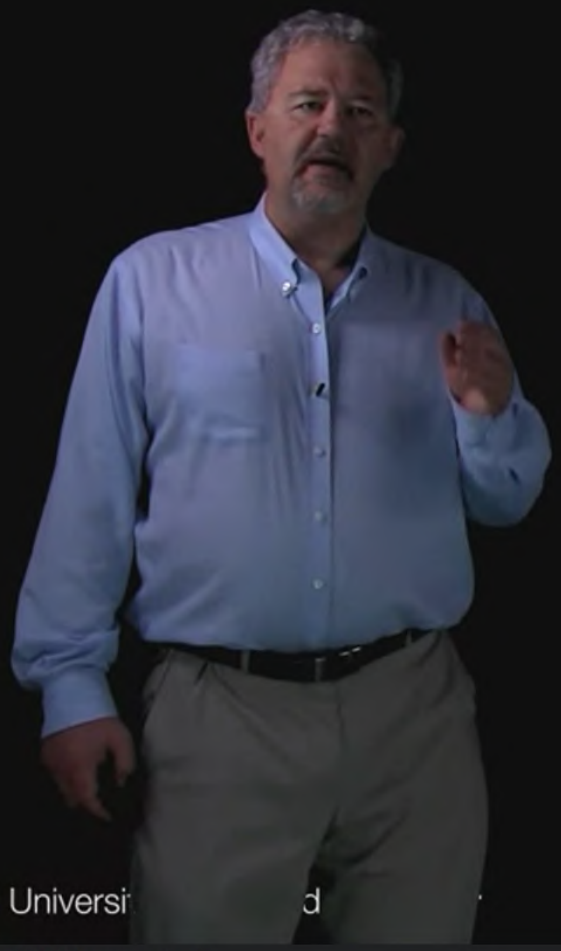
FPGA Design for Embedded Systems

FPGA Softcore Processors and IP Acquisition

University of Colorado **Boulder**

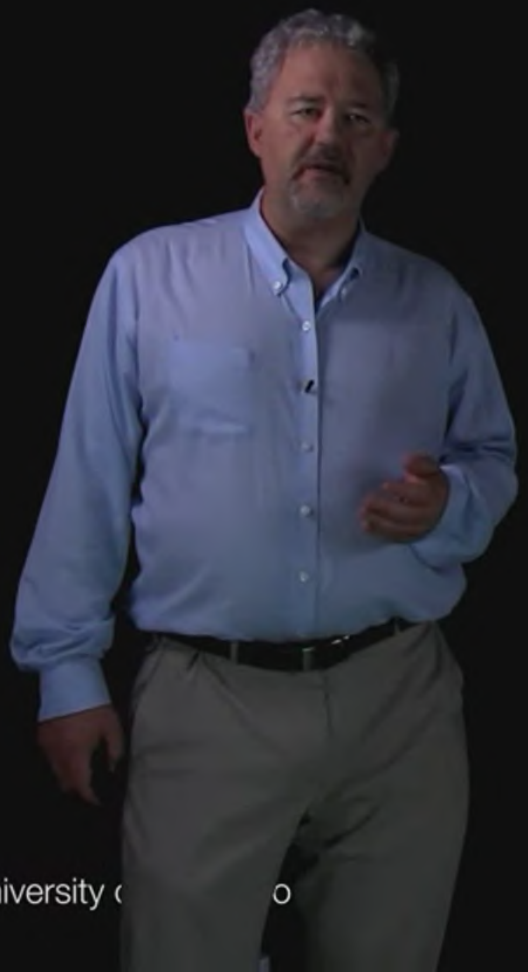
Introducing ModelSim and Simulation for Verification



Up to now, we've just introduced the use of simulation with ModelSim. It's now time to explore the use of simulation in more depth because

- ModelSim is a very powerful tool for debugging our designs.
- Simulation is used throughout industry for Verification – proof that a design is correct. This particularly true of IP testing.
- ModelSim can help automate our verification and debugging processes to a great degree.

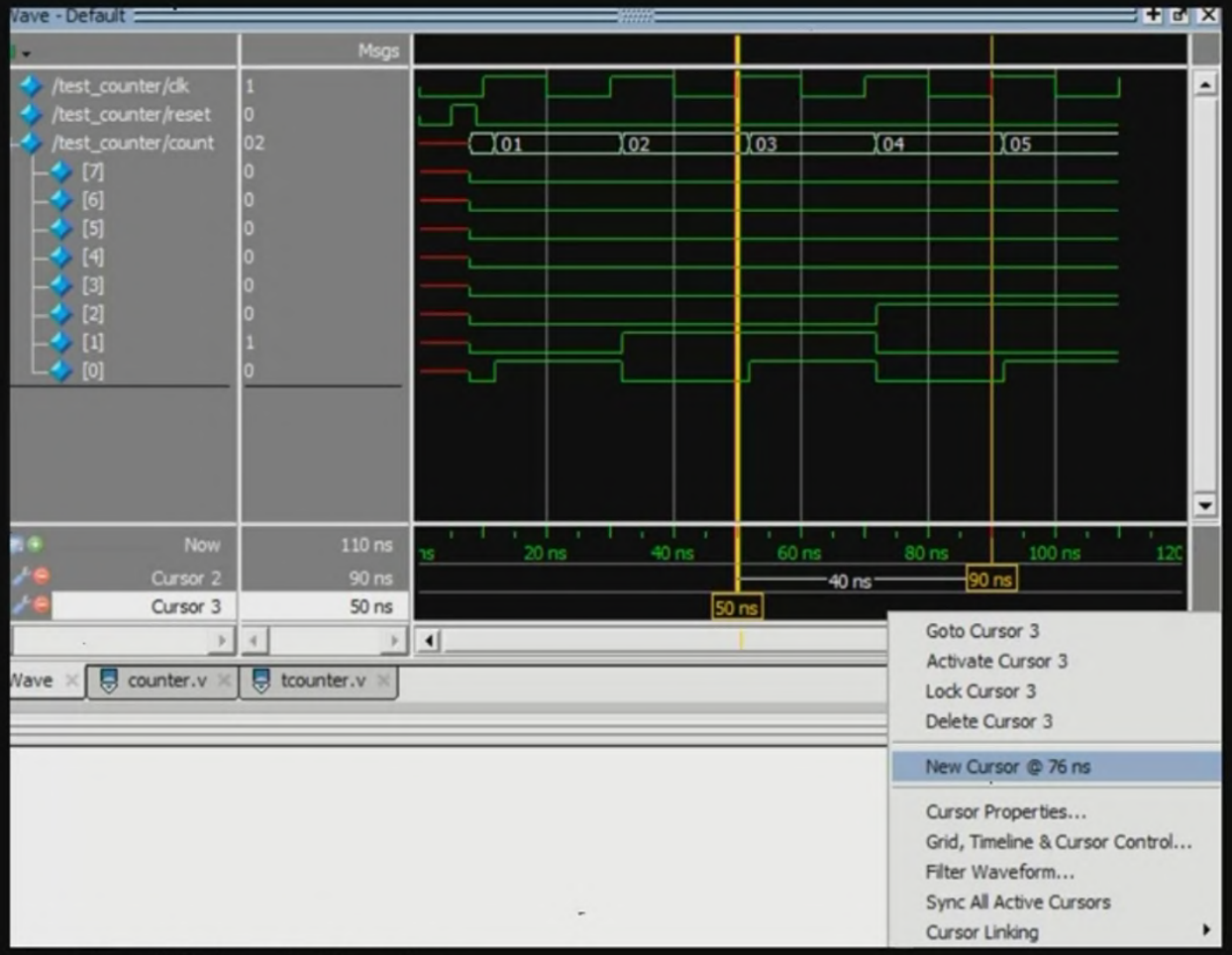
Simulation for Verification and Debugging



As we work on more complex FPGA designs, the challenges to create an error-free design mount exponentially. Having a good grasp of the tools needed to verify correctness of design has become more and more important.

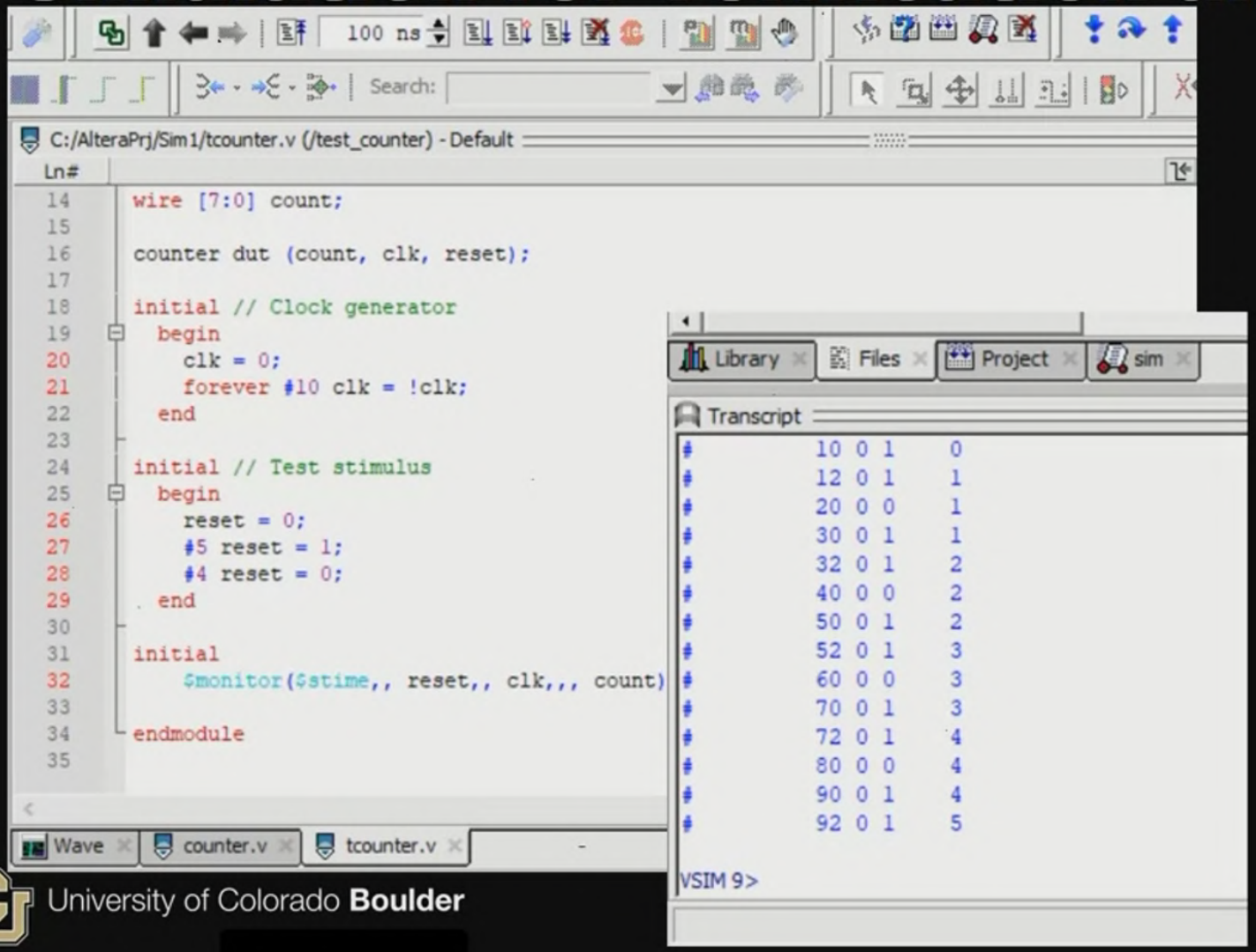
After introducing simulation in previous sessions, we will examine simulation with ModelSim in more depth by working through some examples. This will show the utility of simulation for verification and debugging.

Simulation for Verification and Debugging



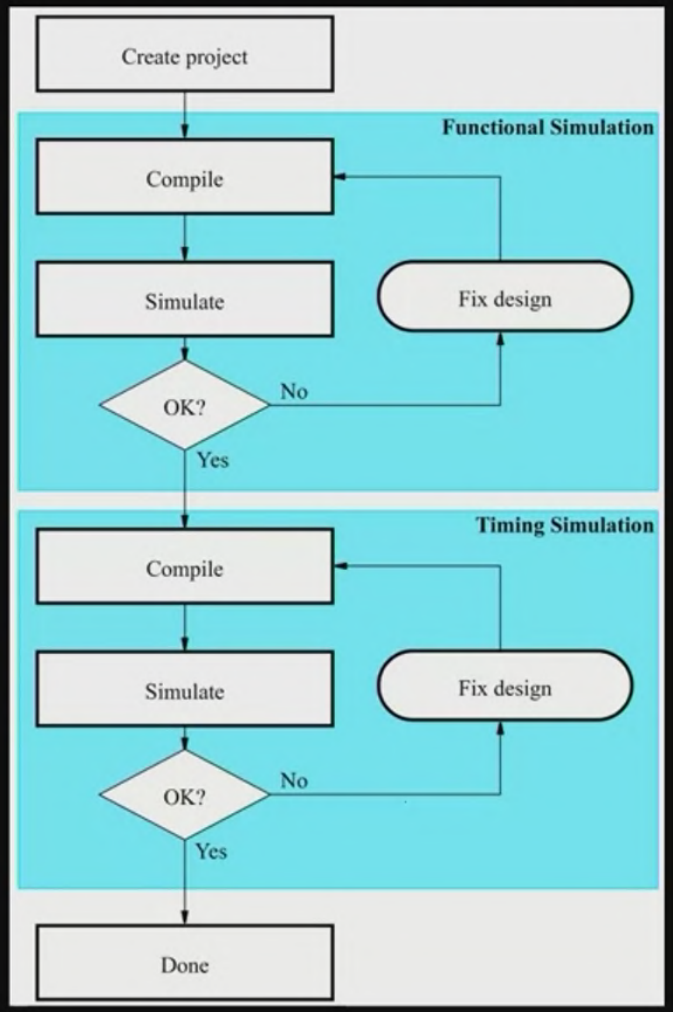
We will look at use of the simulator more closely.

Simulation for Verification and Debugging



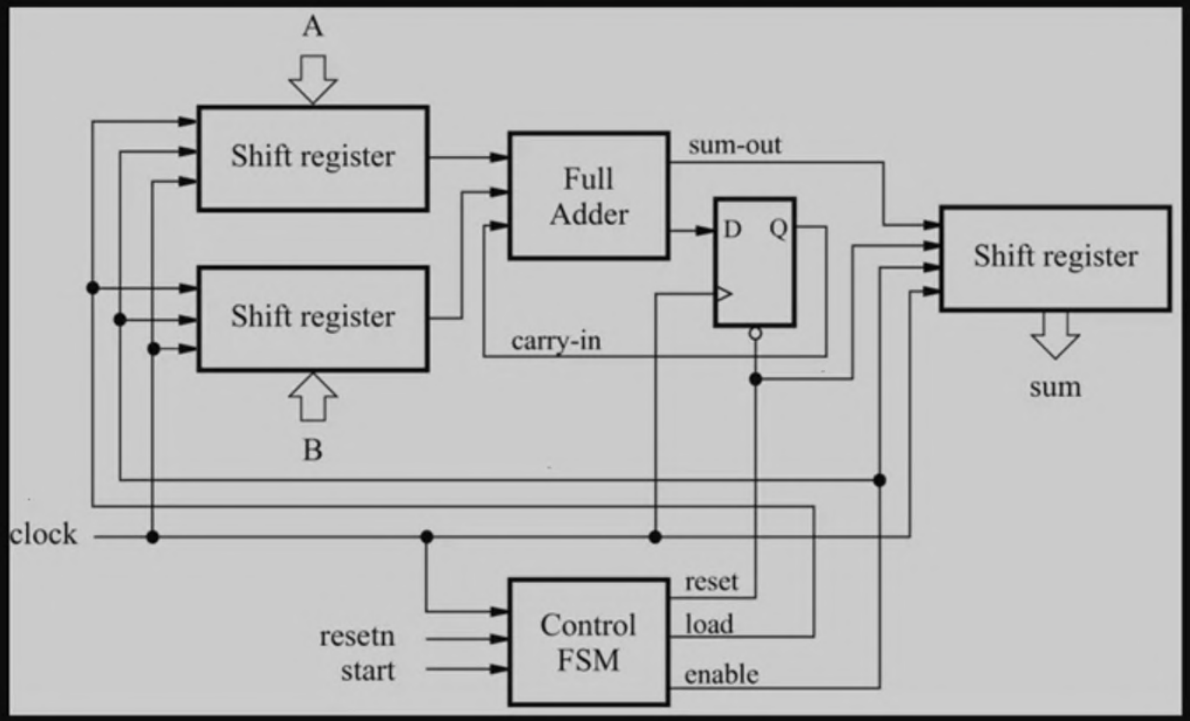
We will demonstrate how the simulator interacts and relates to the code.

Simulation for Verification and Debugging



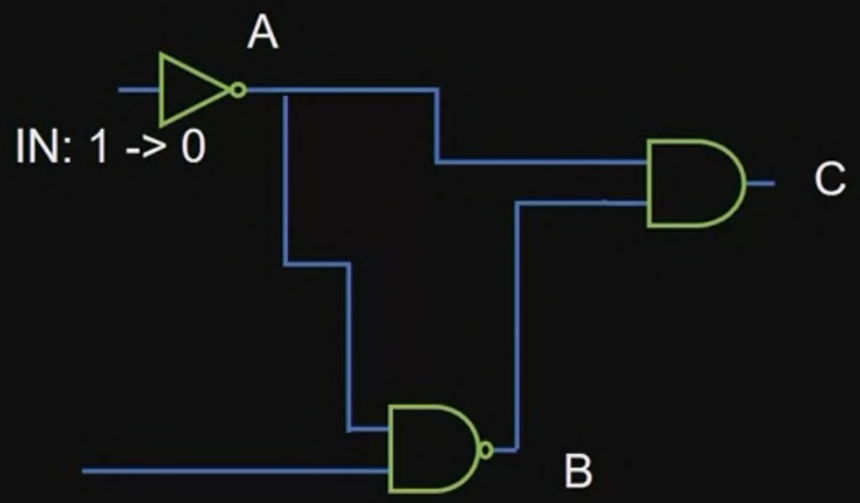
We will investigate both functional and timing simulation

Simulation for Verification and Debugging



We will work with some more interesting examples

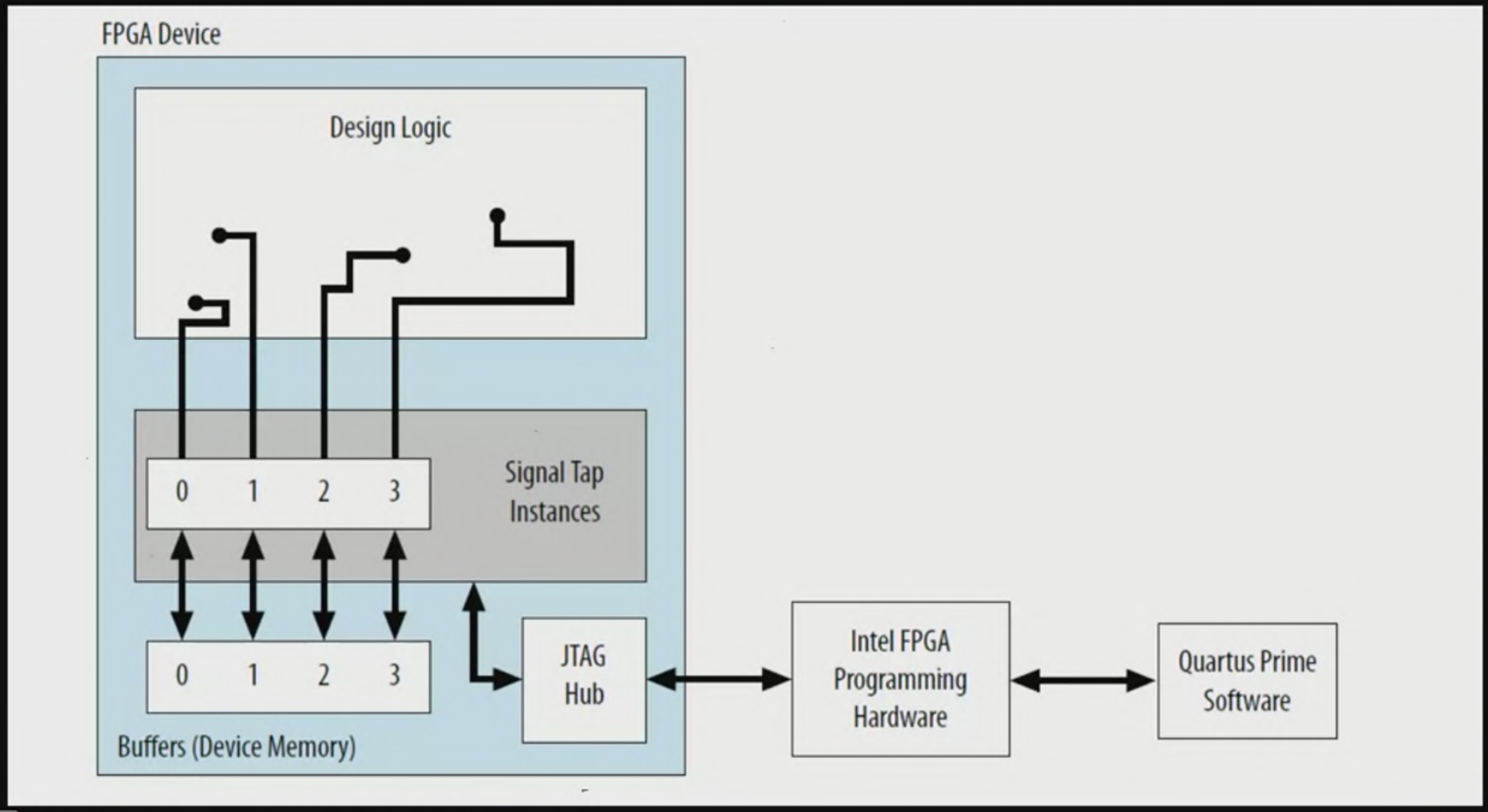
Simulation for Verification and Debugging



Time	Delta Delays	Event
0 ns	1	IN: 1 -> 0
		Evaluate inverter
	2	A: 0 -> 1
		Evaluate NAND, AND
	3	B: 1->0
		C: 0 -> 1
		Evaluate AND
	4	C: 1 ->0
1 ns		

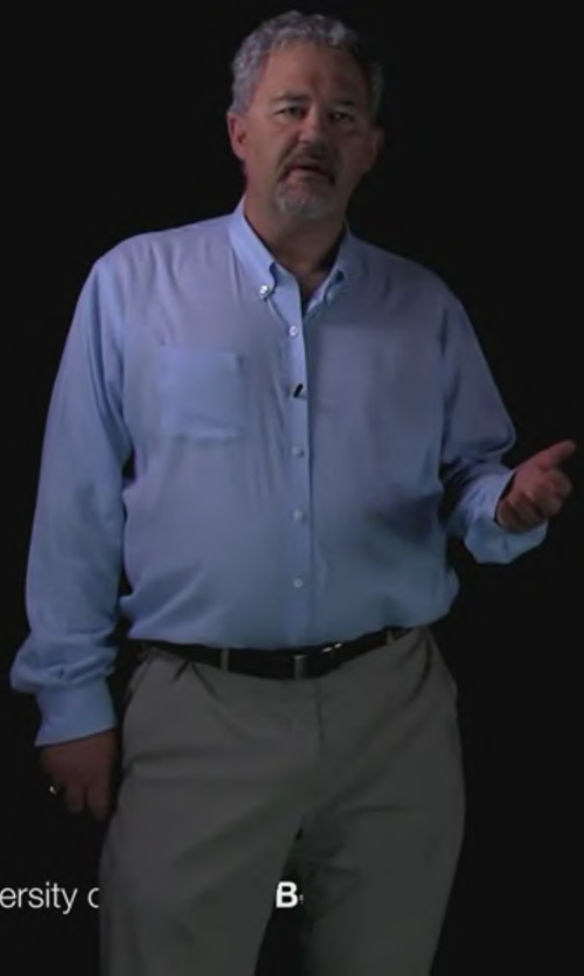
We will discuss how simulators work including the mysteries of delta delays

Simulation for Verification and Debugging



We will introduce the use of an important tool, the internal logic analyzer

Videos in this Module

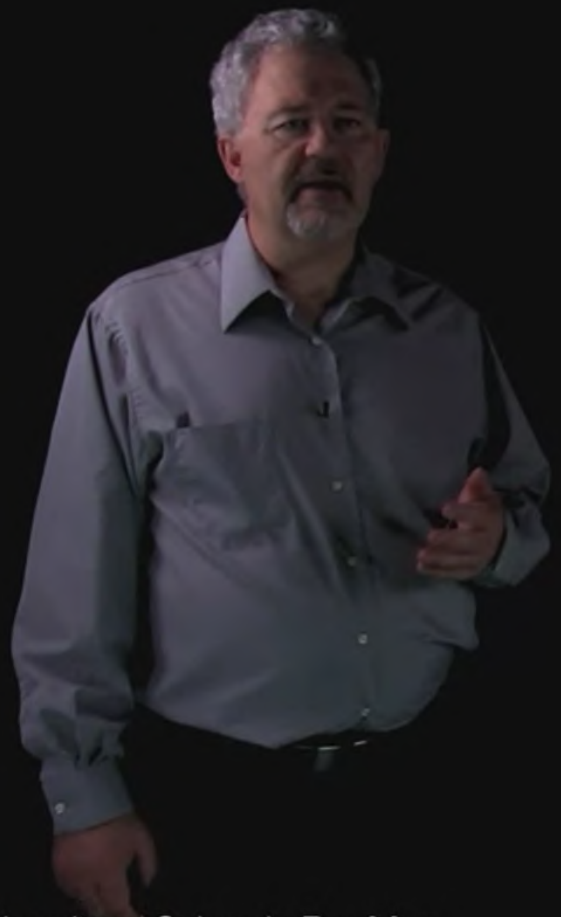


1. Introducing ModelSim and Simulation for Verification
2. Basic RTL Simulation using ModelSim
3. Simulation with Altera ModelSim
4. ModelSim Timing Simulation from Quartus Prime
5. Testbench Verification
6. Design for Simulation
7. Simulation for Verification
8. Logic Analysis with SignalTap II

FPGA Design for Embedded Systems

FPGA Softcore Processors and IP Acquisition

Basics of RTL Simulation



In this video, you will learn:

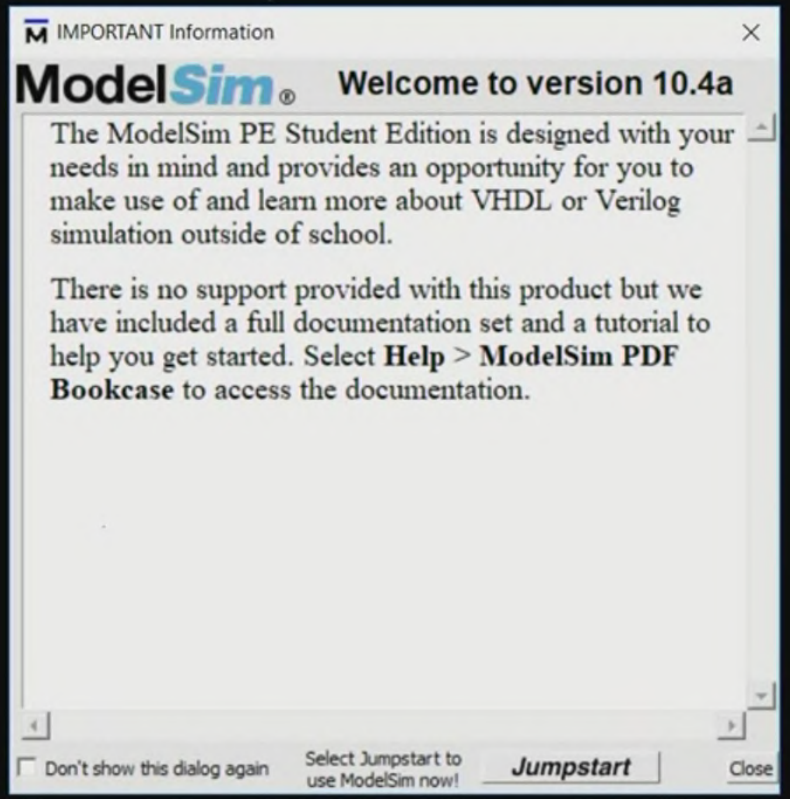
- How to start and run a simulation in ModelSim directly, including setting up the project and libraries, compiling the HDL code and running the simulation.
- Helpful interactions with ModelSim, including setting breakpoints and single stepping through source code, and controlling the Waveform window with Zoom and Cursors.
- How to automate test and debug with ModelSim by the use of DO files and the ModelSim command line.

Basics of RTL Simulation

Come Simulate with us!

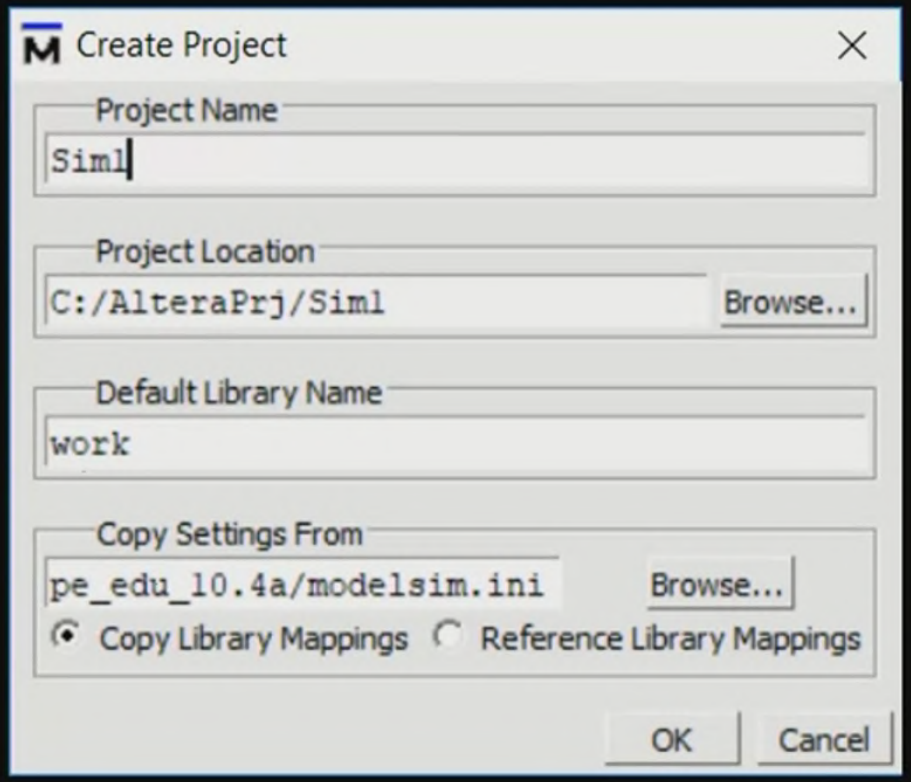
- Install ModelSim Student Edition 10.4a
- Prepare for this exercise by copying files counter.v and tcounter.v from the Modeltech_pe_edu_10.4a directory */<install_dir>/examples/tutorials/verilog/basicSimulation*
- Place these in a new directory, like C:/AlteraPrj/Sim1 or something of your choosing.

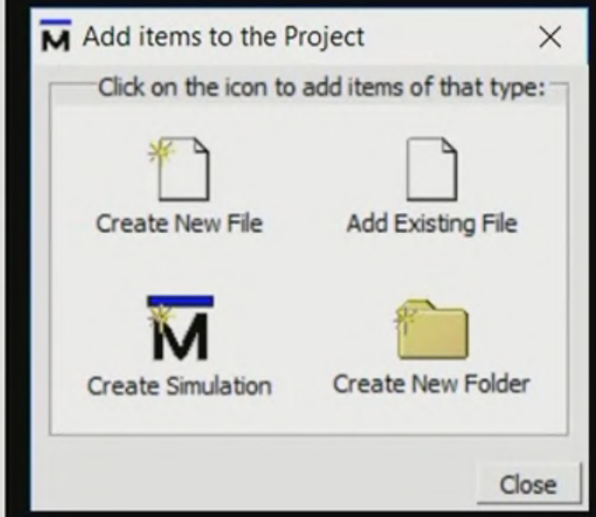
Starting the Simulation



- Start ModelSim, and then click Jumpstart.
- Create a Project

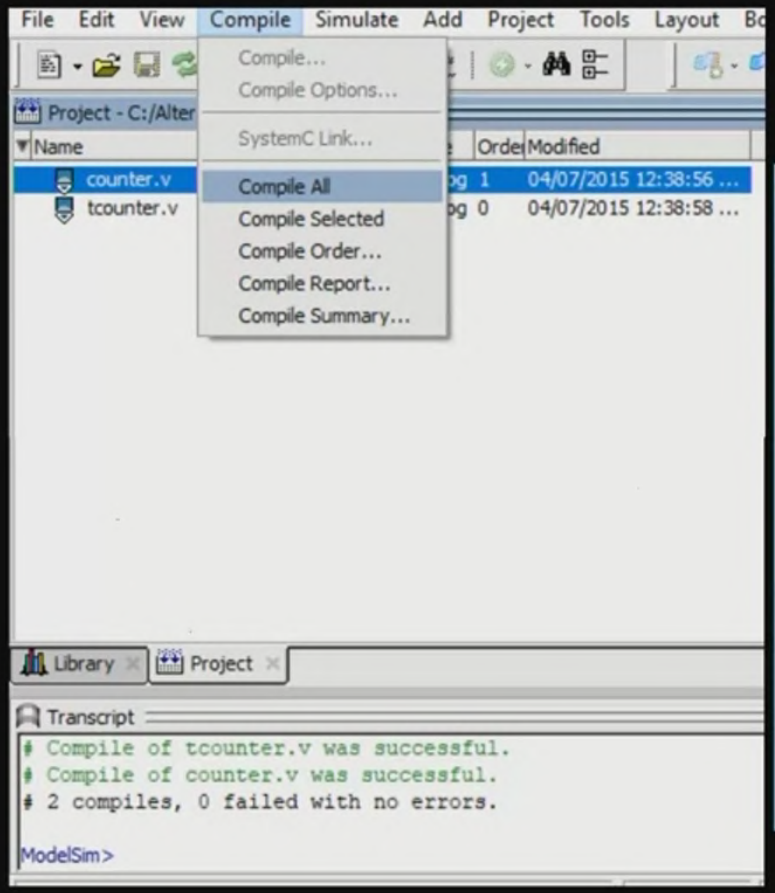
Starting the Simulation

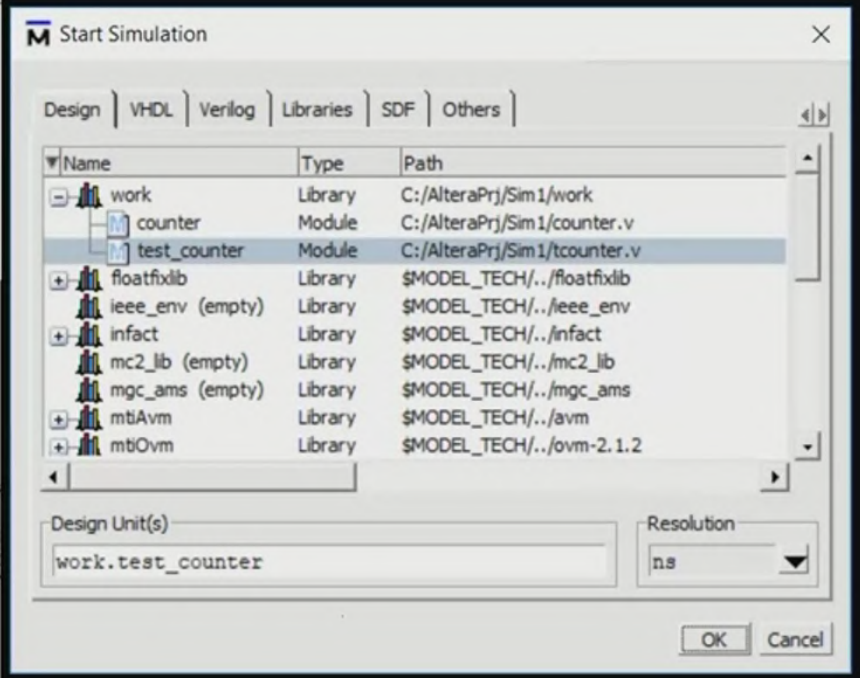




- Name the project, and Browse to your directory.
- Create the Project, and then add the files count.v and tcount.v

