FPGA Development for Radar, Radio-Astronomy and Communications

THE RADAR MASTERS COURSE

Dept. Electrical Engineering, University of Cape Town
Private Bag, Rondebosch, 7701, South Africa

http://www.rrsg.uct.ac.za

Presented by John-Philip Taylor

Convened by Prof Daniel O'Hagan

Tutored by Stephen Paine and Randy Cheng

Day 2 – 18 July 2017

Outline

Outline

Altera Library

Verilog Processes

Finite State Machines

Timing Constraints

Injection Testing

Simulation

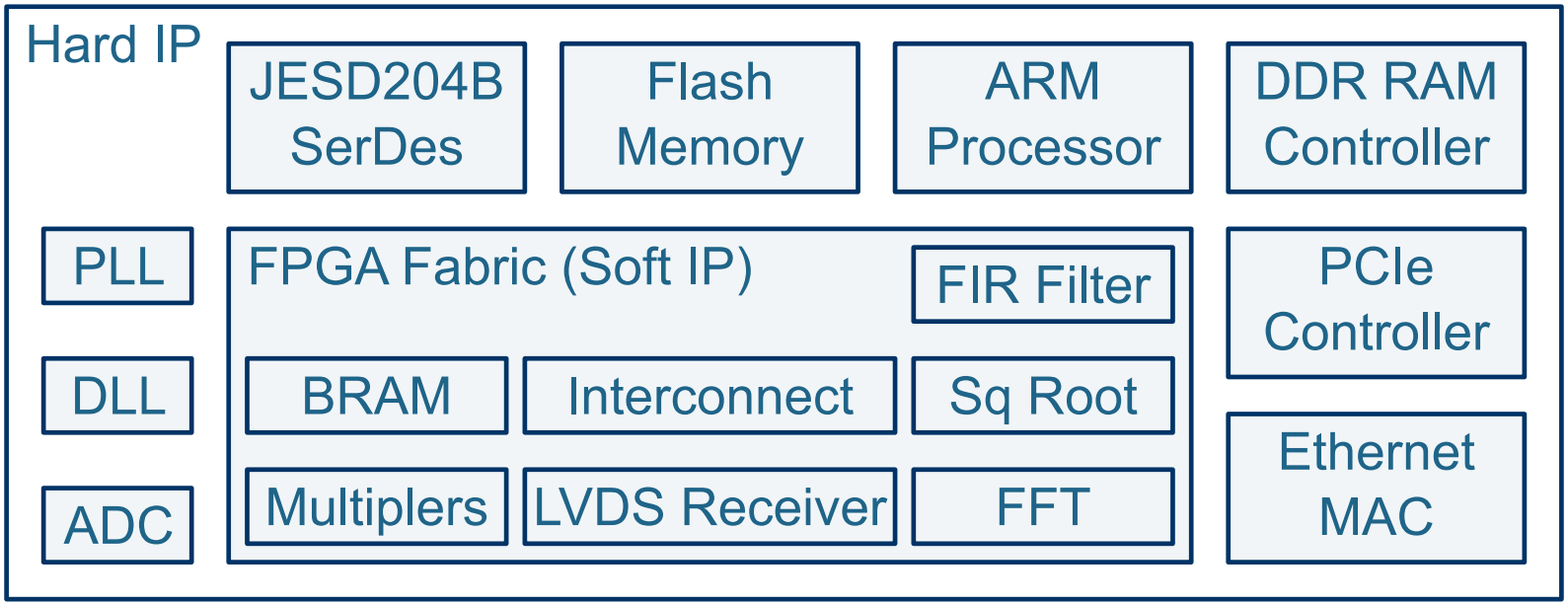
Practical

