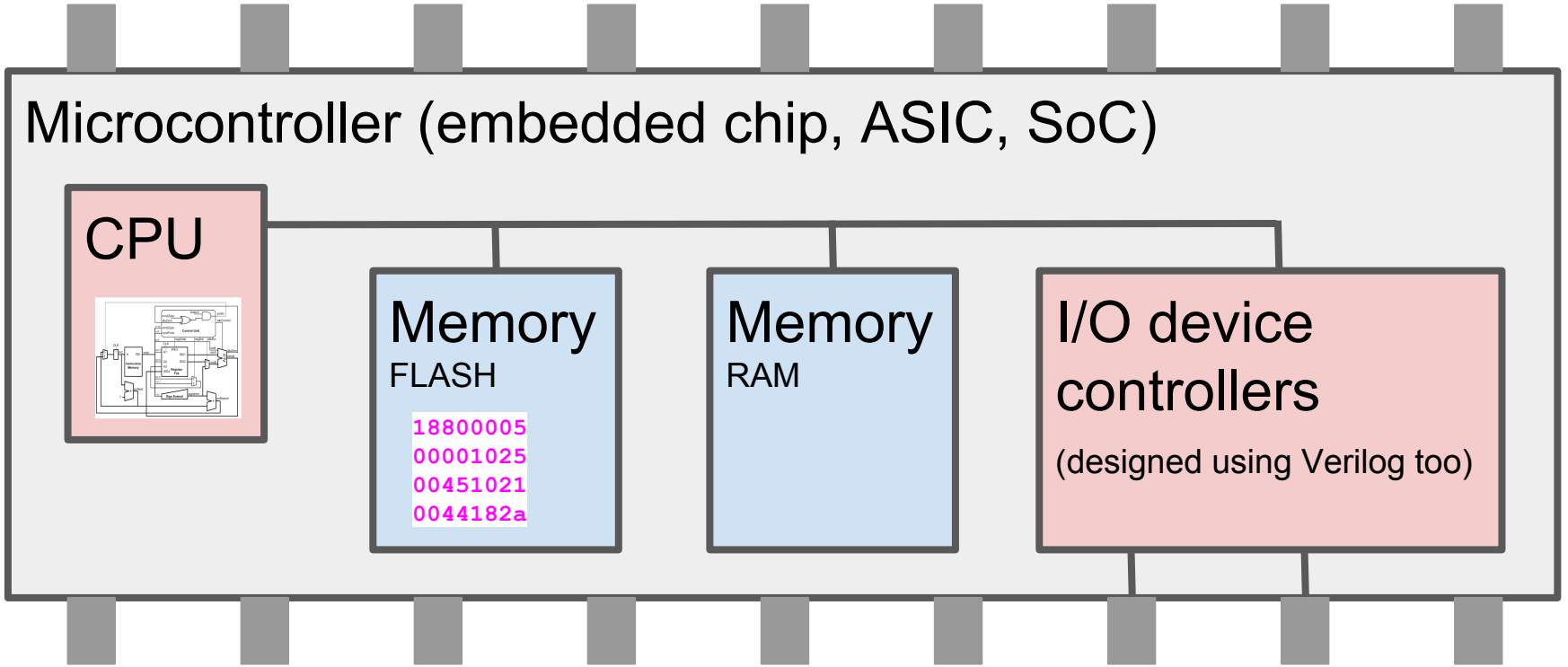
Memory

Contains a program,
a sequence of instructions

Compiled from C
or a similar language

```
18800005
00001025
00451021
0044182a
5460fffe
00451021
03e00008
00000000
```

Hardware/software dualism - an informal example / 2



Software: from C to processor instructions

C:

```
int f (int a, int b)
{
   int s = 0;

   while (s < a)
      s += b;

   return s;
}
```

Assembly:

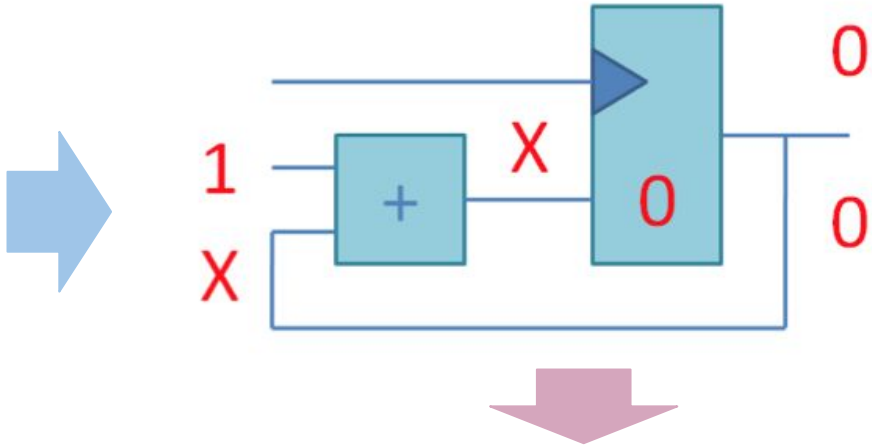
```
sum:
        blez    $4, exit
        move    $2, $0

        addu    $2, $2, $5
loop:
        slt     $3, $2, $4
        bnel    $3, $0, loop
        addu    $2, $2, $5
exit:
        jr      $31
        nop
```

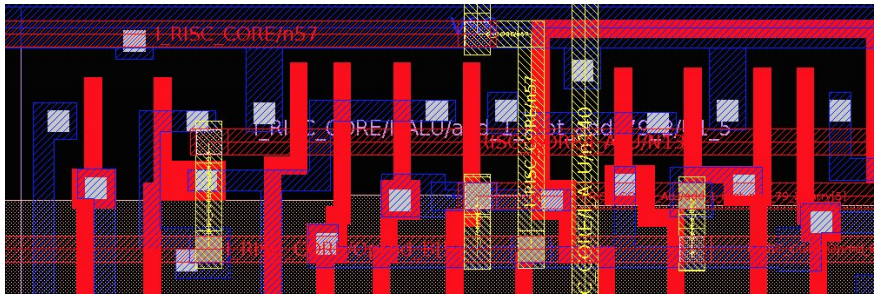
Machine code

```
18800005
00001025

00451021

0044182a
5460fffe
00451021

03e00008
00000000
```

Circuits: from Verilog to transistors (simplified)

```
module counter
(
    input   clock,
    input   reset,
    output logic [1:0] n
);
    always @(posedge clock)
    begin
        if (reset)
            n <= 0;
        else
            n <= n + 1;
    end
endmodule
```





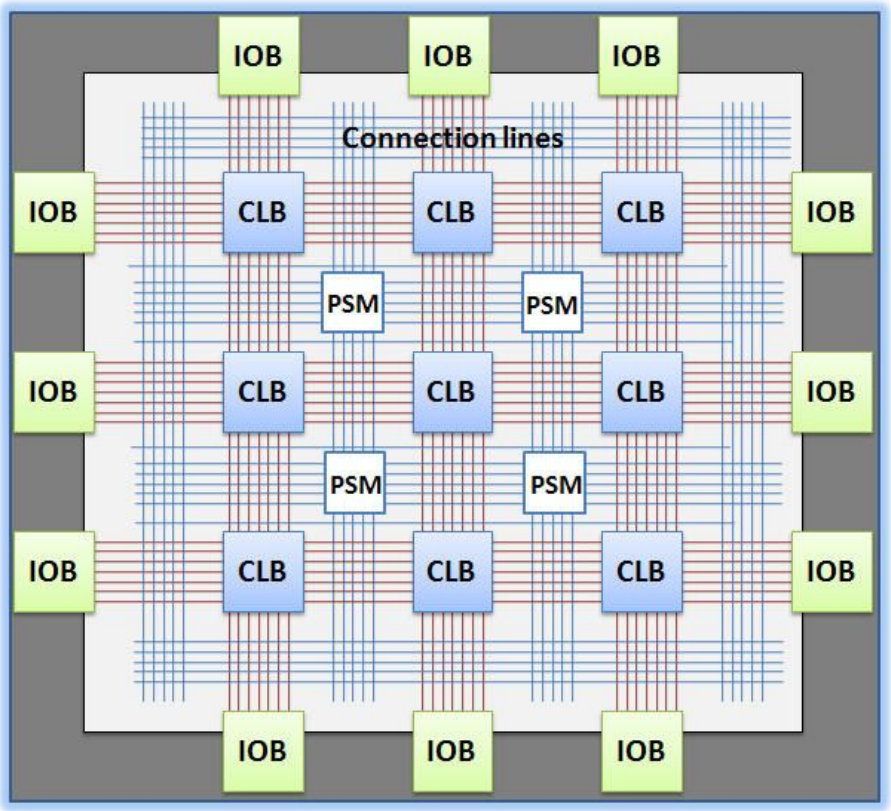
What is FPGA? A simplified explanation

A matrix of cells with changeable function

One cell can become AND, another OR, yet another - one bit of memory

FPGA does not contain fixed CPU, but can be configured to work as CPU

A picture from
http://jjmk.dk/MMMI/PLDs/FPGA/fpga.htm



IOB
Input Output Block

CLB
Configurable
Logic Block

PSM
Programable
Switch Matrix

Connection lines
Single, Long
Double, Direct

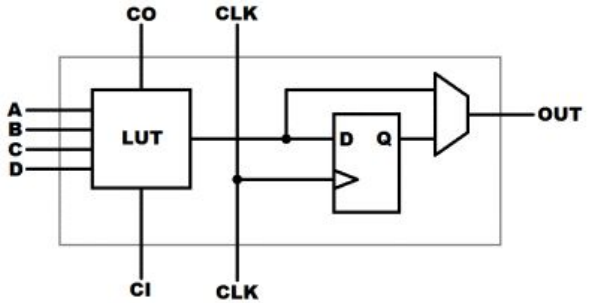
What is inside FPGA cell?