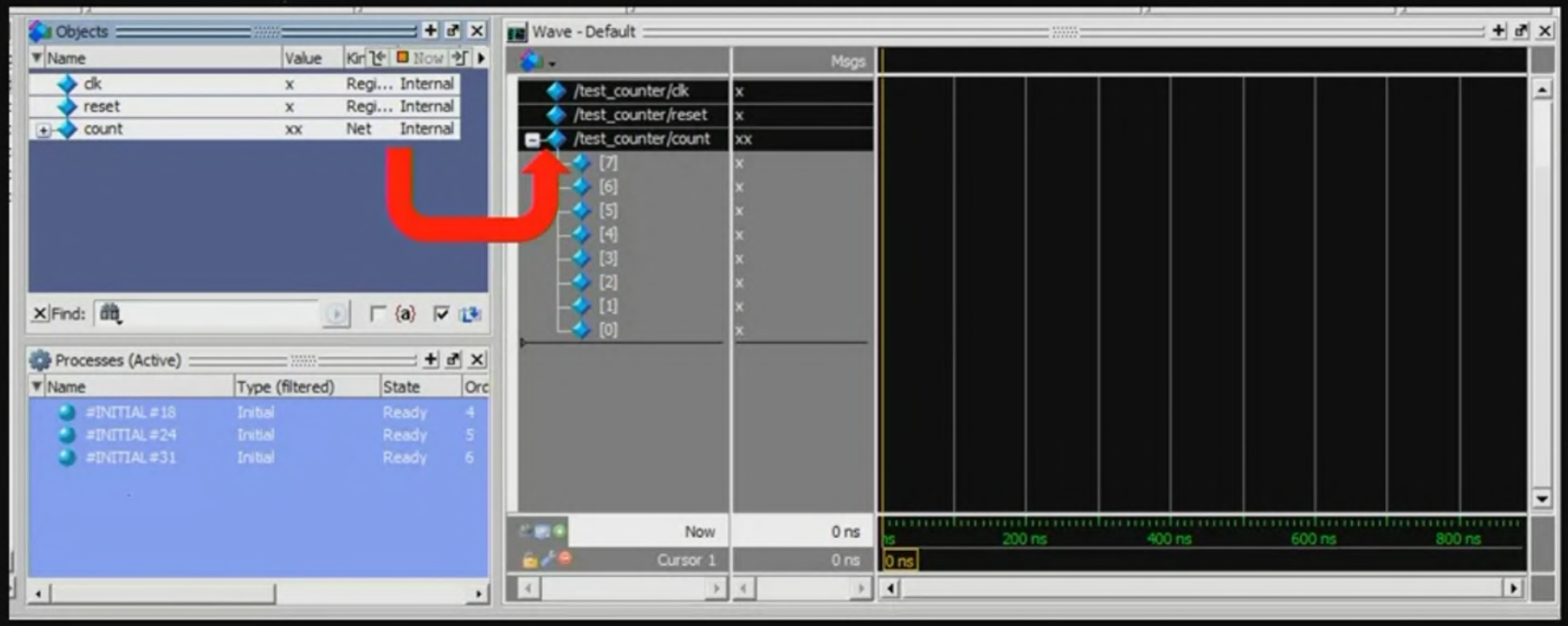
Starting the Simulation

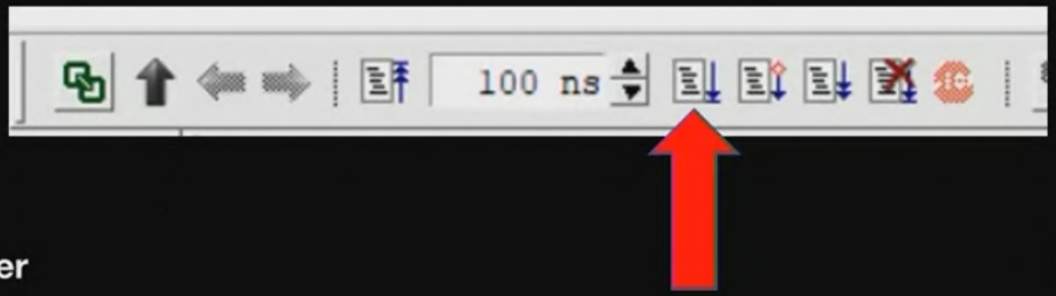




- From the top menu, select Compile All
- Click Simulate, and select tcount.v in the work library

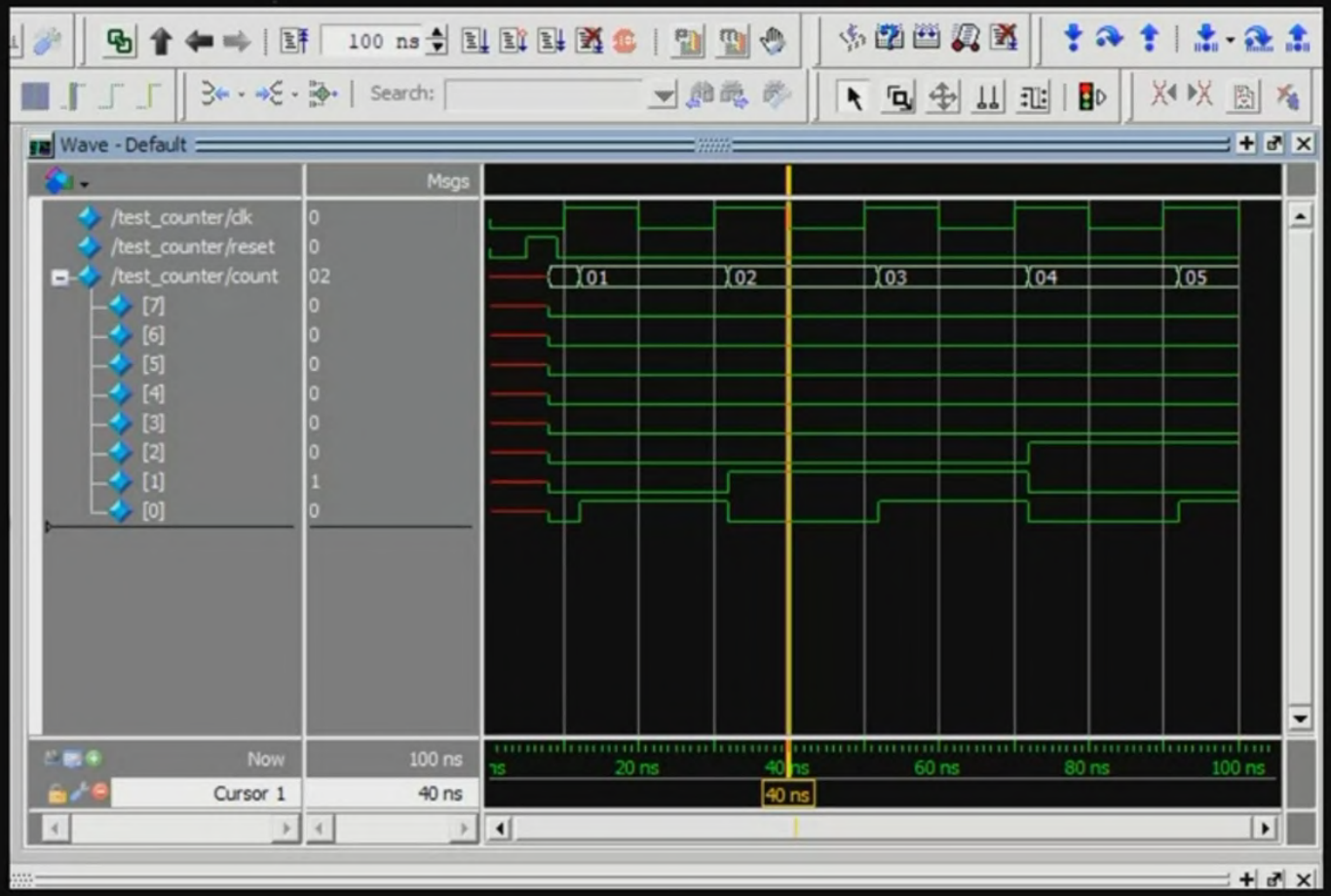
Running the Simulation





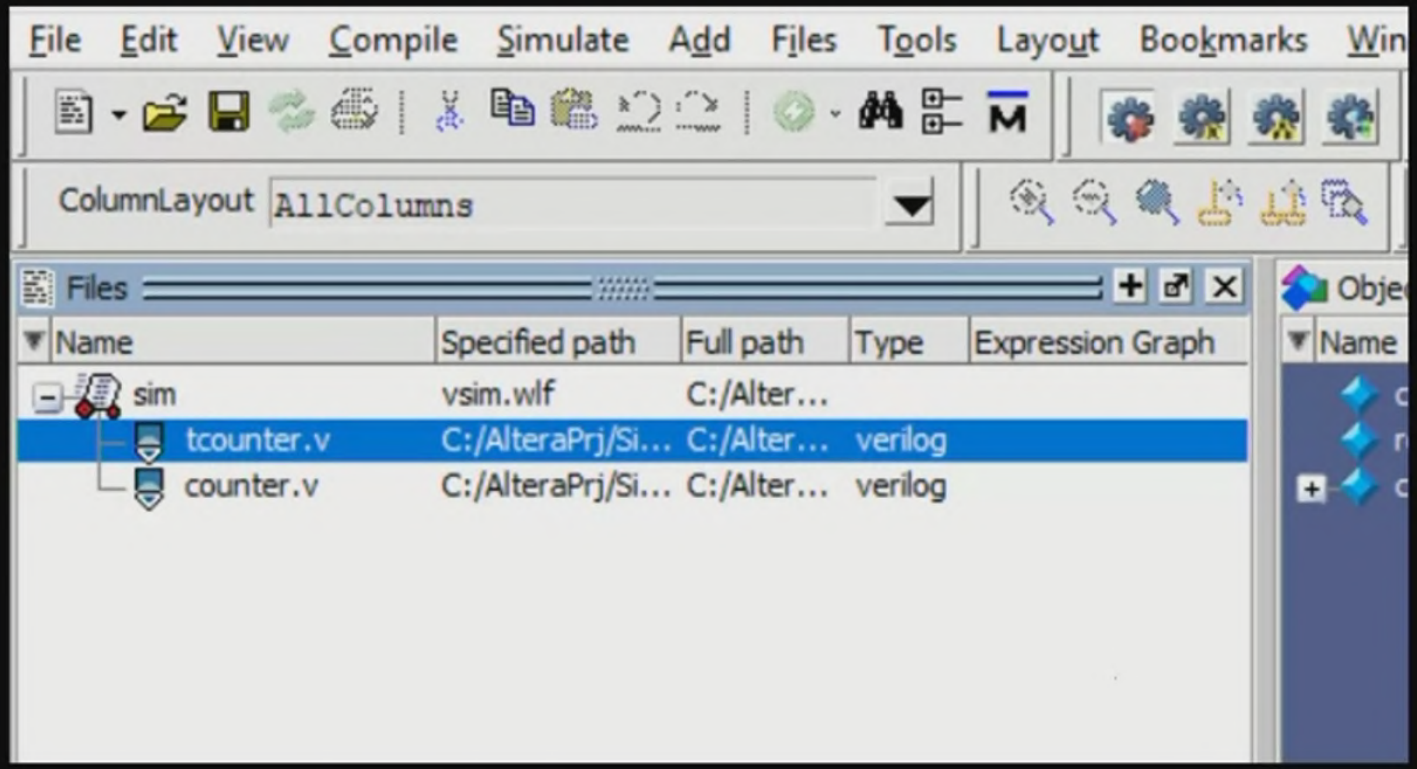
- Click on a signal in the object window, hit cntrl-A and drag signals to the wave window
- Click the down run arrow to run for 100 ns

Running the Simulation



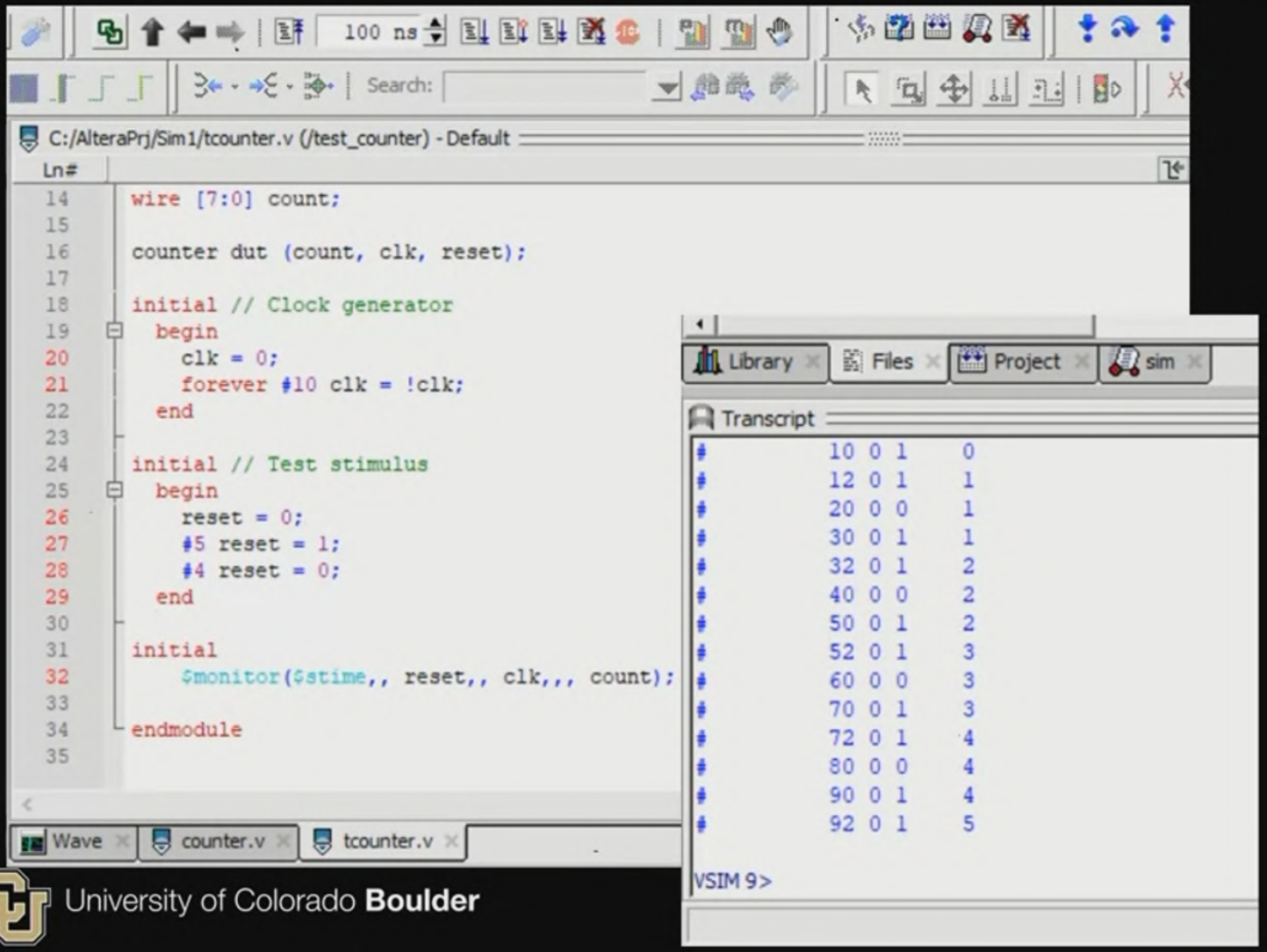
- Right-click in the wave window, and select zoom full. You should see the counter counting.
- Select View->Files to see files

Interacting with the Simulation



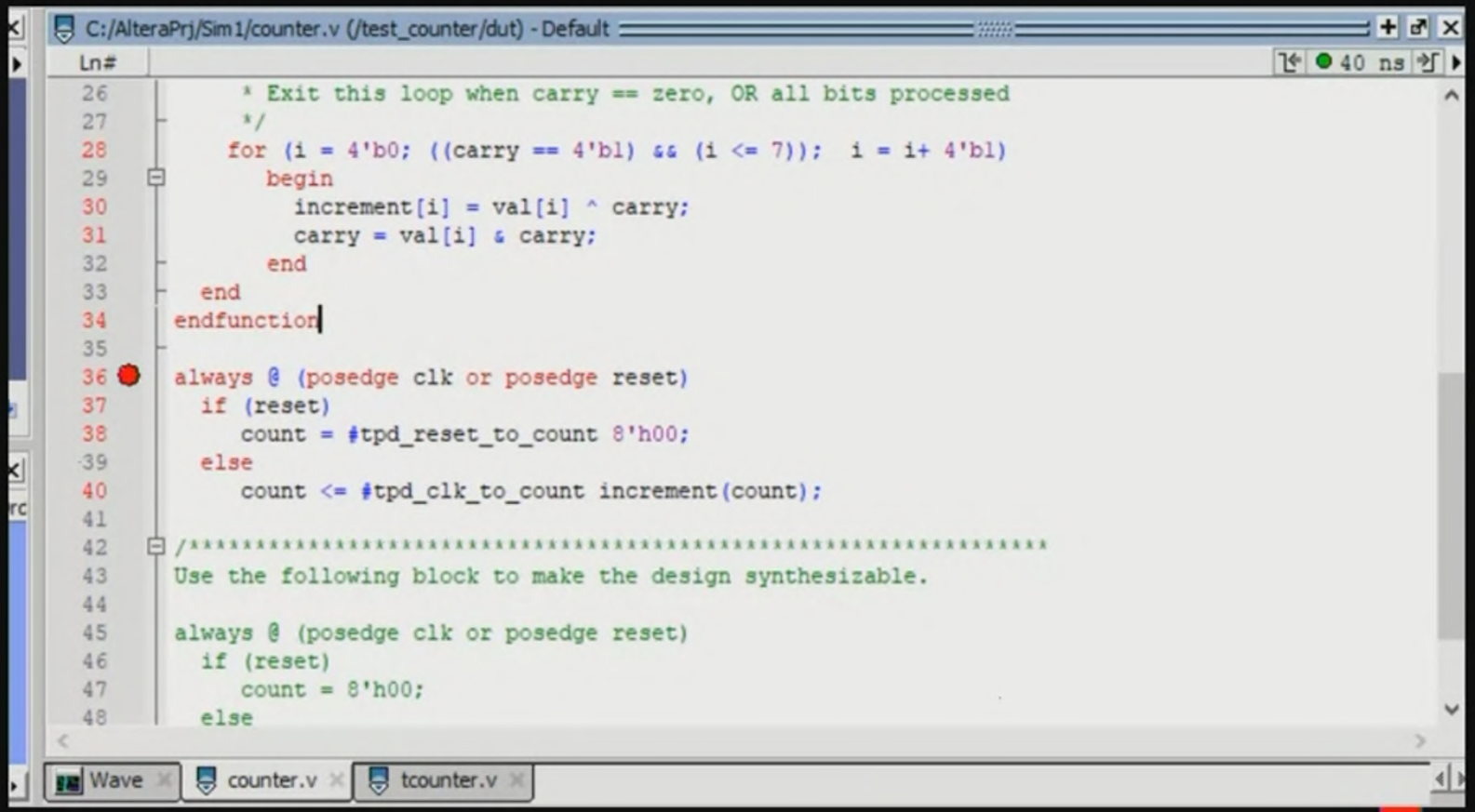
- Double click on the count and tcount.v files to see the source
- Examine the tcount source, with clock and reset stimulus and a monitor

Interacting with the Simulation



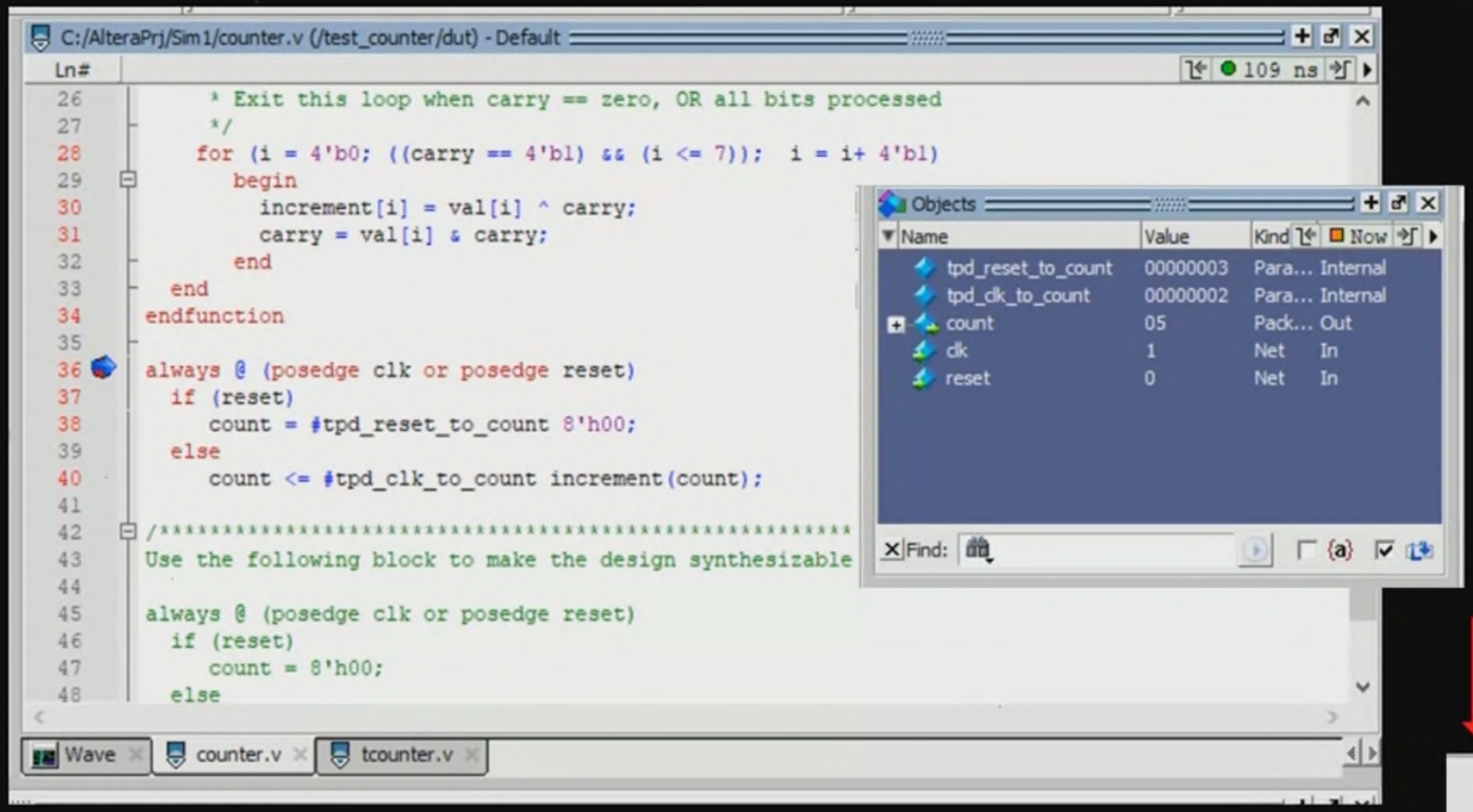
- The stimulus is just a 50 MHz clock and reset
- The monitor causes values to be printed in the Transcript window

Interacting with the Simulation



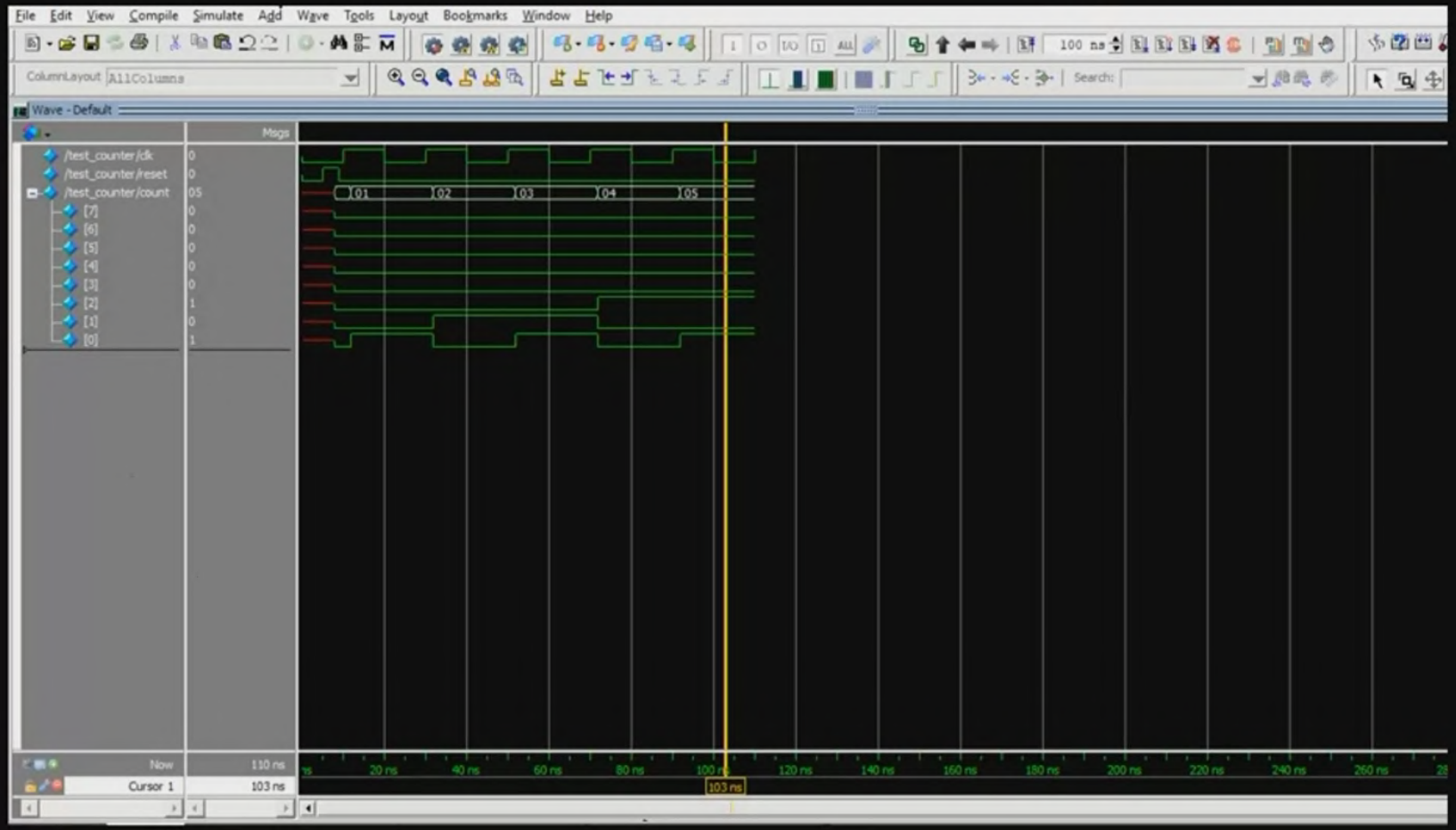
- In the count.v window, click next to line 36 to set a breakpoint.
- Click the run button again, but this time it won't run for 100 ns. Why?

Interacting with the Simulation

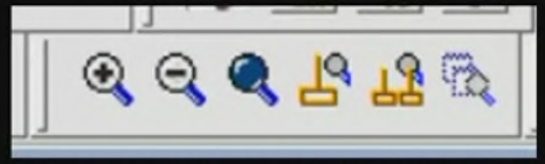


- In fact, it only runs for 10 ns before hitting the breakpoint
- Signal values can be seen in the object or wave window.
- The code can also be single stepped, just like C code in an IDE.

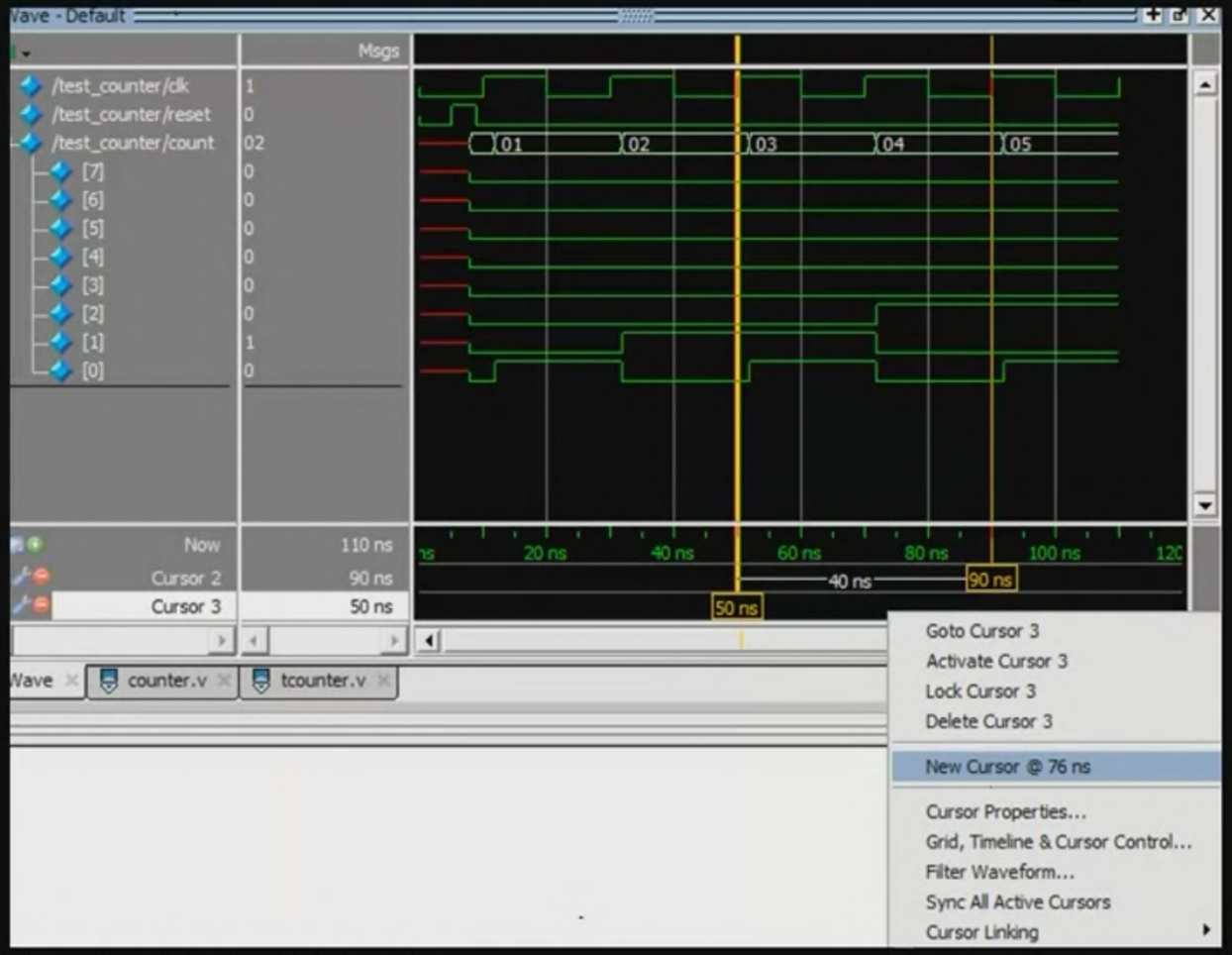
Controlling Zoom



- Click the Zoom mode icon on the wave toolbar 
- The zoom goes to full screen. Clicking again returns it.
- Icons can also be used to zoom in and out.



Controlling Zoom



You can add additional cursors;

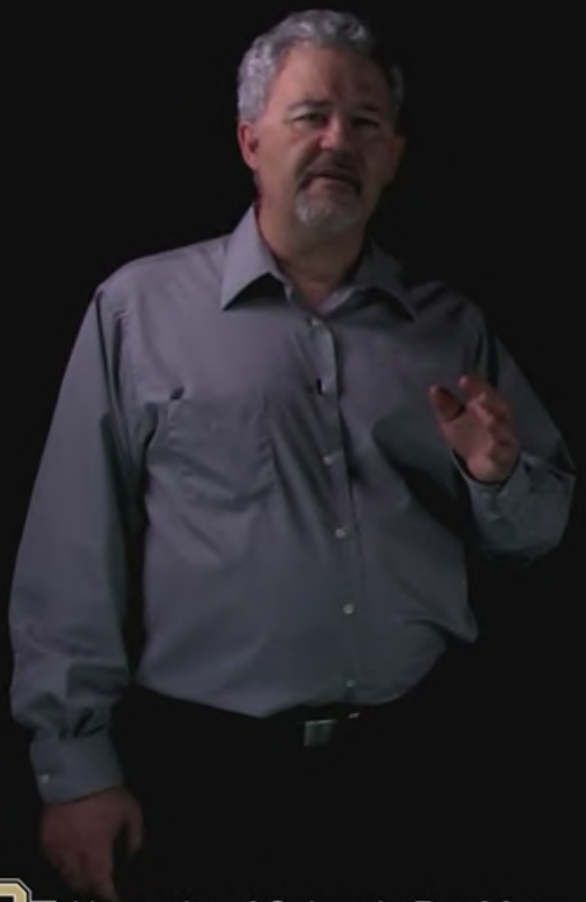
- name, lock, and delete cursors;
- use cursors to measure time intervals; and
- use cursors to find transitions

Make do with DO

Automate simulation by creating DO files

- DO files are scripts that allow you to execute many commands at once.
- You can save your transcript as a DO file to repeat all the steps you've done.
- DO files can be edited just like text files, and in fact are TCL scripts.
- ModelSim can be run from the transcript as a command line by entering TCL commands there, which can be collected into a DO file.

Make do with DO



Saving a Transcript File as a DO File

1. Open a saved transcript file in a text editor, or hit Cntrl-A in the transcript window and save in the editor.
2. Remove all #commented lines leaving only the lines with commands.
3. Save the file as *<name>.do*.

The script can be run by typing the do command in the command window, or from the top menu Tool -> TCL -> Execute Macro

Summary

In this video, you have learned:

- How to start and run a simulation in ModelSim directly, including setting up the project and libraries, compiling the HDL code and running the simulation.
- Helpful interactions with ModelSim, including setting breakpoints and single stepping through source code, and controlling the Waveform window with Zoom and Cursors.
- How to automate test and debug with ModelSim by the use of DO files and the ModelSim command line.

References

[1] Mentor Graphics Corporation. (2019/Nov/27), *ModelSim Tutorial* [Online]. Available: https://www.mentor.com/products/fpga/verification-simulation/modelsim/

[2] Mentor Graphics Corporation. (2019/Jul/17), *ModelSim® User's Manual* [Online]. Available: https://www.mentor.com/products/fpga/verification-simulation/modelsim/

FPGA Design for Embedded Systems

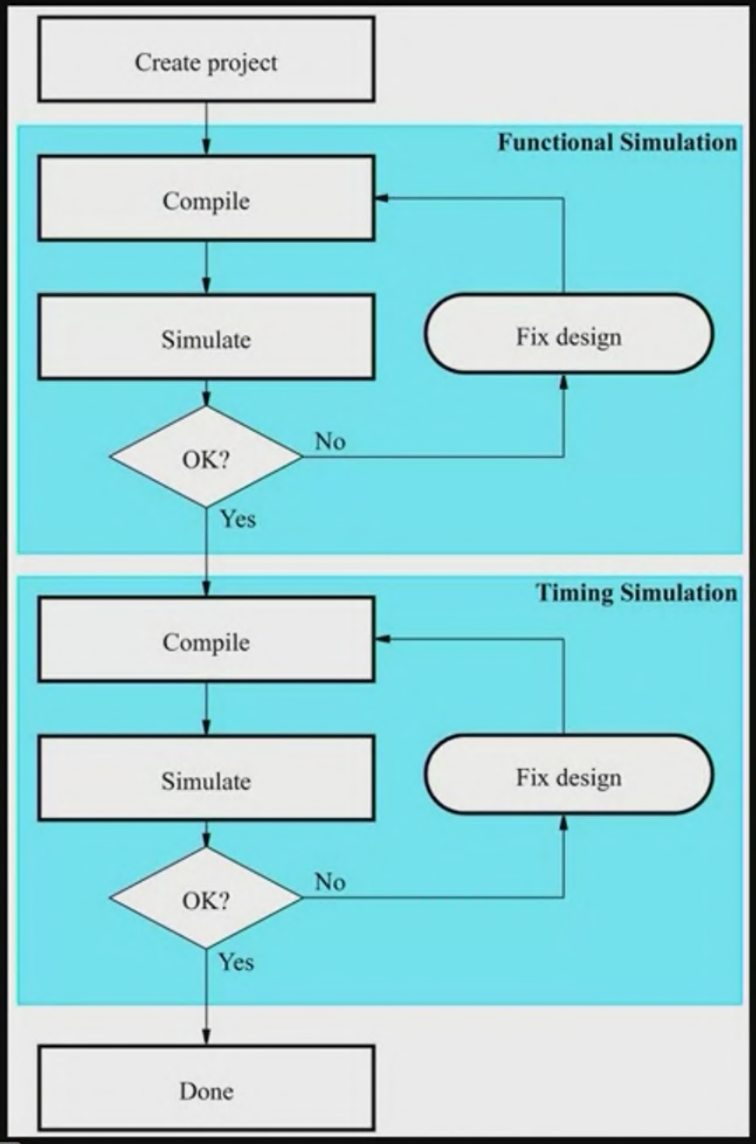
FPGA Softcore Processors and IP Acquisition

University of Colorado **Boulder**

Simulation with Altera ModelSim

In this video, you will learn:

- How to start and run a functional simulation in ModelSim from Quartus prime.
- How to use simulation to verify the correct functional operation of a VHDL serial adder.
- How to interact with the simulation by viewing source code, changing the radix display of signals, and manually forcing signals to values to provide test stimulus.



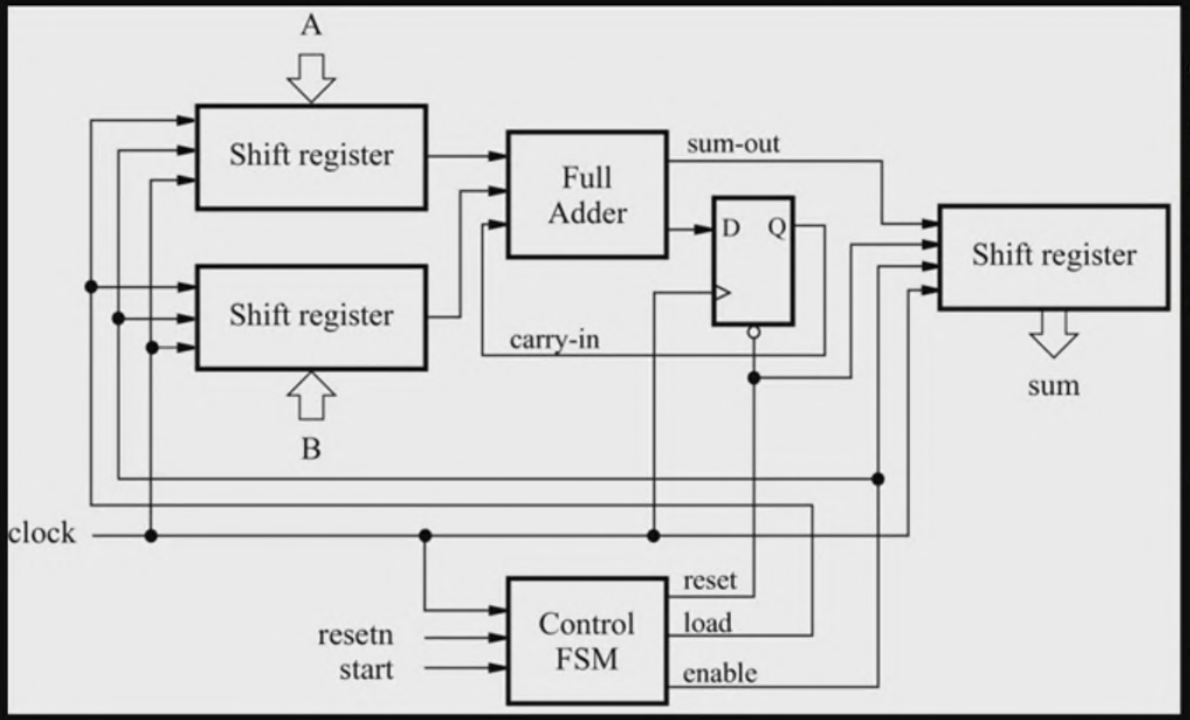
FPGA Simulation Flow

In FPGA development, simulations for design verification are done at 2 separate times in the design flow:

- After Analysis a functional simulation is done, which only determines if the logic is correct and does not include any timing models.
- After fitting (place and route) a timing simulation can be done which will identify timing violations. These will show up in red on the ModelSim waveform viewer. This simulation can help achieve timing closure.
- This process is often iterative as shown in the diagram, where errors are corrected, recompile, resimulate, etc. until no errors remain.