Appendix – High Resolution PWM

THE RADAR MASTERS COURSE

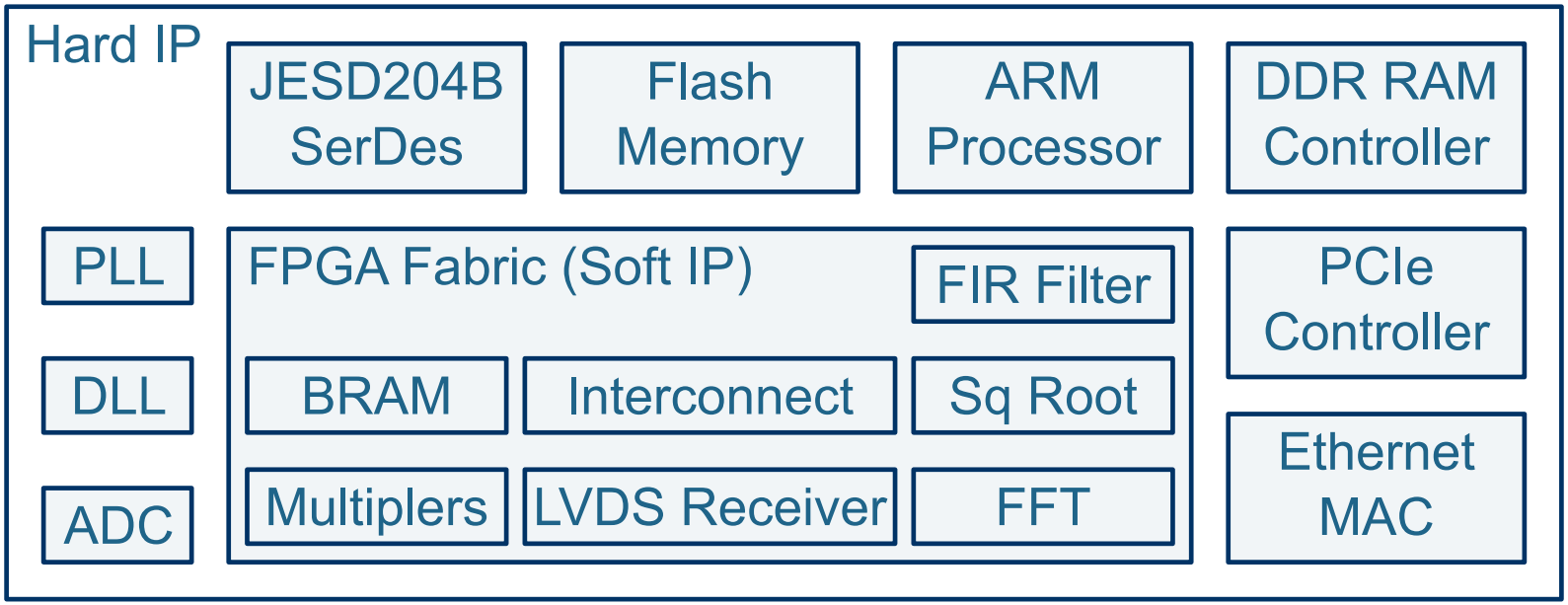
Altera Library

- A combination of soft and hard IP
- Collectively known as “Megafunctions”



Altera Library

- A combination of soft and hard IP
- Collectively known as “Megafunctions”



Wizard

- Generally easier to use the IP Catalogue (or Qsys) wizard to generate wrapper modules
- Or one can instantiate the built-in modules directly
- The practical today uses Qsys to set up the external SDRAM controller and interface