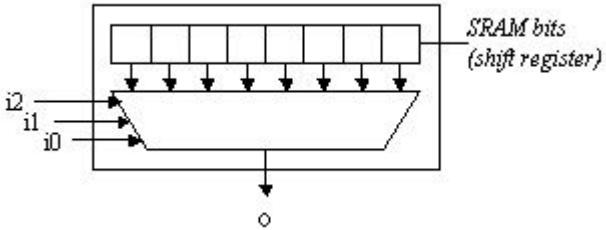
Each cell has multiplexers or muxes

A mux is a device that implements choice, “if ” in hardware

Muxes are connected to bits of memory, loaded from outside FPGA

This allows forming circuits inside FPGA by changing the contents of configuration memory





Pictures are from https://breadboardgremlins.wordpress.com/what-is-an-fpga/ and http://www2.engr.arizona.edu/~rlysecky/courses/cs168-04w/lab7/lab7.html

Twelve basic ideas of digital hardware design

- Gate
- Area
- Delay
- Parallelism
- Module
- Testbench
- Clock
- Reset
- D-Flip-Flop
- Power
- Finite State Machine
- Pipeline

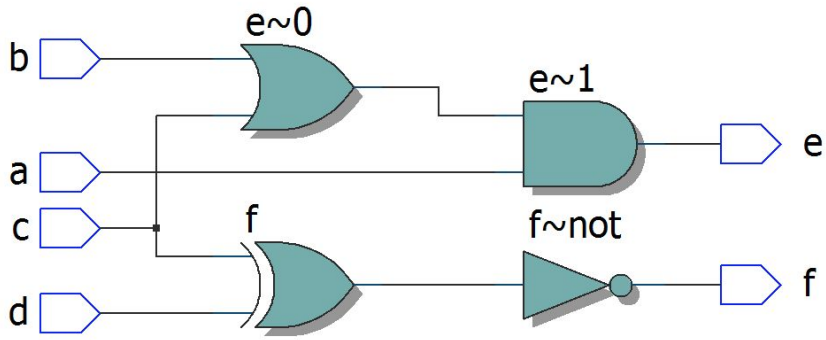
We color-code the topics of Lab 1 in green, the topics of Lab 2 in blue and the topics of Lab 3 in purple.

A hardware design can be partitioned into

- Combinational logic
  - Used to calculate logic and arithmetical functions
- Sequential logic
  - Allows hardware to have memory and repeat actions
- The first lab covers only combinational logic

Combinational logic

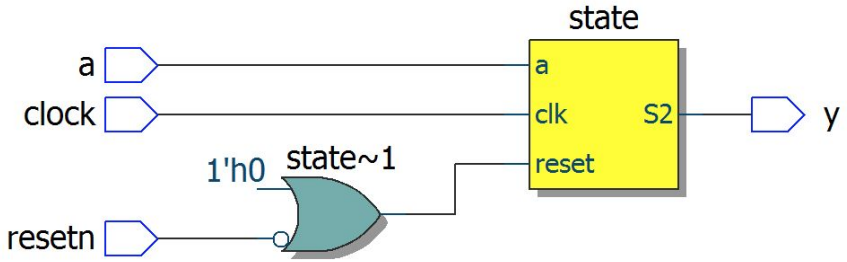
- The outputs of the group of components depend only on inputs
- You set inputs and get outputs after some time
- This groups is called “a combinational cloud”
- Used to calculate logic and arithmetical functions

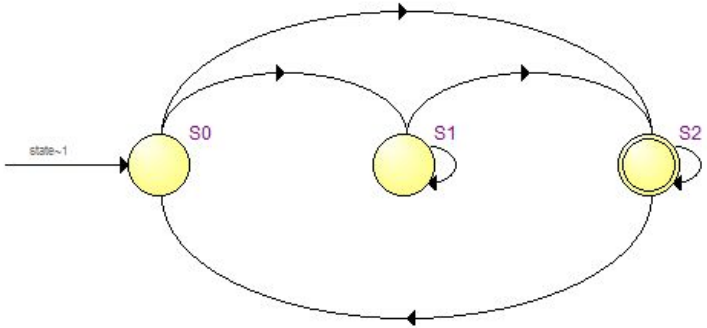



```
module top
(
    input  a, b, c, d,
    output e, f
);

    assign e = a & (b | c);
    assign f = ~ (c ^ d);

endmodule
```

Sequential logic

- The outputs of the partition depend not only on inputs but also on internal state, stored in registers
- Change of the state is synchronized by clock signal
- Allows hardware to be smart - memorize the intermediate results and iterate





Lab 1 exercises

1. Simulating a combinational module
    a. Ports, continuous assignments, testbench, delay, waveforms
    b. We will use ModelSim PE Student Edition from Mentor Graphics
2. Synthesizing a combinational module with basic gates
    a. Logic synthesis, constrains, synthesized schematics, FPGA configuration
    b. We will use Intel Quartus Lite Edition and Terasic DE10-Lite FPGA boards
3. Homework: Synthesize a combinational module that uses input from a button and muxes to output either your first name or last name to a multi-digit 7-segment indicator

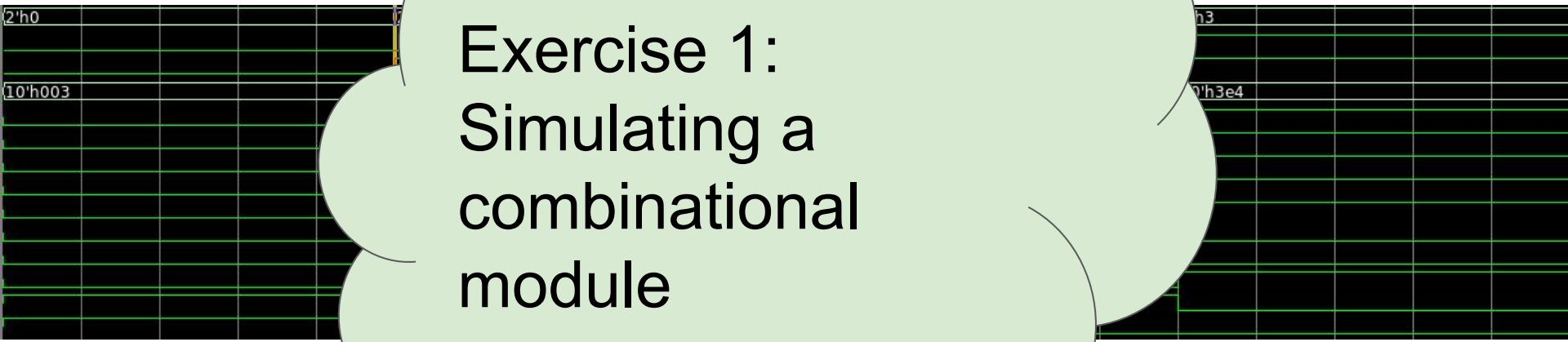
Clone a repository from GitHub

Organization

https://github.com/MIPSfpga

Repository

digital-design-lab-manual



Exercise 1: Simulating a combinational module

The design module - 1

- lab_01/src/lab1_hdl/lab1.v
- The unit of design is called “module”
- Module has a name and a list of ports
- Each port has a direction
  - input, output or inout
- Each port has a width / vector dimension
  - “[9:0]” means “10 bits from LED [9] till LED [0]”
  - If no dimension, 1-bit port assumed, like “input ABC”

```
module lab1
(
    input  [1:0] KEY,
    output [9:0] LED
);
```

The design module - 2

```
wire a = ~ KEY [0];
wire b = ~ KEY [1];
```

- lab_01/src/lab1_hdl/lab1.v
- “wire” declaration creates an intermediate connection between ports, other wires or “reg” variables to be discussed further
- wire can be 1-bit wide, or multiple bit wide, like “wire [4:0] A”
- The assignments to wires are continuous
    - Every time the expression on right hand side changes, it triggers change of the wire on right hand side
- Wires can be optimized away by the synthesis

The design module - 3

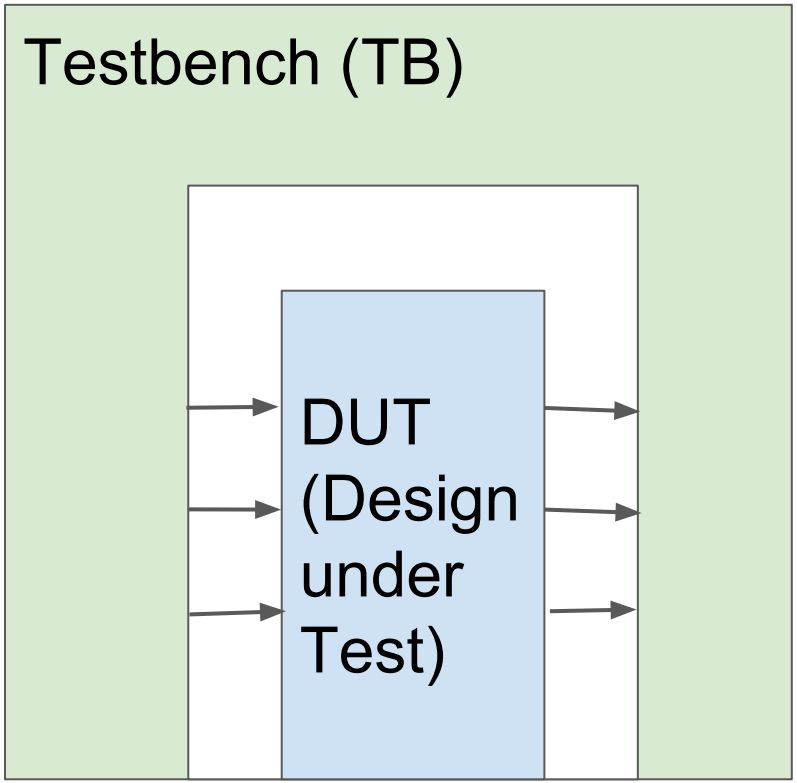
```
// Basic gates AND, OR and NOT
assign LED [0] = a & b;
assign LED [1] = a | b;
assign LED [2] = ~ a;
```