[1]

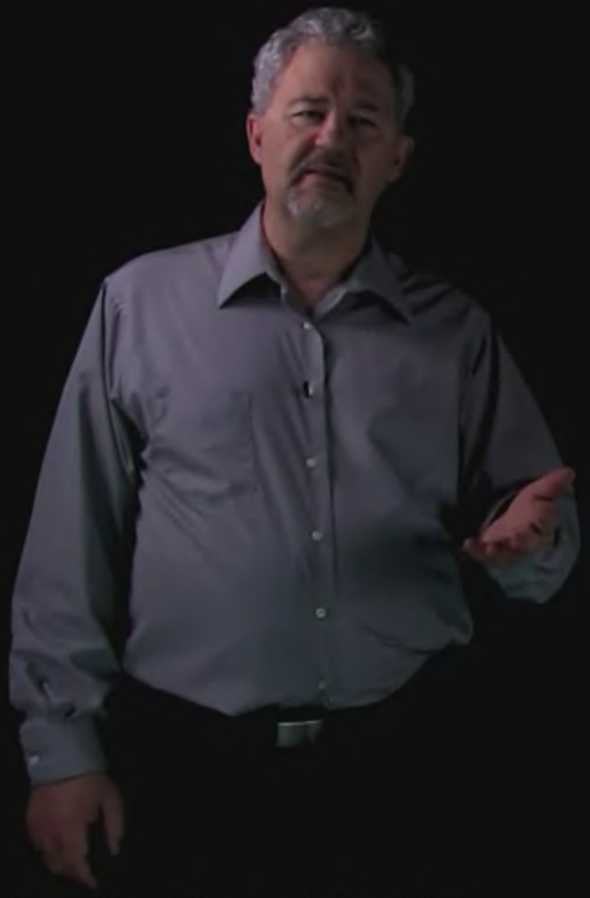
FPGA Simulation Example



To learn about the FPGA flow, we will work through a complete example using Quartus Prime and the Altera version of ModelSim

- Launching ModelSim from Quartus has the advantage of including all the Altera model libraries automatically.
- The example is based on a serial adder circuit as shown in the diagram. We will be using VHDL for this example.
- The code for this example is provided in the including Using_ModelSim.zip file which you can download.

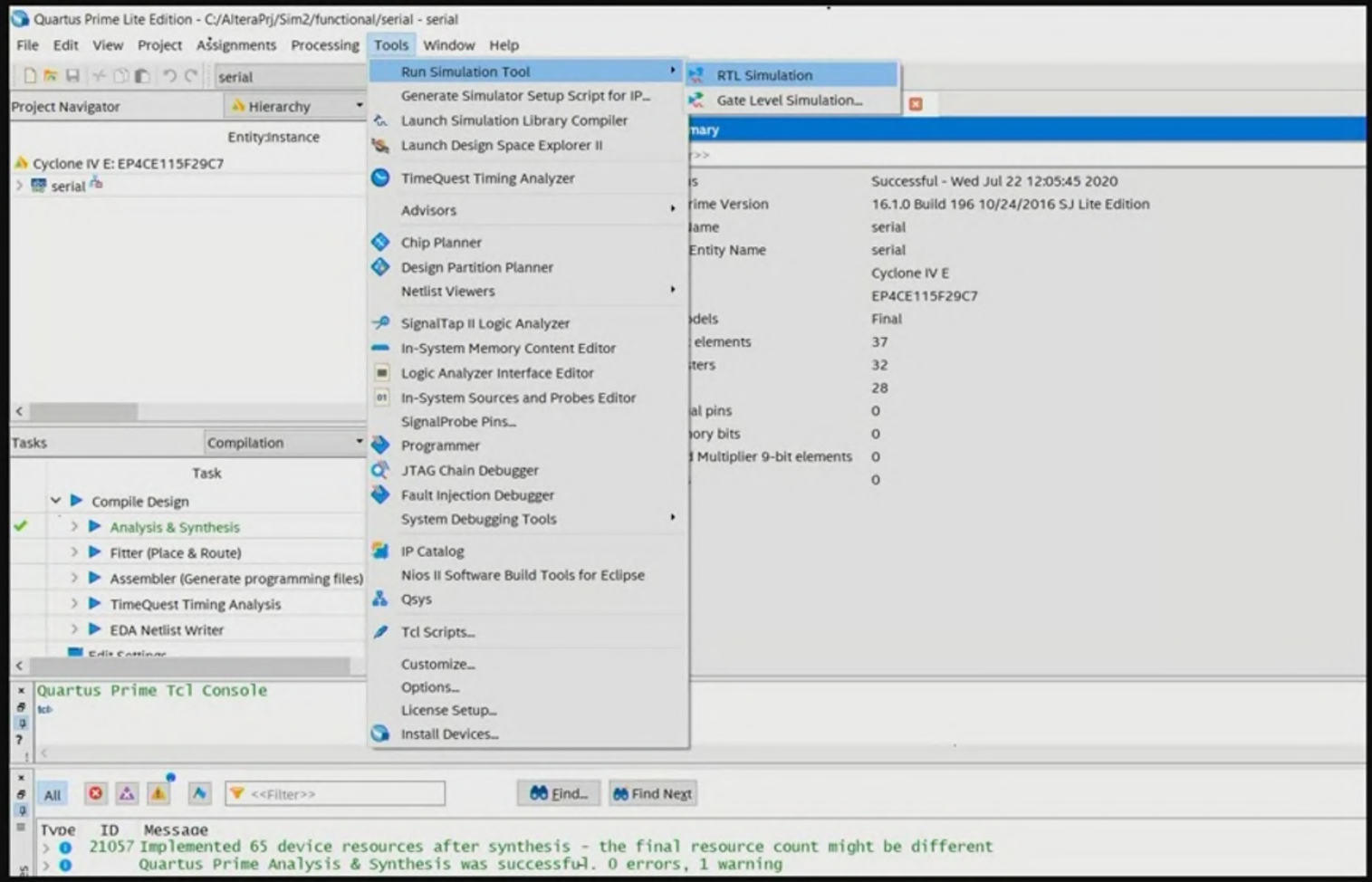
Simulation with Altera ModelSim



Come Simulate with us!

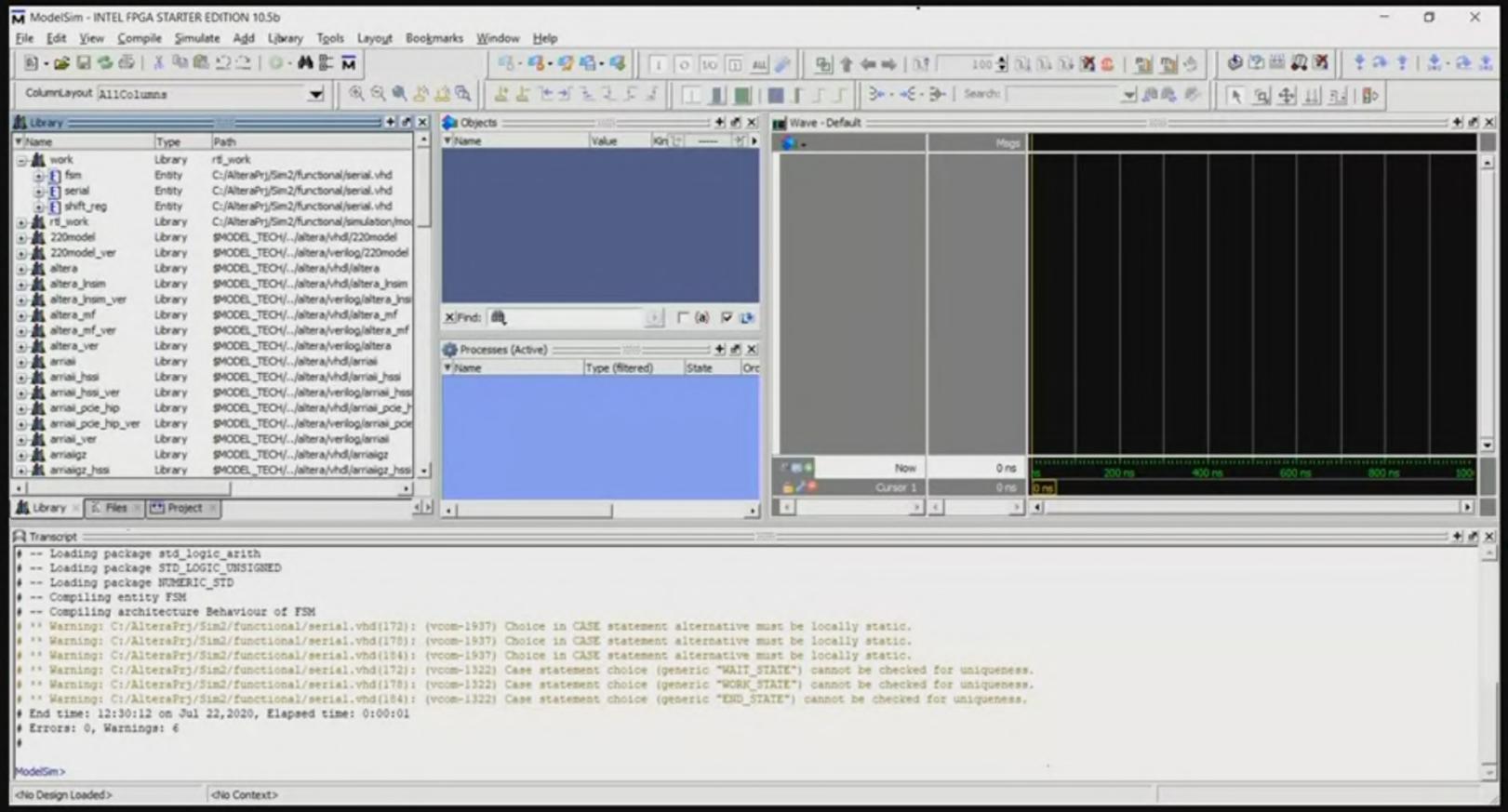
- You should have Quartus Prime installed
- Prepare for this exercise by unzipping the Using_ModelSim.zip to a new project directory, like C:/AlteraPrj/Sim2 or something else of your choosing.
- Start Quartus Prime and select New Project and browse to your new project directory, and select serial.qpf in the functional directory.

Starting the Simulation



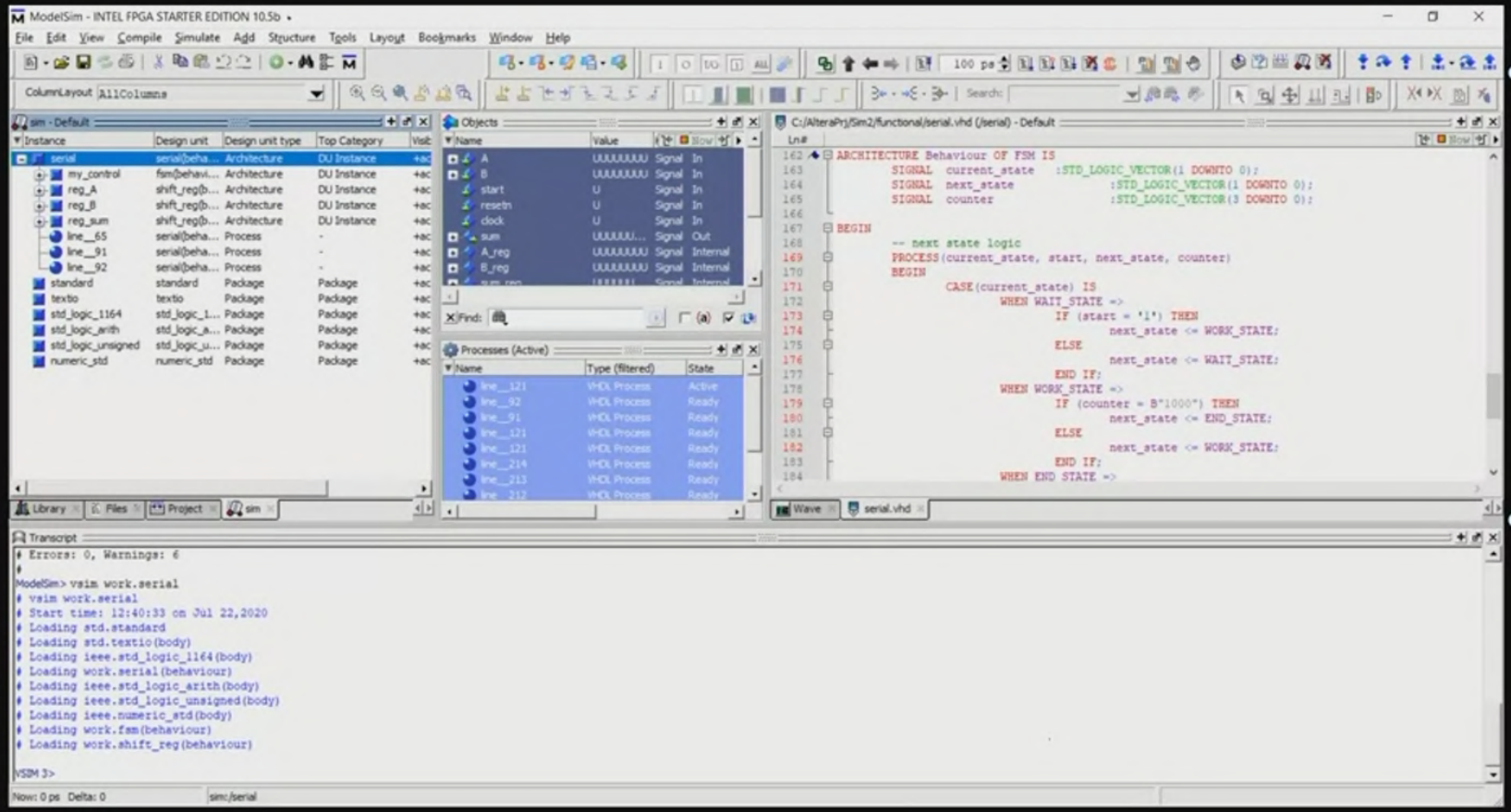
- In the design flow, Run Analysis and Synthesis
- Start the simulation from Quartus by selecting Tools -> RTL Simulation from the top menu

Starting the Simulation



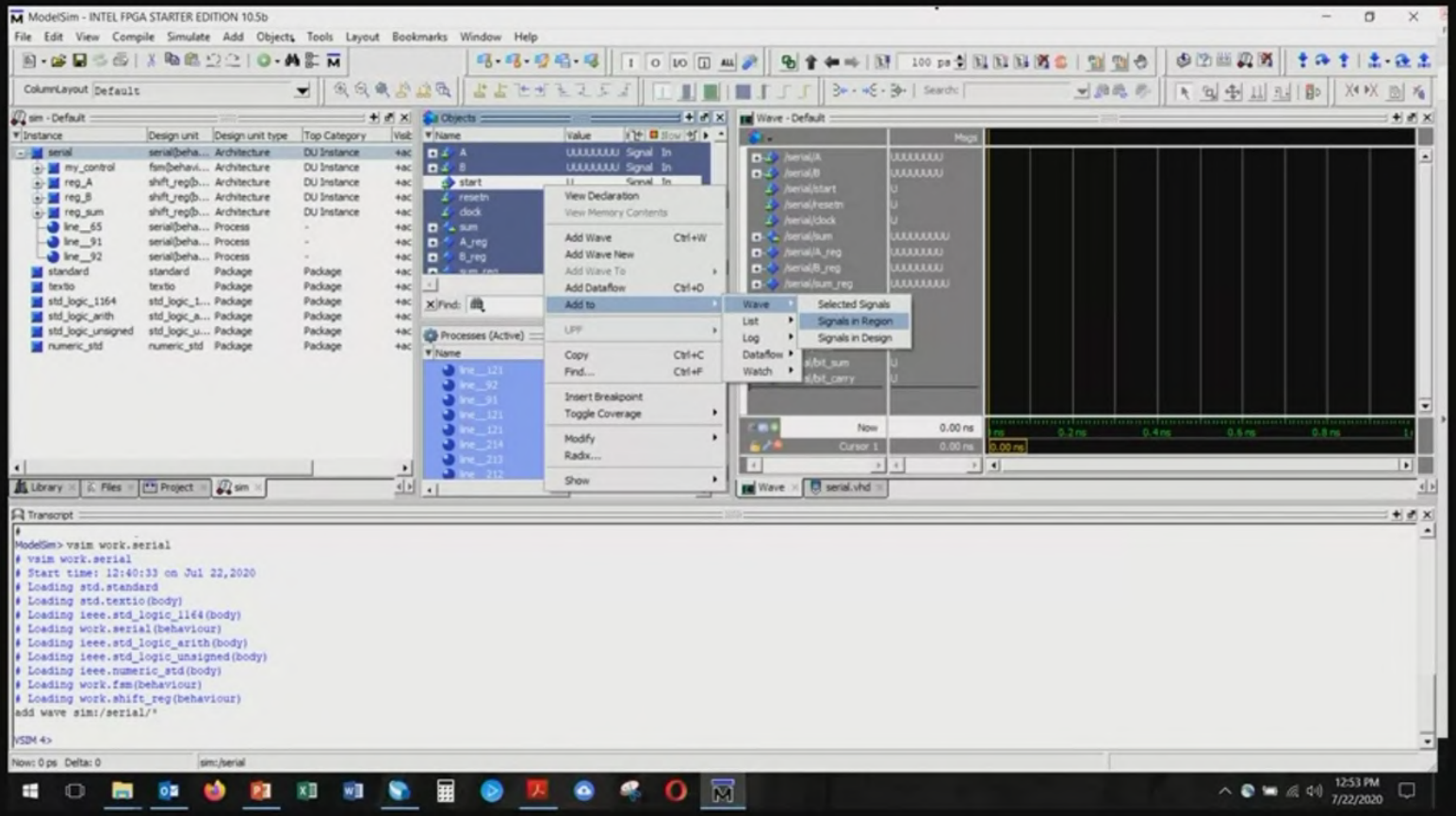
- ModelSim should start. Even though the HDL has been compliled in Quartus, ModelSim can also compile these files.
- Select the serial.vhd file from the work library and right-click it to simulate

Starting the Simulation



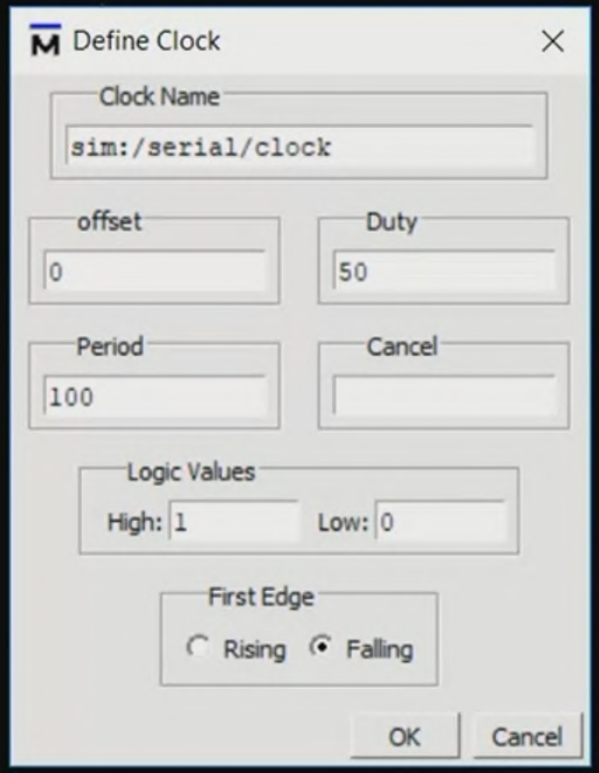
- The simulation should begin. The sim window appears, and signals from the top level appear in the object window.
- Double clicking on entities in the sim window will open the source code for that entity.

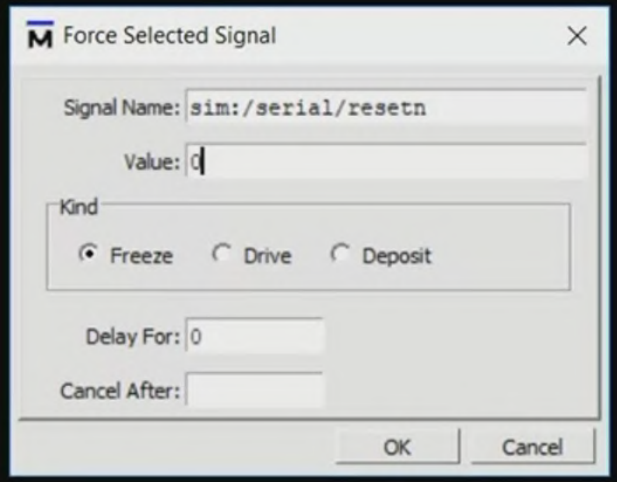
Running the Simulation

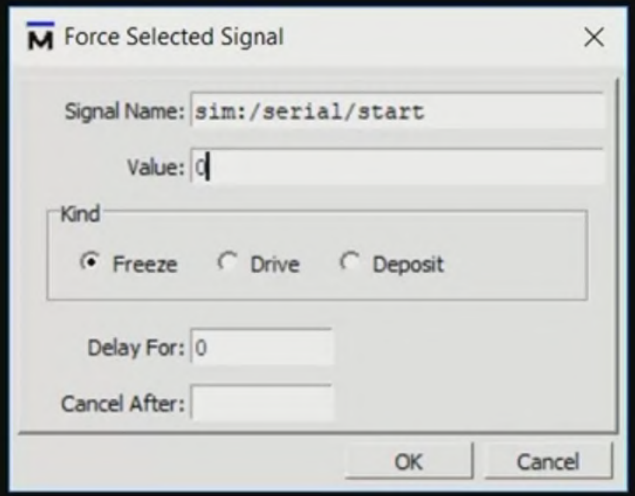


- Add signals to the waveform window by right clicking on a signal in the object window, the Add to -> Wave -> Signals in Region

Running the Simulation

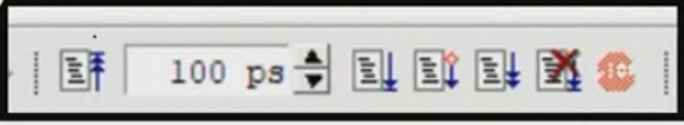


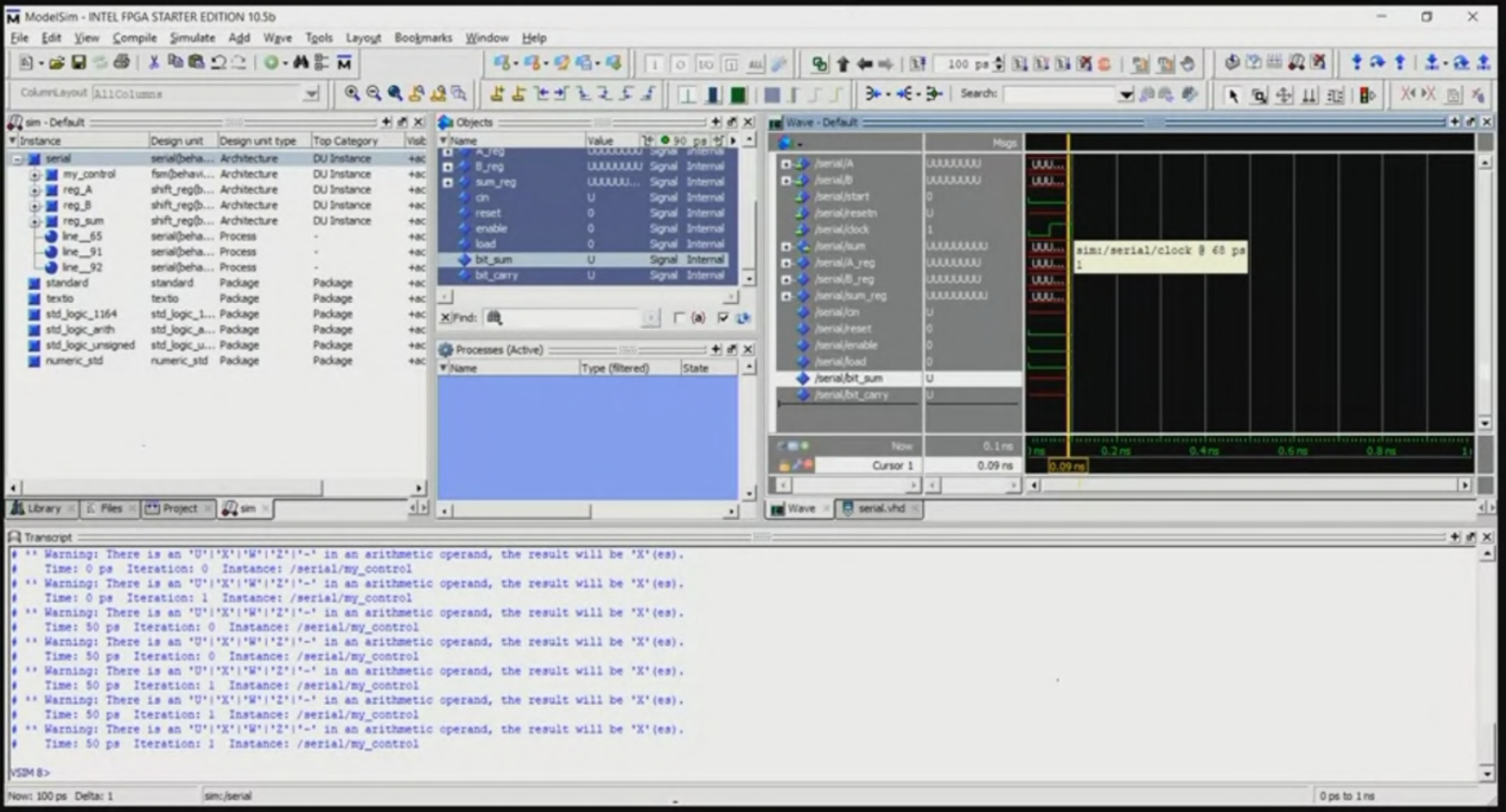




- We have no testbench, so manually add the clock and reset stimulus. Right click on clock in the waveform window and select clock with period 100.
- Right click on resetn and force it to 0. Do the same for start.

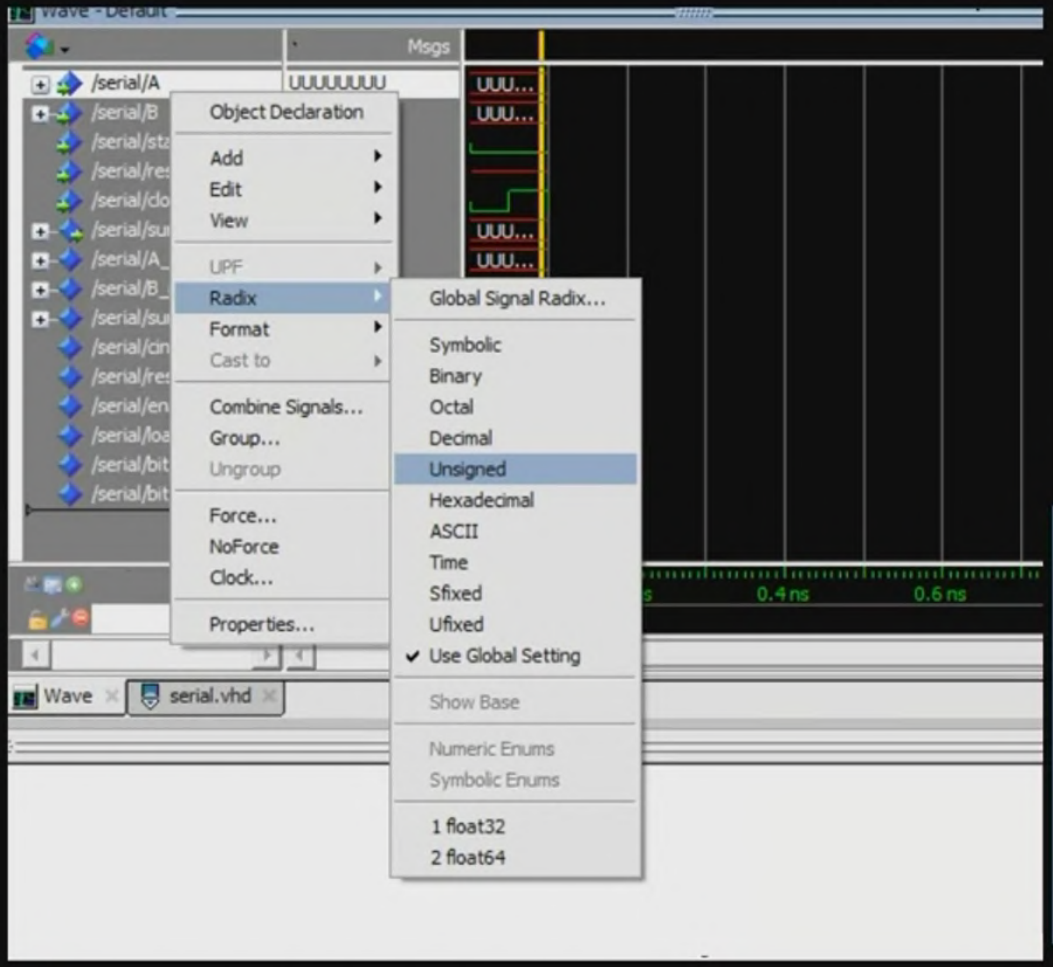
Running the Simulation

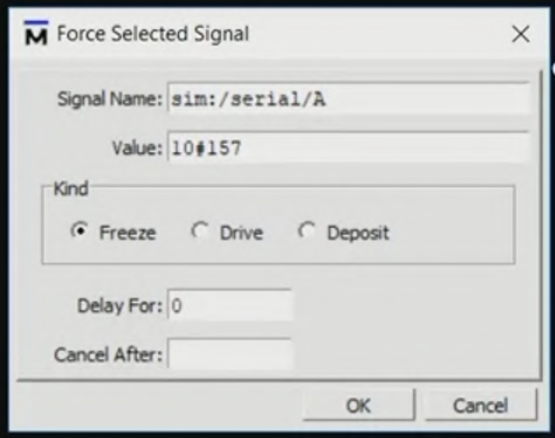


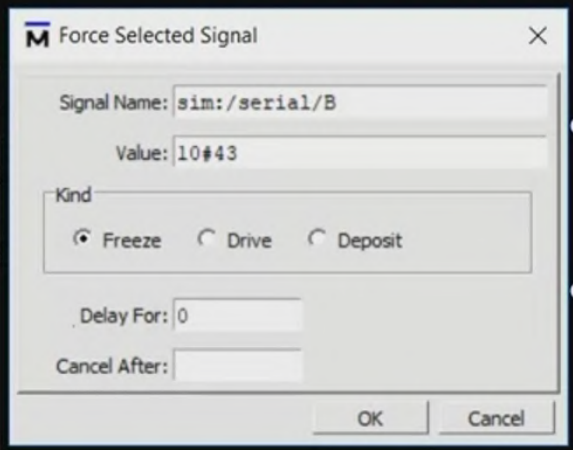


- Click on the run button to the right of the 100 ps menu icon to run the simulation.
- Some signals will be unknown as shown in red and in the transcript window

Running the Simulation

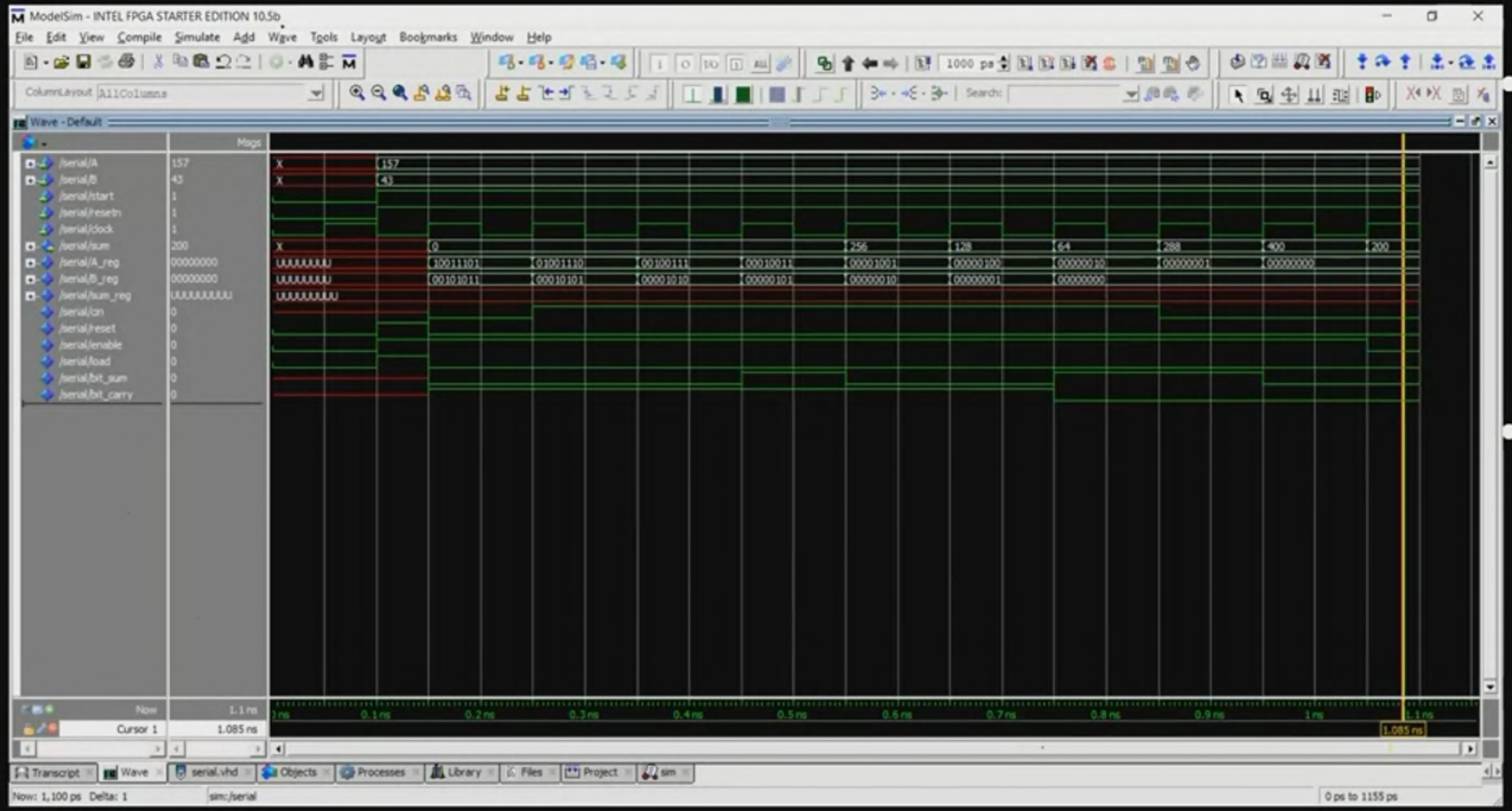




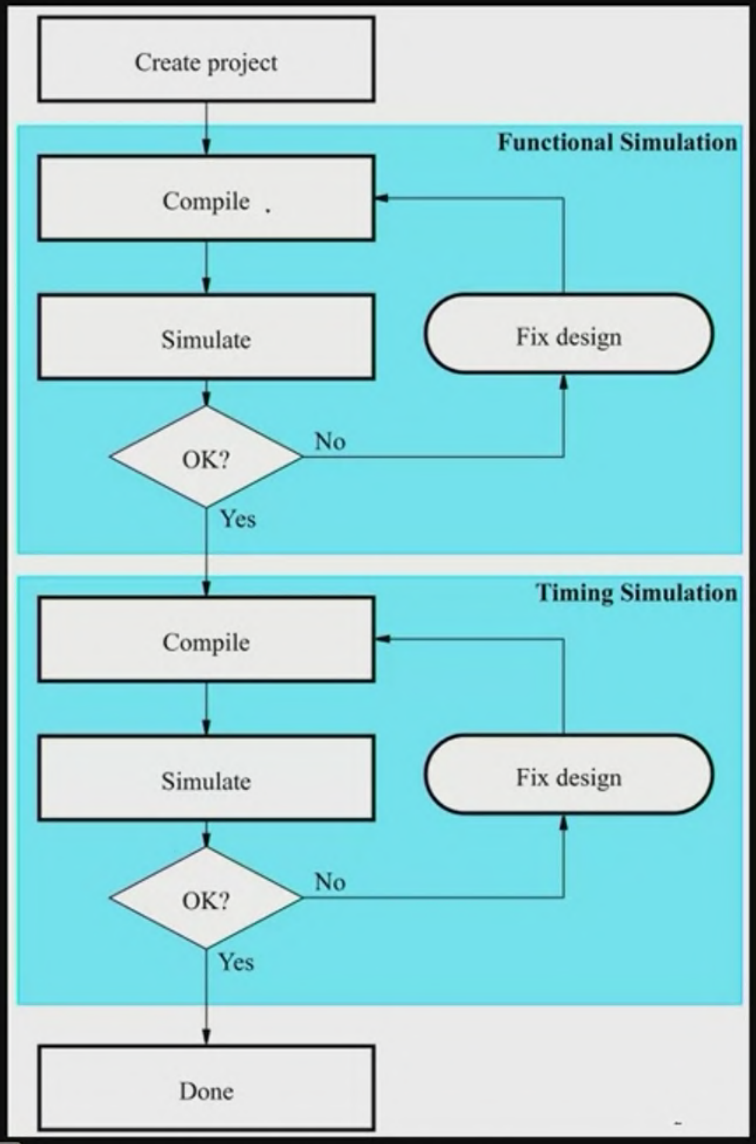


- Force A to 157 and B to 43 as before. Right click on this signals and changes the radix to unsigned
- Set sum to unsigned as well.
- Force resetn and start to 1

Running the Simulation



- Set the simulation time in text field to 1000 ps and hit the Run button.
- Right click in the waveform and select Zoom full. Put the cursor after the last edge to verify the sum is 200.



