

FPGA tutorial

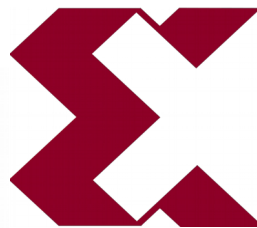
Lecture 2

Monday 07.09.2015 – 16:00

Jochen Steinmann

BND SCHOOL

Belgian Dutch German graduate school in particle physics



XILINX®



Physics
Institute III B

RWTHAACHEN
UNIVERSITY

1st Project – Summary

- Start VIVADO
- First knowledge about Verilog
 - module
 - datatypes & levels
 - logical operators
- connect inputs with outputs
 - using combinatorical logic
 - assign
 - constraint file (XDC)



1st Project – Solution

- There are always multiple ways how to realise a FPGA – Project
- Many solutions can be synthesized in the same way and result in the same configuration



1st Project – Solution

```
module simple_top(  
  input [15:0] sw,  
  output [15:0] led  
);
```

```
  simple_logic log1(sw, led);
```

```
endmodule
```

```
module simple_logic(  
  input [15:0] switch,  
  output [15:0] LED  
);
```

```
  assign LED = switch;  
endmodule
```

You can also skip the submodule!



2nd Project – doing Logic

Now we want to do some logic

- $LED0 = SW0 \& SW1 \leftarrow AND$
- $LED1 = SW0 | SW1 \leftarrow OR$
- $LED2 = SW0 \wedge SW1 \leftarrow XOR$
- $LED3 = \sim LED0 \leftarrow NOT$

just change simple_logic.v and keep the top module from Exercise1!

$LED4 - LED15 = SW4 - SW15$

Hint: merge single Bits to a Bus!



2nd Project – Solution

```
module simple_logic(  
    input [7:0] switch,  
    output [7:0] LED  
);
```

without joining all together to a bus again.

```
    assign LED[0] = switch[0] & switch[1];  
    assign LED[1] = switch[0] | switch[1];  
    assign LED[2] = switch[0] ^ switch[1];  
    assign LED[3] = ~ LED[0];  
    assign LED[15:4] = switch[15:4];  
endmodule
```

Alternative: → Same Function

```
assign LED = {switch[15:4], ~(switch[0] & switch[1]), switch[0] ^ switch[1], switch[0] | switch[1]};
```