- lab_01/src/lab1_hdl/lab1.v
- The keyword “assign” is used for continuous assignments to an already declared port or a wire
- Note that the assignments do not assume a specific order - they semantically happen whenever the right hand side changes
- HDL simulator is an event-driven, it schedules the assignment evaluations by putting them into a queue of events
- HDL synthesis simply creates a netlist (graph) for an electric circuit

The simulation testbench - 1

```
module testbench;
```

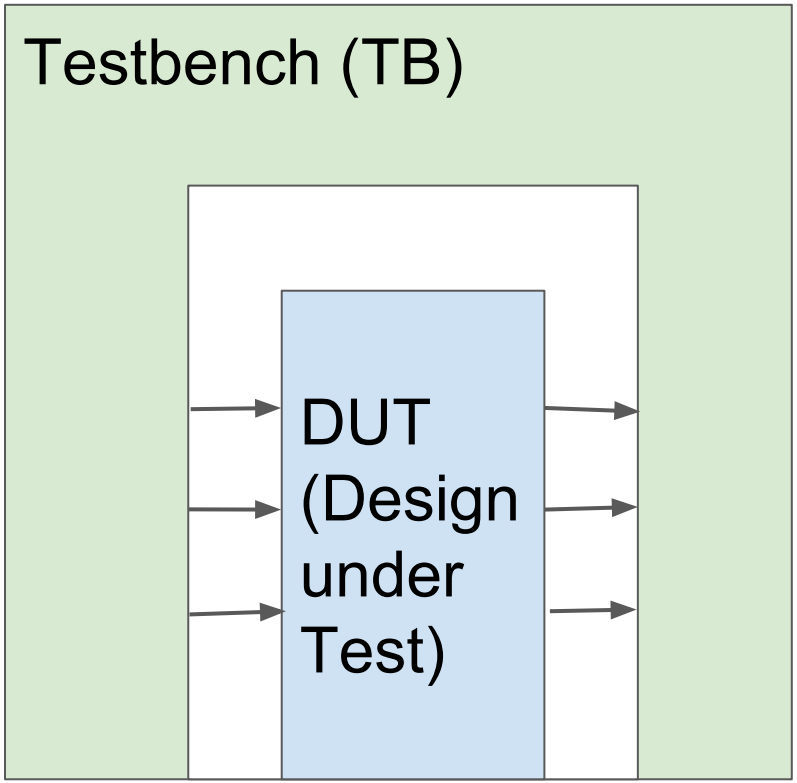
- lab_01/src/lab1_hdl/simulation/testbench.v
- The testbench module (TB) is for simulation only, it is never synthesized
- TB contains a special program that tests the design module
- The design module instantiated inside TB is frequently referred as DUT, Design under Test



The simulation testbench - 2

```
module testbench;
```

- lab_01/src/lab1_hdl/simulation/testbench.v
- Testbench
  - Instantiates Design under Test
  - Drives stimulus
  - Checks the responses
- Testbench may also call functions in other languages (like C) to make verification environments sophisticated



The simulation testbench - 3

- Module instantiation connects the design module's ports to the variables declared in the testbench
- Module instantiation looks like a function call, but it is not
  - DUT instance is physically put inside the testbench module
  - The connections between TB variables and DUT ports are continuous, just like left and right sides of "assign".

- lab_01/src/lab1_hdl/simulation/testbench.v

```
// input and output test signals
reg  [1:0] key;
wire [9:0] led;

// creating the instance of the
//  lab1 - module name
//  dut  - instance name ('dut'
lab1 dut ( key, led );
```

The simulation testbench - 4

- lab_01/src/lab1_hdl/simulation/testbench.v
- Our TB generates stimulus in the simplest way:
  - Assign a value to a variable connected to DUT input port
  - Wait using #delay construct that causes simulator to postpone the execution of the next statement for a number of specified time units

```
initial
    begin
        key = 2'b00;
        #10;
        key = 2'b01;
        #10;
        key = 2'b10;
        #10;
        key = 2'b11;
        #10;
    end
```

The simulation testbench - 5

- lab_01/src/lab1_hdl/simulation/testbench.v
- Keyword “initial” means that the next statement is executed at the beginning og the simulation.
- “begin/end” means that the statements inside are executed sequentially.
- The assignments denoted with “=” inside “begin/end” are called “blocking assignments”. They are executed sequentially, not in parallel (“block the execution”).

```
initial
    begin
        key = 2'b00;
        #10;
        key = 2'b01;
        #10;
        key = 2'b10;
        #10;
        key = 2'b11;
        #10;
    end
```

The simulation testbench - 6

- lab_01/src/lab1_hdl/simulation/testbench.v
- The constant 2'b10 means "two bit-wide binary number 10".
- The time unit in #10 and simulation precision can be specified using `timescale directive. We will see this time later on simulation waveforms. If we put this directive to the beginning of the file, #10 means "10 nanoseconds":

```
`timescale 1 ns / 100 ps
```

```
initial
    begin
        key = 2'b00;
        #10;
        key = 2'b01;
        #10;
        key = 2'b10;
        #10;
        key = 2'b11;
        #10;
    end
```

The simulation testbench - 7

- lab_01/src/lab1_hdl/simulation/testbench.v
- reg declaration means “a variable that can be assigned inside begin/end block”
- $monitor statement in initial block tells the simulator to print the specified text into the simulation log whenever any of the variables change

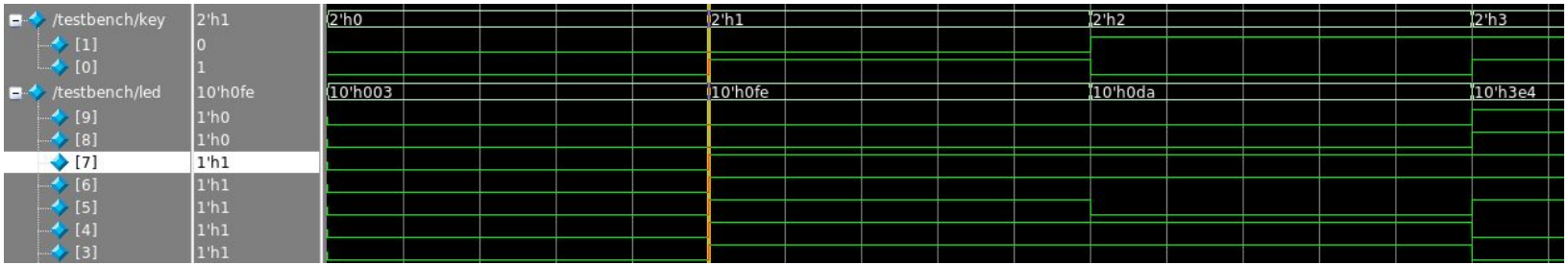
```
reg   [1:0] key;
```

```
initial
    $monitor("key=%b led=%b", key, led);
```

The simulation testbench - 8

```
initial
    $dumpvars;
```

- lab_01/src/lab1_hdl/simulation/testbench.v
- $dumpvar statement in initial block tells the simulator to generate the simulation waveforms.
- Waveform is a database of signal changes for debugging the design.
- The waveform can be viewed in a waveform viewer, either standalone (like GTKWave), or integrated into a Verilog simulator.



Batch file to run the simulation

- lab_01/src/lab1_hdl/simulation/01_simulate_with_modelsim.bat
- Creates a directory for temporary files, produced by ModeSim simulator
- Invokes the simulator executable *vsim.*
- *vsim* reads a stream of its proprietary commands from a script file written in TCL (Tool Command Language).

```
rem recreate a temp folder for
rd /s /q sim
md sim
cd sim

rem start the simulation
vsim -do ../modelsim_script.tcl
```

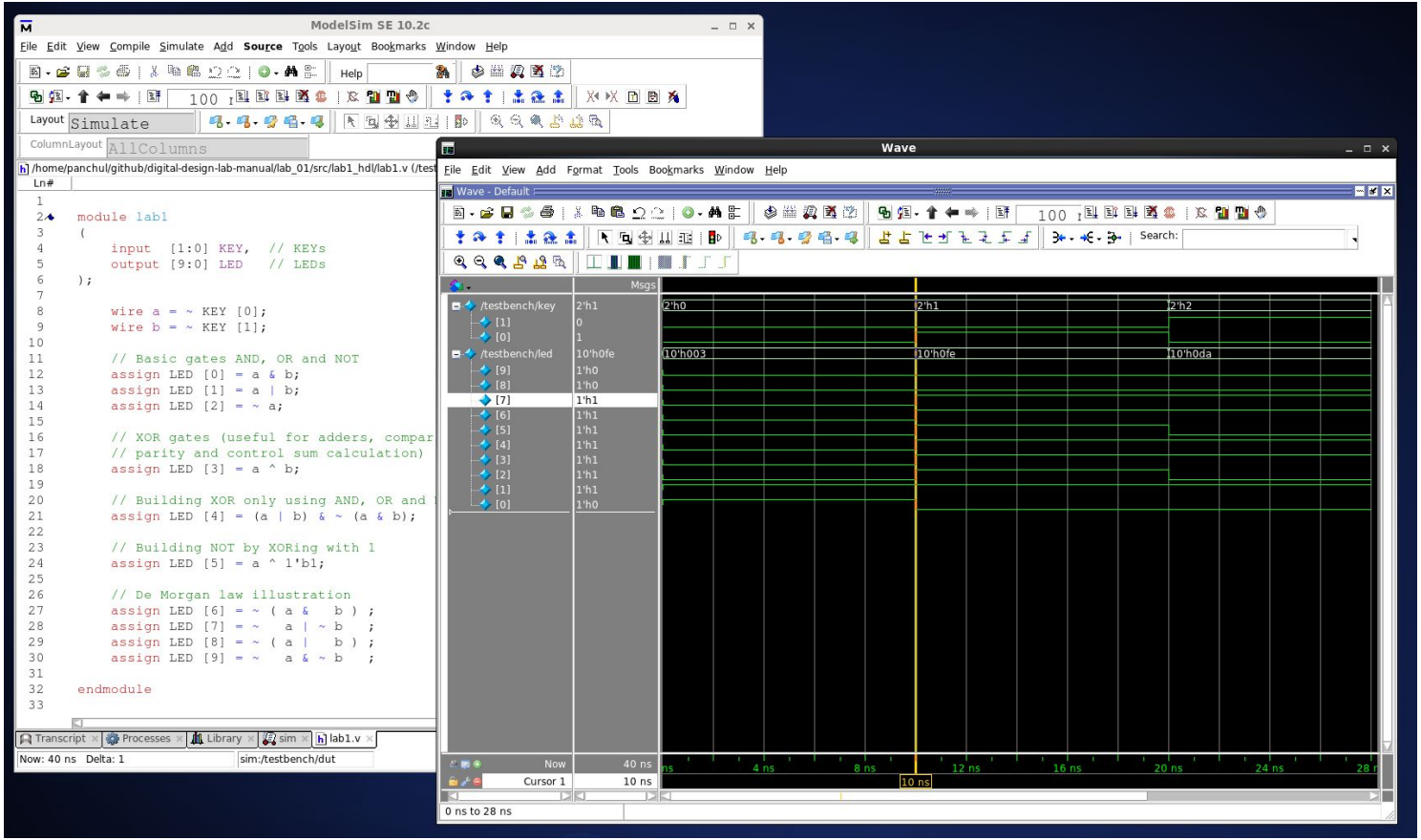
- If you work under Linux, you can write a shell script that does the same.