Wizard

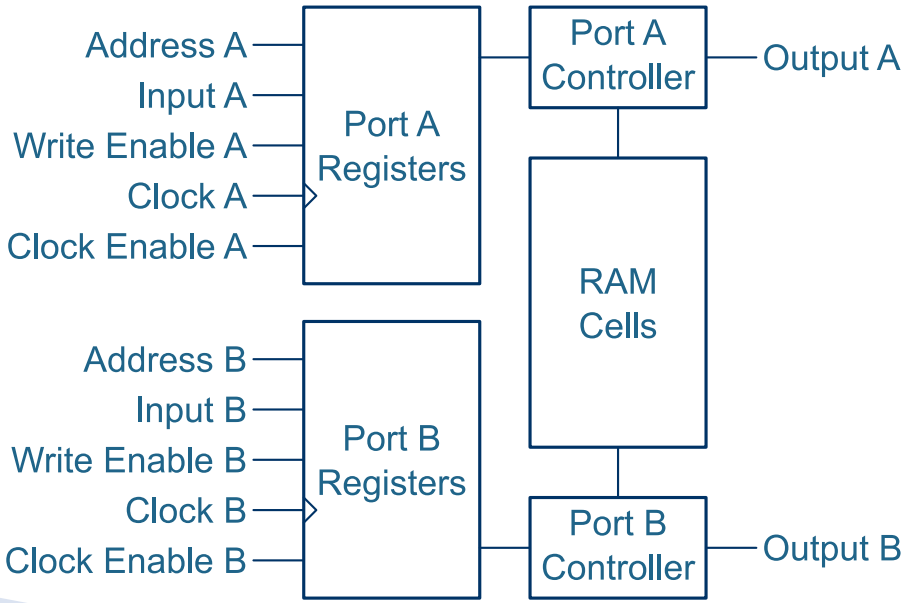
- Generally easier to use the IP Catalogue (or Qsys) wizard to generate wrapper modules
- Or one can instantiate the built-in modules directly
- The practical today uses Qsys to set up the external SDRAM controller and interface

Wizard

- Generally easier to use the IP Catalogue (or Qsys) wizard to generate wrapper modules
- Or one can instantiate the built-in modules directly
- The practical today uses Qsys to set up the external SDRAM controller and interface

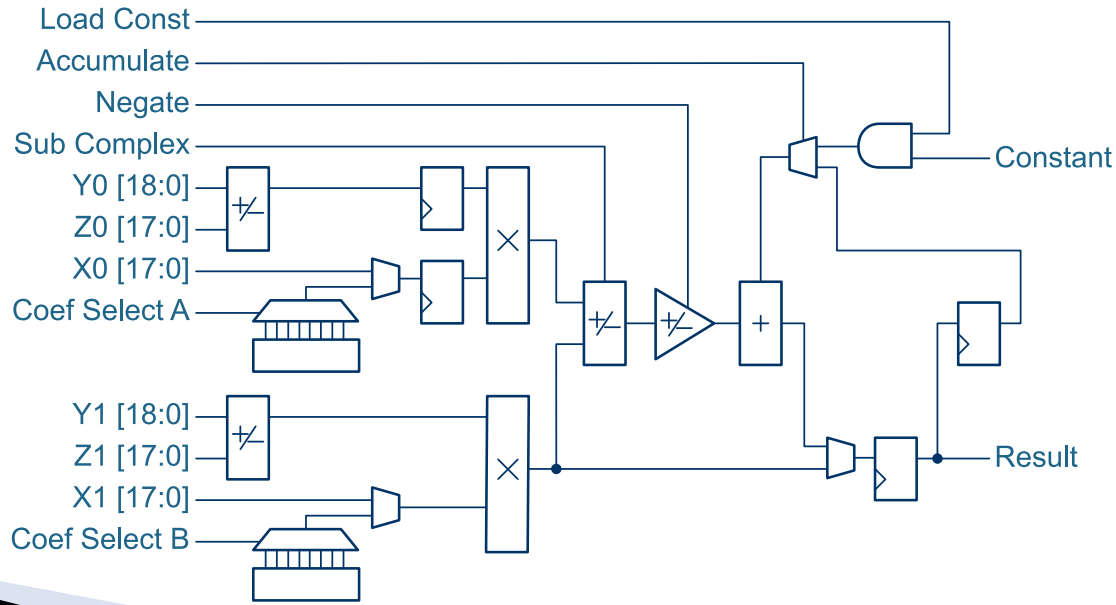
Embedded Components

- RAM / ROM
- DSP blocks
- PLL / DLL blocks
- Processors / SoC (ARM) with bus infrastructure
- Interfaces (DDR Memory / PCIe / SerDes (JESD204) / etc.)