Timing vs. Functional

We have now completed the functional simulation of the circuit. You may have some thoughts about the veracity of the results

- The time scale is in picoseconds. This does not mean the circuit will run with a GHz clock. In the functional simulations, Time is just an index.
- To get information about timing, timing models must be inserted and used in the simulation. This requires a different simulation setup. To determine timing of path delays, a place and route must be done before the timing simulation.

[1]

Summary

In this video, you have learned:

- How to start and run a functional simulation in ModelSim from Quartus prime.
- How to use simulation to verify the correct functional operation of a VHDL serial adder.
- How to interact with the simulation by viewing source code, changing the radix display of signals, and manually forcing signals to values to provide test stimulus.

References

[1] Intel Altera. (2015/Oct/18), *Using ModelSim to Simulate Logic Circuits in VHDL Designs* [Online]. Available: https://www.mentor.com/products/fpga/verification-simulation/modelsim/

[2] Mentor Graphics Corporation. (2019/Jul/17), *ModelSim® User's Manual* [Online]. Available: https://www.mentor.com/products/fpga/verification-simulation/modelsim/

FPGA Design for Embedded Systems

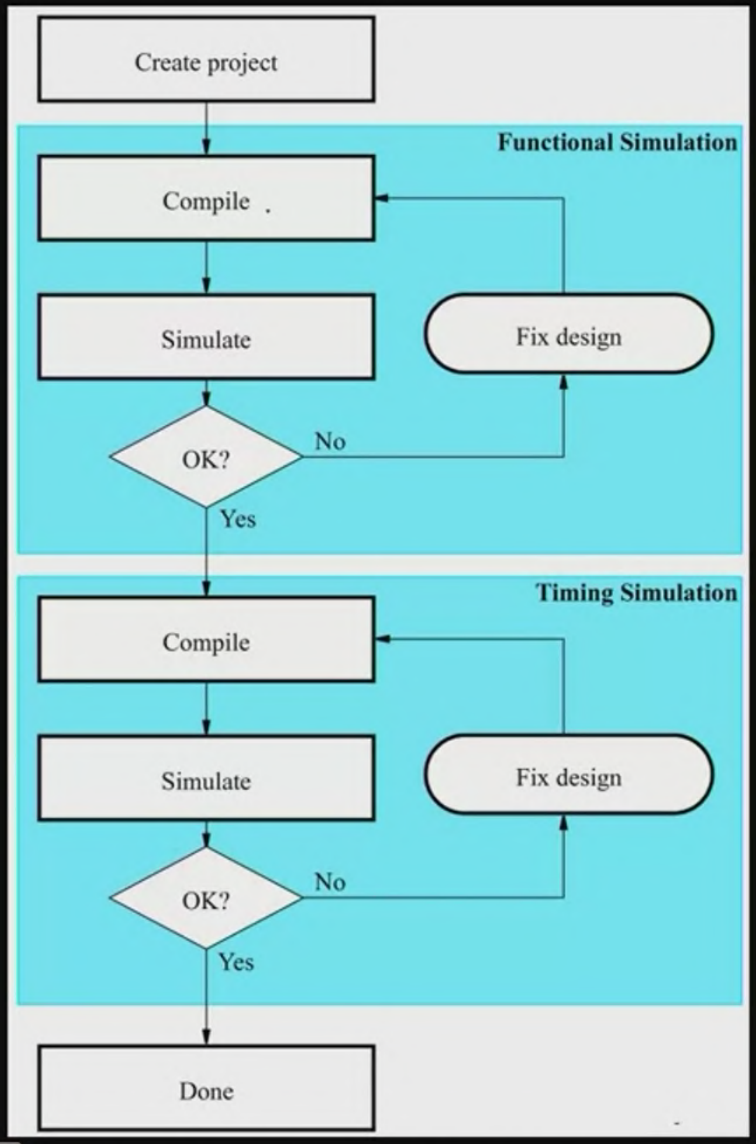
FPGA Softcore Processors and IP Acquisition

University of Colorado **Boulder**

Simulation with Altera ModelSim

In this video, you will learn:

- How to start and run a timing simulation in ModelSim from Quartus prime.
- How to use simulation to verify the correct timing of a VHDL serial adder.
- How to interact with the simulation by zooming in to signal waveforms to discovery timing details.



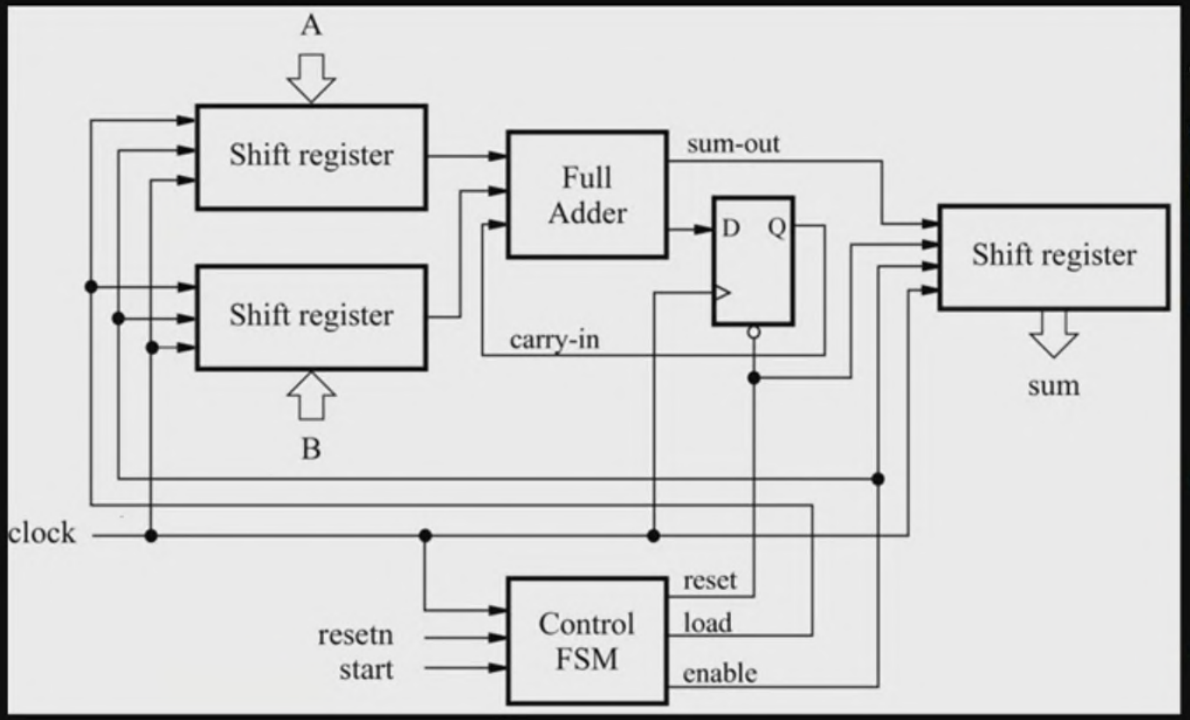
FPGA Simulation Flow

In FPGA development, simulations for design verification are done at 2 separate times in the design flow:

- After Analysis a functional simulation is done, which only determines if the logic is correct and does not include any timing models.
- After fitting (place and route) a timing simulation can be done which will identify timing violations. These will show up in red on the ModelSim waveform viewer. This simulation can help achieve timing closure.
- This process is often iterative as shown in the diagram, where errors are corrected, recompile, resimulate, etc. until no errors remain.

[1]

FPGA Simulation Example



To learn about the FPGA flow, we will work through a complete example using Quartus Prime and the Altera version of ModelSim

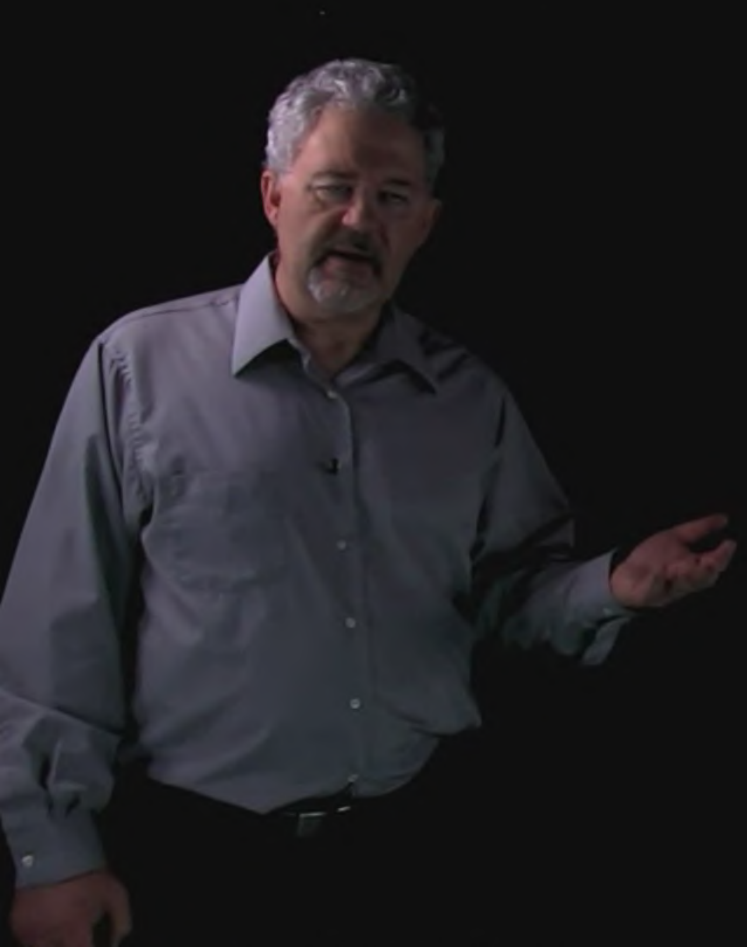
- Launching ModelSim from Quartus has the advantage of including all the Altera model libraries automatically.
- The example is based on a serial adder circuit as shown in the diagram. We will be using VHDL for this example.
- The code for this example is provided in the including Using_ModelSim.zip file which you can download.

Simulation with Altera ModelSim

Come Simulate with us!

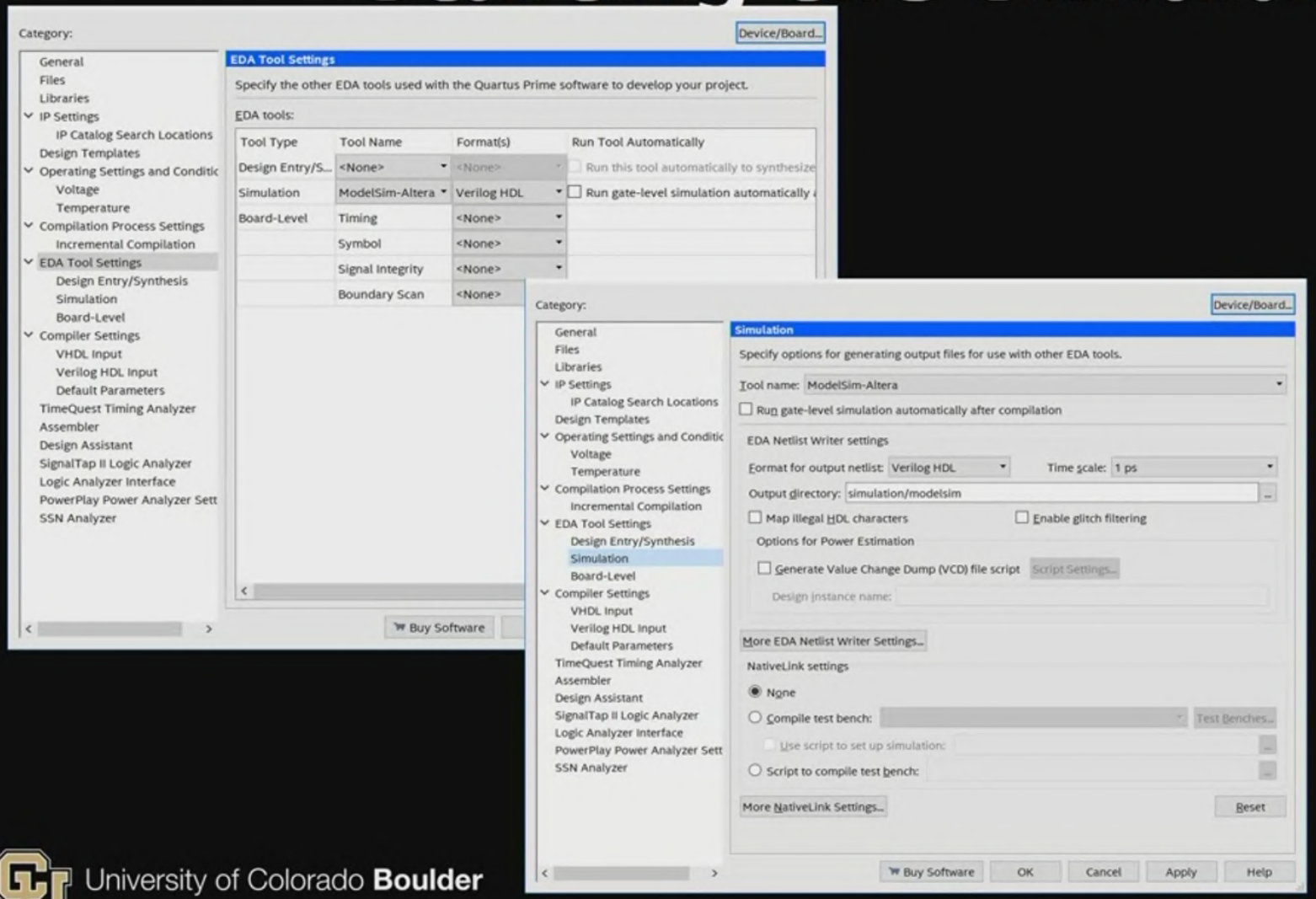
- You should have Quartus Prime installed
- Start Quartus Prime and select New Project and browse to your new project directory, and select serial.qpf in the timing directory.

Starting the Simulation



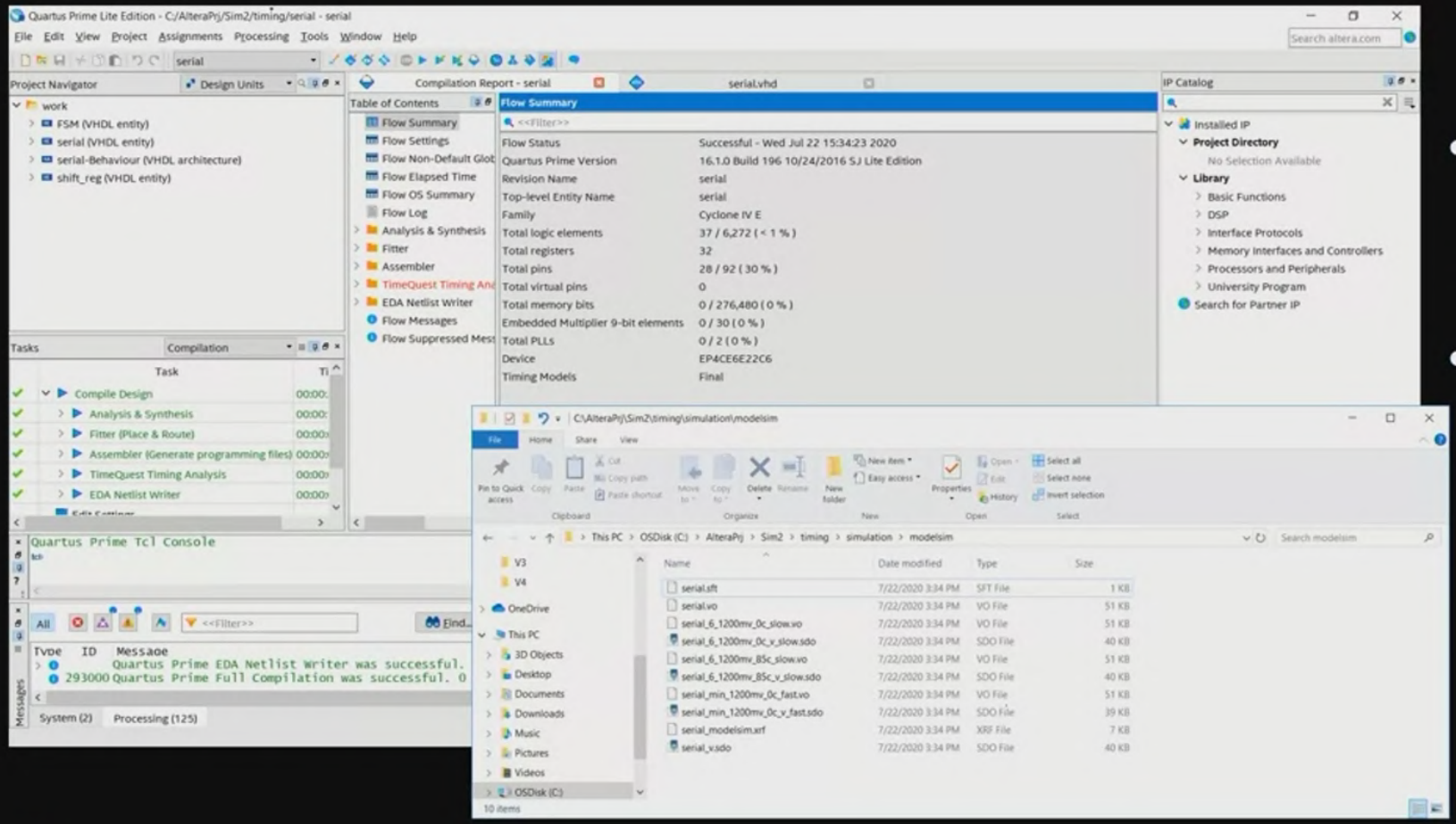
- You may see a message about use of the Cyclone II family in this design. Select Cyclone IV E instead. Exploring use of other devices with this project at a later time is encouraged

Starting the Simulation



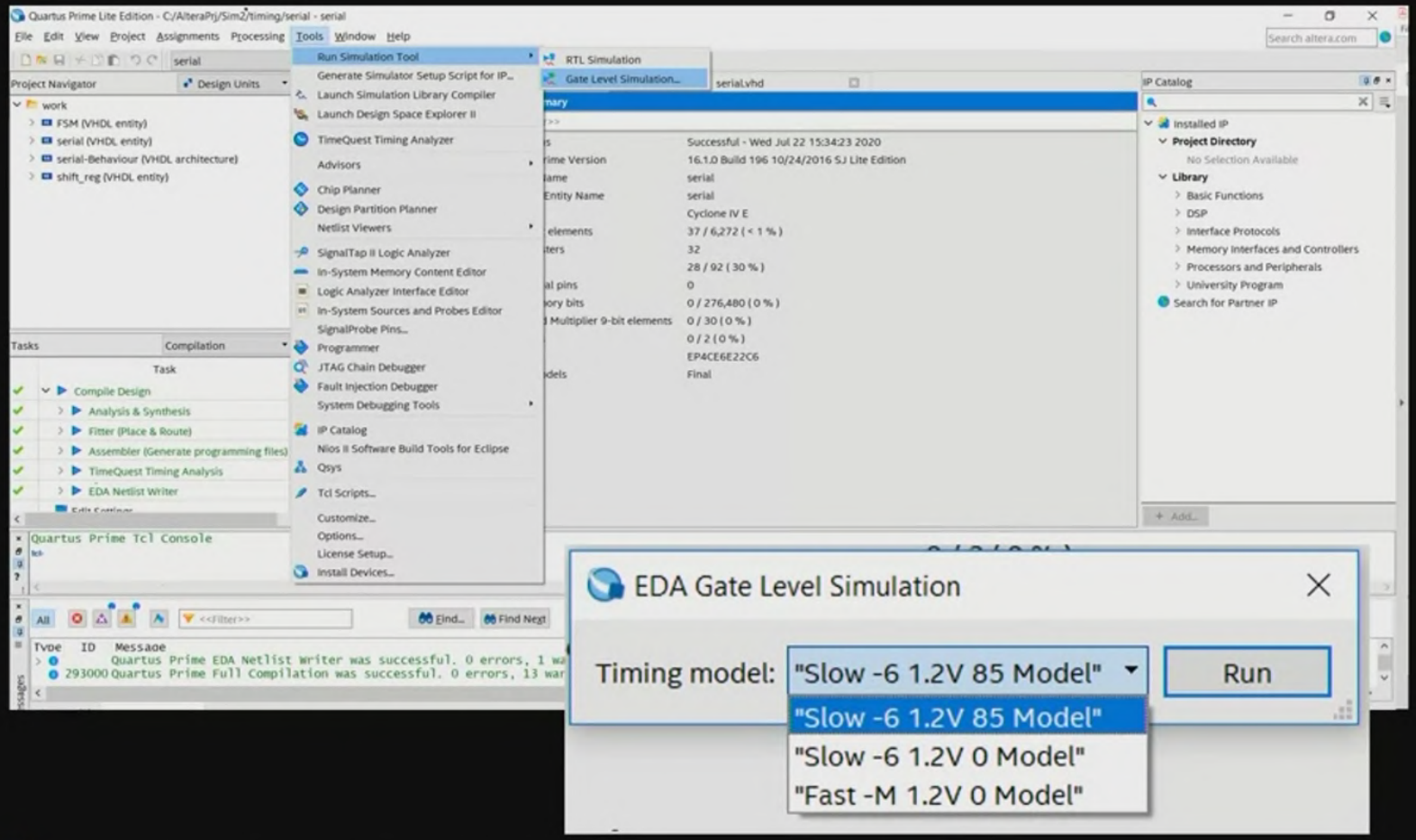
- In Quartus, Open the project serial.qpf in the timing directory
- Use Assignments -> Settings -> EDA Tools and then Simulation to setup the simulation

Starting the Simulation



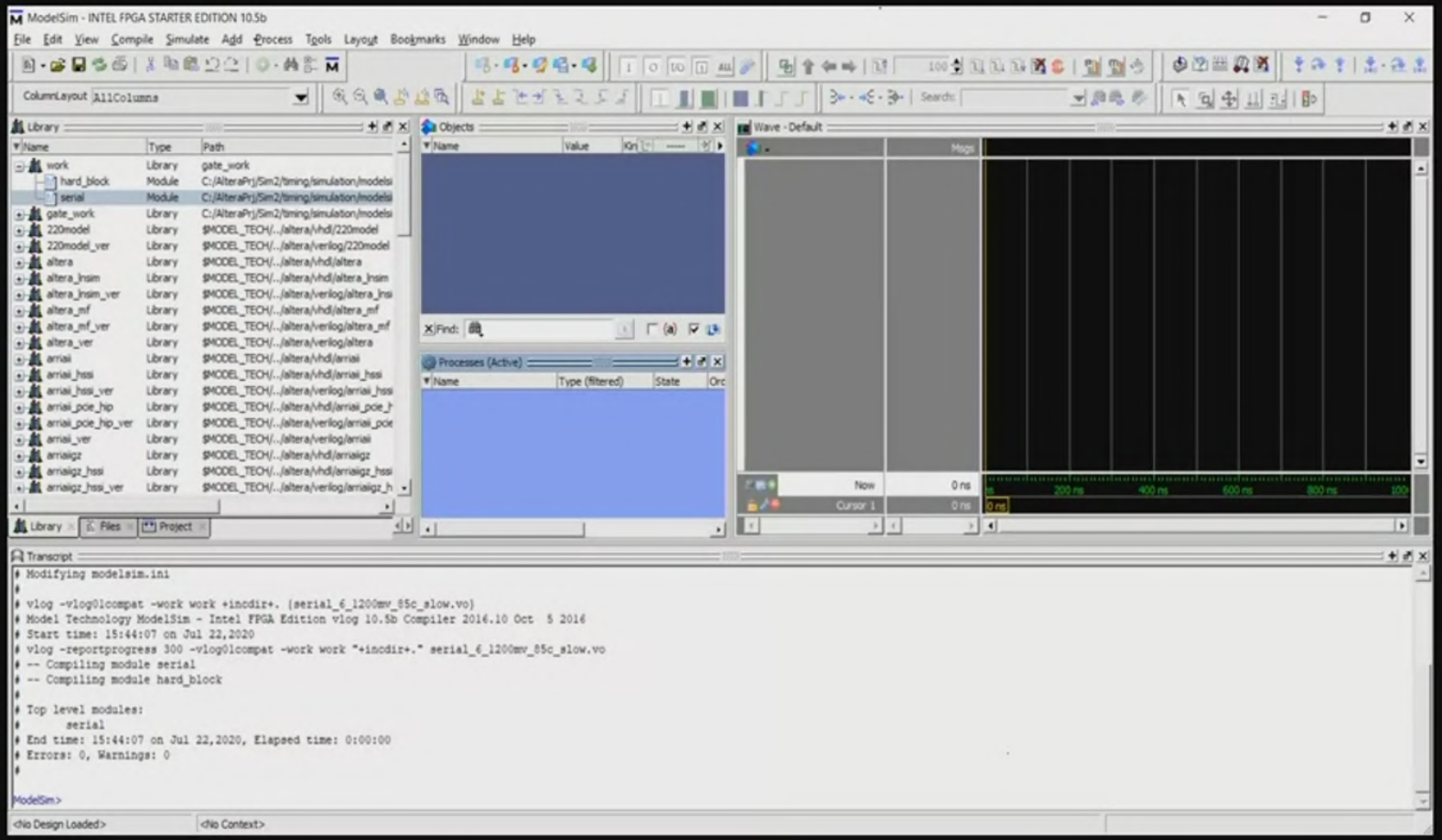
- In the design flow, Run Full Compilation
- Note the creation of .vo and .sdo files for 3 timing corner cases

Starting the Simulation



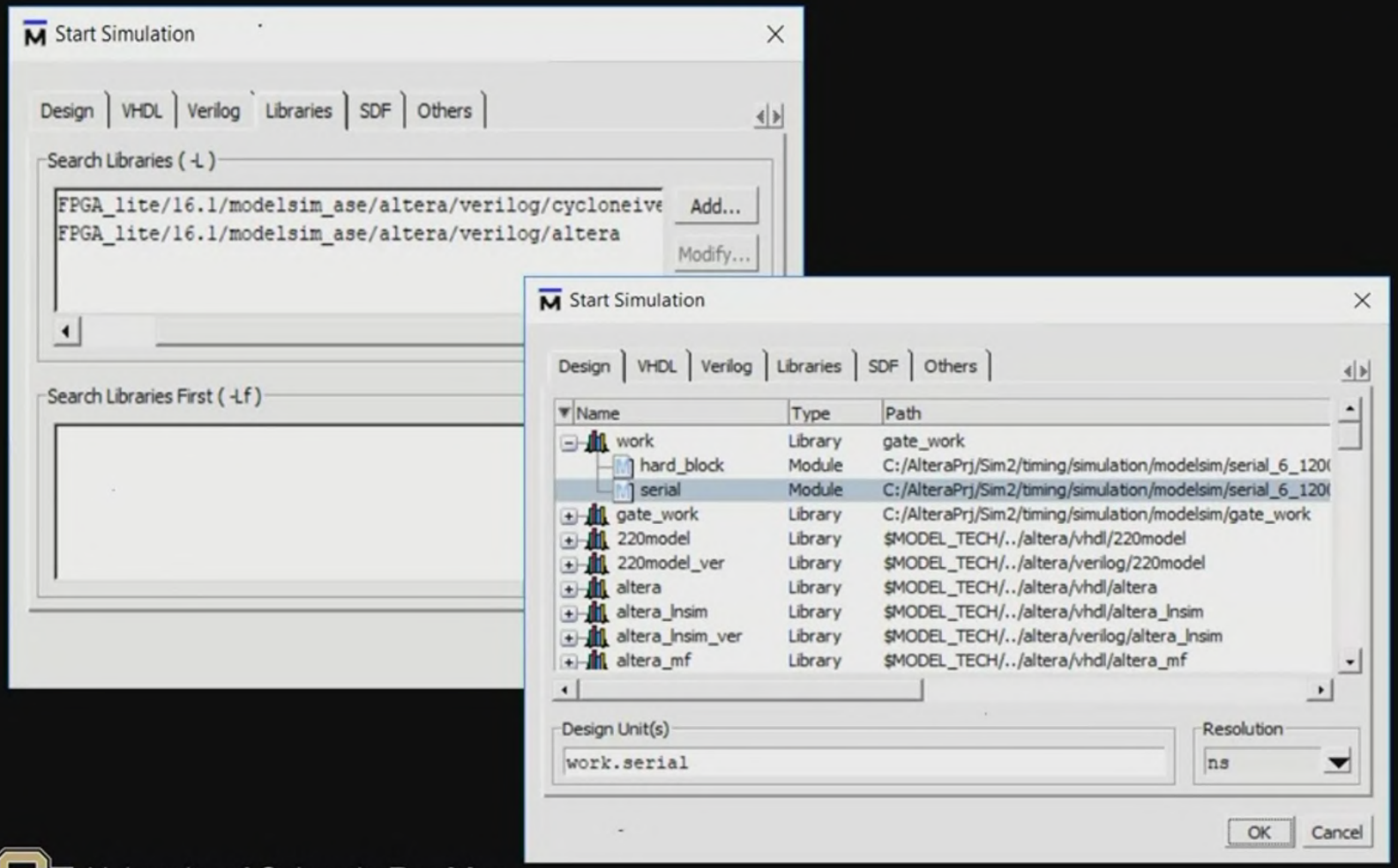
- Start the simulation from Quartus by selecting Tools -> Run Simulation -> Gate Level Simulation from the top menu
- Note the choice of timing model, choose slow 85

Starting the Simulation



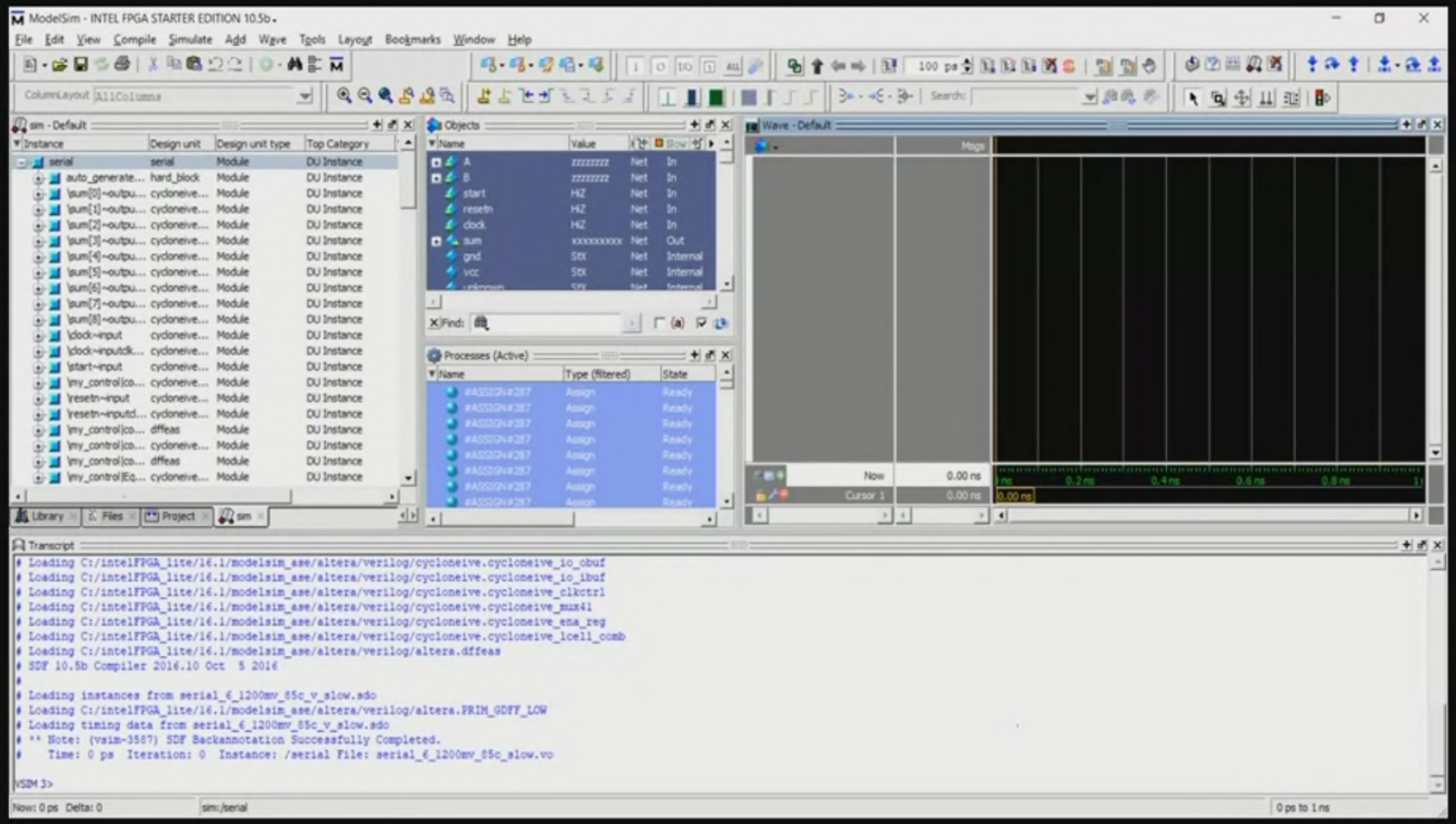
- ModelSim should start. Transcript messages should indicate the top level is serial

Starting the Simulation



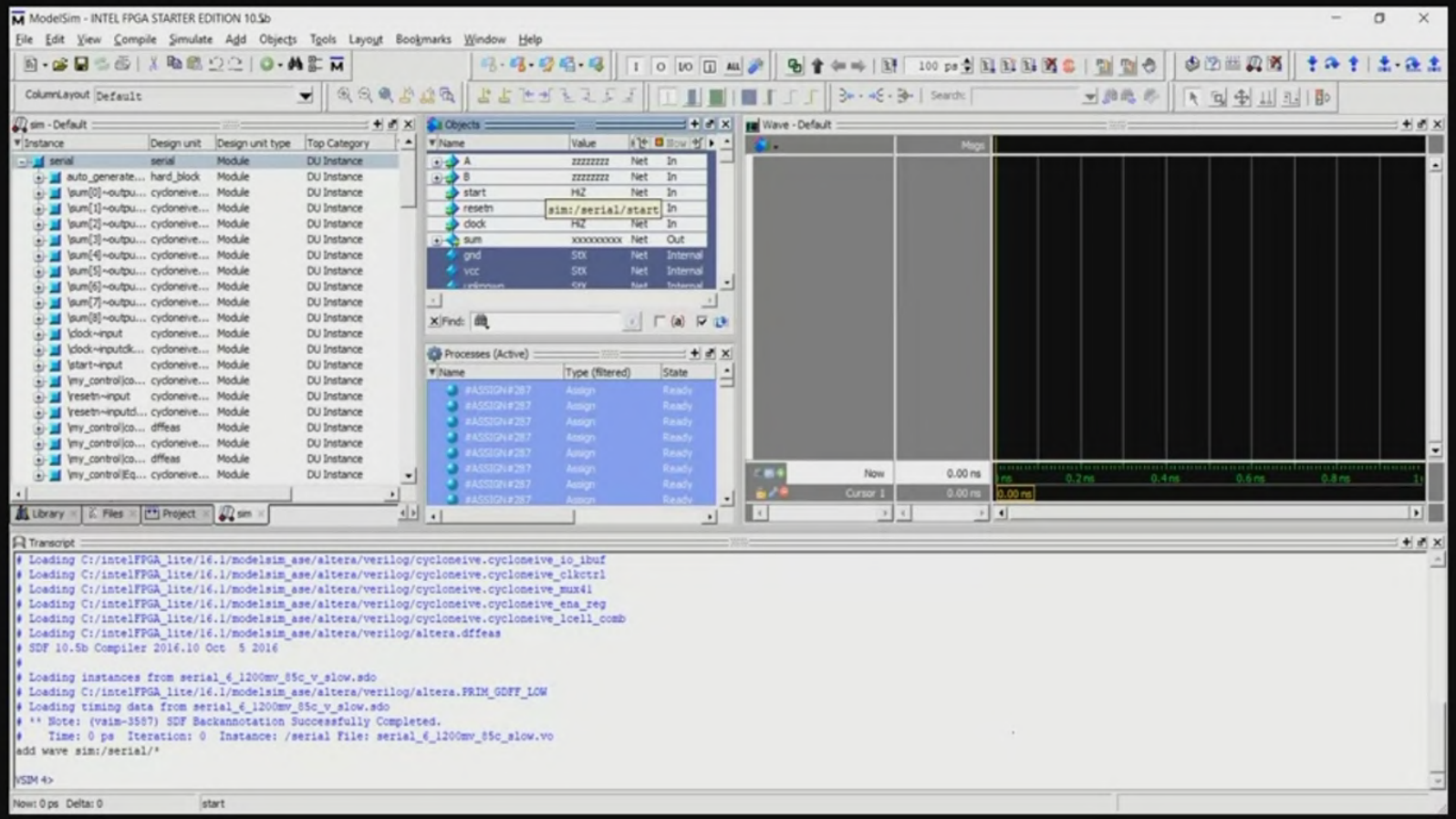
- Click on Simulate, -> Start Simulation.
- In the library tab, add the cycloneive and altera libraries by browsing the intelFPGA lite 16.1 -> modelsim directory
- Select the serial.vdh file from the work library with resolution to ps and hit OK to start simulating

Starting the Simulation



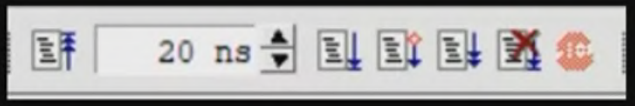
- The simulation should begin. The sim window appears, and many more signals appear in the object window.
- The transcript shows the loading of library timing models