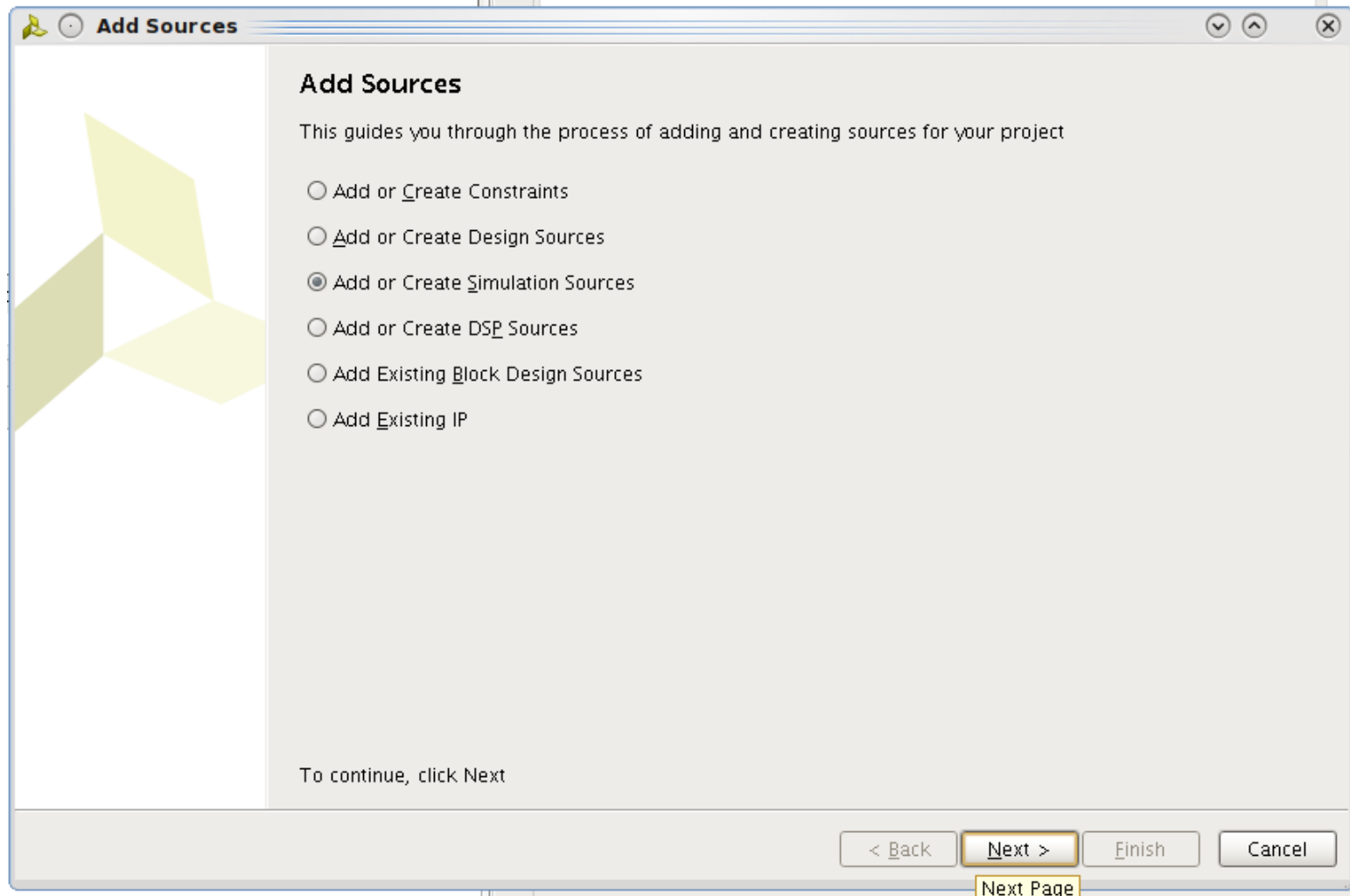


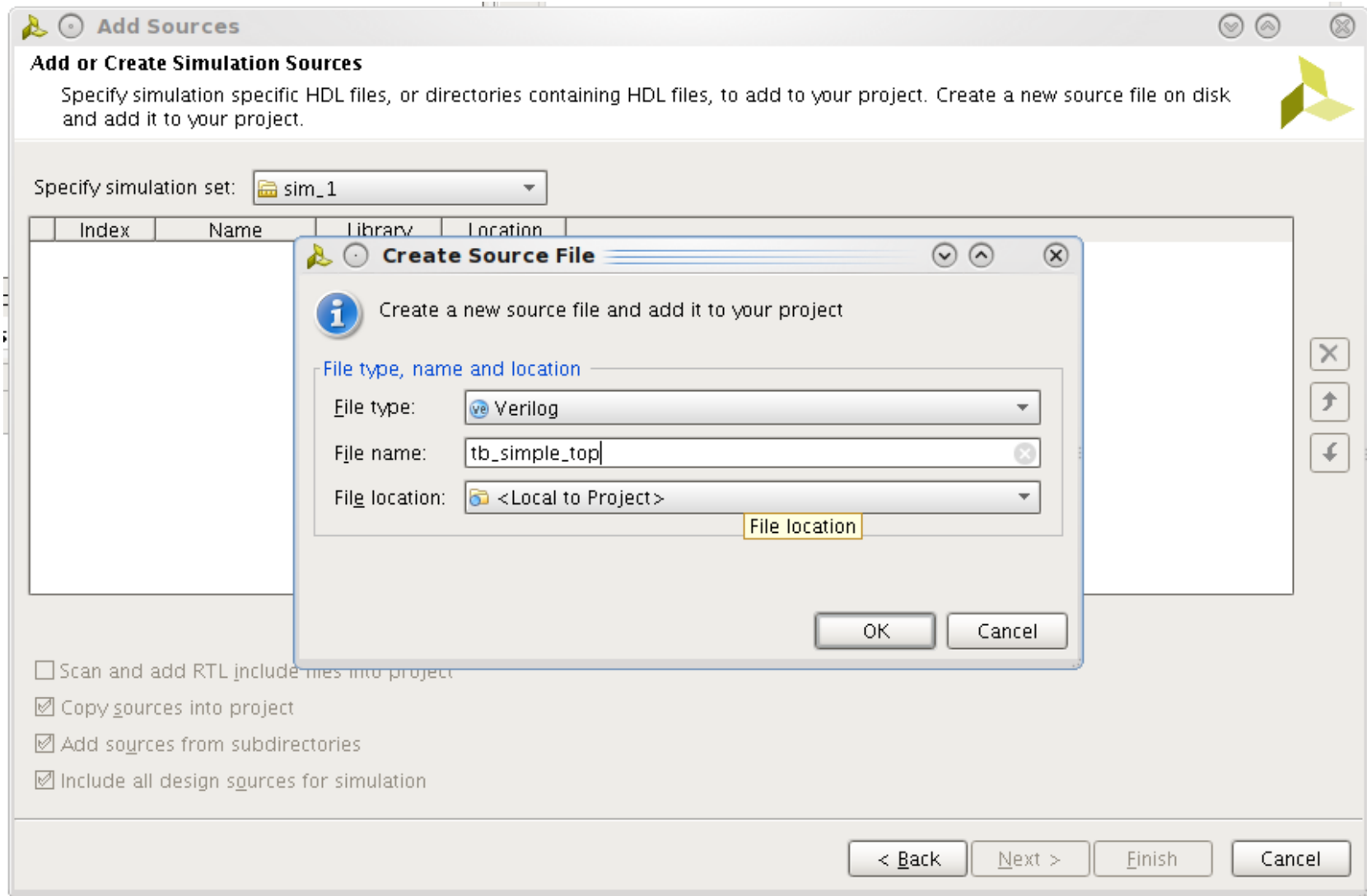
Simulation

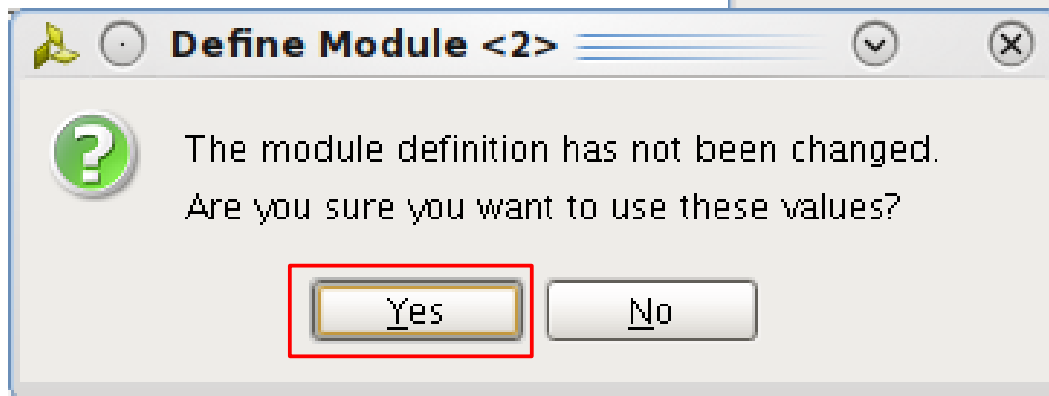