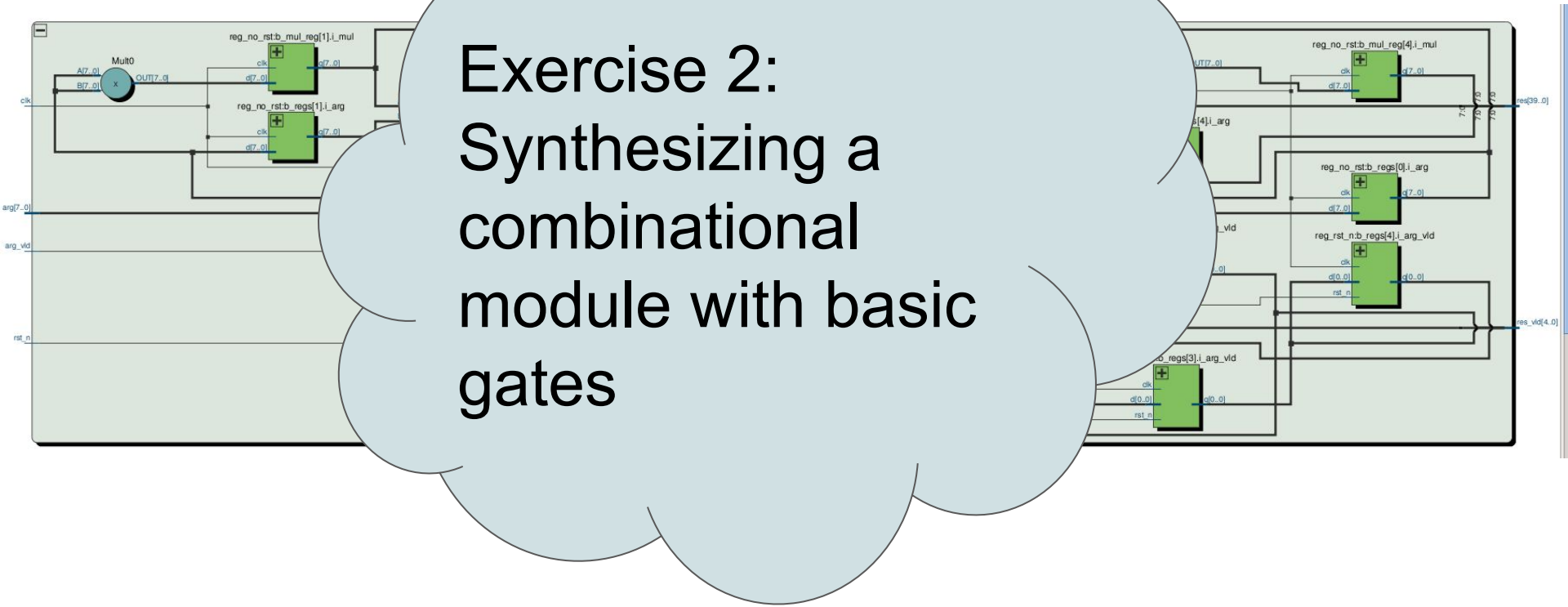
TCL script file for ModelSim

- lab_01/src/lab1_hdl/simulation/modelsim_script.tcl
- TCL (pronounced "tickle" or tee cee el) is a scripting language used since 1990s in many EDA tools (EDA - Electronic Design Automation).
- You don’t need to know it well, just copy somebody else’s scripts for your projects.

```
# create modelsim working library
vlib work

# compile all the Verilog sources
vlog  ../testbench.v ../../lab1.v

# open the testbench module for simulation
vsim -novopt work.testbench

# add all testbench signals to time diagram
add wave sim:/testbench/*

# run the simulation
run -all

# expand the signals time diagram
wave zoom full
```

Now run everything, modify the code and run again





Exercise 2: Synthesizing a combinational module with basic gates

The main Quartus project file

- lab_01/src/lab1_hdl/synthesis/lab1.qpf
- It is OK for the Quartus project file to be empty.
- Quartus usually stores some dates and versions there for Intel / Altera customer support.
- All the essential synthesis settings are not in .qpf, but in other files, .qsf and .sdc.

```
QUARTUS_VERSION = "17.0"
DATE = "12:11:55  November 06, 2017"
PROJECT_REVISION = "lab1"
```

TCL script with synthesis settings - 1

- lab_01/src/lab1_hdl/synthesis/lab1.qsf
- For this project you need only four directives below, and some signal mapping assignments (see the next slide).
- You don’t need to know all the script commands in QSF file.
- Just copy somebody’s script and use bare minimum settings.

```
set_global_assignment -name DEVICE 10M50DAF484C7G
set_global_assignment -name TOP_LEVEL_ENTITY lab1
set_global_assignment -name VERILOG_FILE ../../lab1.v
set_global_assignment -name PROJECT_OUTPUT_DIRECTORY .
```

TCL script with synthesis settings - 2

- lab_01/src/lab1_hdl/synthesis/lab1.qsf
- You need to map the port names in your design to the physical pins of FPGA.
- Synthesis tool know the location of PIN_B8 for chip 10M50DAF484C7G, but it has no idea that the board connects it to a button and you call it KEY[0].

```
set_location_assignment PIN_B8 -to KEY[0]
set_location_assignment PIN_A7 -to KEY[1]
set_location_assignment PIN_A8 -to LED[0]
set_location_assignment PIN_A9 -to LED[1]
. . . . . .
```

TCL script with synthesis settings - 3

- lab_01/src/lab1_hdl/synthesis/lab1.qsf
- Your QSF file has to specify a so-called I/O standard for each signal connected to an FPGA pin.
- You don’t need to know the meaning of I/O standard property in this course.
- You can use wildcards for signal names when setting IO_STANDARD.

```
set_instance_assignment -name IO_STANDARD \
    "3.3 V Schmitt Trigger" -to KEY*

set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LED*
. . . . . .
```

Before synthesis create a copy of project files

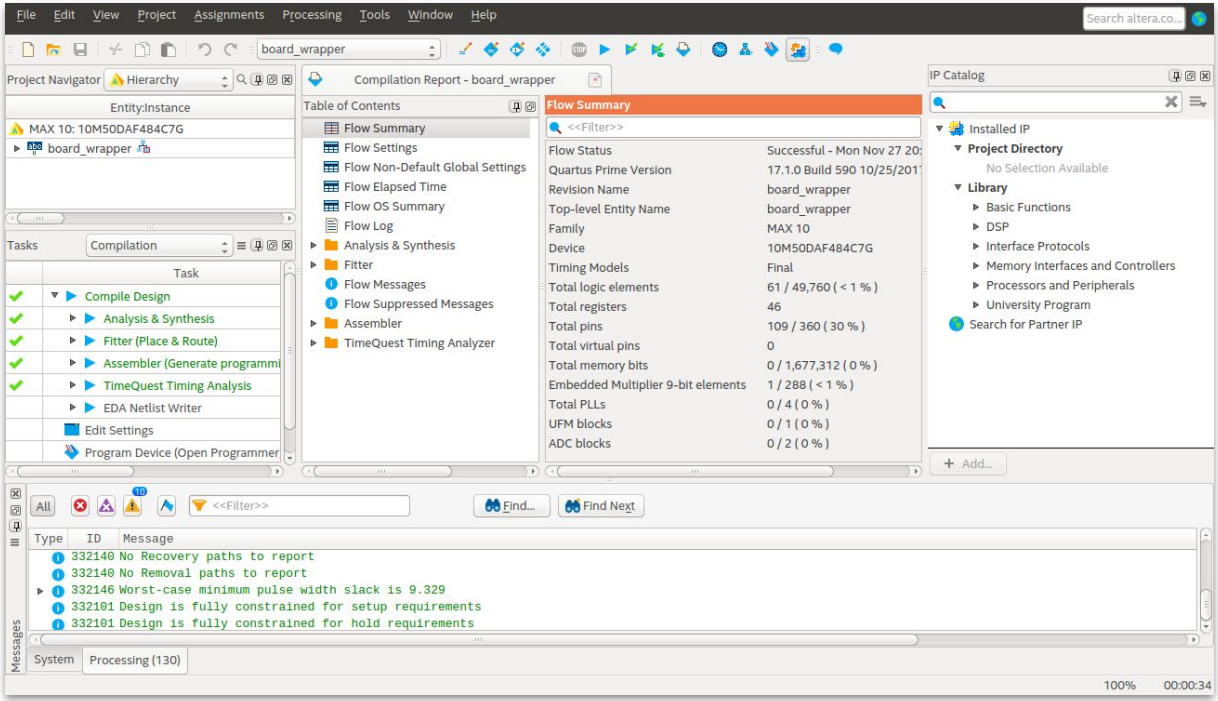
- lab_01/src/lab1_hdl/synthesis/make_project.bat
- Intel / Altera Quartus Lite Edition creates a lot of temporary files
- It is convenient to run synthesis in a separate directory
- You have to copy project files into that directory
- To do this, run make_project.bat [Windows] / run make_project.sh

```
rd /s /q project
mkdir project
copy *.qpf project
copy *.qsf project
```

```
#!/bin/sh

rm -rf project
mkdir project
cp *.q[ps]f project
```