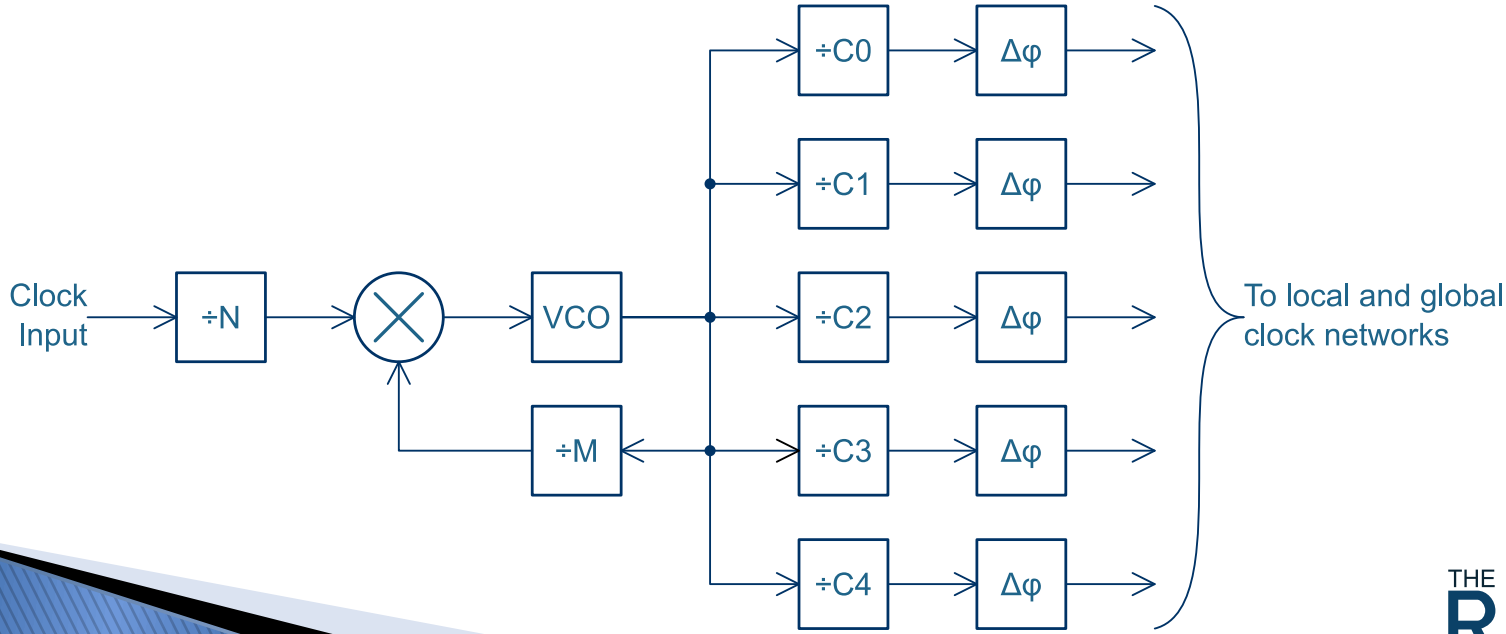
Embedded Components

- RAM / ROM
- DSP blocks
- PLL / DLL blocks
- Processors / SoC (ARM) with bus infrastructure
- Interfaces (DDR Memory / PCIe / SerDes (JESD204) / etc.)