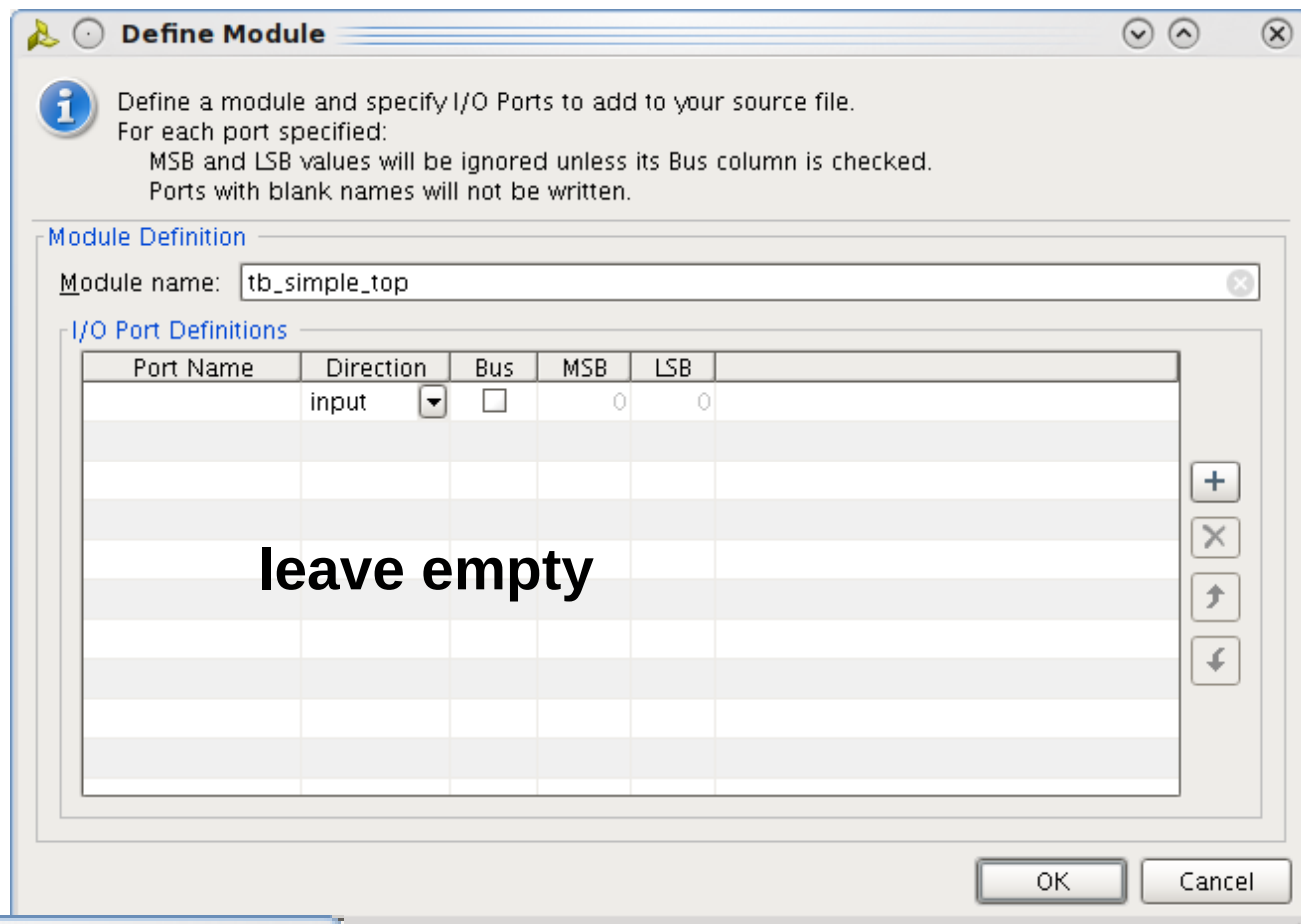
- Very useful to test Design
 - can also display “inner” signals