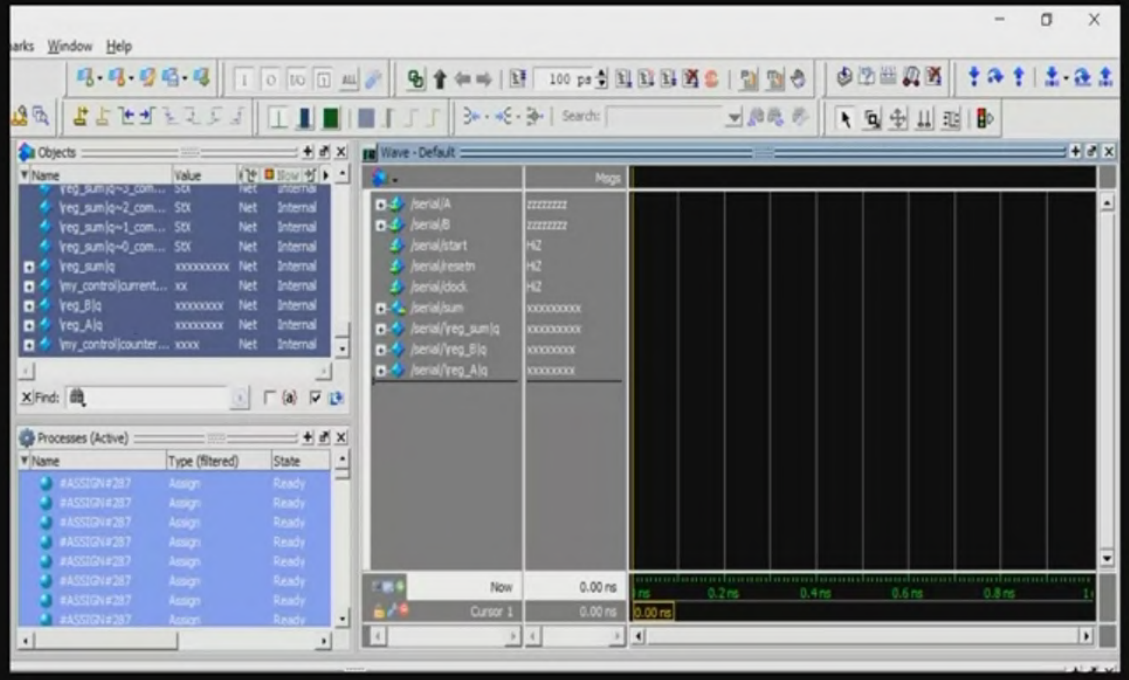
Running the Simulation

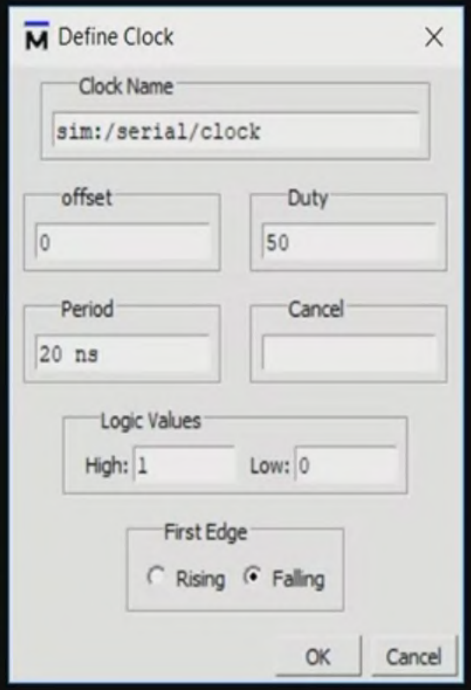


- Add signals to the waveform window by selecting and dragging the 1st 6 and then scrolling down to the bottom to get reg_sum|q, reg_A|q, and reg_B|q

Running the Simulation

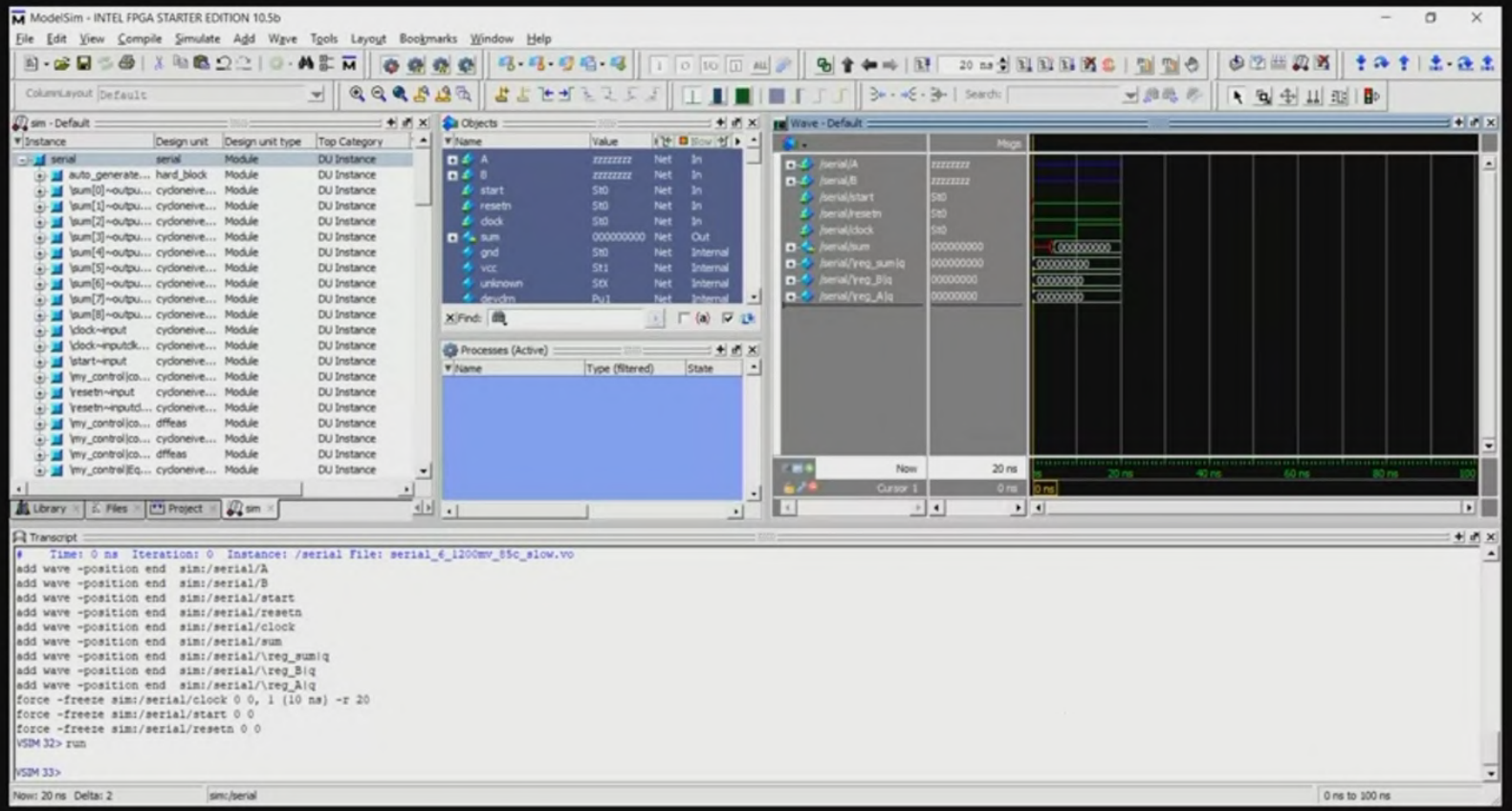






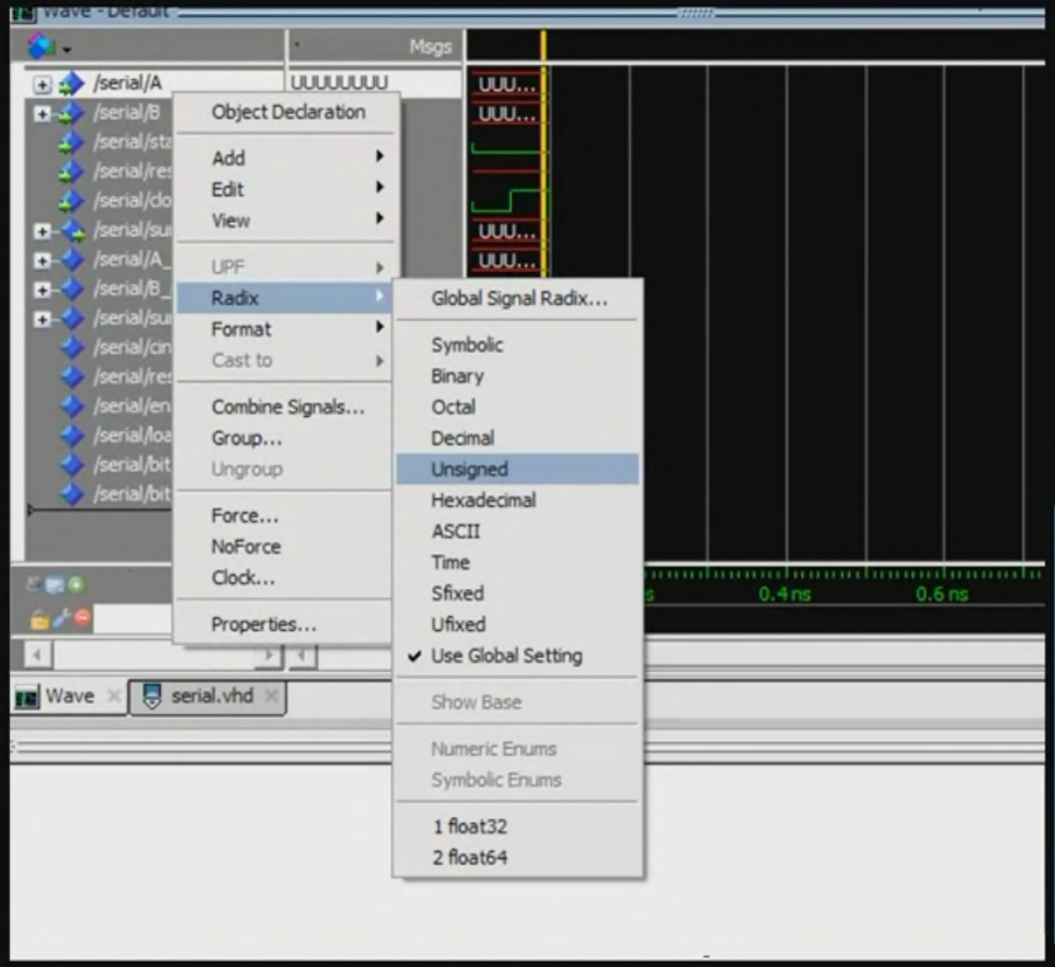
- Set the simulation time to 20 ns.
- We have no testbench, so manually add the clock and reset stimulus. Right click on clock in the waveform window and select clock with period 20 ns (include units).
- Right click on resetn and force it to 0. Do the same for start.

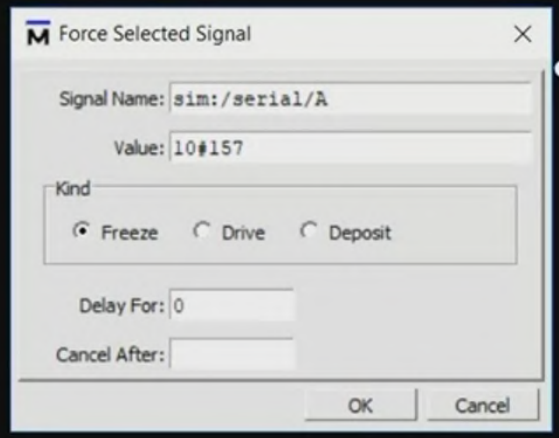
Running the Simulation

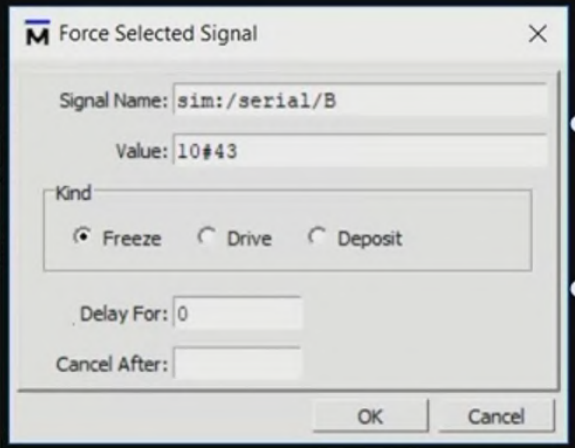


- Click on the run button to the right of the 20 ns menu icon to run the simulation.
- Some signals will be unknown as shown in the red and in the transcript window

Running the Simulation

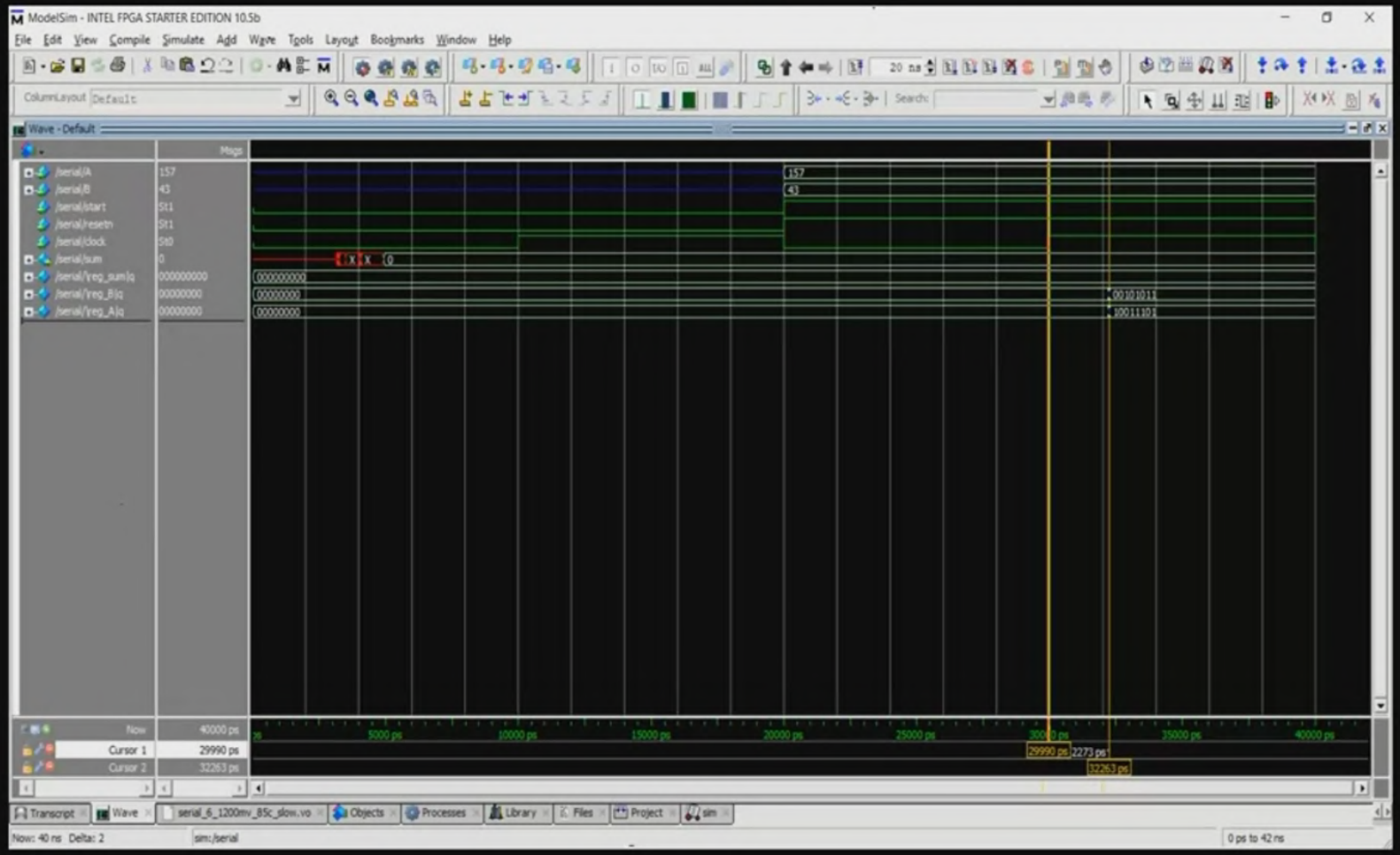






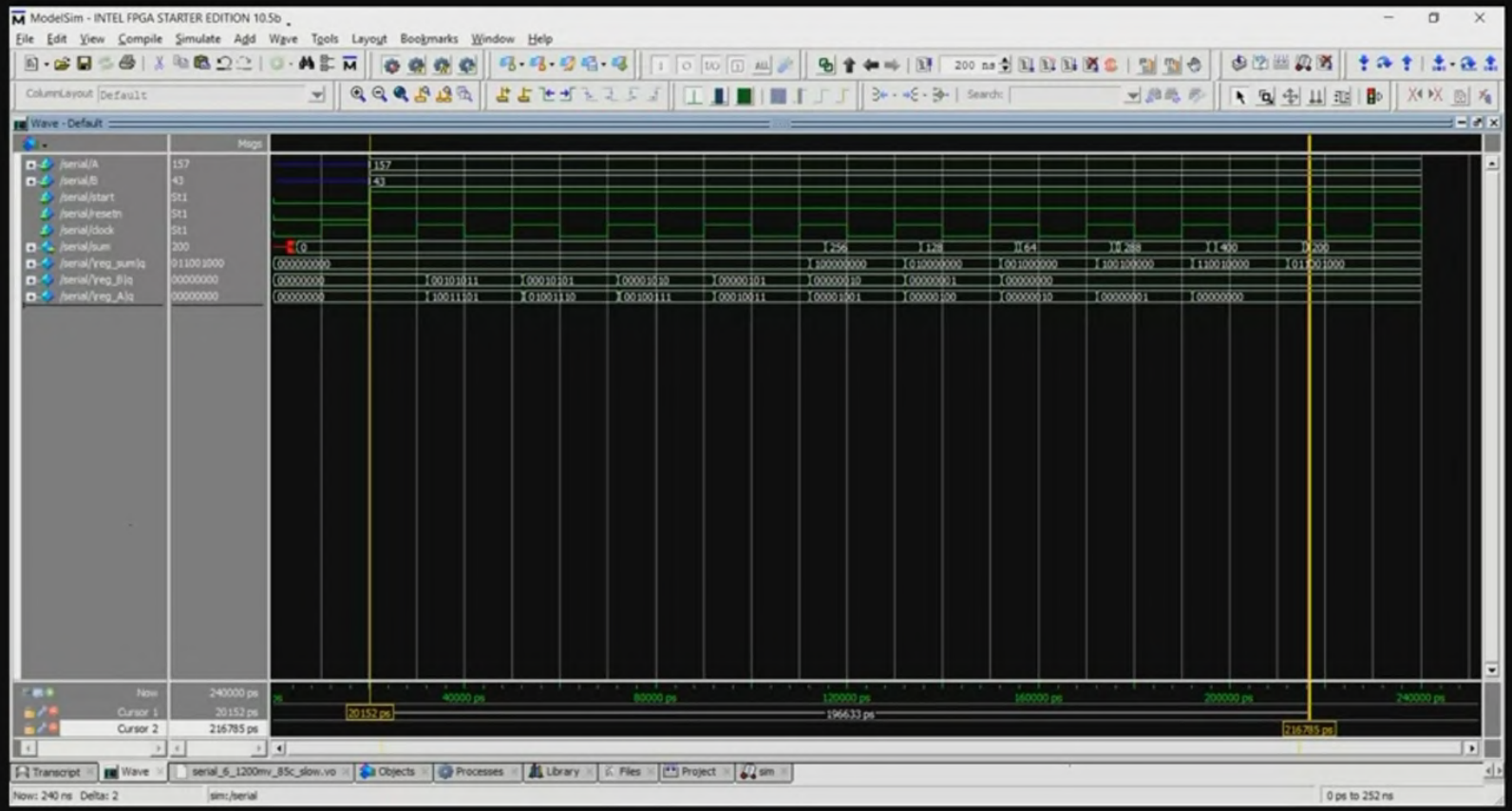
- Force A to 157 and B to 43 as before. Right click on this signals and changes the radix to unsigned
- Set sum to unsigned as well.
- Force resetn and start 1

Running the Simulation



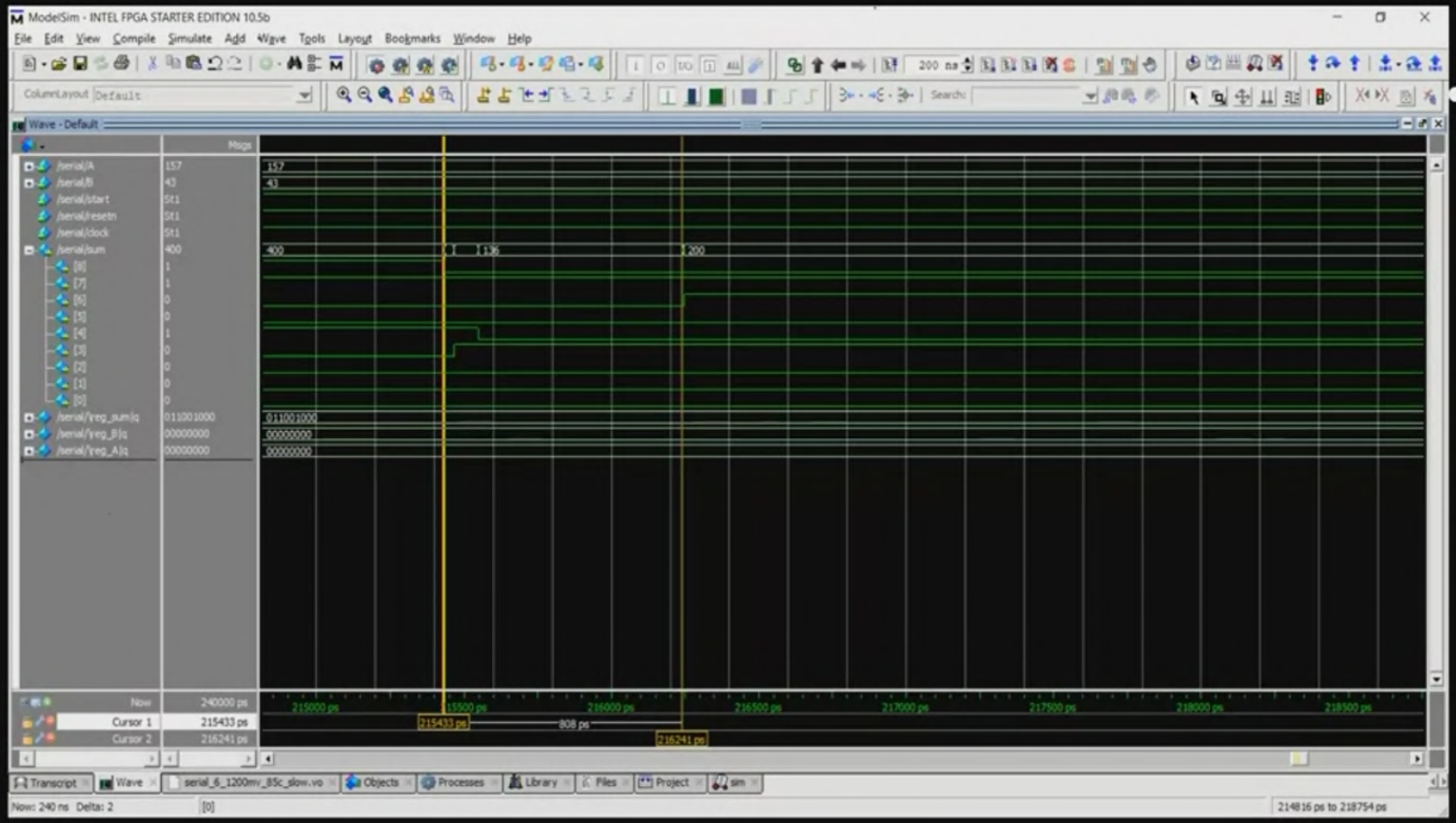
- Run for another 20 ns.
- Use the cursors to measure the time delay from the clock edge to the A and B register data. It's about 2.3 ns.

Measuring time delays

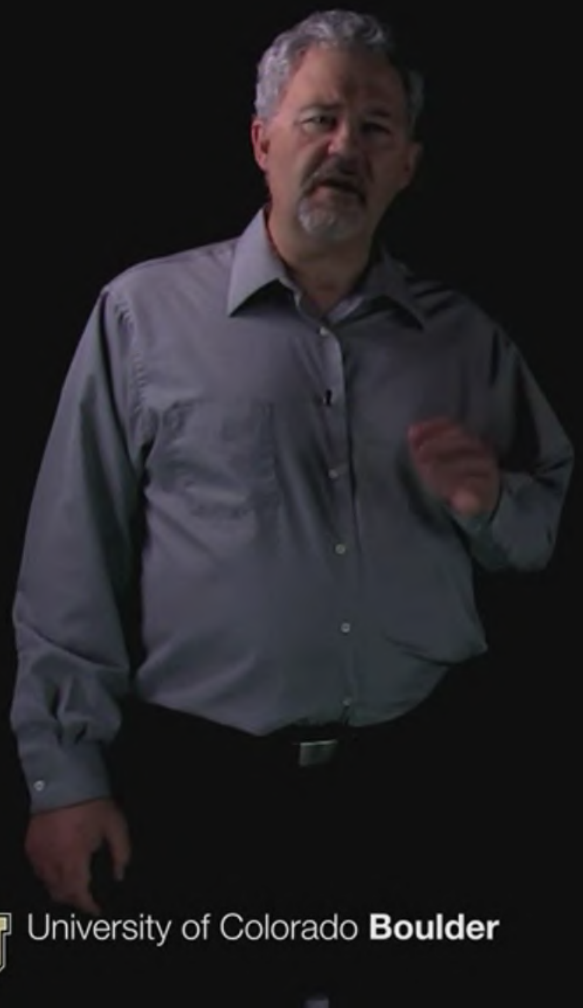


- Change the Run time to 200 ns and run
- Use the cursors to the measure the time delay from the first appearance of the input data until the sum, about 196 ns.

Measuring time delays



- Zoom in on the final sum transition. The data goes through several values before settling to 200 over about 0.8 ns. The data bits do not all change at the same time.



Summary

In this video, you have learned:

- How to start and run a timing simulation in ModelSim from Quartus prime.
- How to use simulation to verify the correct timing of a VHDL serial adder.
- How to interact with the simulation by zooming in to signal waveforms to discovery timing details.

References

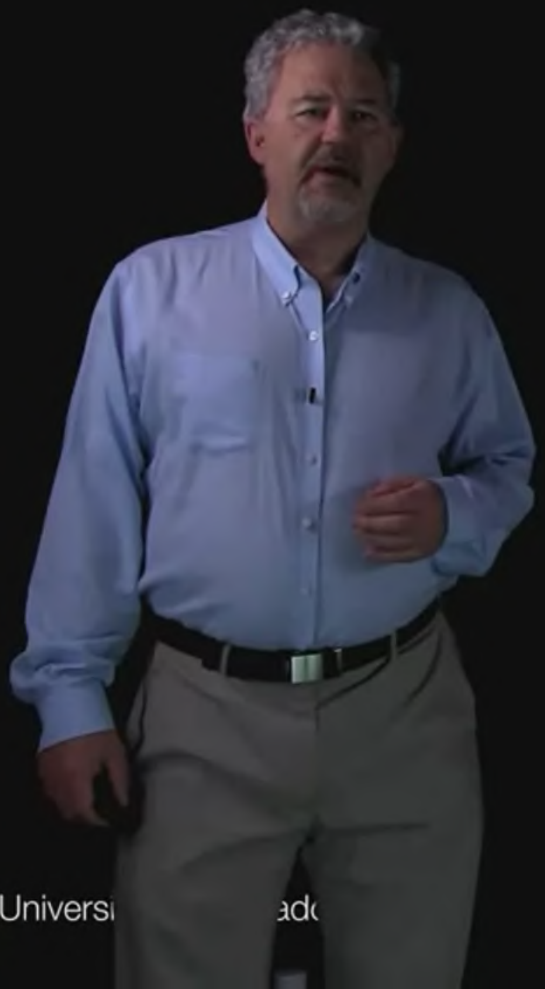
[1] Intel Altera. (2015/Oct/18), *Using ModelSim to Simulate Logic Circuits in VHDL Designs* [Online]. Available: https://www.mentor.com/products/fpga/verification-simulation/modelsim/

[2] Mentor Graphics Corporation. (2019/Jul/17), *ModelSim® User's Manual* [Online]. Available: https://www.mentor.com/products/fpga/verification-simulation/modelsim/

FPGA Design for Embedded Systems

FPGA Softcore Processors and IP Acquisition

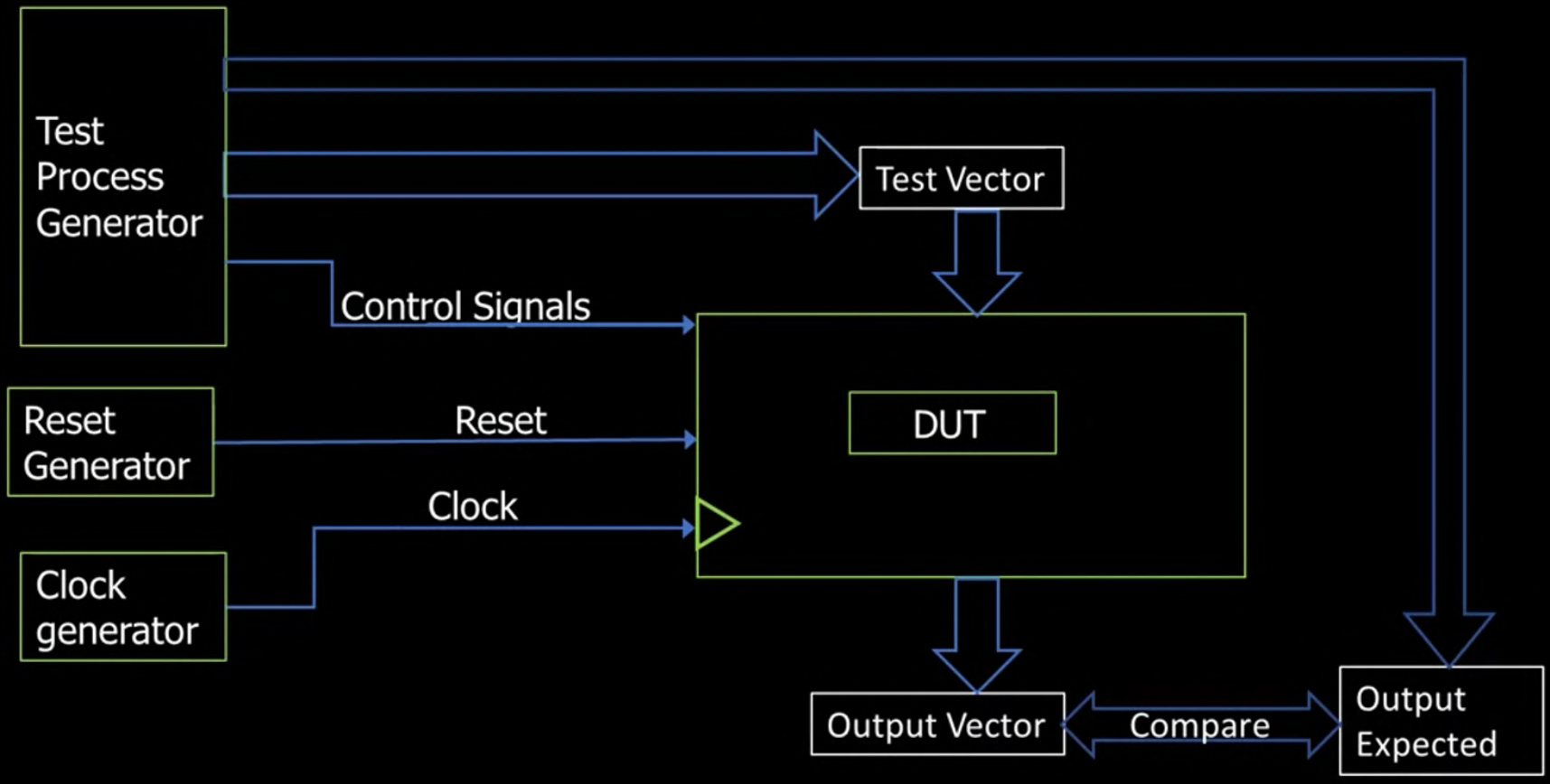
Testbenches for Verification



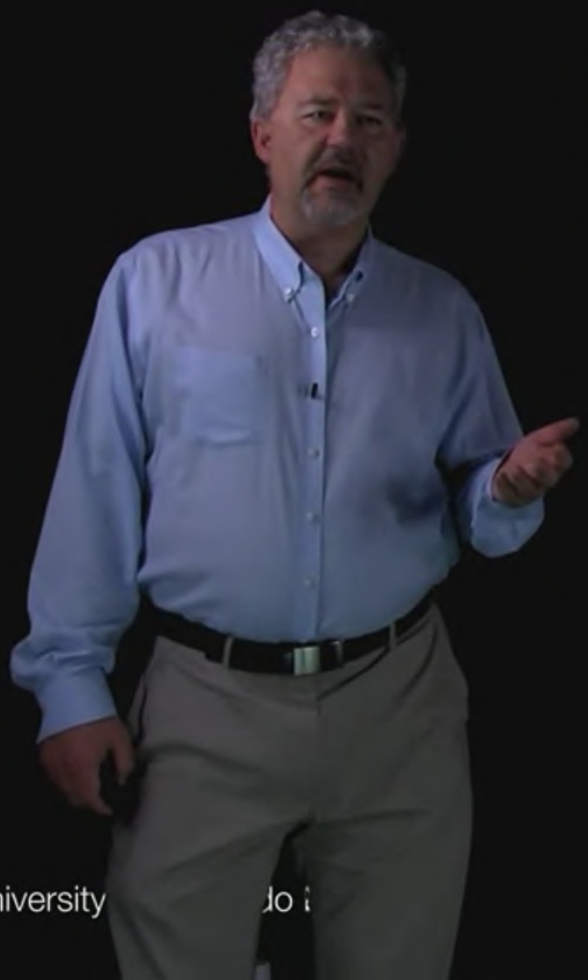
In this video, you will learn:

- How to write simple testbenches for synchronous circuits
- How to use external signal generators to create stimulus for synchronous circuits.
- How to extend testbenches to provide verification of circuit design functional correctness.