Run synthesis in Intel Quartus

- File:Open Project
- Do not confuse with File:Open
- Find project file in your temporary project directory
- Do not open project from the original directory
- Double-click “Compile Design” in Tasks window



File Edit View Tools Window Help

Search altera.com

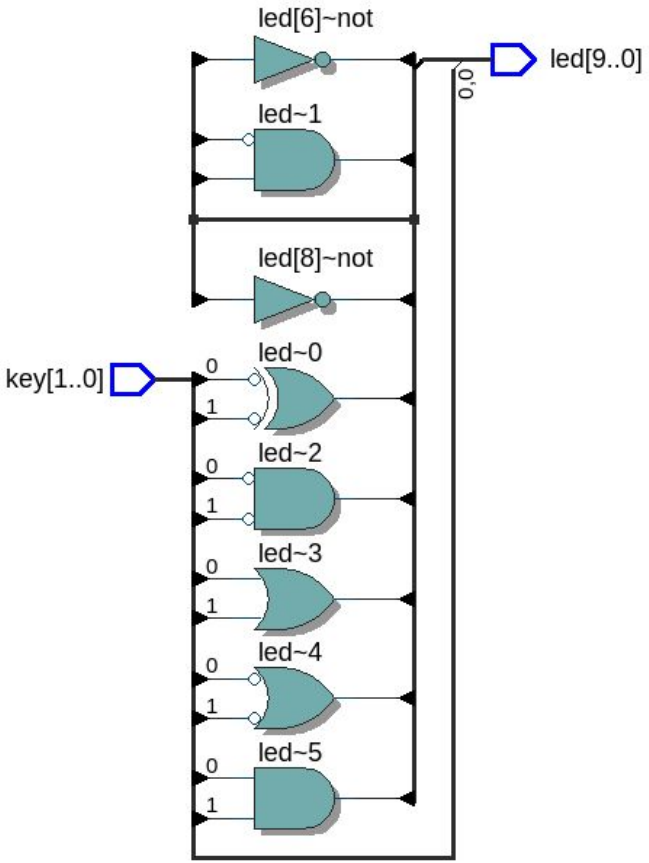
Page: 1 of 1

Netlist Navigator

top

top:1

- Task window menu has a viewer of FPGA-independent schematics.
- It shows the representation of your design after RTL analysis,
- but before mapping RTL to specific cells available in this particular FPGA



100% 00:00:01

File Edit View Tools Window Help

Search altera.com

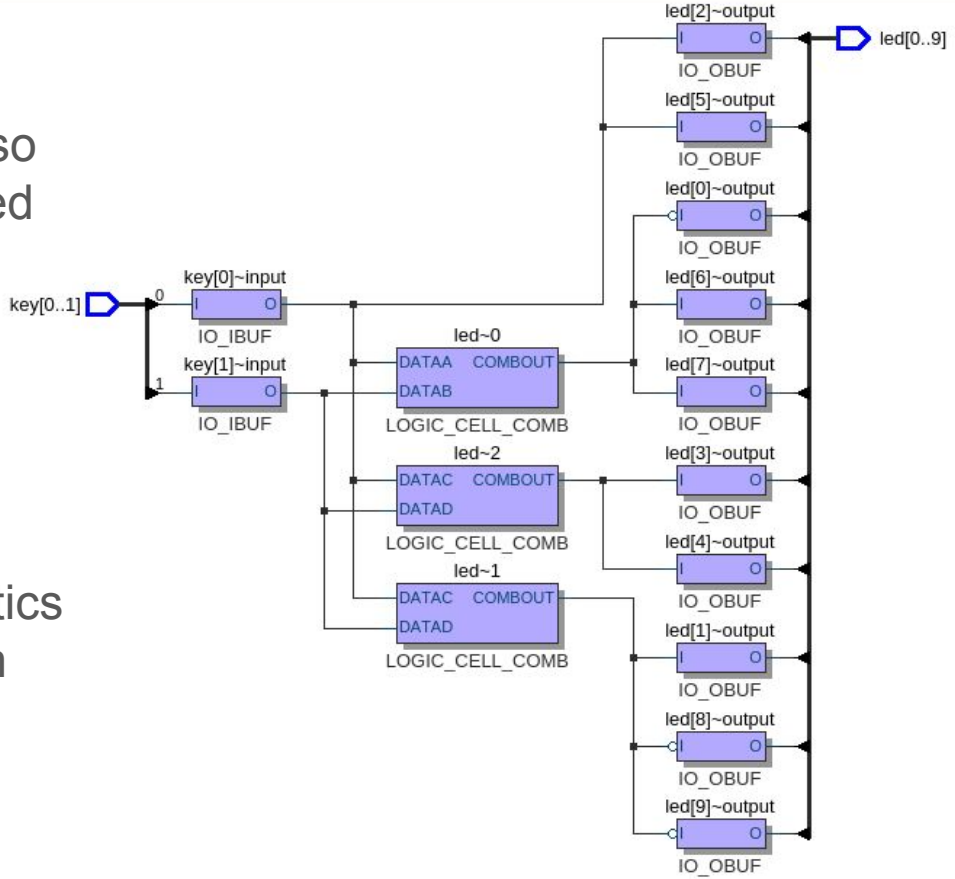
Page: 1 of 1

Netlist Navigator

top

top:1

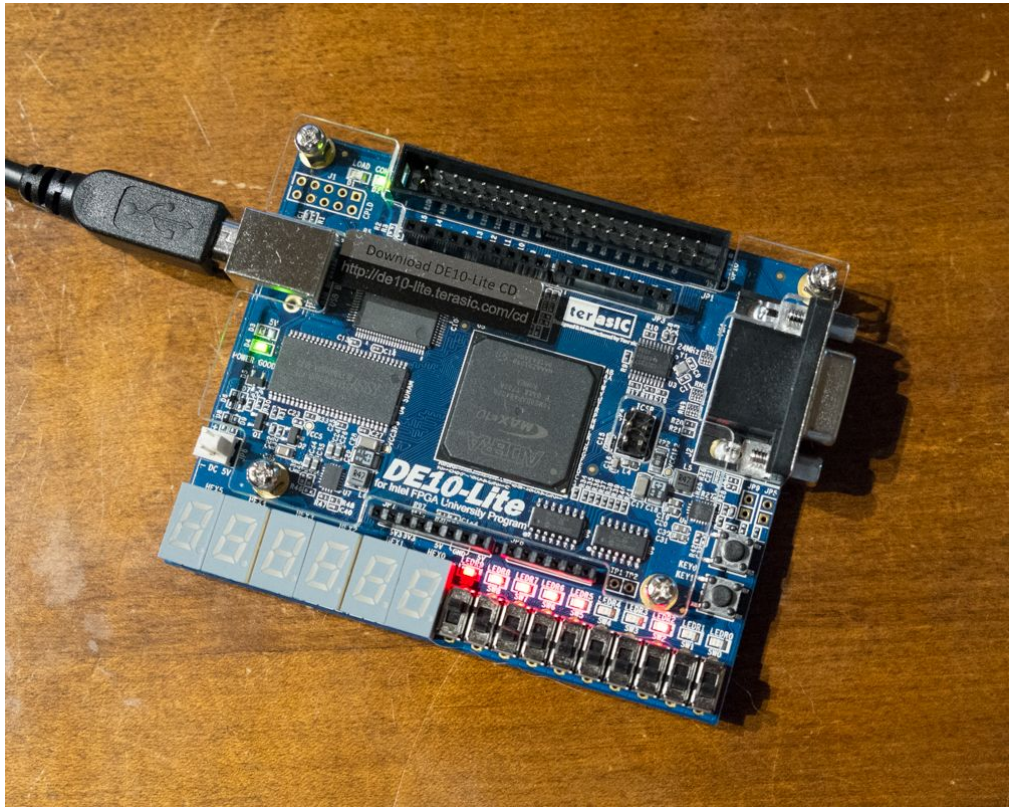
- Task window menu also has a viewer of mapped schematics,
- where all your logic is mapped to blocks that directly correspond to cells in target FPGA.
- Compare this schematics to the schematics from the previous slide.