Add new Source







Content of Testbench File

```
22
23 module tb_simple_top(); dummy module
24
25     reg SW0; create a register
26
27     initial begin
28         #100; delay 100 time units
29         SW0 = 1; set SW0 to 1
30         #100;
31         SW0 = 0;
32     end
33
34     Module under Test
35     simple_top simpleTop(SW0, LED0);
36 endmodule
37
```

sometimes also called:
stimulus

for the BND01 example:
module tp_BND01();

```
reg [15:0] sw;
wire [15:0] led;
```

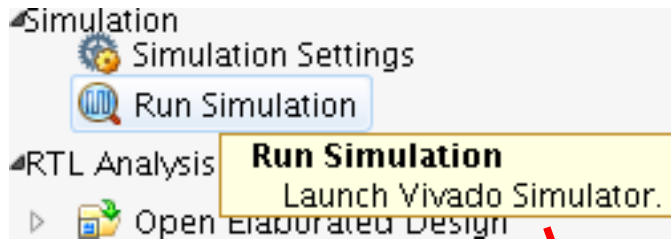
```
initial begin
    #100;
    sw = 'hFFFF;
    #100;
    sw = 'b0;
end
```

```
BND01 dut(sw, led);
endmodule
```

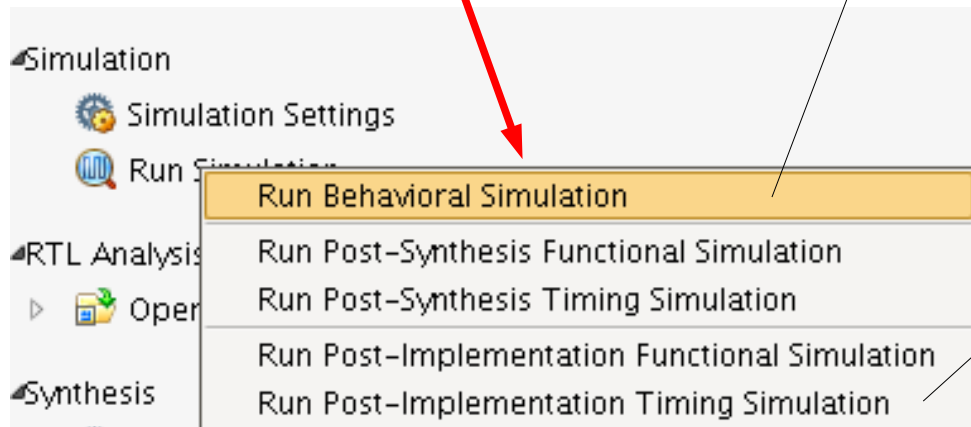
for normal, we should create a wire for LED0!
When you use a BUS you have to create a wire
for the signal!



Start Simulation



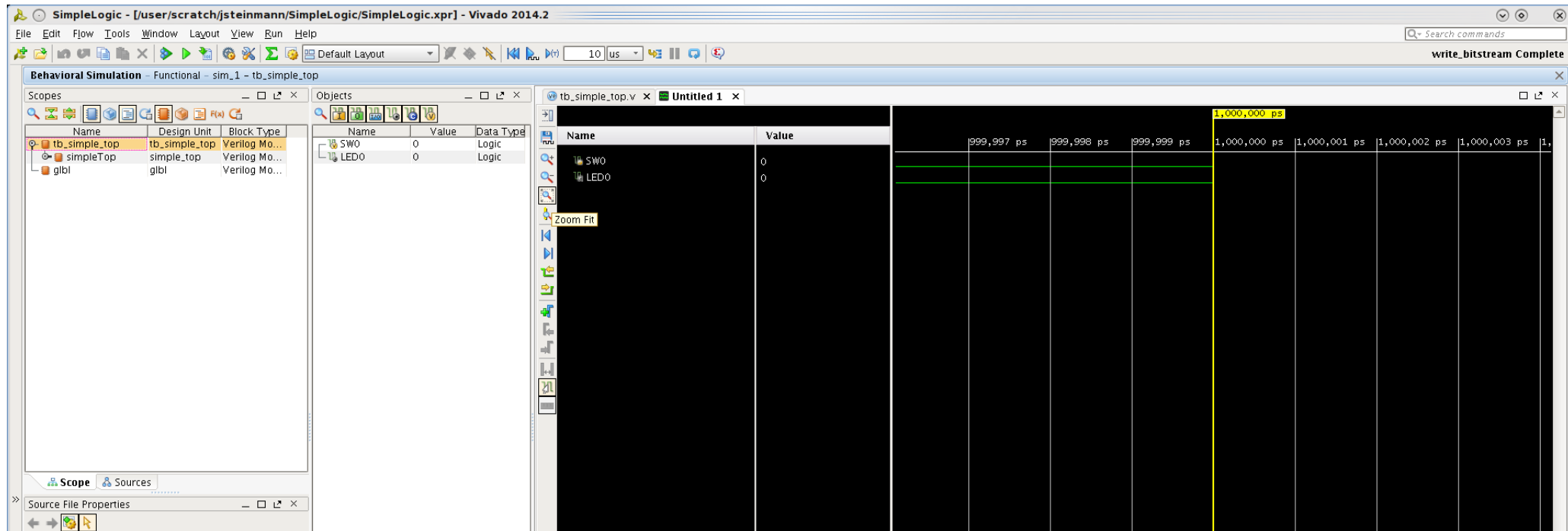
simulate just the Verilog file, without FPGA



can simulate the timing on the FPGA chip. You can observe the gate delay times etc.

takes a lot of time





Zoom to fit





undefined

logic 1

logic 0



Clock



Clock – Why?