Generic Test Bench

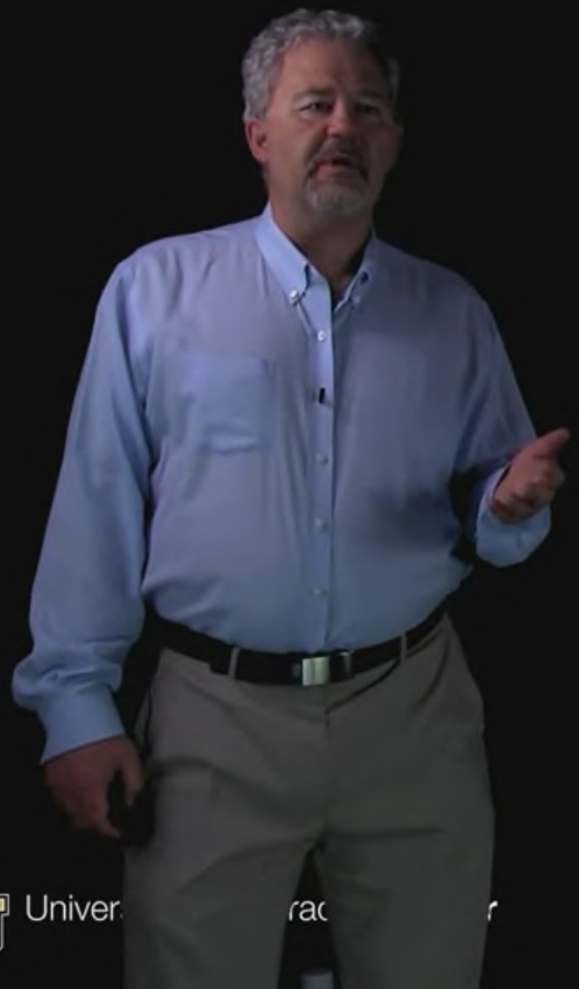


Counter Test Bench



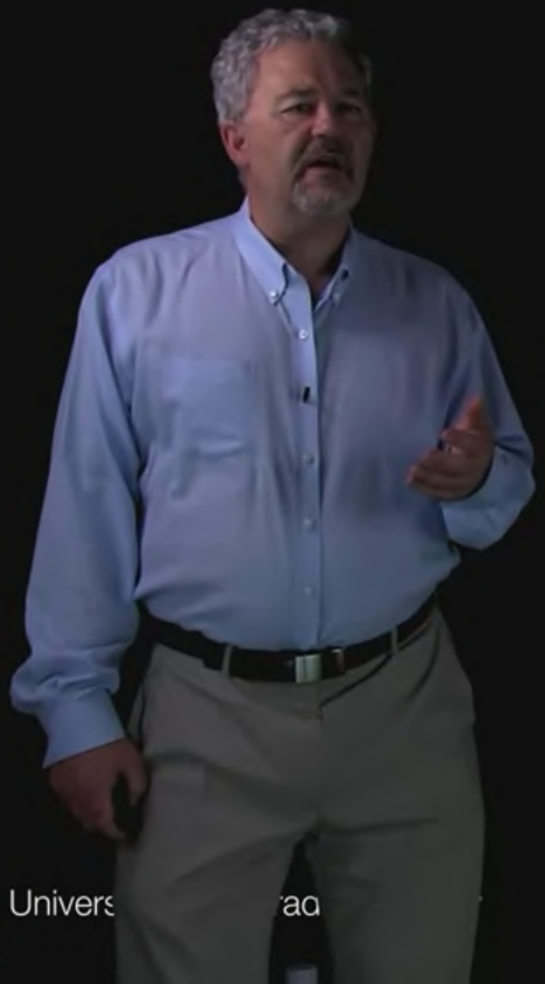
```
timescale 1 ns / 1 ps    // set timescale
                         // to nanoseconds, ps precision
//Testbench entity declaration
module Counter_tb ();  // top level, no external ports
//constant declarations
  parameter delay = 10;  //ns  defines the wait period.
  localparam n = 4;  // width of counter in bits
  localparam T = 20;  // clock period
//  signal declarations
  reg clock = 0; //clock if needed, from generator model
  reg reset = 0;    // reset if needed
  reg [n-1:0] data_tb = 4'b0000;  // data input stimulus
  reg load = 0, en = 0;  // input stimulus
  wire [3:0] q;  // output to check
  reg [n-1:0] checkout= 4'b0000;// variable to compare
                               // to count or load
```

Counter Test Bench



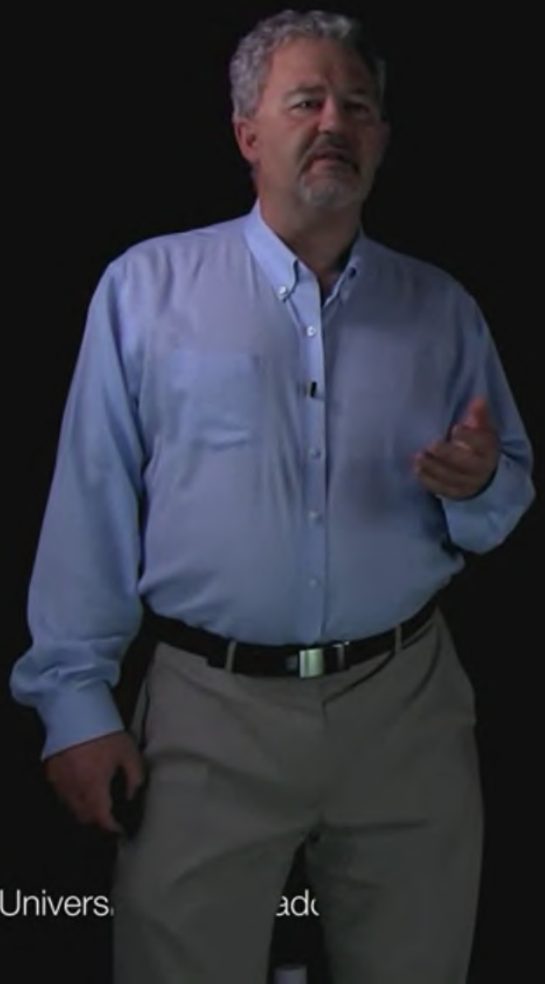
```
// Component Instances
// instantiate the device under test
Counter DUT (     // Device under Test
        // Inputs
        .d(data_tb),
        .clk(clock),
        .reset(reset),
        .load(load),
        .en(en),
         // Outputs
        .q(q)
        );
```

Counter Test Bench



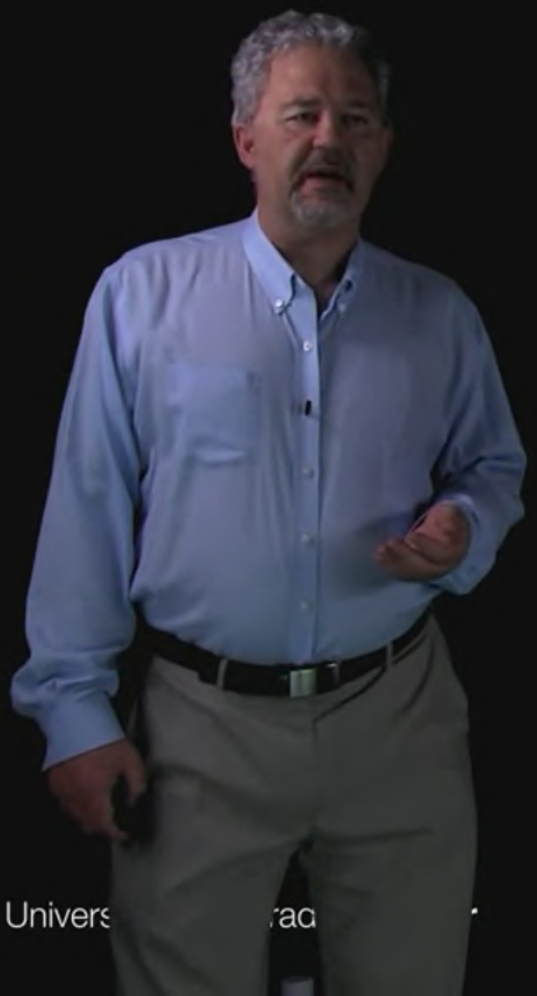
```
// External Device Simulation Processes
// clock driver
  always
  begin
    clock = 1'b1;
    #(T/2);
    clock = 1'b0;
    #(T/2);
  end

// reset driver
  initial
  begin
    reset = 1'b1;
    #(T/2);
    reset = 1'b0;
  end
```

Counter Test Bench



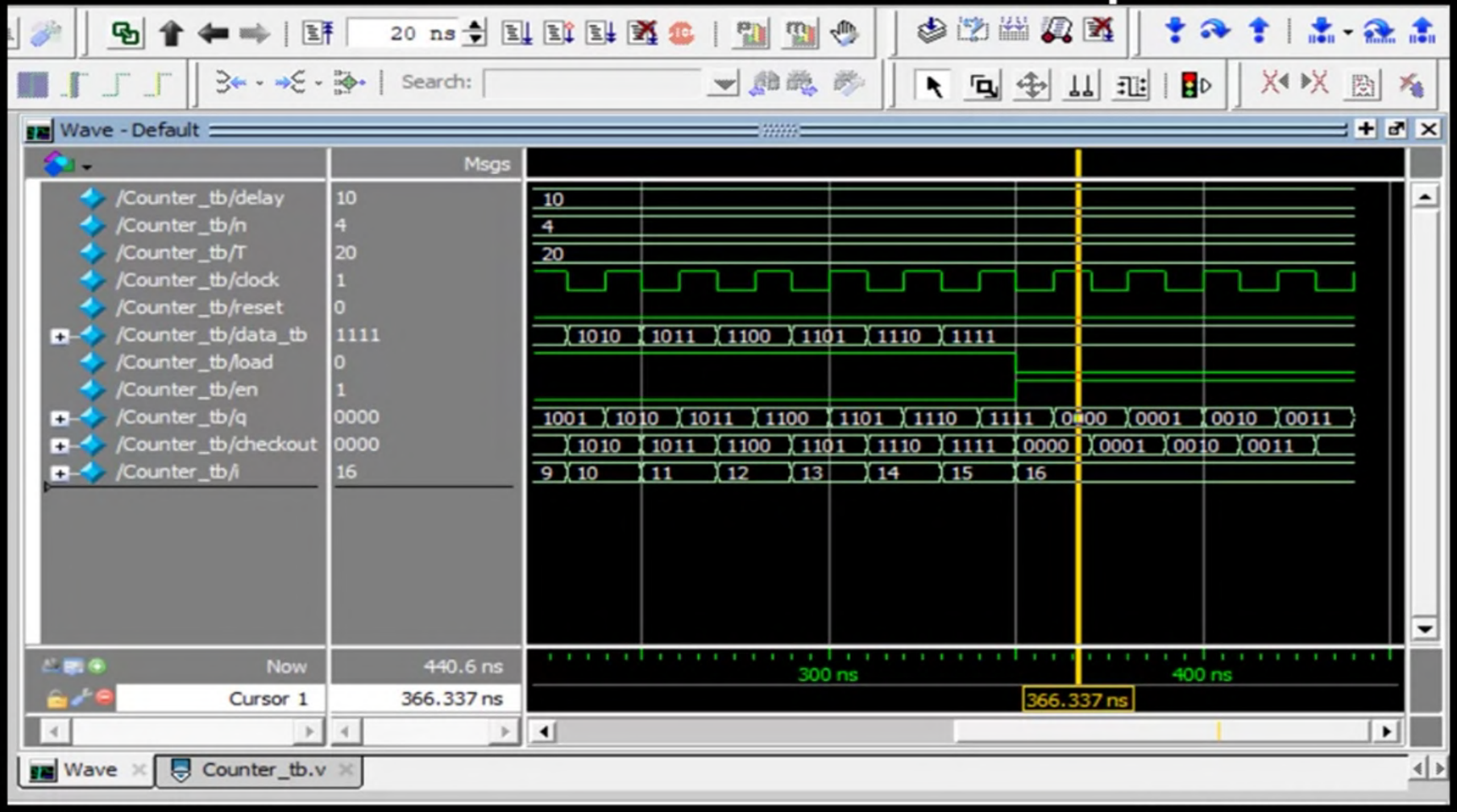
```
// Test Process
  initial    // test generation process
  begin
    @(negedge reset) // wait for reset inactive
    @(negedge clock) // wait for one clock
  // Test load
      load = 1'b1;
      en = 1'b0;
      for (i = 0; i<2**n; i = i+1)
      begin
        checkout = i;
        data_tb = i;
        @(negedge clock) // wait for one clock
        if (q != checkout)
          $display("Load failure %b", q);
      end
```

Counter Test Bench

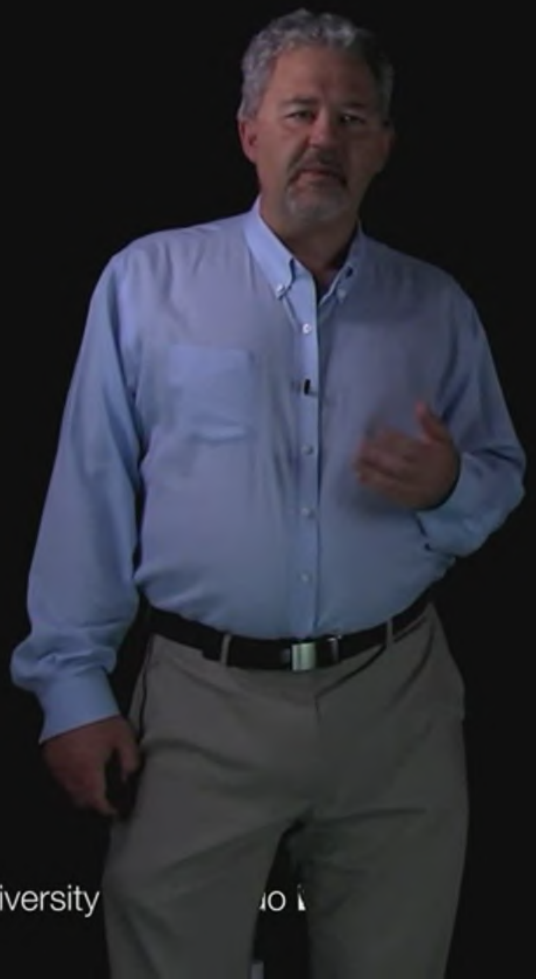


```
// Test Process, continued
   // test count
    checkcount = 4'b1010;  // compare variable
    load = 1'b0;
    en = 1'b1;
    repeat (2*2**n)
    begin
      checkcount = checkcount + 1;   // count
      @(negedge clock) // wait for one clock
      if (q != checkcount)
        $display("Count failure at time %g/t at
                  count %b", $time, q);
    end
    $stop;   // end simulation
  end
endmodule
```

Counter Test Bench Output



Summary – Testbenches in Verilog II



In this video, you have learned:

- **How to write simple testbenches for synchronous circuits**
- **How to use external signal generators to create stimulus for synchronous circuits.**
- **How to extend testbenches to provide verification of circuit design functional correctness.**

References

[1] Pong P. Chu, "Regular Sequential Circuit" in *Embedded SOPC Design with NIOS II Processor and Verilog Examples*, Hoboken, NJ, Wiley, 2012, ch. 5, sec. 5.2, pp. 107-110

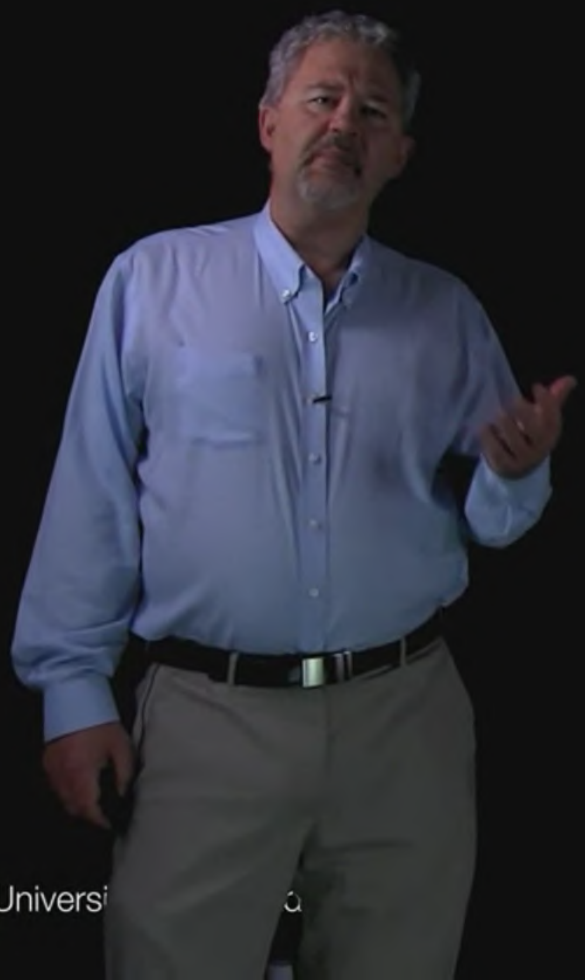
[2] D. Smith, "Test Harnesses" in *HDL Chip Design, A practical guide for designing, synthesizing and simulating ASICs and FPGAs using VHDL or Verilog,* Madison, AL, Doone Publications, 1996, ch. 11, pp. 323-344.

FPGA Design for Embedded Systems

FPGA Softcore Processors and IP Acquisition

University of Colorado **Boulder**

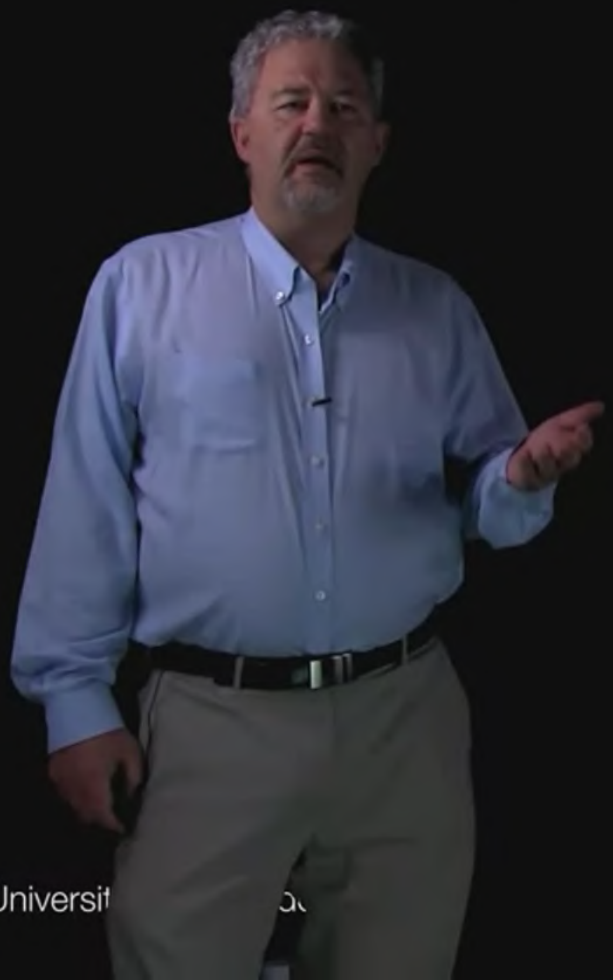
Designing for Simulation



In this video, you will learn:

- How the simulator works so that you can design HDL code that works in simulation
- An understanding of the techniques used to model concurrency using sequential software
- How to write your code so that it works for both synthesis and simulation
- Tips for avoiding common problems in simulation code

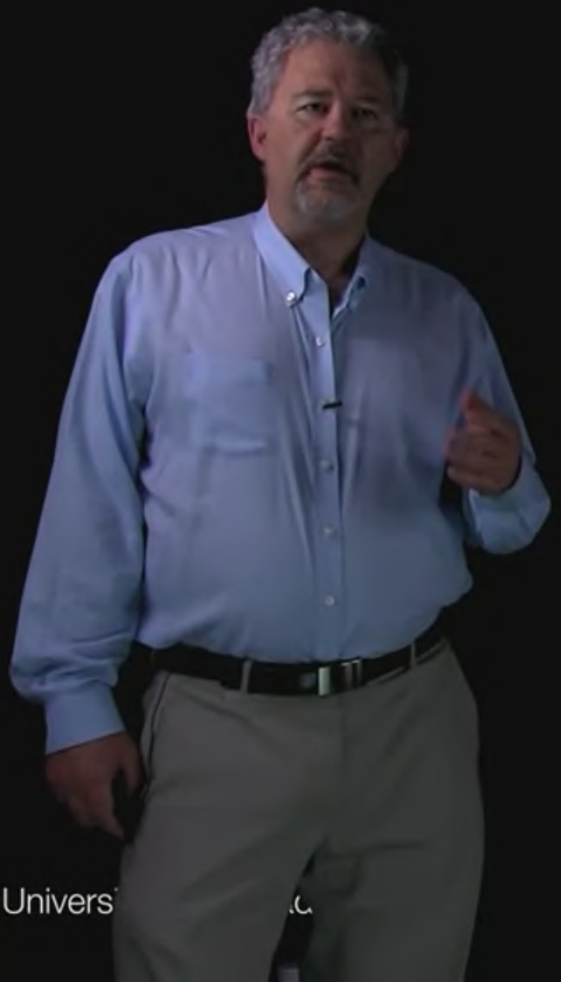
Designing for Simulation



An HDL Simulator is just a Software Program

- It models the actions of hardware, but also includes programmatic constructs and interactive elements. Based on TCL scripting, it is a complete and powerful development environment.
- It uses clever programming techniques to create the appearance of concurrency to correctly model how real hardware behaves.
- Some HDL code will simulate but not synthesize. Some HDL code will synthesize but not simulate. Your HDL code describing hardware should both simulate and be synthesizable.

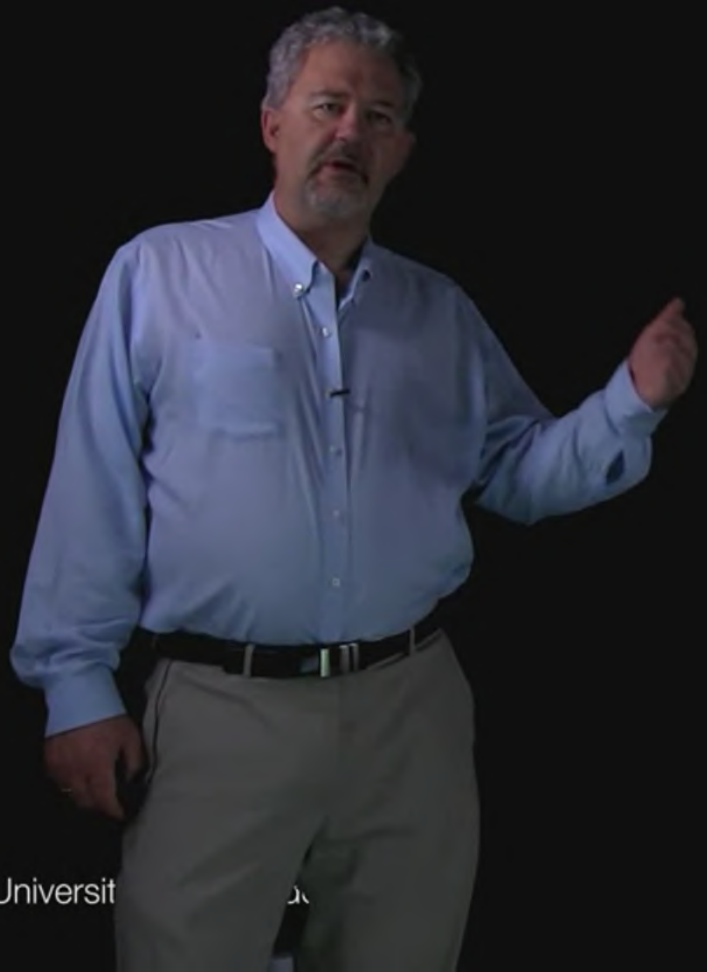
Modeling for Simulation



- Modeling for synthesis is primary emphasis; but we need to simulate the code as well. Code must satisfy simulation syntax and semantics

  Main simulation concepts
  - Simulation Cycle – Stimulus and Response
  - Simulation Timing
  - Sensitivity Lists
  - Signal Drivers
  - Resolution Functions

Simulation Cycle

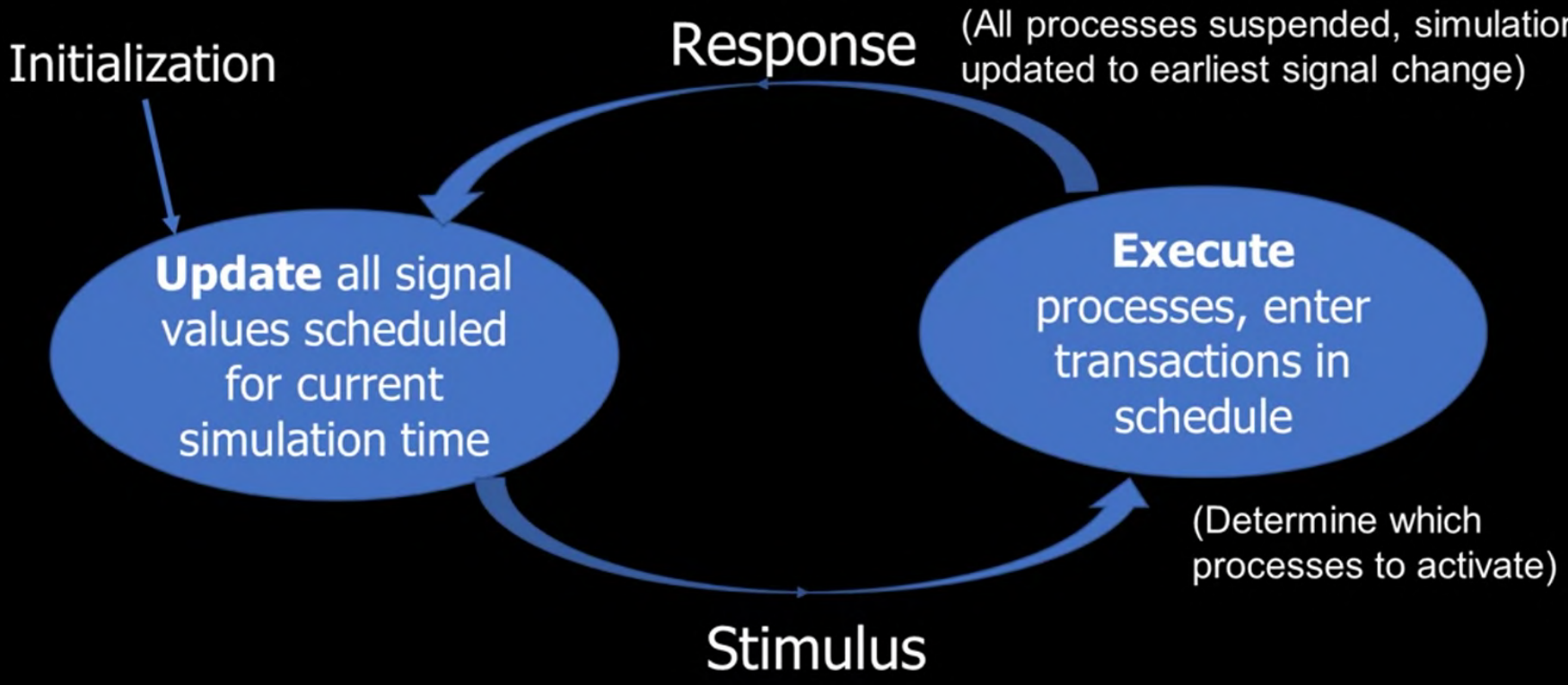


Modeling for Simulation

VHDL code consists of numerous **concurrent** statements or processes

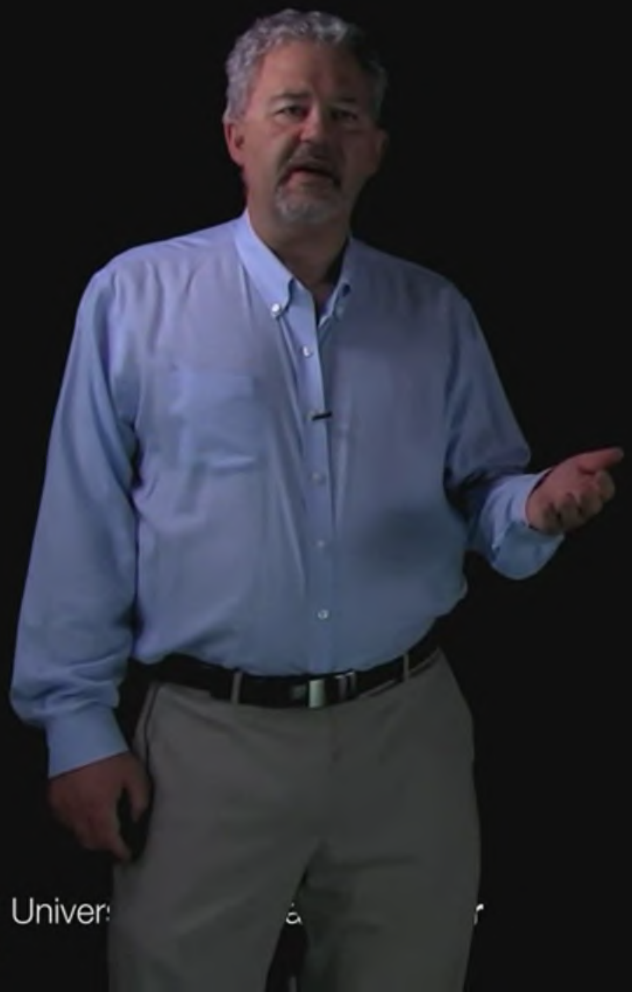
- Stimulus/Response paradigm
  - Stimulus (inputs) changes at specific simulation time
  - Processes evaluate resulting logic changes
  - Signal values change after specific simulation time has elapsed
- All concurrent processes repeatedly execute in parallel (concurrently), following a standard simulation cycle

Simulation Cycle



[1]

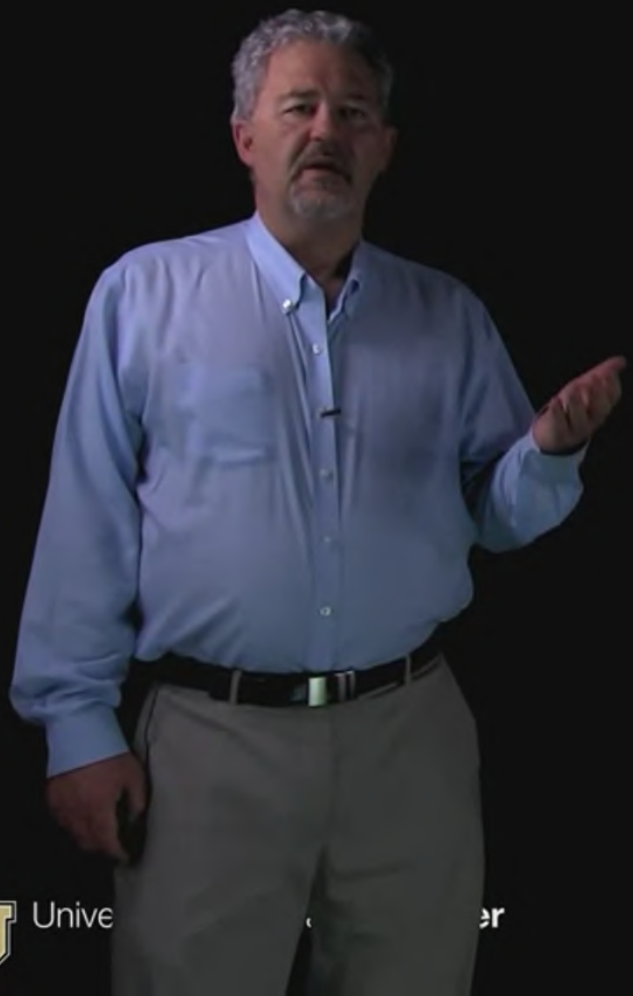
Simulation Cycle 1: Initialization Phase



- Simulation time set to zero
- Signals initialized to
  - explicit values specified in code, if they exist
  - else, the following:
    - `'0'` for bit
    - `FALSE` for boolean
    - `'U'` (unknown) for type std_logic and std_ulogic
    - `-2,147,483,647` ($-2^{31}-1$) for 32-bit integer
    - `first value` for enumeration type
- Every process runs until it suspends
- Simulation time is still zero

[1]

Simulation Cycle 2: Update Phase

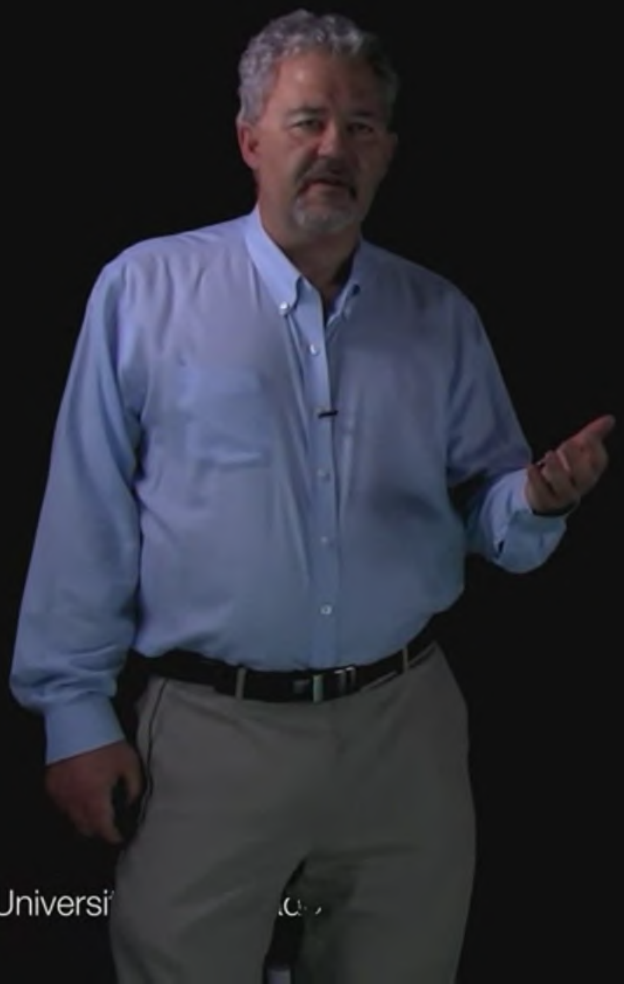


All signals are updated that have a transition scheduled at the current simulation time (i.e. signal events occur)

- Other signals do not change (i.e. no signal events occur)
- Simulation time does not change

[1]

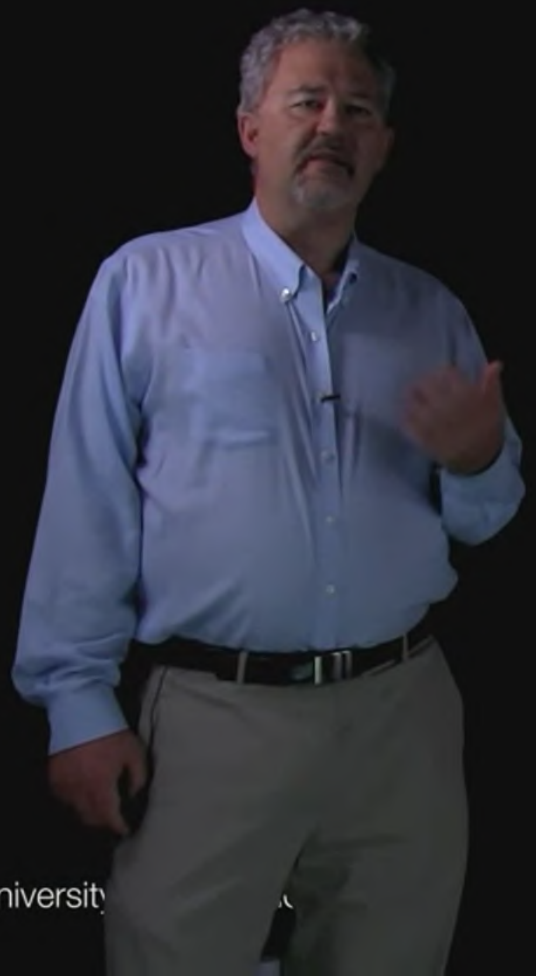
Simulation Cycle 3: Process Execution Phase



Each process is executed IF it is sensitive to a signal that had an event in the current simulation cycle

- The simulation time of the next cycle is determined. This is the earliest time among these possibilities:
  - the earliest time that any signal has an event scheduled
  - the earliest time at which any process is scheduled to resume
- If the new time is a delta delay, a new simulation cycle is begun with the same simulation time as the current simulation cycle
- Otherwise the simulation time is updated and a new simulation cycle begins

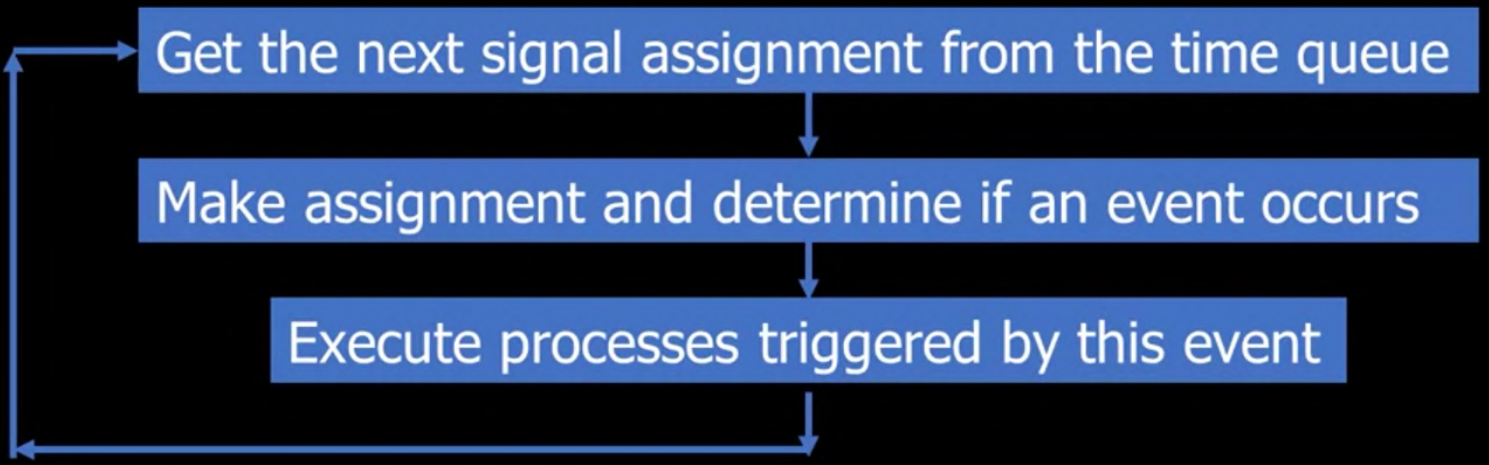
Simulation Process Execution



A process is either being executing or it is suspended

- Execution begins when any signal in its sensitivity list has an event (change in value)
- During execution, sequential statements in the process are executed in succession
- Execution continues until the process is suspended, due to:
  - execution of the last sequential statement in the process, or
  - execution of a WAIT statement

Simulation Loop



Time queue: A <= '1' at 2ns, B <= '0' at 4 ns, C <= '1' at 10 ns
Actually represented by ordered pairs (value, time) on the time queue

A: ('1', 2 ns), B: ('0', 4ns), C: ('1', 10 ns)

Process Triggering

A process can be triggered by resumption after a wait statement or by an event on a signal in its sensitivity list:

```
process (a, b, s)
begin
…
end process;
process
begin
…
  Wait for 5 ns;
…
  Wait;
end process;
```

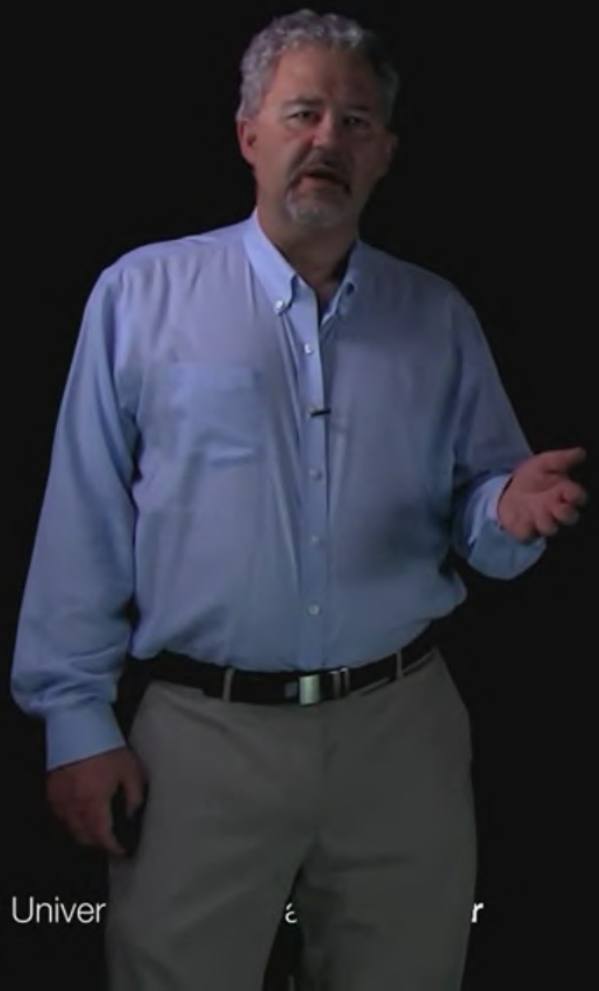
Sensitivity list for process – process executed when an event occurs on any signal in this list

Process with no sensitivity list will always be triggered initially at time 0.

Suspend for 5 ns

Suspend forever

Process Guidelines



1. If a process has a sensitivity list, then it cannot contain a 'wait' statement.

2. A process with a sensitivity list is always triggered at time 0 because all signals always have an initial event placed on them at time 0.

3. A process without a sensitivity list is always triggered at time 0 initially.

4. If a process without a sensitivity list 'falls out the bottom' then it immediately loops back to the top until it hits a wait statement.