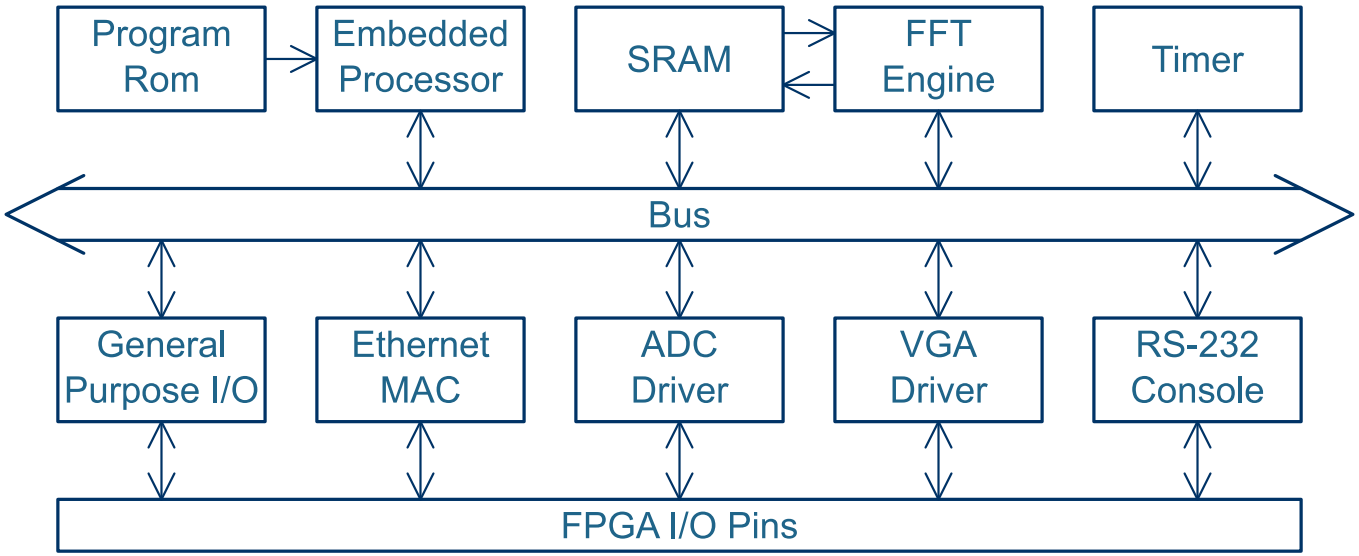
Embedded Components

- RAM / ROM
- DSP blocks
- PLL / DLL blocks
- Processors / SoC (ARM) with bus infrastructure
- Interfaces (DDR Memory / PCIe / SerDes (JESD204) / etc.)



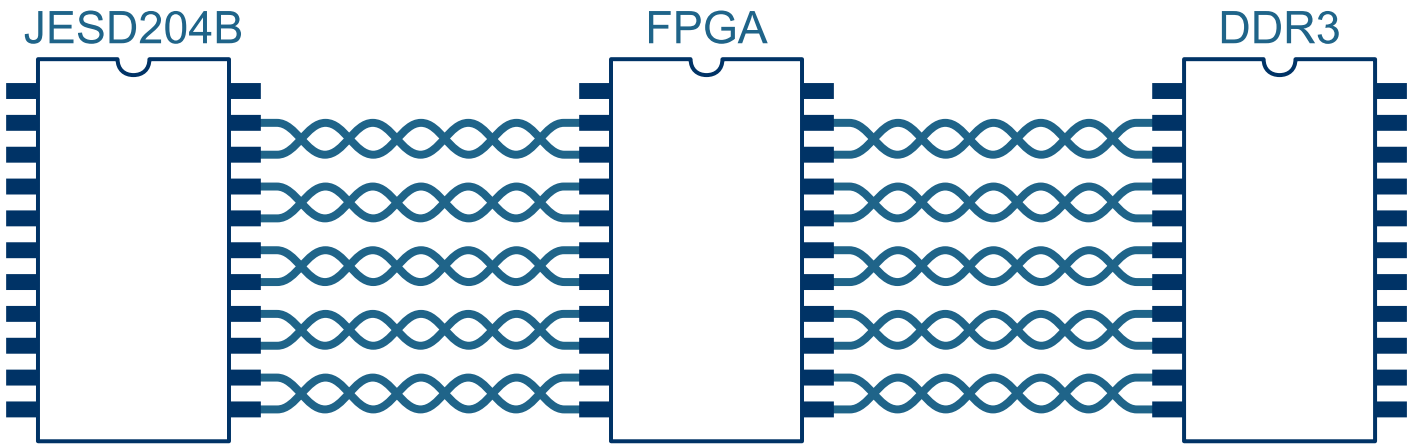
Embedded Components

- RAM / ROM
- DSP blocks
- PLL / DLL blocks
- Processors / SoC (ARM) with bus infrastructure
- Interfaces (DDR Memory / PCIe / SerDes (JESD204) / etc.)