- Until now only “Glue Logic”
 - simple logic only
 - output action is related to input
 - This “mode” is used for trigger generation:
 - $\text{trg} = (A \ \& \ B) \ || \ (A \ \& \ C)$
 - But if we want to introduce timing, e.g. dead-time after a trigger
 - we need to react on something but not the input
 - **CLOCK**



Use of Clock

- We want to let the FPGA doing thing for us
 - Memory storage RAM need a clock to refresh
 - Digital communication to other electronics or PC
- Delays
 - compensate cable length
 - artificial dead times → Trigger Veto
- Measurements of the time between two events:
 - Drift velocity measurement in a TPC
 - Time of Flight measurements to distinguish particles



Clock Requirements

- Requirements:
 - stable frequency
 - stable phase to other clocks
 - stable signal
 - reproducible
 - low temperature dependency
 - low dependency on environment

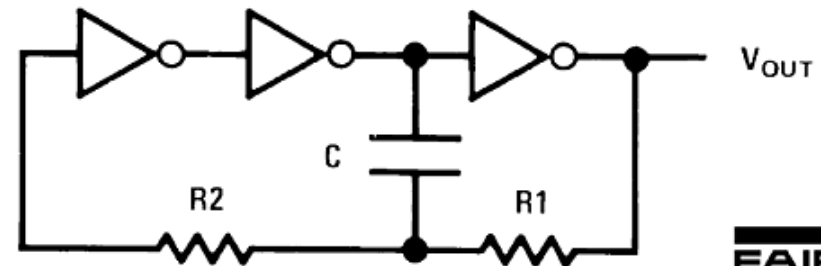
Most FPGAs do not have an internal clock generator!
Need external reference Clock generator



Clock generation

- Driven LC / RC resonator

- huge tolerances
- not very stable
- no high frequencies

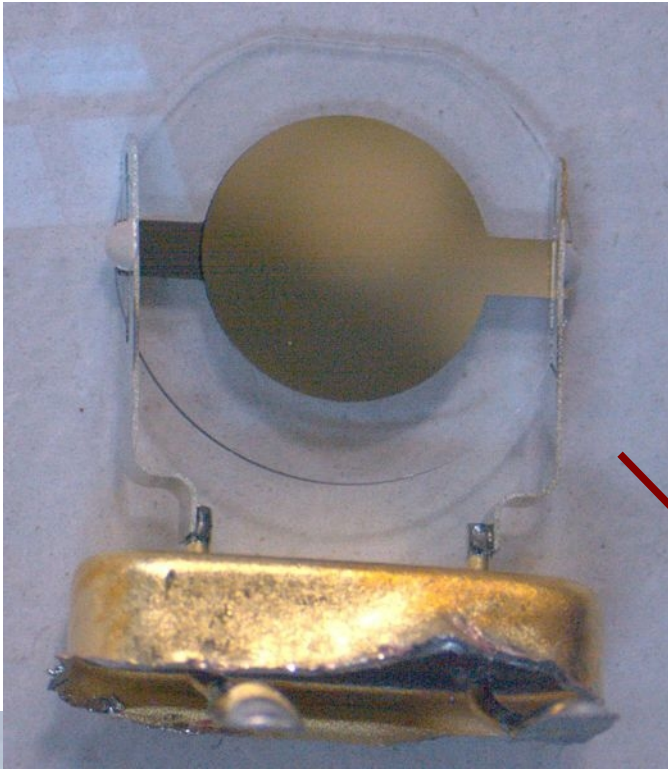


- Tuned Voltage Controlled Oscillator

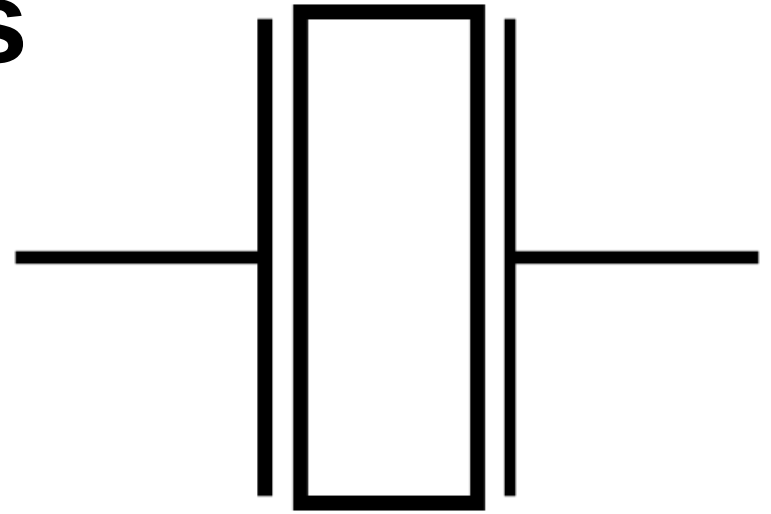
- can be easily tuned by a voltage

VCOs		RF Freq. (MHz)		V _{CC} (V)	Footprint (mm x mm)	Budgetary Price
		min	max			See Notes
<input checked="" type="checkbox"/>	MAX2750 2400MHz to 2500MHz Monolithic VCO with Integrated Tank and Low-Power Shutdown Mode.	2400	2500	2.7 to 5.5	3.0 x 4.9	\$1.18 @1k
<input checked="" type="checkbox"/>	MAX2751 2120MHz to 2260MHz Monolithic VCO with Integrated Tank and Low-Power Shutdown Mode.	2120	2260	2.7 to 5.5	3.0 x 4.9	\$1.24 @1k
<input checked="" type="checkbox"/>	MAX2752 2025MHz to 2165MHz Monolithic VCO with Integrated Tank and Low-Power Shutdown Mode.	2025	2025	2.7 to 5.5	3.0 x 4.9	\$1.24 @1k