Process Tips

An Infinite Loop

```
process
Begin

   A <= '1';

end process;
```

This process generates an infinite loop because it will 'fall out the bottom', loop back, and never encounter a wait statement.

You will get a compiler warning about this – if you execute it, the ModelSim simulator may hang.

Process Tips

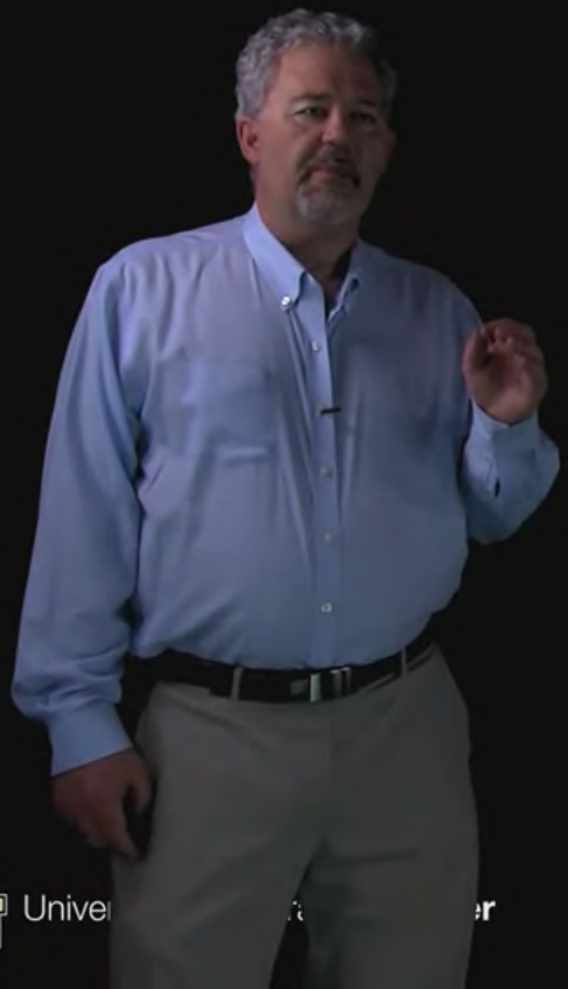
A Common Problem with sensitivity

```
process (A, S)
begin
  if (S = '1') then
    Y <= A;
  else
    Y <= B;
end process;
```

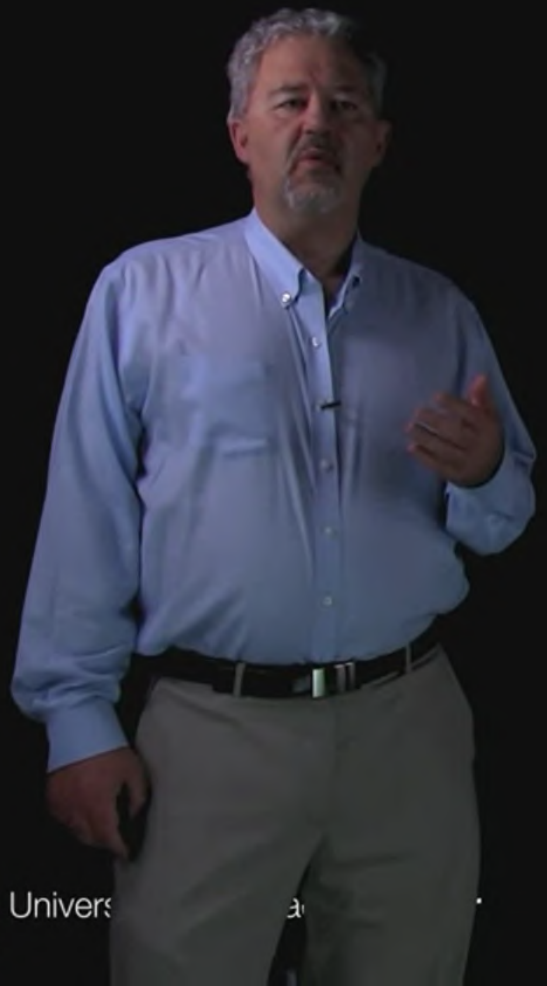
A common problem is to not include a needed signal on the sensitivity list as shown here.
This is implementing a 2:1 mux. Signal 'B' has been left off the sensitivity list by mistak – if 'S=0' and a change occurs on 'B', this change will no be propagated to the Y output! This can be hard to debug – be careful with sensitivity lists!

This is fine for synthesis – you will still get a 2:1 mux, but not for simulation – simulation won't match the circuit created by synthesis.

Designing for Simulation



- Basic granularity of concurrency is the *process*
  - Processes are executed concurrently
  - Concurrent signal assignment statements are one-line processes
- Mechanism for achieving concurrency:
  - Processes communicate with each other via signals
  - Signal assignments require delay before new value is assumed
  - Simulation time advances when all active processes complete
  - Effect is concurrent processing
    - i.e. order in which processes are actually executed by simulator does not affect behavior



Summary

In this video, you have learned:

- How the simulator works so that you can design HDL code that works in simulation
- An understanding of the techniques used to model concurrency using sequential software
- How to write your code so that it works for both synthesis and simulation
- Tips for avoiding common problems in simulation code

References

[1] Dr. Hakduran Koc (October 2015), *VHDL Identifiers and Types* [Online]. Available: http://sceweb.sce.uhcl.edu/koch/ceng4354

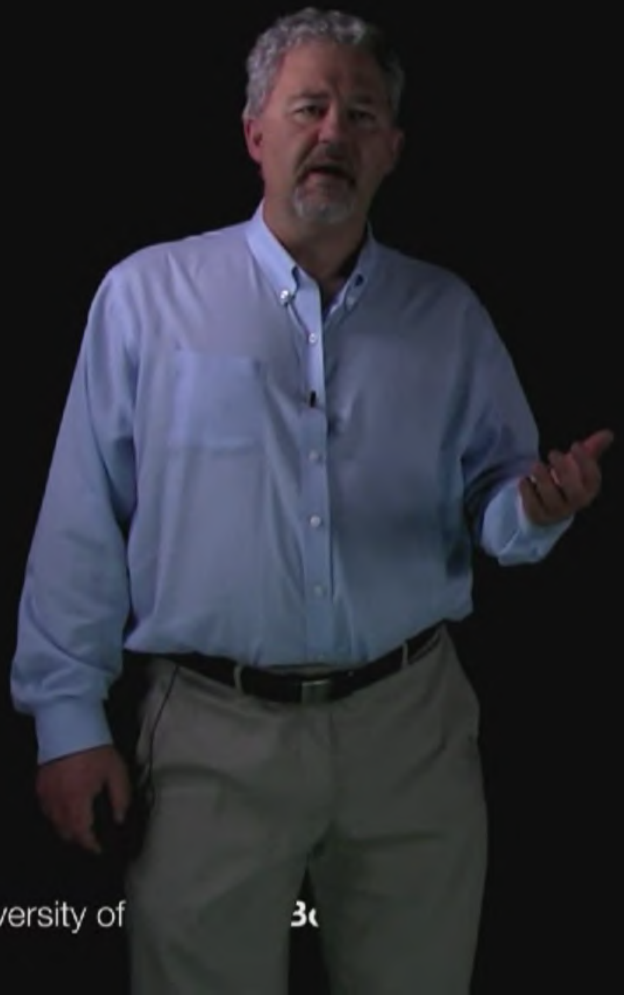
[2] MAREK ANDRZEJ PERKOWSKI (October 2015), *Timing and Simulation* [Online]. Available: http://web.cecs.pdx.edu/~mperkows/CLASS_VHDL_99/tran888/lecture004-timing-and-simulation.pdf

FPGA Design for Embedded Systems

FPGA Softcore Processors and IP Acquisition

University of Colorado **Boulder**

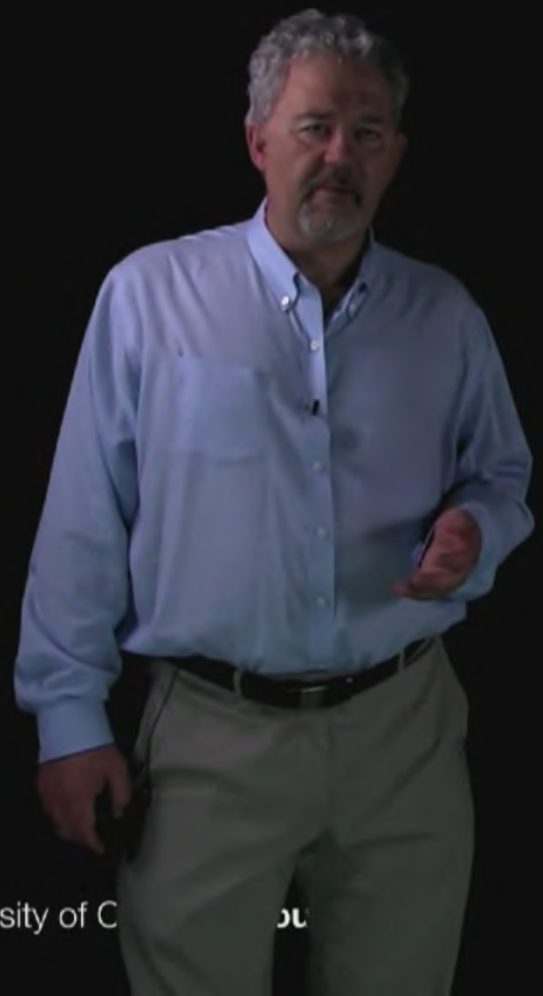
Designing for Simulation



In this video, you will learn:

- How the important concept of delta delays are used to model concurrency in hardware processes
- How VHDL simulators model variables versus signals
- How Verilog simulators model blocking versus non-blocking statements

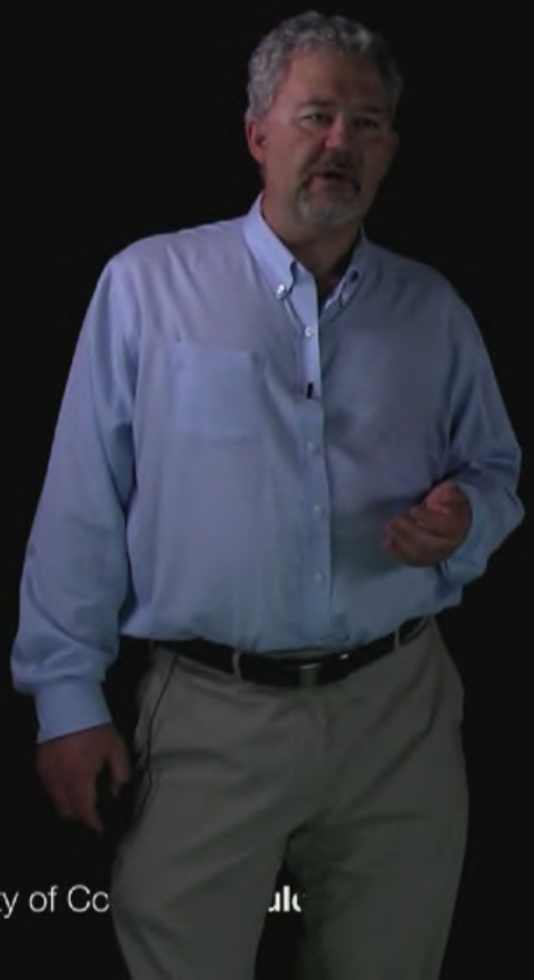
Designing for Simulation



An HDL Simulator is just a Software Program

- It models the actions of hardware, but also includes programmatic constructs and interactive elements. Based on TCL scripting, it is a complete and powerful development environment.
- It uses clever programming techniques to create the appearance of concurrency to correctly model how real hardware behaves.
- Some HDL code will simulate but not synthesize. Some HDL code will synthesize but not simulate. Your HDL code describing hardware should both simulate and be synthesizable.

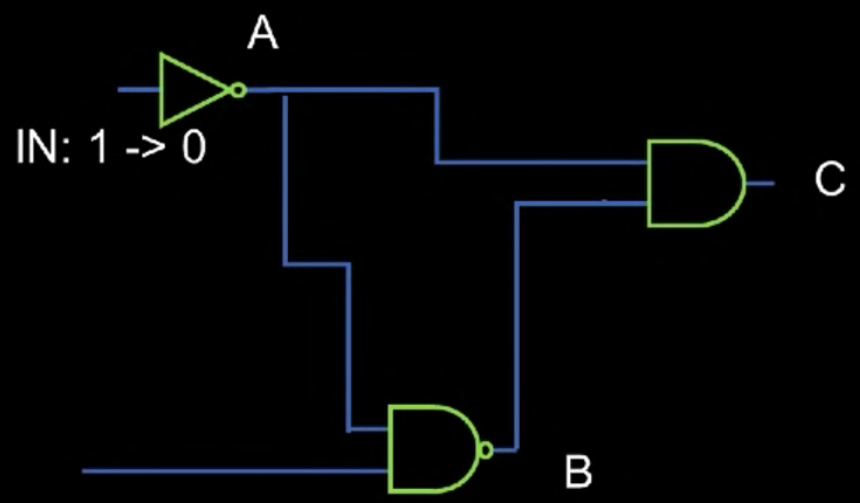
Designing for Simulation – Delta Delays



The Delta Delay is a very important concept to understand in simulation

- The Delta Delay is the default assignment propagation delay in the case that no delay is explicitly defined
- Delta is an infinitesimal time unit so that all signal assignments can result in signals assuming their values at some future time
- For example Output <= NOT Input; -- the output assumes the input in one delta cycle
- Delta Delay supports, in fact makes possible, a model of concurrent HDL process execution. As a consequence, the order in which processes are executed by the simulator does not affect the simulation output

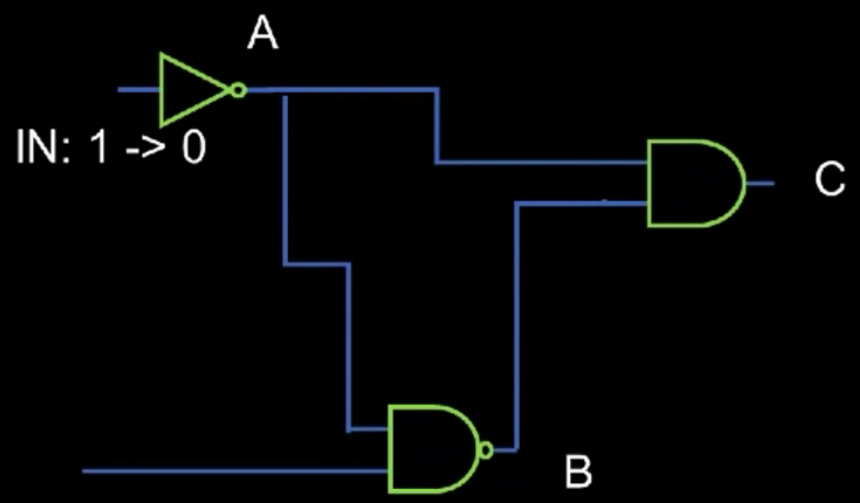
Designing for Simulation – Delta Delays



In this example circuit, how are changes through multiple levels propagated in the simulator. As In goes from 1 to 0, if the NAND gate is evaluated first, then C remains at 0 as we would expect. But if the AND gate is evaluated first, C will briefly be a 1 before then changing to zero.

Is there a way to resolve this behavior consistent with how the devices work?

Designing for Simulation – Delta Delays



Time	Delta Delays	Event
0 ns	1	IN: 1 -> 0
		Evaluate inverter
	2	A: 0 -> 1
		Evaluate NAND, AND
	3	B: 1->0
		C: 0 -> 1
		Evaluate AND
	4	C: 1 ->0
1 ns		

Designing for Simulation – Delta Delays

Delta Delays also help model the different between signals and variables in VHDL In this example, the variable changes immediately, but the signal does not until after 1 delta delay.

```
Architecture var_sig of test is
  signal a, b, c, out4: Bit;
Begin
  process(a, b, c)
  variable out3: Bit;
  begin
    out3 := a NAND b;
    out4 <= out3 XOR c;
  end process
End var_sig
```

Time	a	b	c	out3	out4
0	0	1	1	1	0
1	1	1	1	0	0
1 + delta	1	1	1	0	1

Designing for Simulation – Verilog

Verilog also has the concept of delta delays. In Verilog, delays are modeled using the delay control operator #n, which has the simulation effect of suspending simulation for n time units

```
assign #10 a = x & y;  // delays by 10 time units

always
  begin
    clk = 0;
  #50 clk = 1;
  #50;              // null statement delays 50 time units
  #10 a = x + y; // blocking delay
  a = #10 x + y; // interassignment delay
```

Designing for Simulation – Verilog

Unlike a blocking delay, in a non-blocking assignment the right hand side is evaluated before the delay. These type of statements good for simulation only code, but will not synthesize with delays.

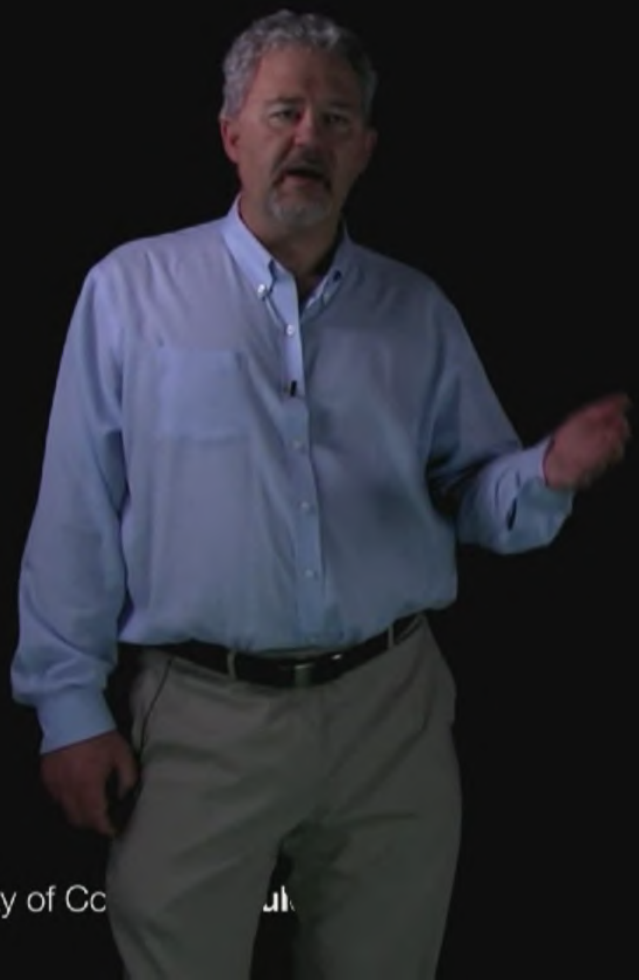
```
// with blocking assignments:
a = #5 b + c;  // b + c evaluated;  change scheduled for 5 time
               // units later;
d = a;         // delayed until 5 time units elapse (blocked)

// with nonblocking assignments:
a <= #5 b + c;  // b + c evaluated;  change in a scheduled
d = a;         // executes immediately;  d gets old value of a!
```

Designing for Simulation

See also…

Verilog Nonblocking Assignments With Delays, Myths & Mysteries by Cliff Cummings
http://www.suburstdesign.com/papers/CummingsSNUG2002Boston_NBAwithDelays.pdf
Nonblocking Assignments in Verilog Synthesis, Coding Styles That Kill! by Cliff Cummings
http://www.sunburstdesign.com/papers/CummingsSNUG2000SJ_NBA.pdf

Summary



In this video, you have learned:

- How the important concept of delta delays are used to model concurrency in hardware processes
- How VHDL simulators model variables versus signals
- How Verilog simulators model blocking versus non-blocking statements

References

[1] MAREK ANDRZEJ PERKOWSKI (October 2015), *Timing and Simulation* [Online]. Available: http://web.cecs.pdx.edu/~mperkows/CLASS_VHDL_99/tran888/lecture004-timing-and-simulation.pdf

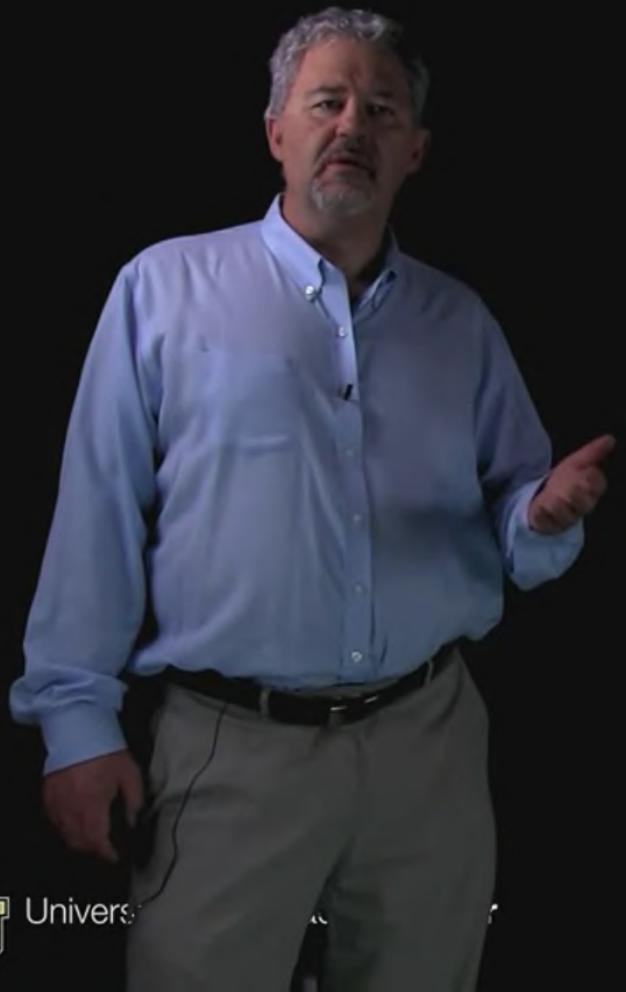
[2] John Nestor, (October 2015), *Verilog Delays* [Online]. Available: workbench.lafayette.edu/~nestorj/cadapplets/ece426/notes/03_426_S03.ppt

FPGA Design for Embedded Systems

FPGA Softcore Processors and IP Acquisition

University of Colorado **Boulder**

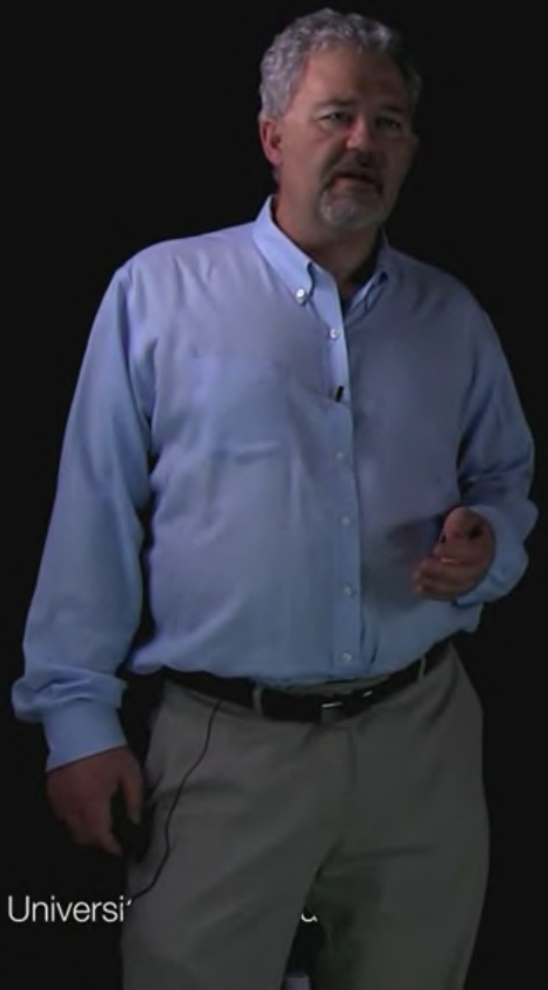
Logic Analysis with SignalTap II



In this video, you will learn:

- How to use the SignalTap II internal logic analyzer (ILA) for FPGA hardware debugging
- How to analyze signals internal to the FPGA without external probes that may cause signal integrity issues
- To understand the limitations of internal logic analyzers, which require FPGA fabric resources

Debugging Tools



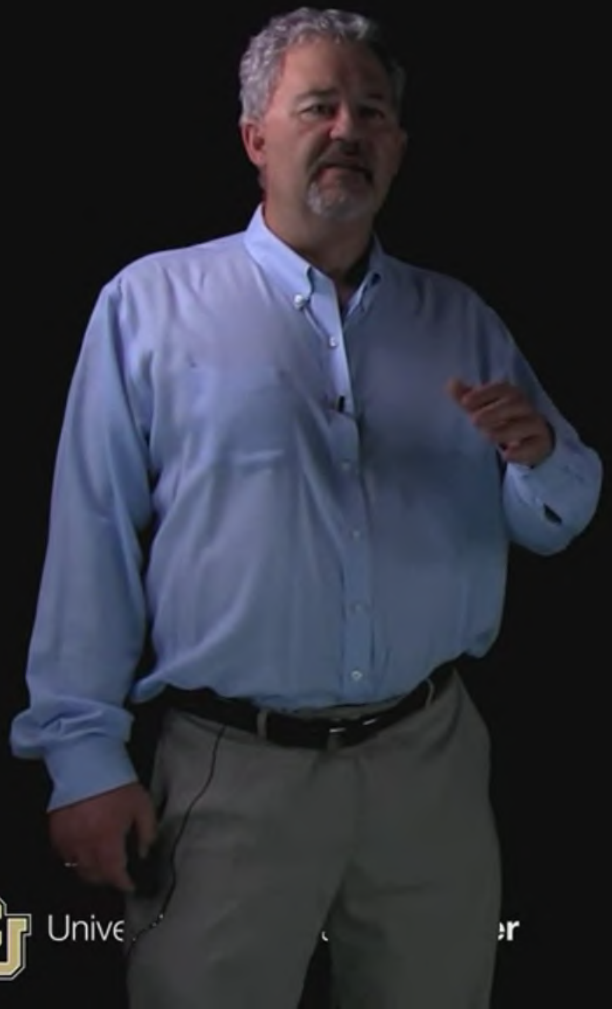
FPGAs and Programmable SoC devices are beginning to receive more and more support, both from the manufacturers themselves and from tool vendors.

Manufacturers tools not only include

- the HDL compilers and fitters,
- Programming tools
- IDEs for software development and
- internal logic analyzers for hardware debugging.
- The offering of IP cores has vastly increased in just the past few years.

[1]

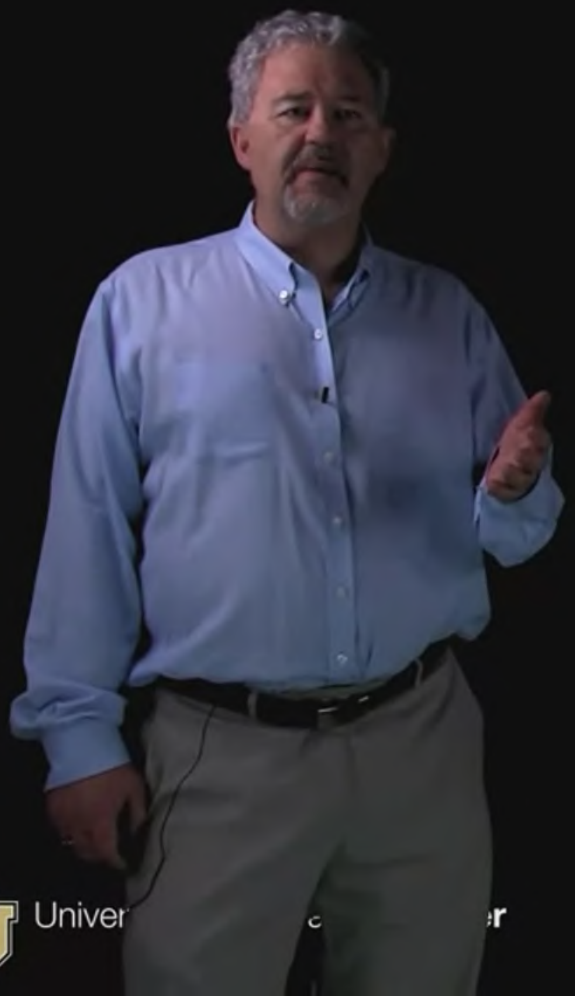
Hardware Debugging for FPGAs



Problem: As the density of FPGA devices increases, so does the impracticality of attaching test equipment probes to these devices under test.

Answer: Altera provides the SignalTap® II Logic Analyzer to help with the process of design debugging. This logic analyzer is a solution that allows you to examine the behavior of internal signals, without using extra I/O pins, while the design is running at full speed on an FPGA device.

Introducing SignalTap II

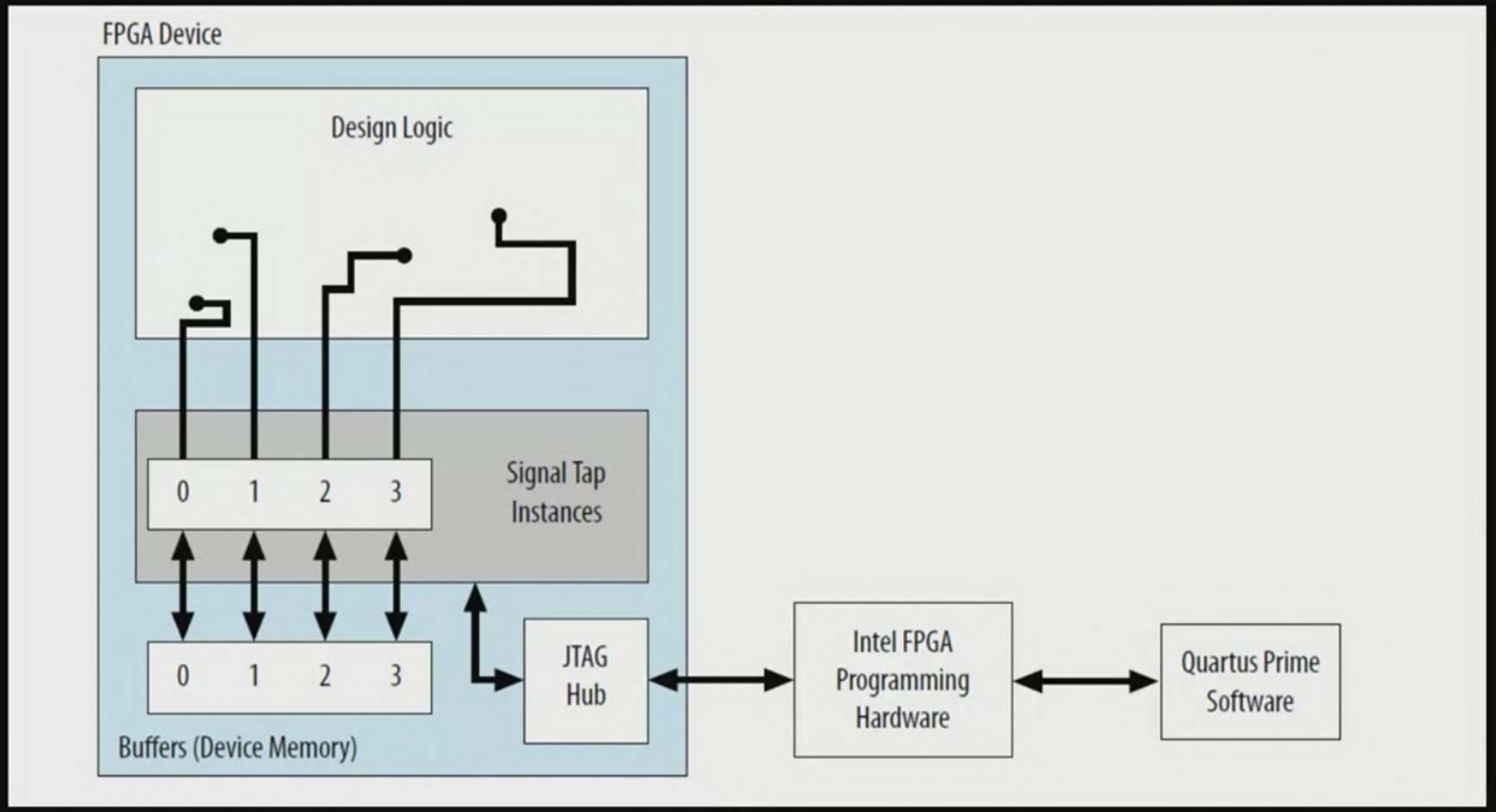


The SignalTap II Logic Analyzer is scalable, easy to use, and is available as a stand-alone package or included with the Quartus software tool .

This logic analyzer helps debug an FPGA design by probing the state of the internal signals in the design without the use of external equipment. Defining custom trigger-condition logic provides greater accuracy and improves the ability to isolate problems. The SignalTap II Logic Analyzer does not require external probes or changes to the design files to capture the state of the internal nodes or I/O pins in the design. All captured signal data is conveniently stored in device memory until you are ready to read and analyze the data.