100% 00:00:00

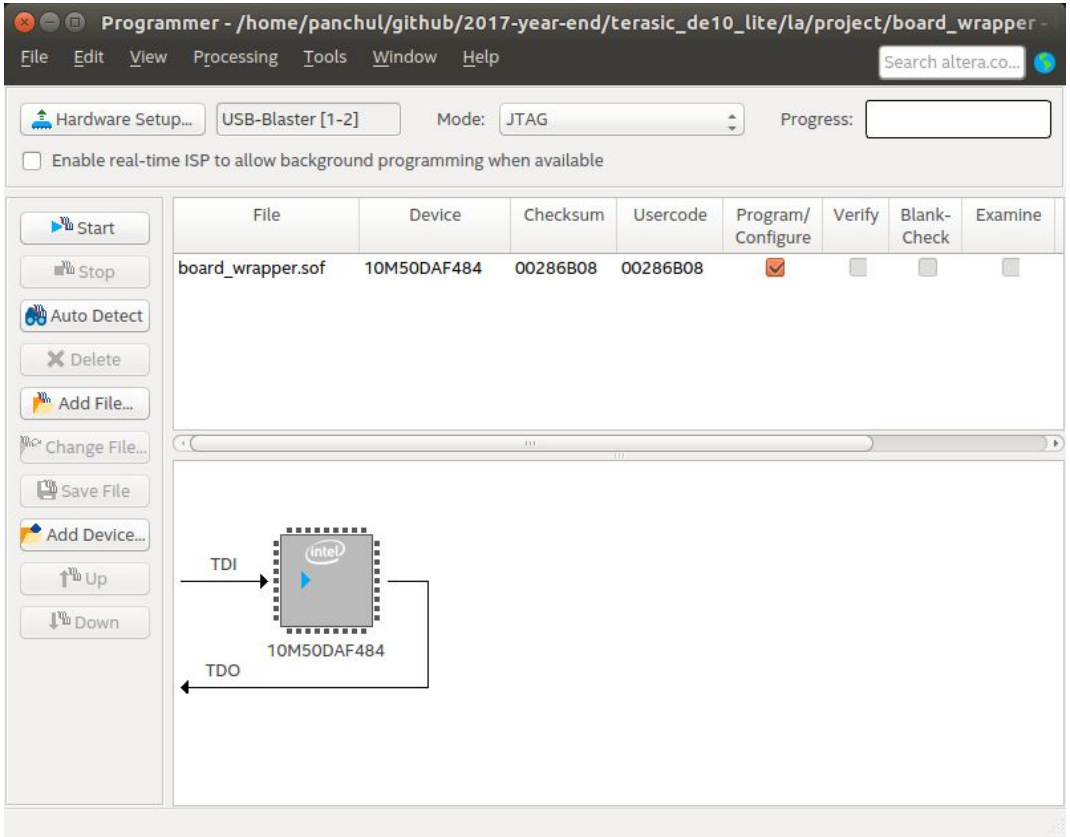
Terasic DE10-Lite FPGA board

- Inexpensive: $85
- Academic price $55
- Intel FPGA MAX 10 10M50DAF484C7G
- 50K cells, enough for designs with CPU cores like MIPSfpga
- On-board 64MB SDRAM
- Connects directly to PC for configuration



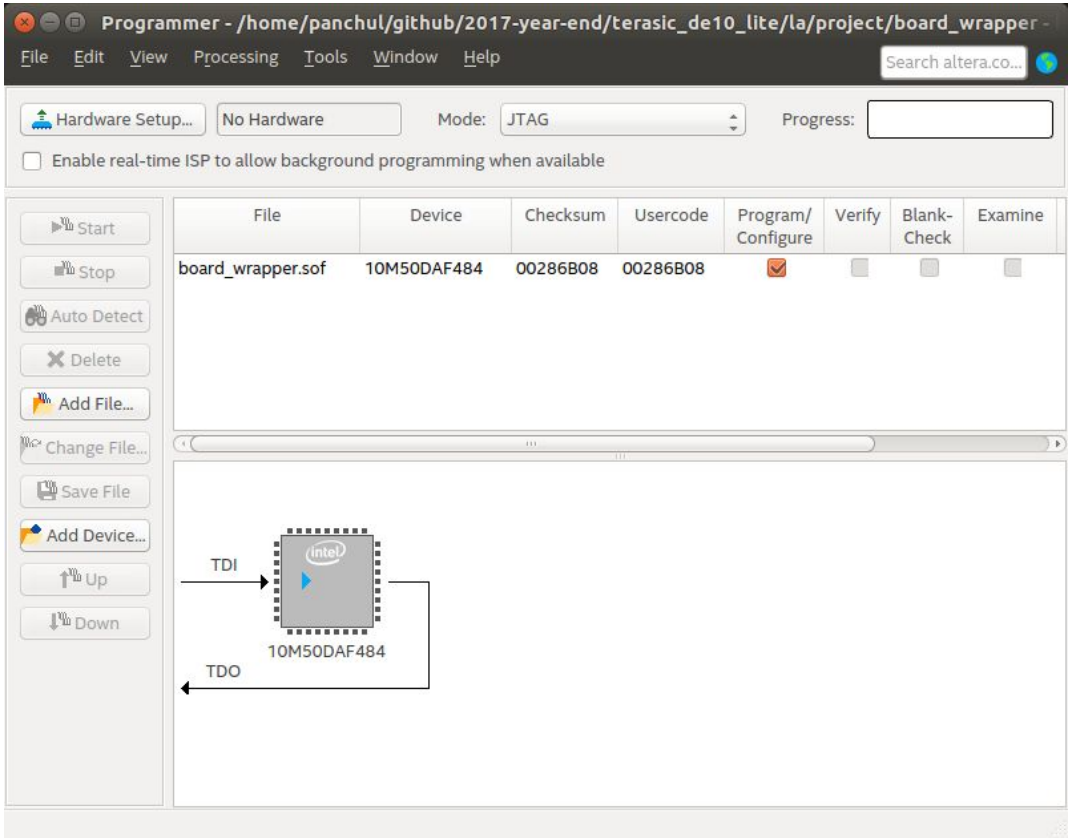
Configure FPGA on the board - 1

- Find and click on “Program Device” in Tasks window
- Programmer window should appear
- If you see “USB-Blaster” right from “Hardware Setup”, the board is connected and found by software
- In this case just press “Start” button



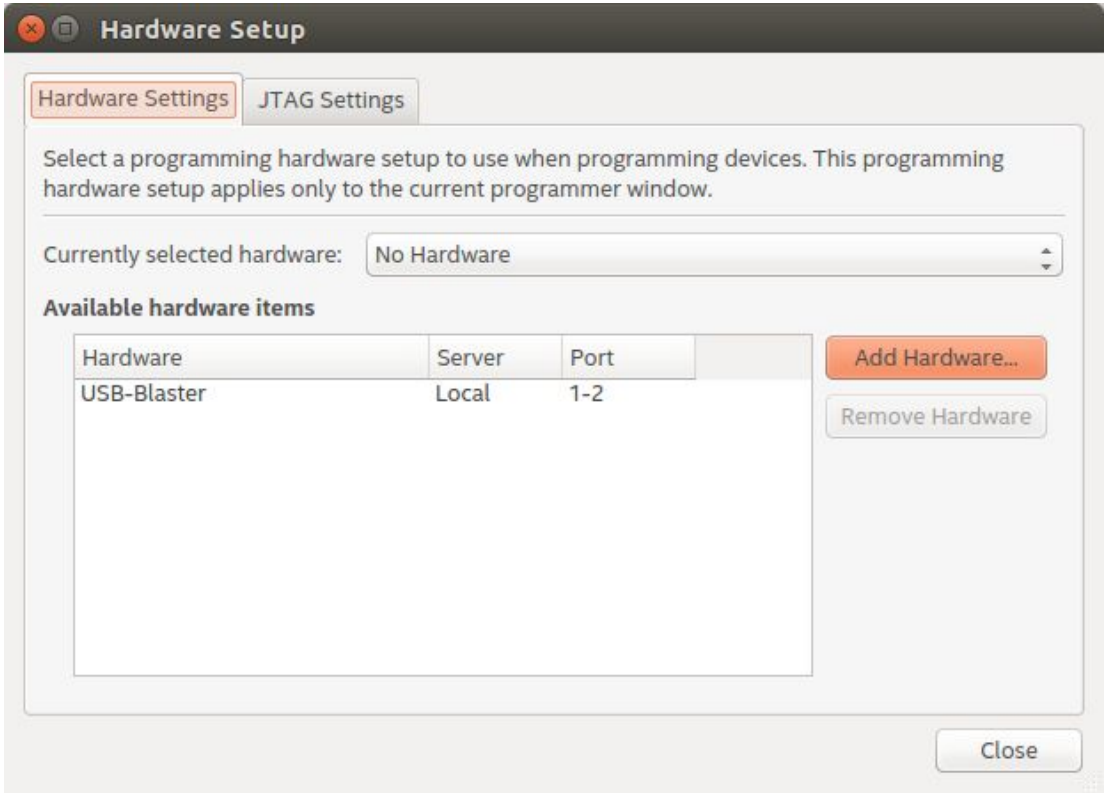
Configure FPGA on the board - 2

- If you see “No hardware”, press “Hardware Setup”



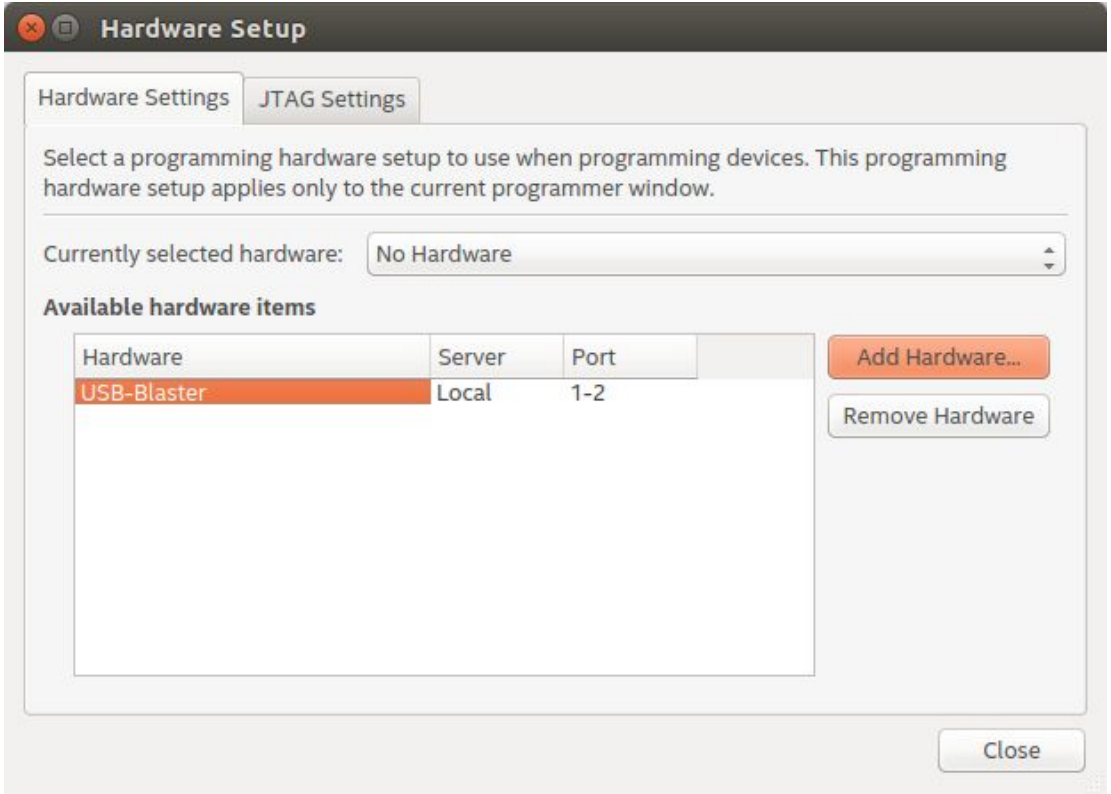
Configure FPGA on the board - 3

- “Hardware Setup” window should appear with “USB-Blaster” listed under “Hardware”.
- If nothing is listed, check your USB cable connection.
- If the board is connected but nothing is listed, install USB-Blaster driver or resolve driver conflict.