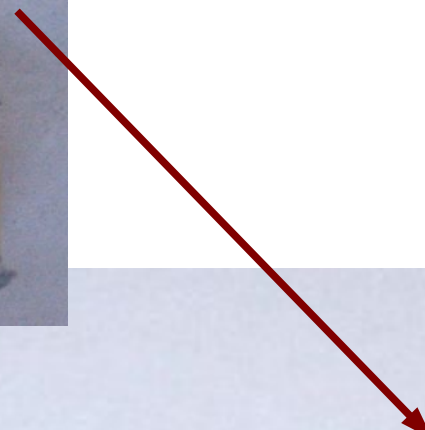
Crystals



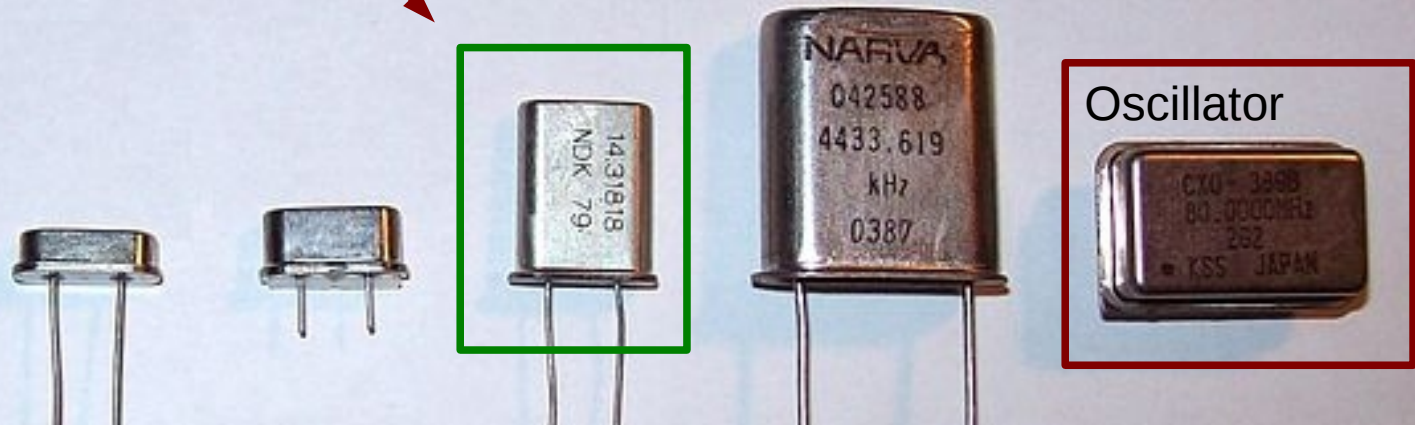
HC49 Inside



Frequencies up to 200 MHz



Many different housings

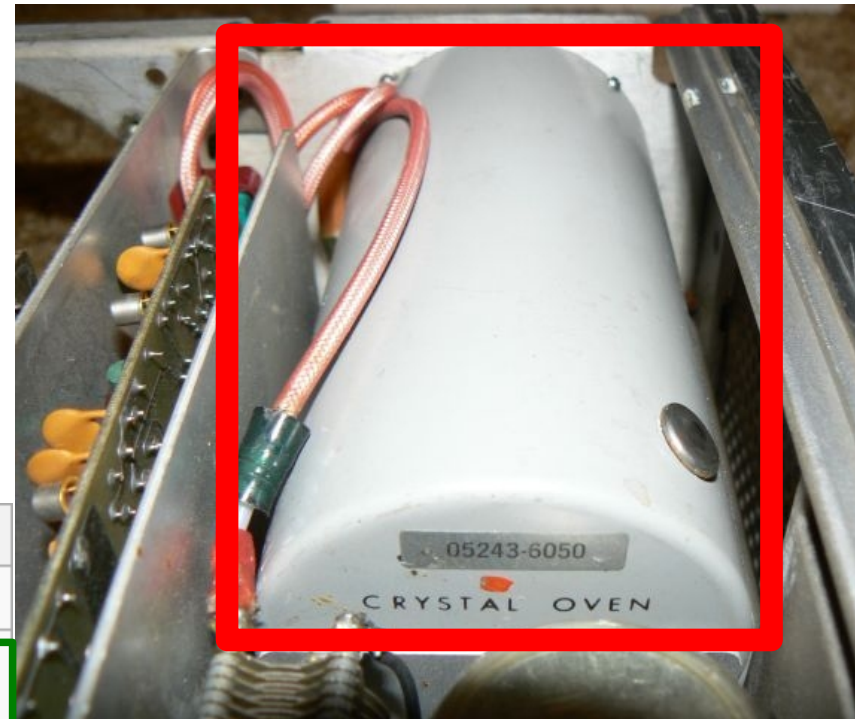


Oscillator

Temperaturesensor ?

- Precise Oscillators
 - keep Temperature constant
 - is used for high precision clock sources

$$T = \text{const.} \rightarrow f = \text{const.}$$

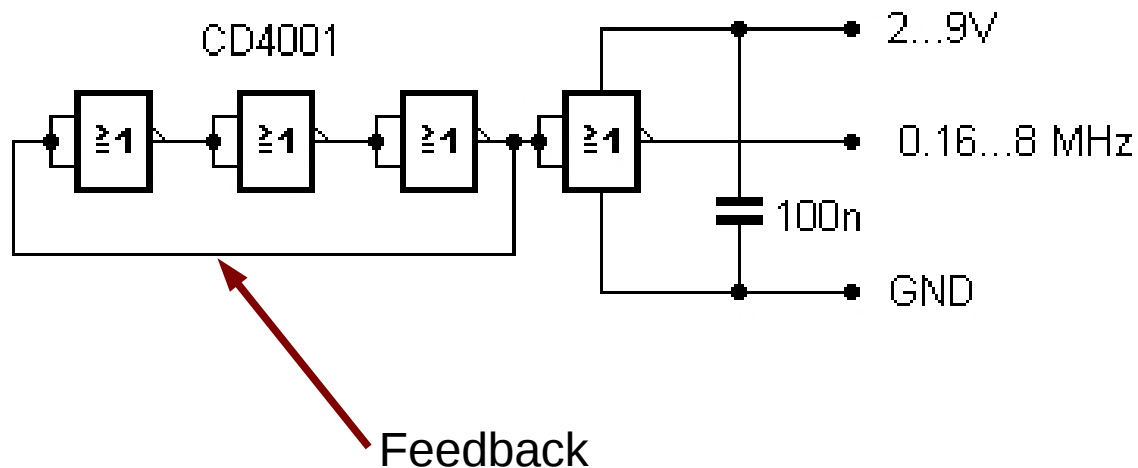


Oszillatortyp	Genauigkeit	Alterung / 10 Jahre
Quarzoszillator	10^{-5} bis 10^{-4}	10 bis 20 ppm
Quarzofen (OCXO)		
5 bis 10 MHz	2×10^{-8}	2×10^{-8} bis 2×10^{-7}
15 bis 100 MHz	5×10^{-7}	2×10^{-6} bis 11×10^{-9}
Atomuhr (Cs)	10^{-10} bis 10^{-11}	10^{-12} bis 10^{-11}
Global Positioning System (GPS)	4×10^{-8} bis 10^{-11}	10^{-13}



other clock sources

- resonator made from single gates:



$$T = 2n \cdot t_D \leftarrow 2\text{ns}$$
$$f = \frac{1}{T} = \frac{1}{2n \cdot t_D} \rightarrow 83 \text{ MHz}$$

3

Not very stable
Difficult to compute



More speed ...