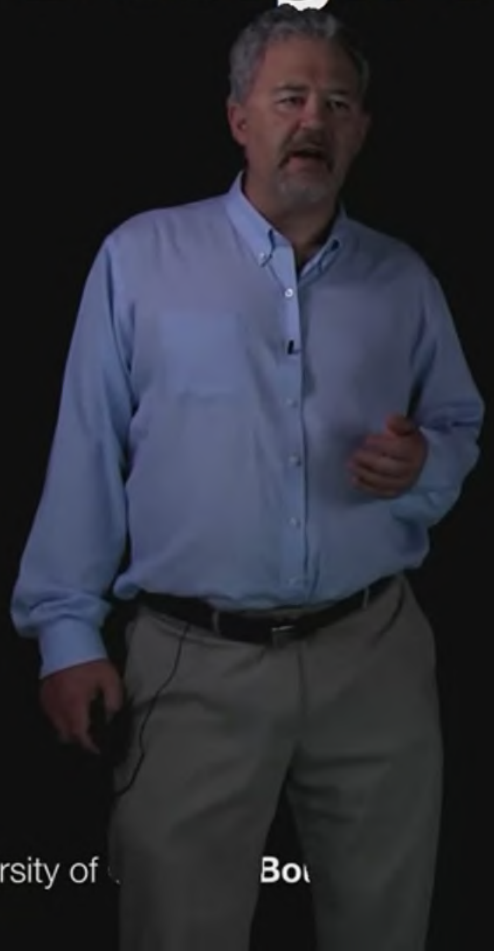
[1]

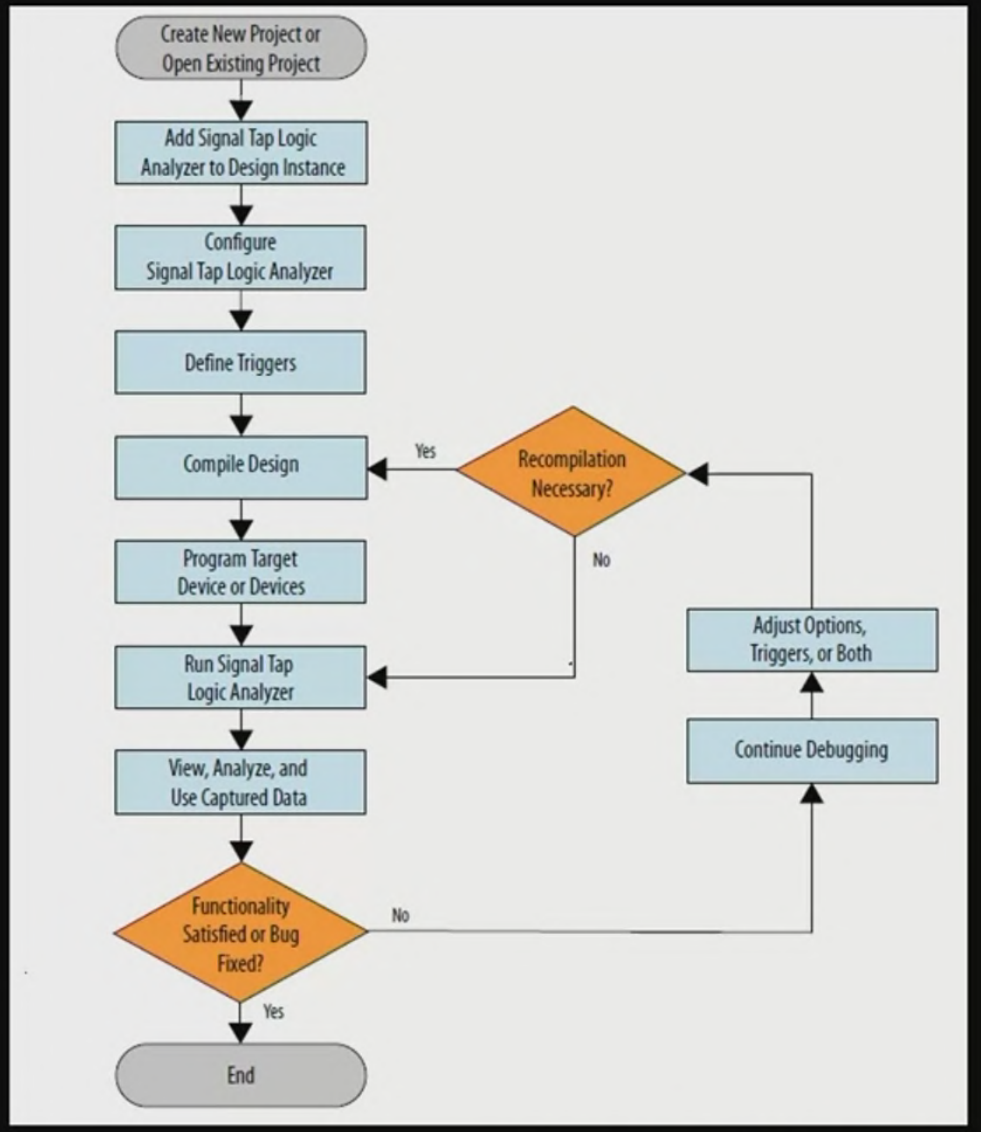
Introducing SignalTap II



[1]

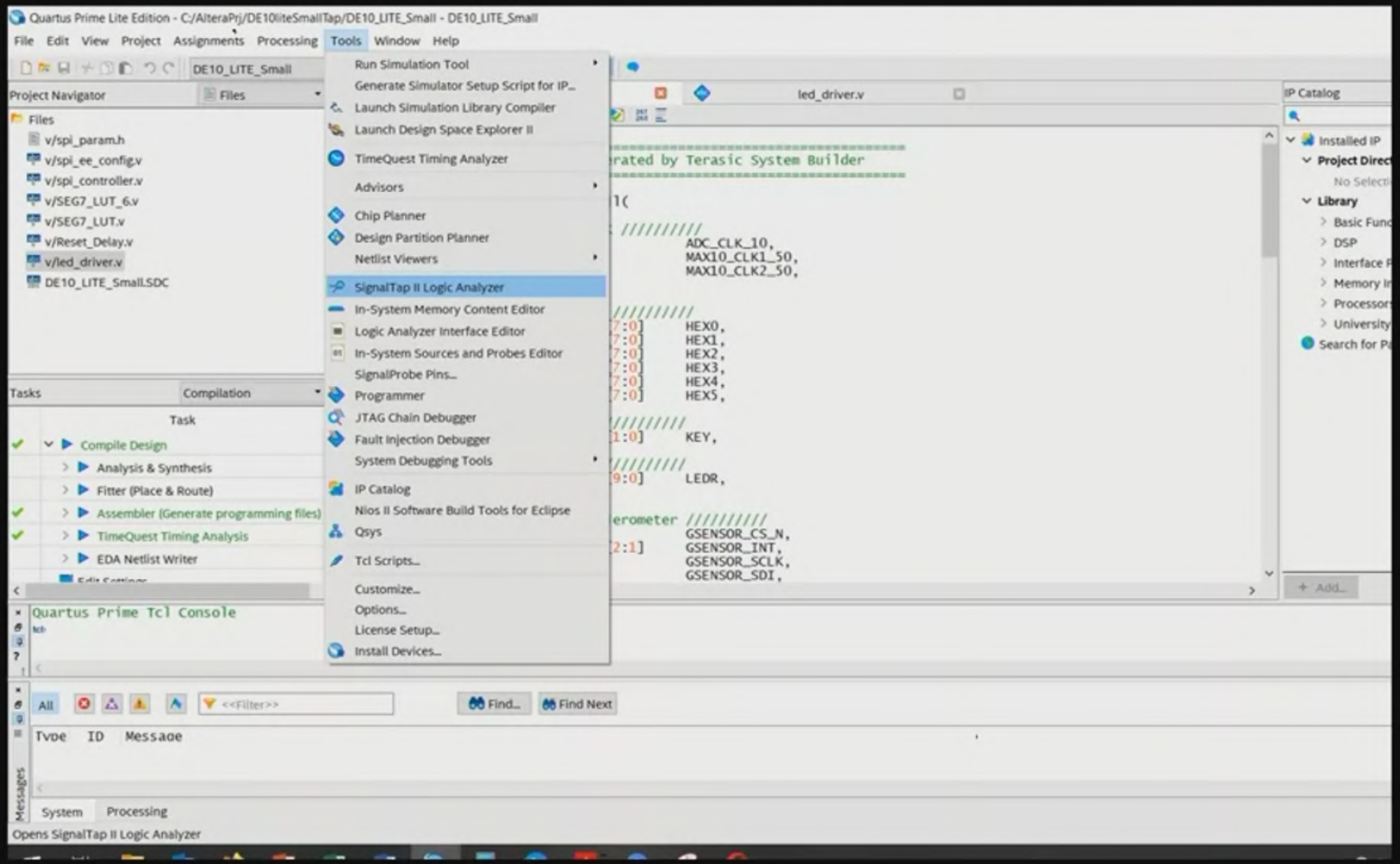
SignalTap II Design Flow





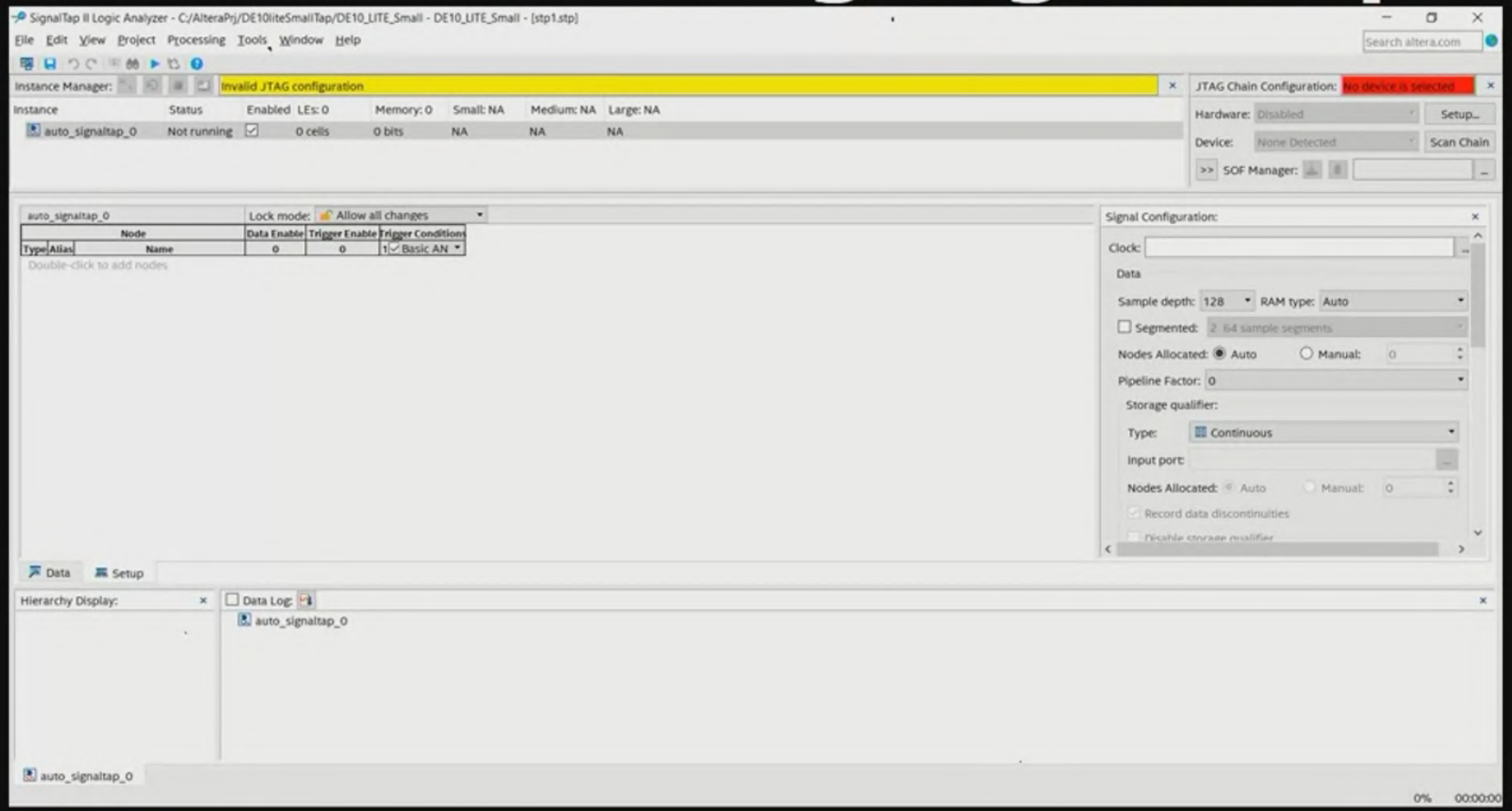
[1]

Starting SignalTap II



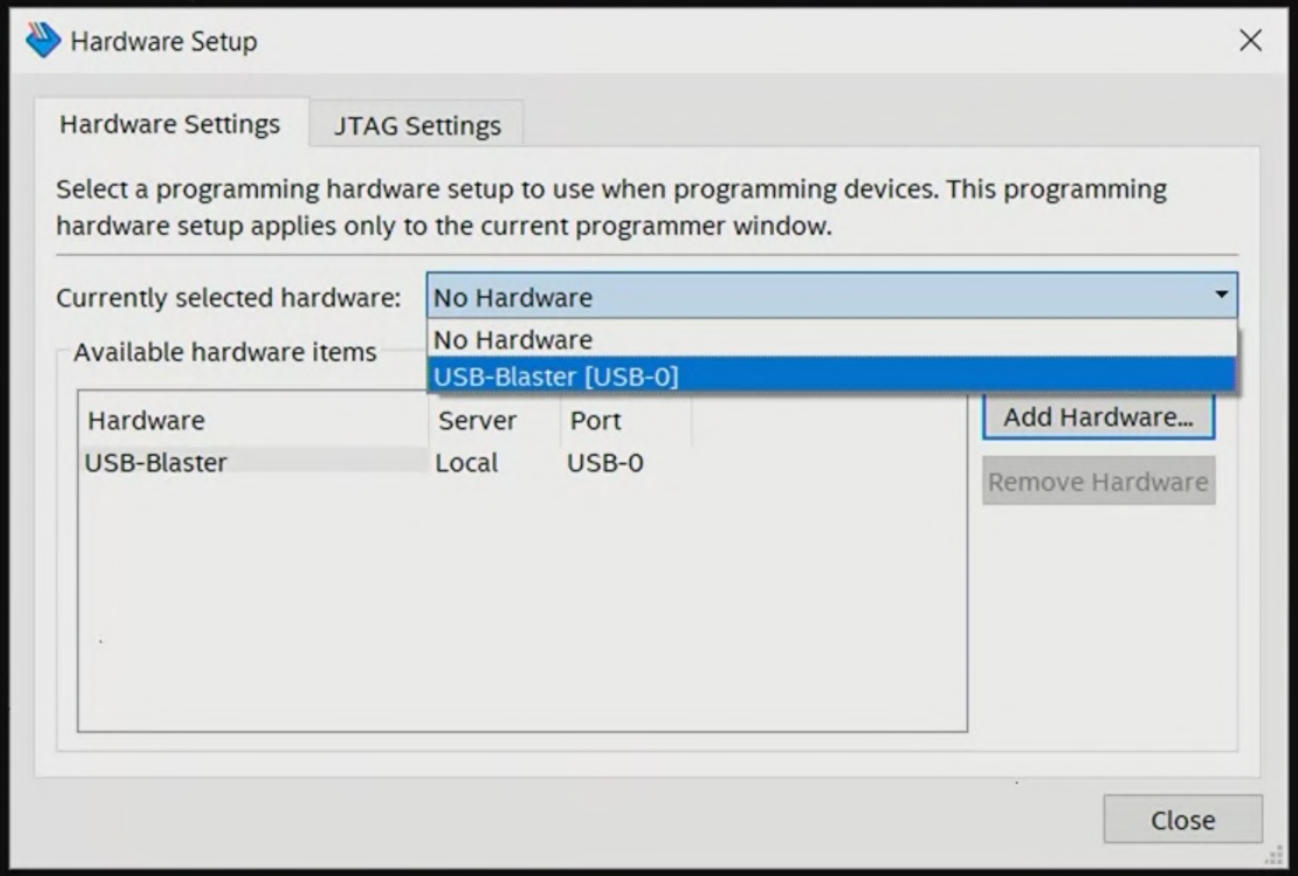
- Start Quartus Prime with an Existing Project
- Select Tools -> SignalTap II Logic Analyzer

Starting SignalTap II



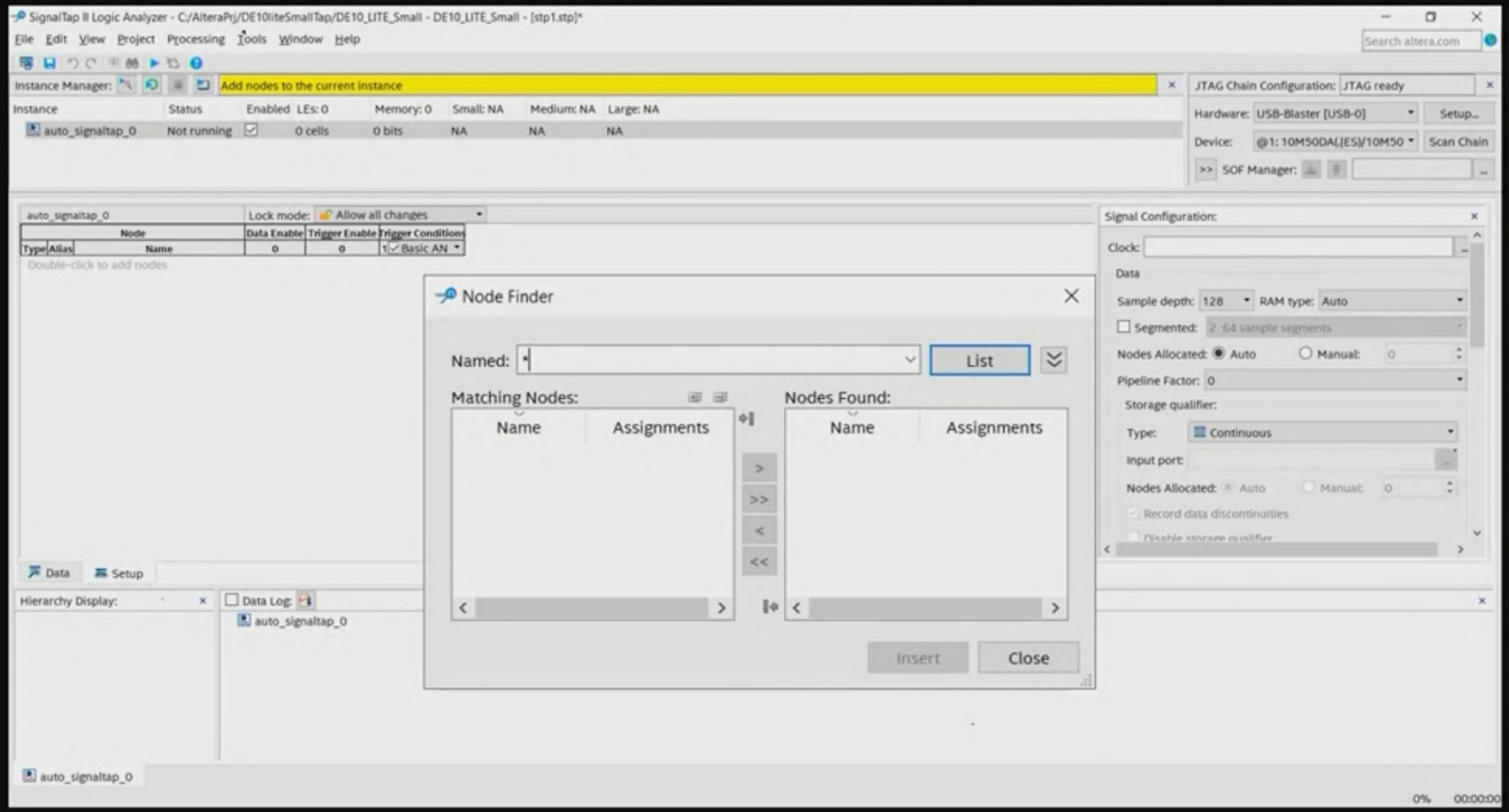
- The SignalTap II analyzer starts with a pre-loaded instance.
- Attach a development board with existing design

Starting SignalTap II



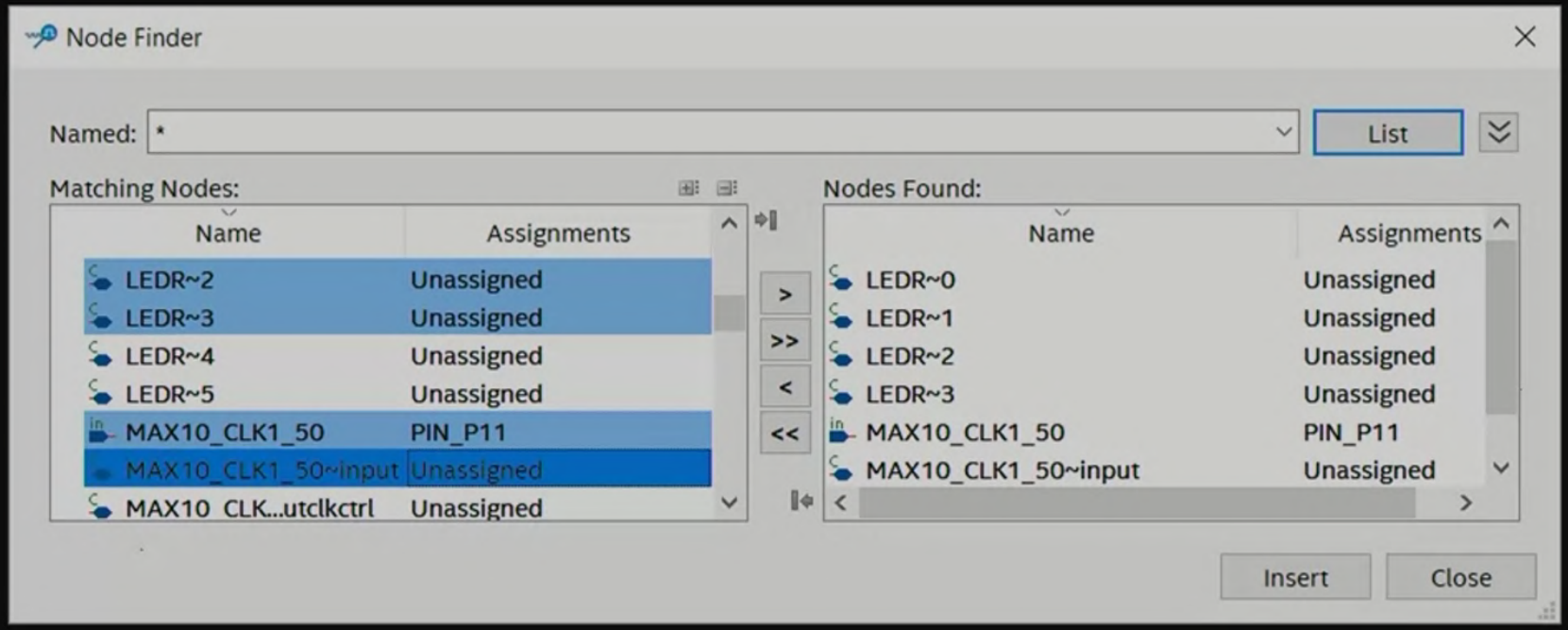
- Under JTAG configuration in the upper right, choose setup, USB Blaster.

Configuring SignalTap II



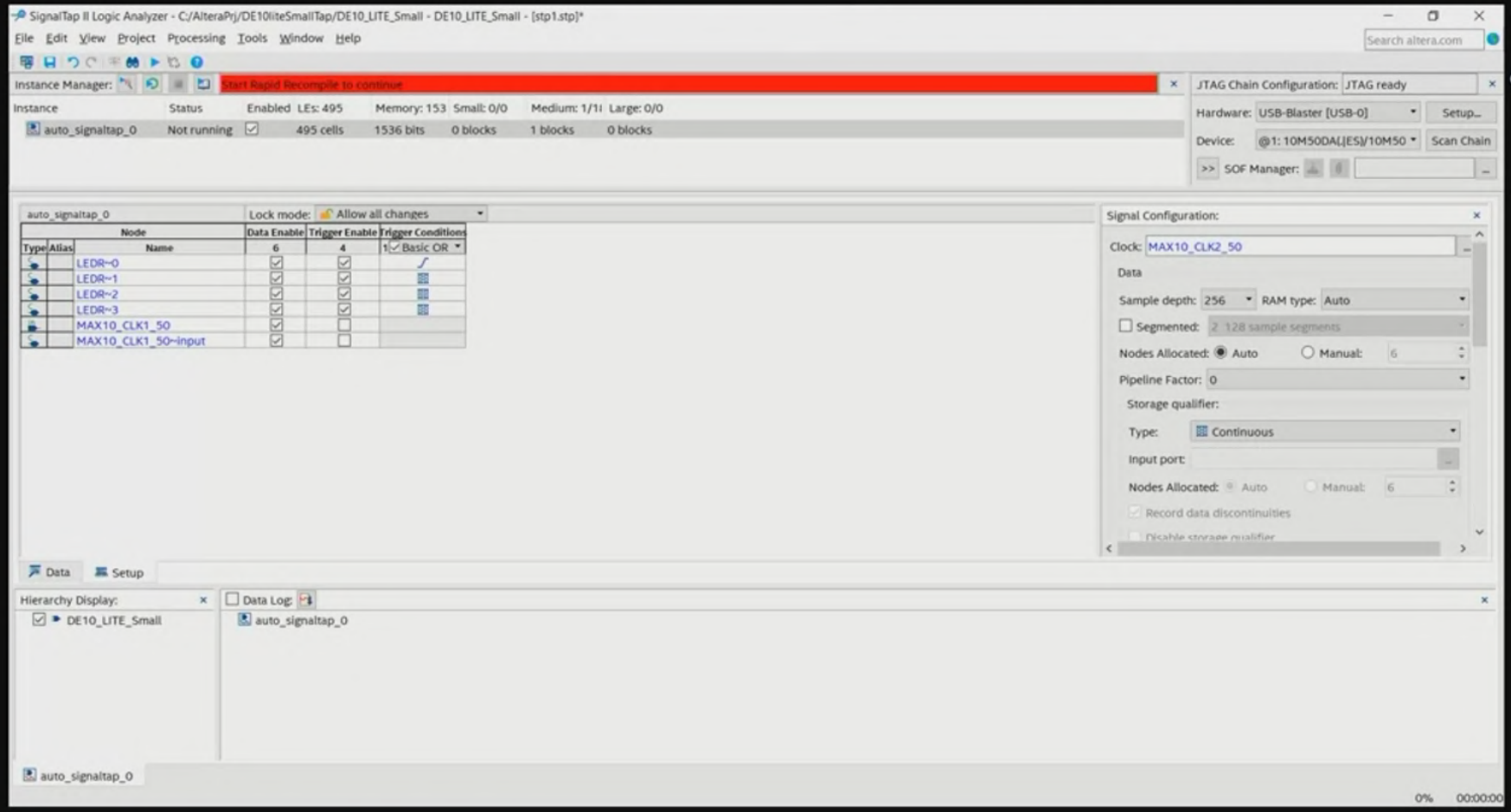
- Now with JTAG ready, double click in the empty space to add nodes to view in the logic analyzer.
- Choose List to see the list of signals

Configuring SignalTap II



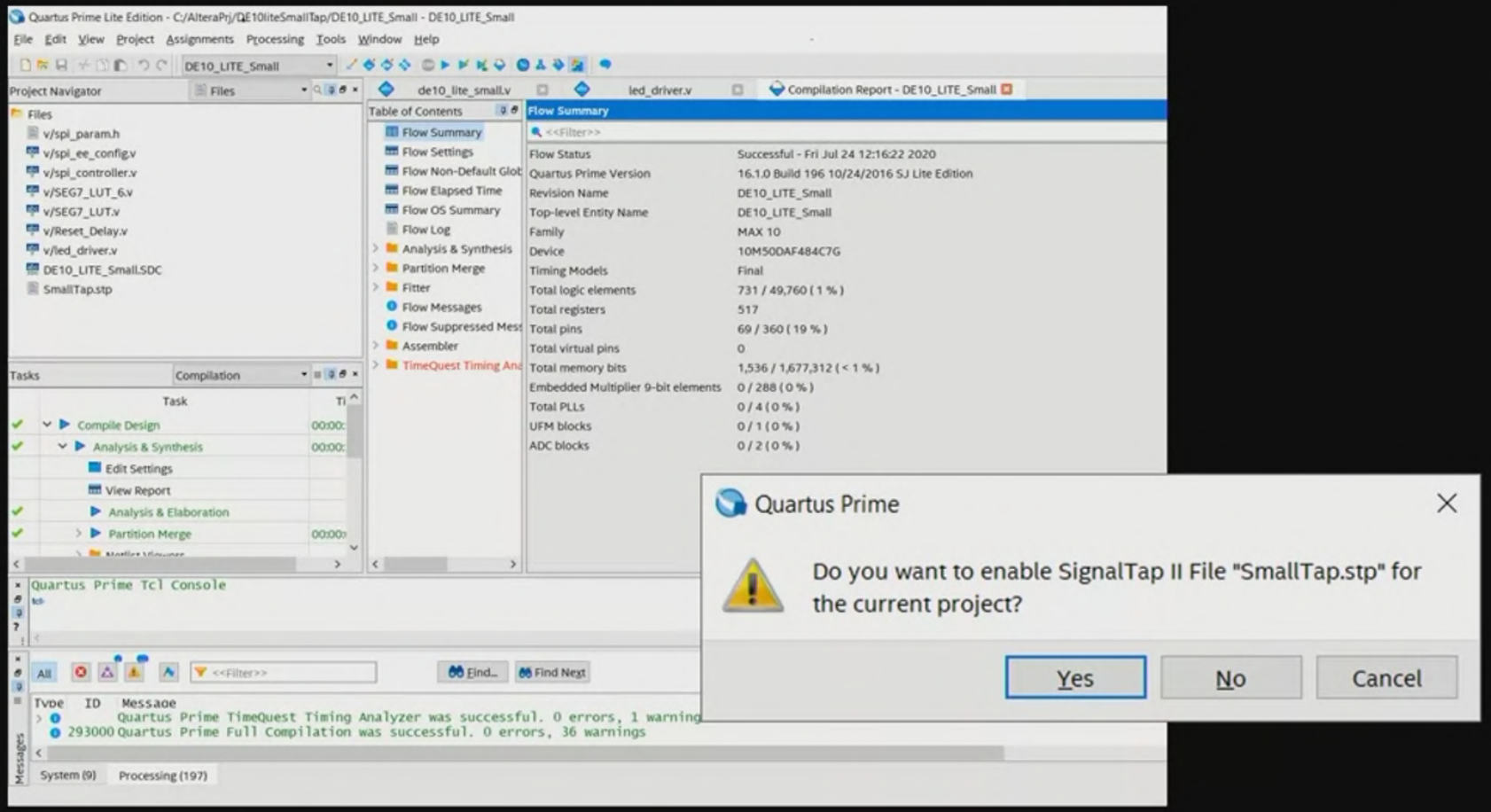
- Add signals to be analyzed

Configuring SignalTap II



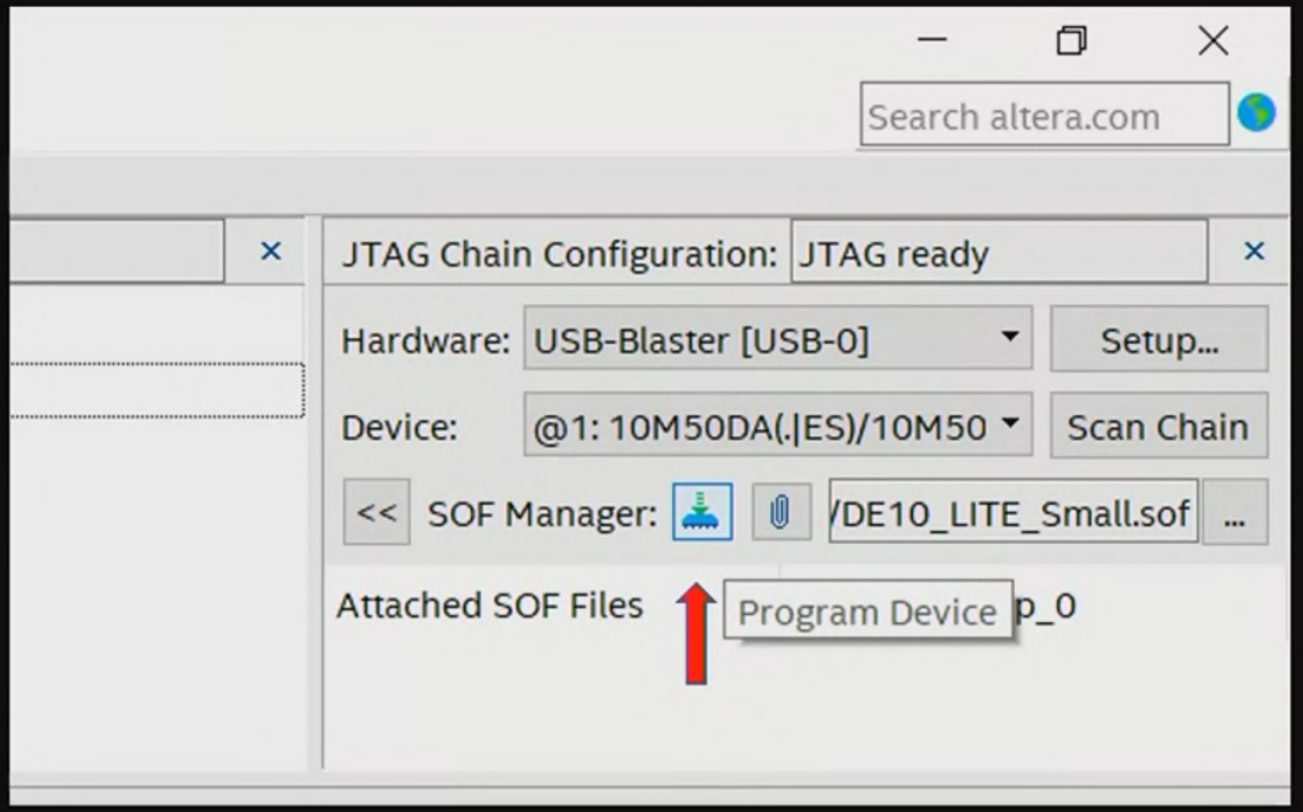
- Select which signals to collect trace data, and which to use for triggering
- Next to “Clock” click “ … ”
- Click “List” and choose clock source

Recompiling SignalTap II



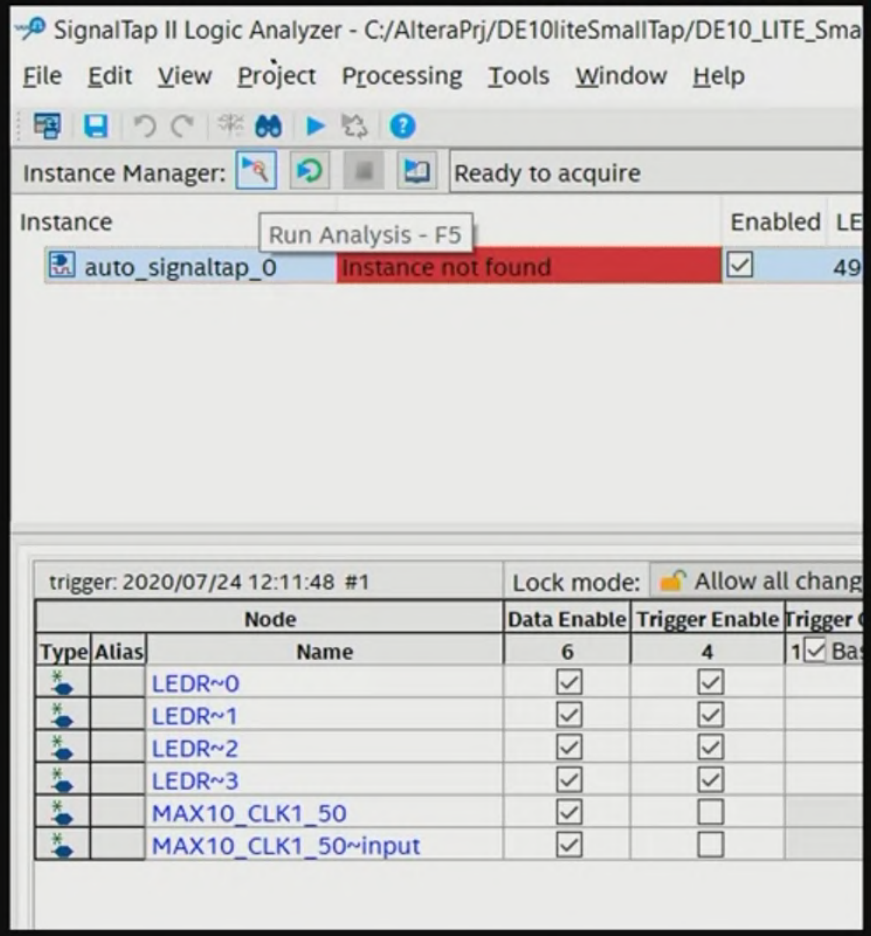
- Select File -> save and name the stp file.
- Start Rapid Recompile to recompile the design to load the logic analyzer into the FPGA

Loading SignalTap II



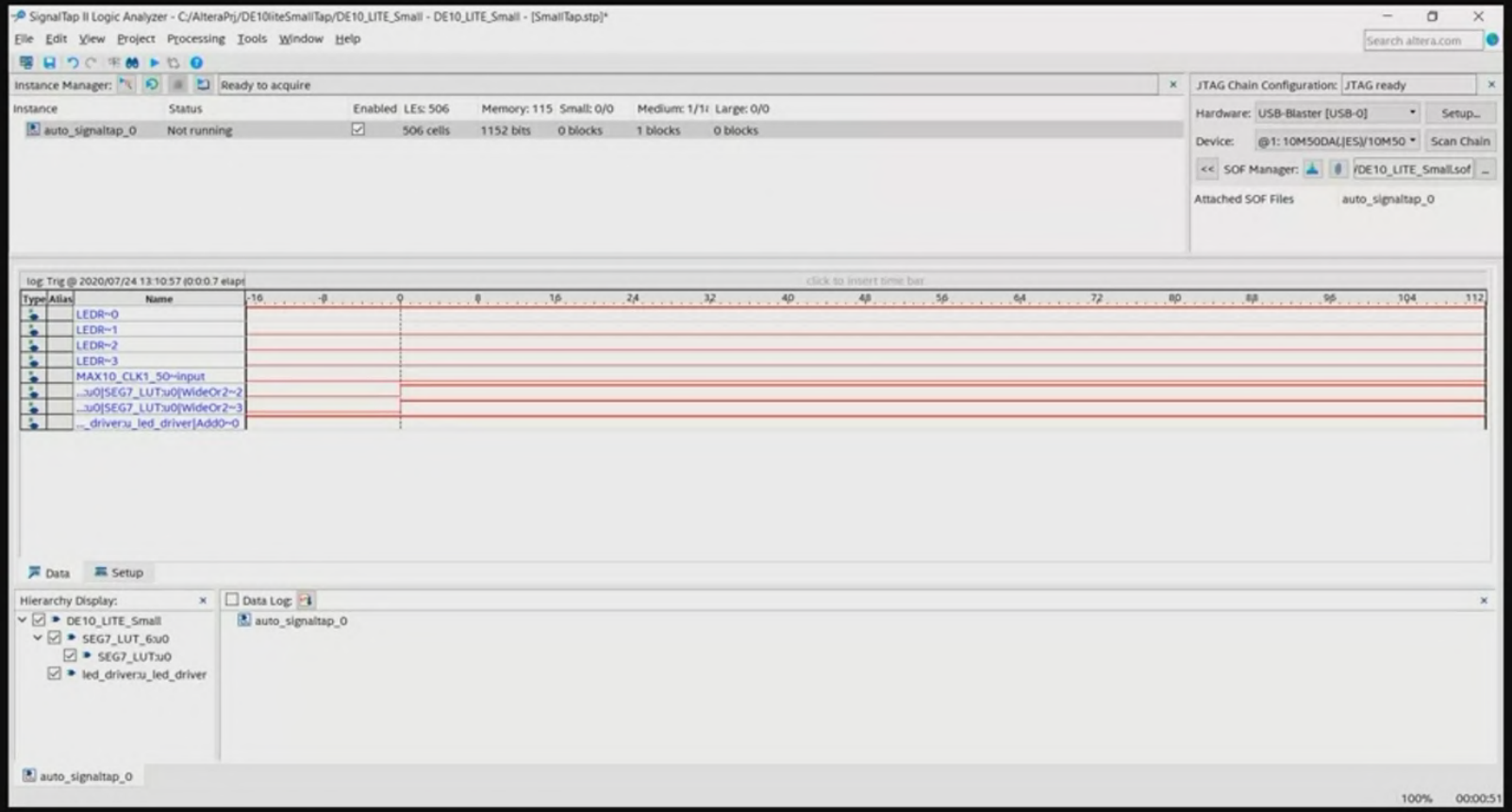
- Program the Device from the SignalTap interface

Running the Analyzer

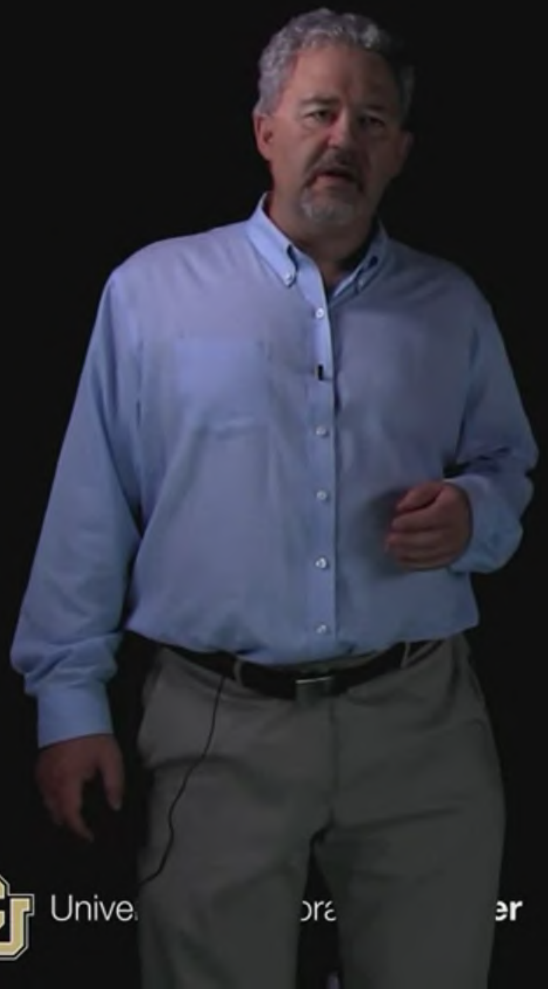


- Click the Run Analysis button to gather data.

Running the Analyzer



- The acquired data is displayed as a time waveform
- Note the number of LE's and memory bits used by the analyzer



Summary

In this video, you have learned:

- How to use the SignalTap II internal logic analyzer (ILA) for FPGA hardware debugging
- How to analyze signals internal to the FPGA without external probes that may cause signal integrity issues
- To understand the limitations of internal logic analyzers, which require FPGA fabric resources

[2]