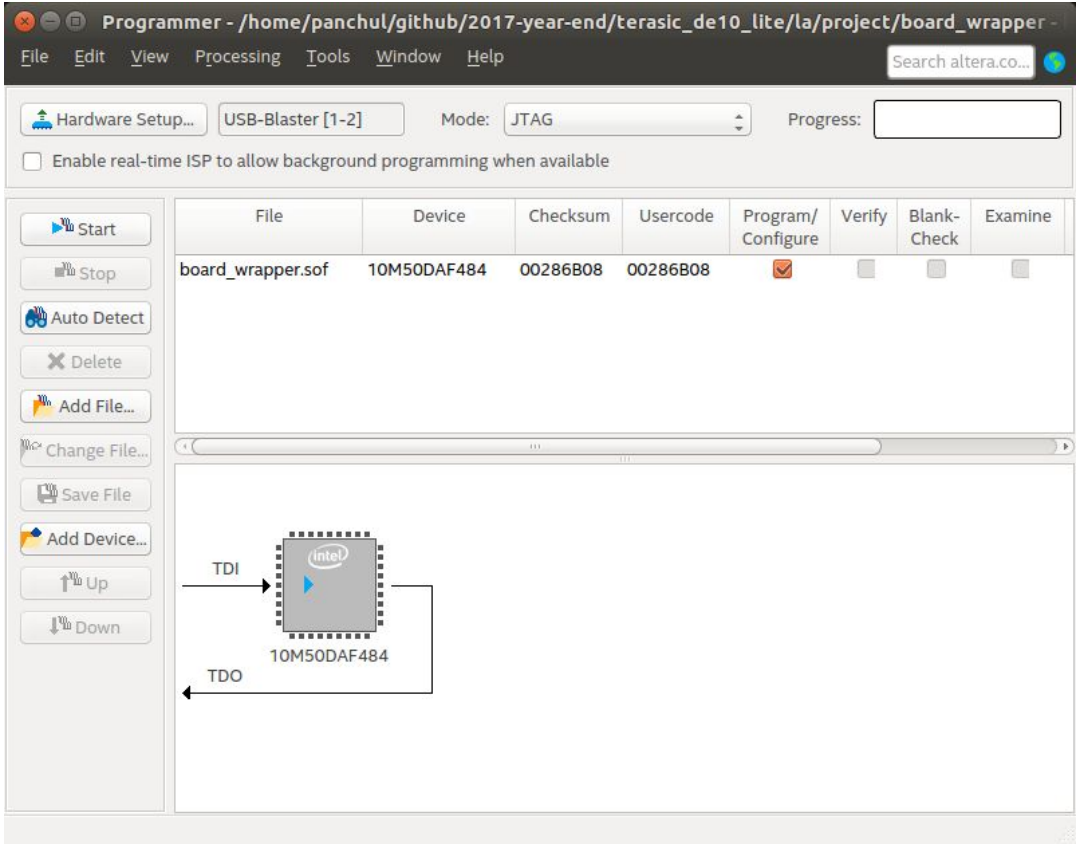
Configure FPGA on the board - 4

- Double-click on “USB-Blaster”, then press “Close”



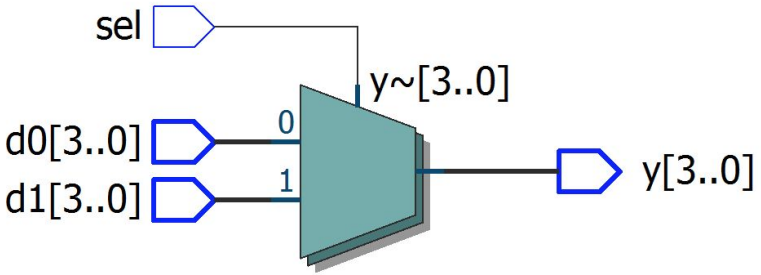
Configure FPGA on the board - 5

- Now the hardware setup problem should be resolved
- Just press “Start” button and see the “Progress” indicator becoming green.
- If “Progress” got stuck, unplug USB cable, plug it again and repeat.



Add a new combinational element - MUX

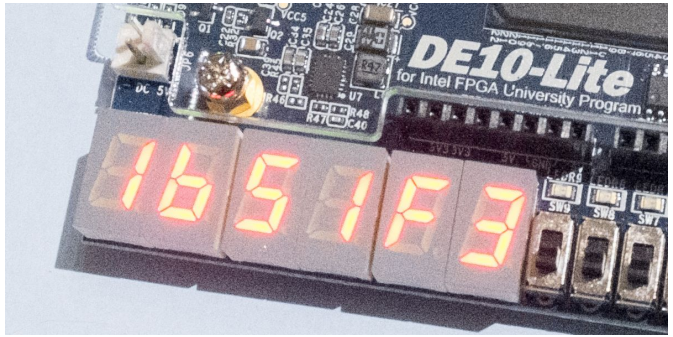
- Multiplexers, or muxes, implement the selection between values in hardware.
- The simplest way to infer mux in hardware is to use ternary ?: operation.
- Modify the design with a mux, and re-run synthesis and FPGA configuration.

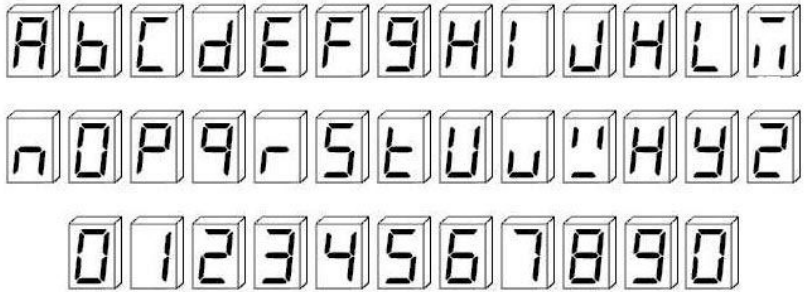


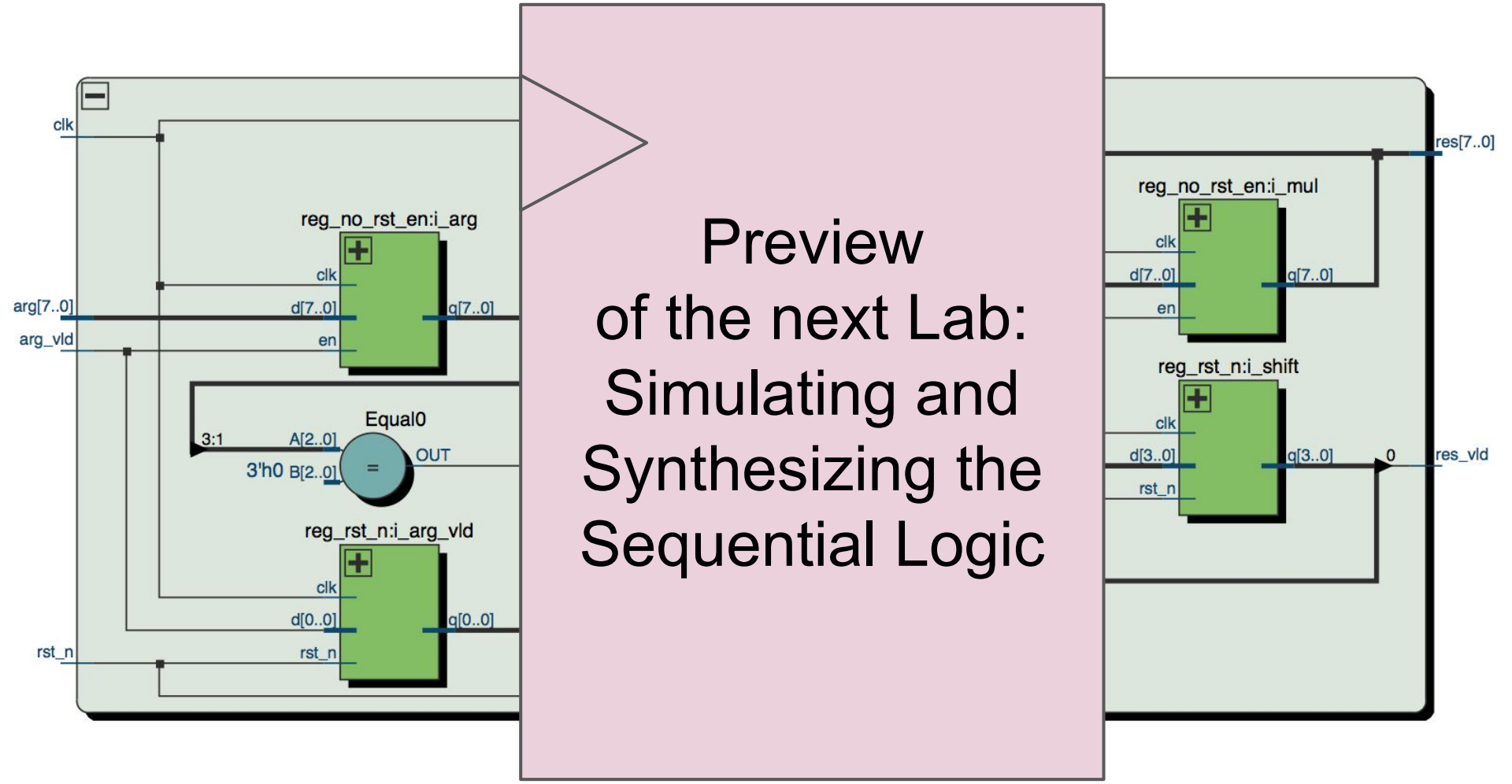

```
module mux_2_1
(
    input  [1:0] d0,
    input  [1:0] d1,
    input        sel,
    output [1:0] y
);

    assign y = sel ? d1 : d0;

endmodule
```