... higher frequency!

Stable high speed clock...

- Crystals and Oscillators are only available up to $\sim 200\text{MHz}$!
- What if we need $\sim 1\text{GHz}$?

**All you need is a Voltage Controlled Oscillator
and a slower reference clock!**

Frequency Multiplication!



Howto?

- Constraints for Multiplication
 - After n Periods the phase must be the same!
- Possible for integer steps only!
 - for fractional Multiplication a divider has to be used!
 - Example: $100 \text{ MHz} = 3 / 2 * 66,6 \text{ MHz}$

Output

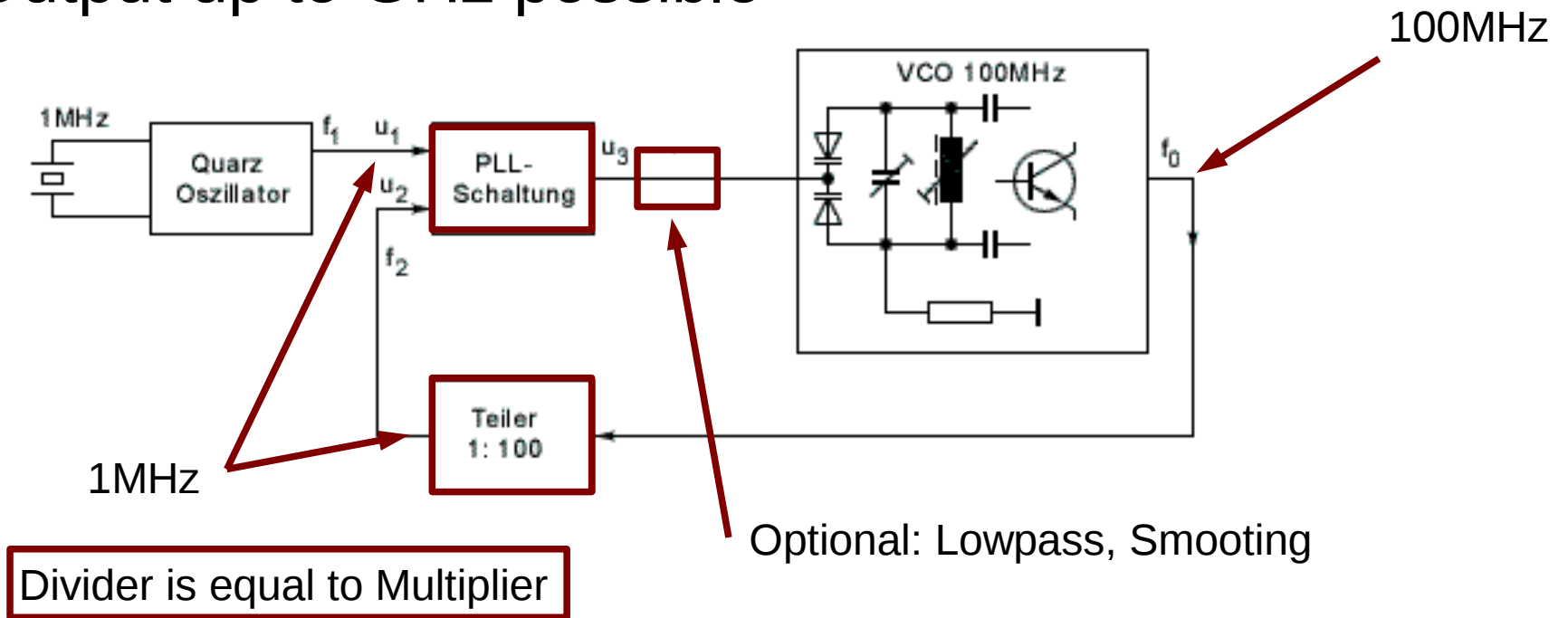
Input

multiply by 3 and divide by 2

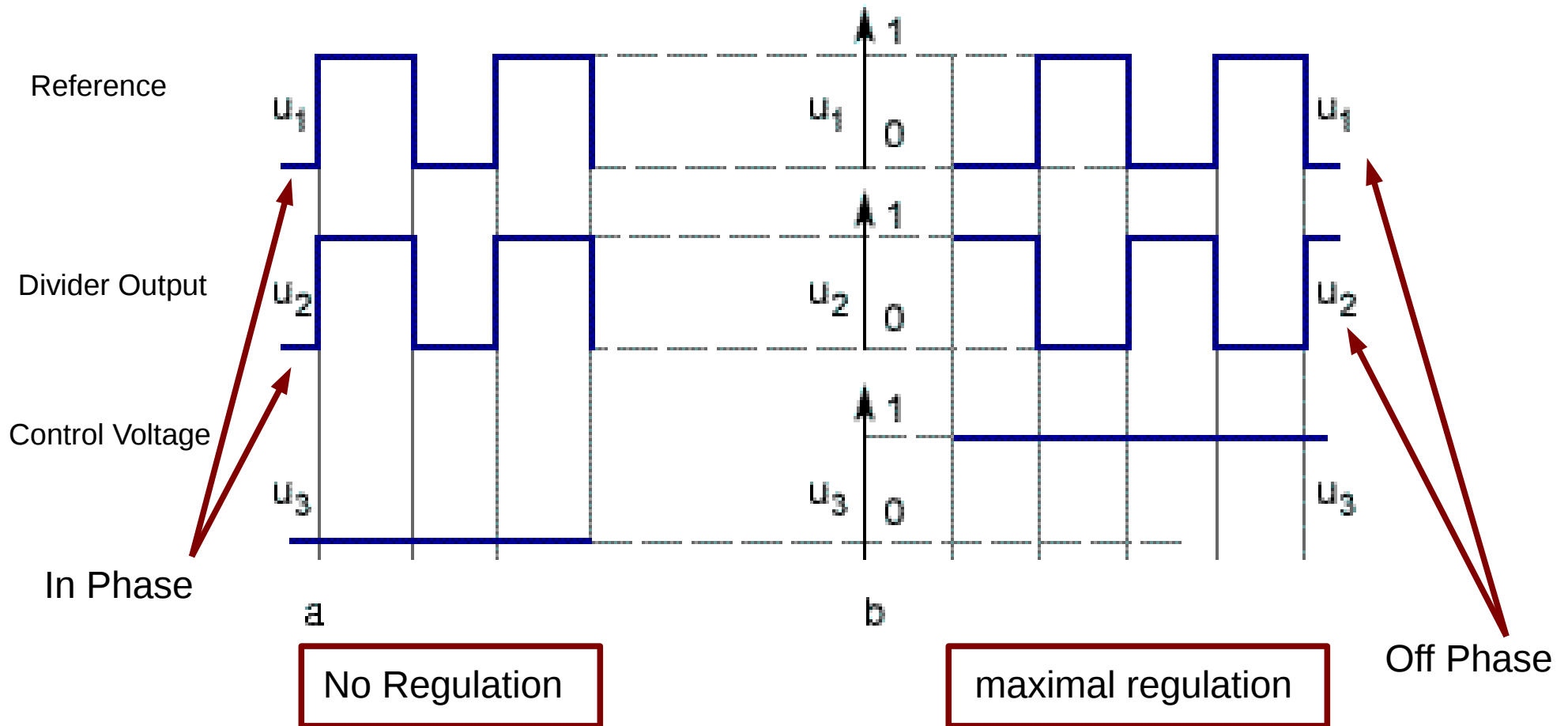


Phase Locked Loop (PLL)

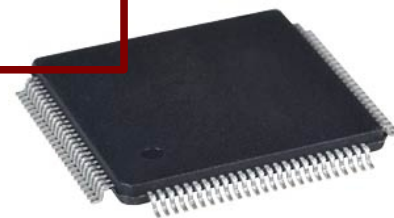
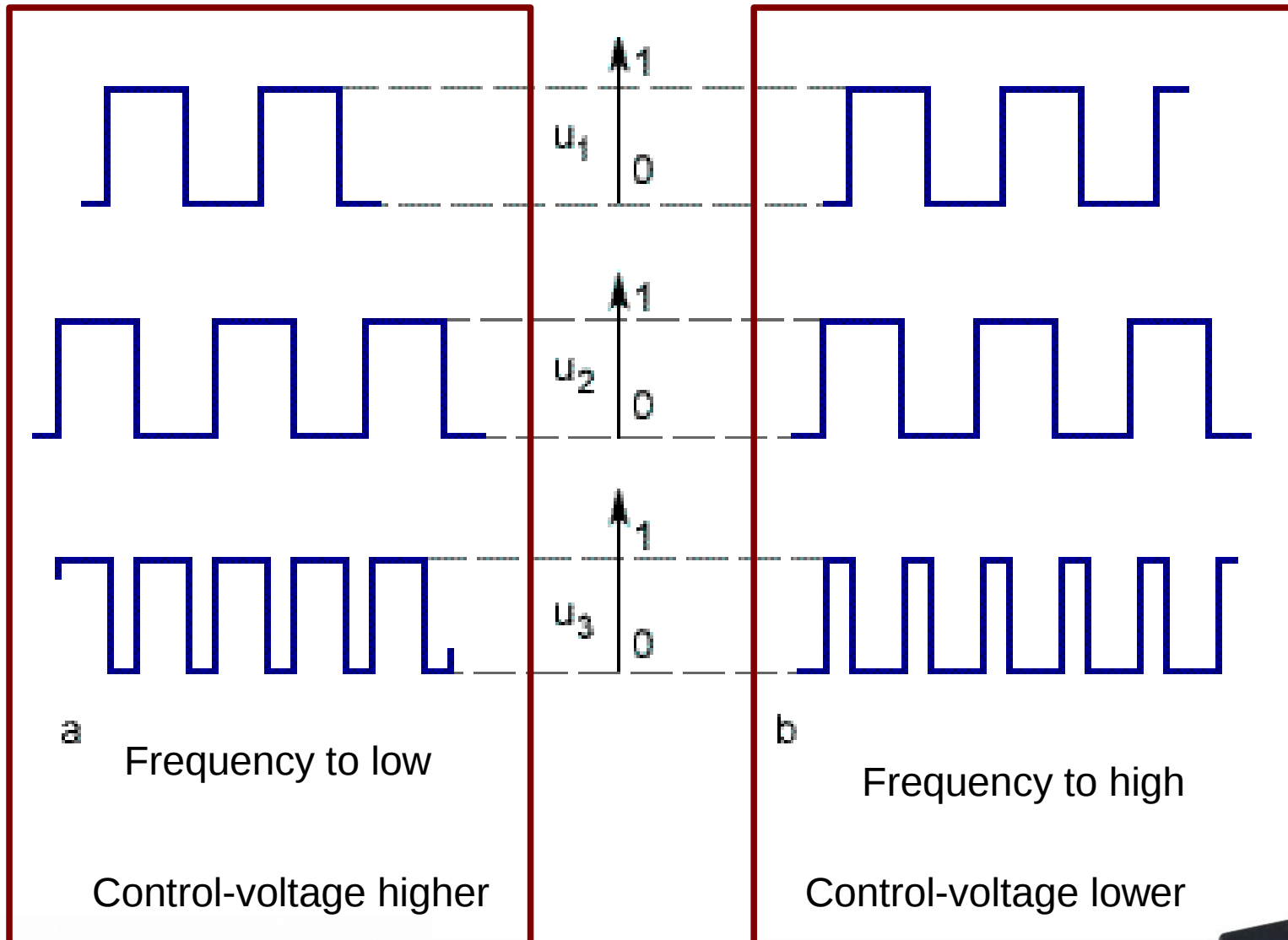
- Comparison of the phase with a reference oscillator:
 - Output up to GHz possible



Function I



Function II



Function III & FPGA

- PLL is stable operating, when the output voltage is stable. ($u_3 = 0$)
- There are so called Lock-In detectors available, which are able to detect this case.
- FPGAs have build in PLL blocks, which can be used to create a stable clock.
 - mainly used to derived secondary (slower) clocks



External PLLs



PLLs/PLLs with Integrated VCO

	V_{SUPPLY} (V)	Oper. Freq. (MHz)	Oper. Freq. (MHz)	Differential RF Outputs	RMS Jitter (ps RMS)	Oper. Temp. (°C)	Budgetary Price
		min	max		typ		See Notes
Sort by: Part Number: ▲▼ ▲▼ ▲▼ ▲▼ ▲▼ ▲▼ ▲▼ Default = Newest First							
<input checked="" type="checkbox"/> MAX2871 NEW! 23MHz to 6000MHz Fractional/Integer-N Synthesizer/VCO	3.0 to 3.6	23.5	6000	2	0.2	-40 to +85	\$6.44 @1k
<input checked="" type="checkbox"/> MAX2880 12.4GHz Fractional-N PLL	3.0 to 3.6	250	12400	-	0.14	-40 to +85	\$7.35 @1k
<input checked="" type="checkbox"/> MAX2870 23.5MHz to 6000MHz Fractional/Integer-N Synthesizer/VCO	3.0 to 3.6	23.5	6000	2	0.25	-40 to +85	\$6.53 @1k

- tunable over a wide frequency range
- up to 12.4 GHz



Clock in Experiments

- Stability is observed by a slower “reference Clock” and a counter
 - Possible reference clock sources
 - GPS → many GPS modules supply a 1 PPS pulse
 - Atomic Clocks → DCF77
 - Rubidium Clock Generators which can also supply any Master Clock signal



GPS as Reference

- Often used in experiments
 - OPERA
 - T2K
- Precise clock output by
 - Rubidium Reference Clock → 10MHz
- Huge accuracy and precision

Helpful, when timestamps should be matched over large distances.



Clock inside FPGA

- Routing of clock inside the FPGA is a challenge for VIVADO
- Requirements:
 - small clock delay
 - at every point inside the FPGA
 - phase must match signal
 - most modules latch their data on rising or falling edge
- There is a special high speed net inside the FPGA, which is only used for clock distribution



FPGA Clocking Challenge

Virtex™ -7 2000T

- 2.000.000 Logic Cells
- 6.800.000.000 Transistors
(Switching Freq > 3.8GHz)
- DSP48E1 Slices with 741MHz global clock
- 12.5Gb/s Serial Transceivers
- 1066MHz Fmax Analog PLLs

The internal PLL can not cover all requirements.
This fact has to be considered and calculated before the PCB design is done.

Largest Problem: Jitter

Applications:

PCIe GEN3 Jitter < 1ps
SDR Jitter < 300 fs
DDR3 Memory Jitter < 150ps



Clock-Distribution in FPGA



I/O Columns

CMT Columns

Clock Routing

CLB, BRAM, DSP Columns

GT Columns

Clock Management Tile

Gigabit Serial Transceiver



Clock routing in FPGA

- very complex topic
 - for our applications it is sufficient to trust VIVADO
 - we want to concentrate on the function and logic inside! keep it simple.



For those, who are interested in detail:

http://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf

Clock Domain Crossing

- Sometimes it is necessary to transfer data between different clock domains.
 - special requirements on the design
 - Use of storage elements:
 - FlipFlops
 - FiFos (auch LiFo, FiLo)
 - RAM, ...



Gated Clock

- If the clock is switched off for non used areas / modules, power can be saved.
- Clock switching must be done via special clock gates.
- **Never use a single AND-gate to switch off the clock.**
- If you need to use a gated clock, always have a look at the RTL and check for right implementation.



Why is clock gating problematic?

- When a simple clock gate is switched on
 - it is not done synchronous to the main clock
 - Very short pulses might appear → **Glitch**
- Runtime delays inside the gate
 - a phase difference between input and output



Some more ...

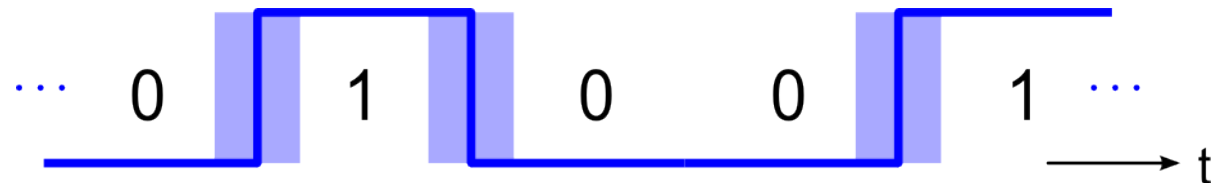
- If high speed frequencies are used, one has to think a bit about the design and how it should be realised.
- Data should be transferred on a defined edge of the clock (**posedge** or **negedge**)



Digital effects I

- Jitter

- Variation of a constant clock in time domain



- Glitch

- Change of output due to timing issues

$$Q = A \text{ and } B \text{ and } \mathbf{NOT} C$$

causes additional run_time

FPGAs will synthesize this simple equation without a glitch because all three signals are sent into one LUT



Verilog Parameter

- Many Code Snippets can be re-used for other applications
- But often the needs are slightly different
 - need to use 10 instead of 8 Bit
 - uses another clock frequency
 - wants a different output value

—————→ **Verilog code can be parameterized!**



Verilog Parameter

- inside module
 - output reg [DW-1:0] cnt
 - use parameter to define data width of cnt
 - parameter DW = 8;
 - define a parameter called 'DW'
- inside upper module
 - counter cnt(sw0, btnD, led);
 - instantiate module
 - defparam cnt.DW = 16;
 - now the output reg cnt is 16 bit wide
- Parameters are set during synthesis!



3rd Project - counting

- Verilog can do math!
 - need a bus
- First use Addition (+) and Subtraction (-)
 - need a register with 16-bit (assigned to LEDs)
 - need a Clock input (assigned to Button)
 - always @(posedge button)

Exercise:

Create a counter with 16-bit output, which should count on pressing a button!



3rd Project

- Counter
 - CLK Clock input
 - DIR Up / Down
 - EN enable
 - RST Reset (synchronous)
will be executed on next clock event
 - VAL 16bit wide **register** for counting value



HowTo Synchronise to Clock?

- sequential logic
- Verilog:

```
initial begin
```

```
    // init all your stuff here
```

```
end
```

```
always @(posedge CLK) begin
```

```
    // logic is triggered on rising edge
```

```
end
```



Doing Math in Verilog

- It is possible to add 1 to a bus
 - `mybus = mybus + 'b1;` If you write 1 instead of 'b1 you get a warning, because 1 is treated as a 32bit number!
 - `mybus = mybus - 'b1;`
- When reaching the maximum value the value will flip to 0 again.
 - `reg [1:0] mybus → 0, 1, 2, 3, 0, 1, 2,`



If – Verilog

- How to use if ?

```
if ( condition ) begin
```

```
    // do something
```

```
end
```

```
else begin
```

```
    // do something other
```

```
end
```

- You might also see:

```
myval = condition ? 42 : 21; → similar to C/C++
```



Hands-On



Physics
Institute III B

RWTHAACHEN
UNIVERSITY