Add new signals to expand the design with MUXes

- Two keys are not enough to demonstrate a mux because it is a ternary operation. You need to add news signals, connected to buttons and switches on the board
- You can also use on-board 7-segment display to output digits or even letters
- Get the full .qsf file for DE10-Lite board from https://github.com/yuri-panchul/2017-year-end/tree/master/terasic_de10_lite

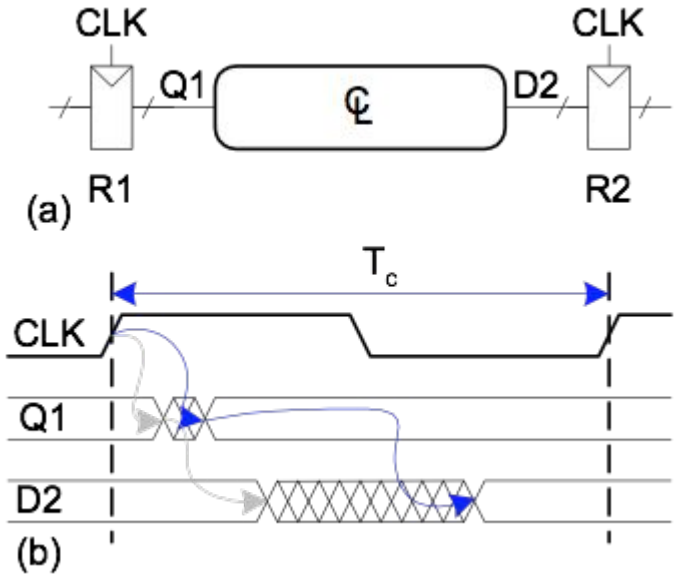





clk
reg_no_rst_en:i_arg
clk
arg[7..0]
d[7..0]
q[7..0]
arg_vld
en
Equal0
3:1
A[2..0]
OUT
3'h0
B[2..0]
=
reg_rst_n:i_arg_vld
clk
d[0..0]
q[0..0]
rst_n
rst_n
Preview
of the next Lab:
Simulating and
Synthesizing the
Sequential Logic
res[7..0]
reg_no_rst_en:i_mul
clk
d[7..0]
q[7..0]
en
reg_rst_n:i_shift
clk
d[3..0]
q[3..0]
0
res_vld
rst_n

Computation in combinational logic is not instant

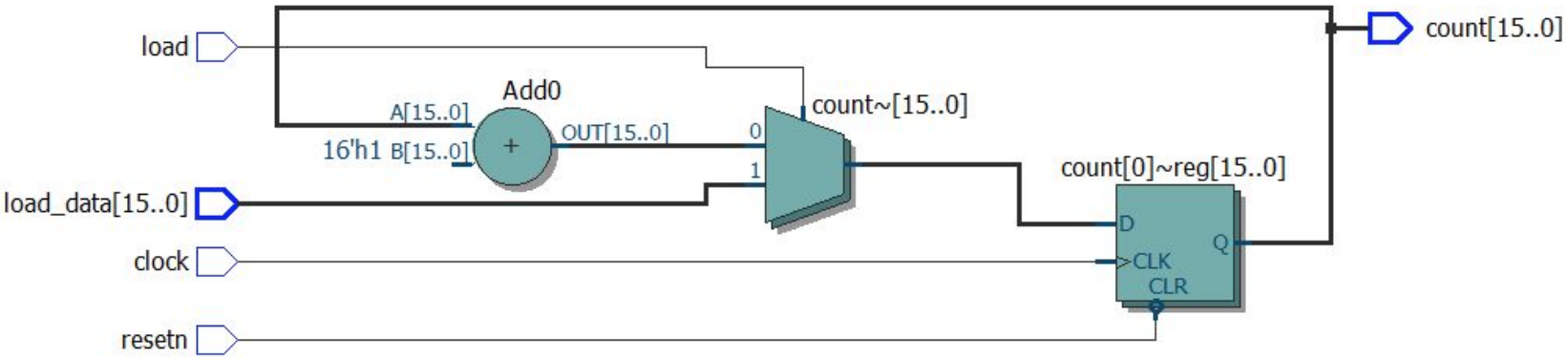
- It takes time for electric signal to propagate.
- Before the results are ready, the outputs may contain random values.
- How to find when the results are ready?
- We can synchronize the computations with a special signal called clock.

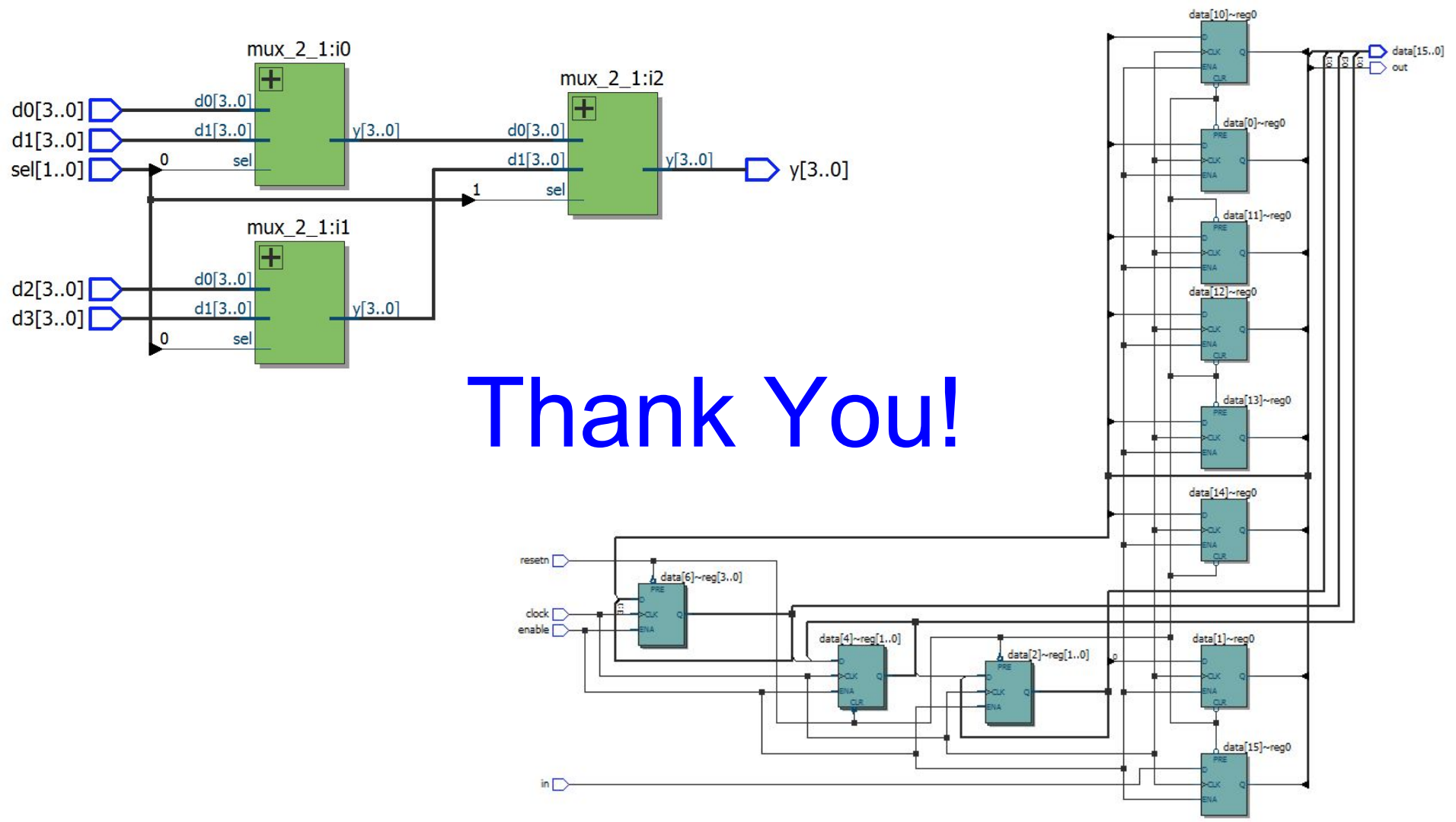


The picture is from Digital Design and Computer Architecture, 2nd Edition by David Harris and Sarah Harris. Elsevier, 2012

Clocks are useful not only to synchronize outputs

- Circuits synchronized with clocks are called sequential.
- Sequential circuits can also have memory and perform iterative computations.
- We will learn about sequential circuits in Lab 2.




mux_2_1:i0
mux_2_1:i1
mux_2_1:i2
d0[3..0]
d1[3..0]
sel[1..0]
d2[3..0]
d3[3..0]
y[3..0]
sel
Thank You!
data[10]~reg0
data[0]~reg0
data[11]~reg0
data[12]~reg0
data[13]~reg0
data[14]~reg0
data[1]~reg0
data[15]~reg0
data[15..0]
out
resetn
clock
enable
in
data[6]~reg[3..0]
data[4]~reg[1..0]
data[2]~reg[1..0]