Embedded Components

- RAM / ROM
- DSP blocks
- PLL / DLL blocks
- Processors / SoC (ARM) with bus infrastructure
- Interfaces (DDR Memory / PCIe / SerDes (JESD204) / etc.)



Outline

Altera Library

Verilog Processes

Finite State Machines

Timing Constraints

Injection Testing

Simulation

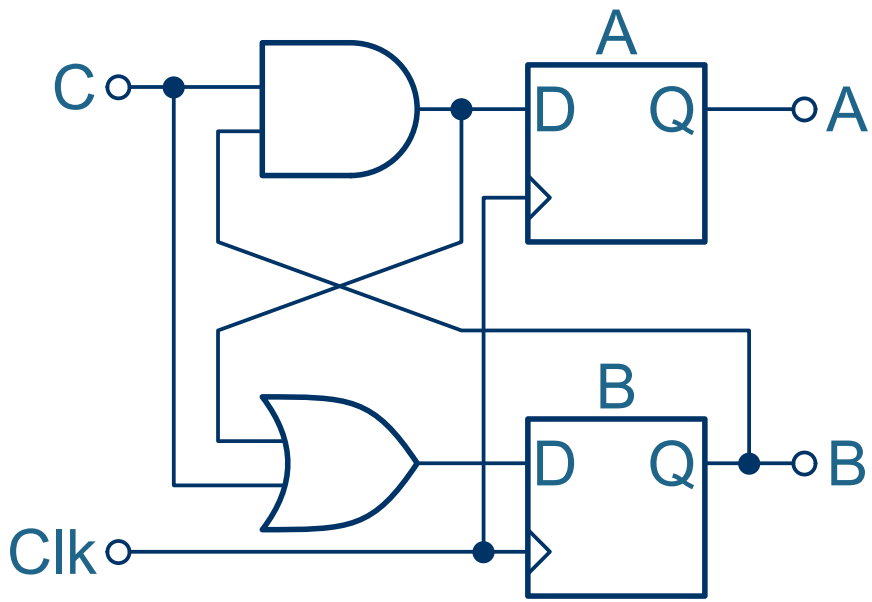
Practical

Appendix – High Resolution PWM

THE RADAR MASTERS COURSE

Blocking Statements

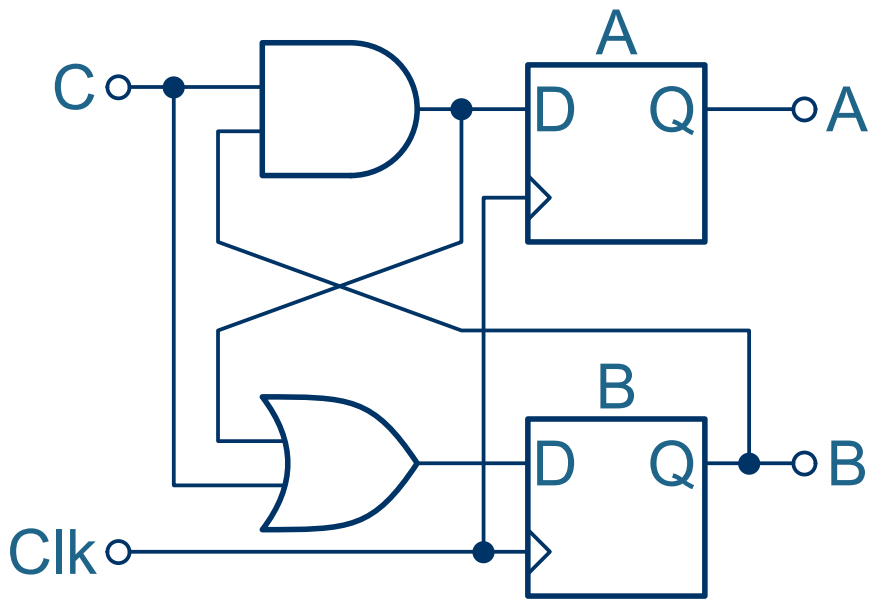

```
reg A, B;

always @(posedge Clk) begin
 A = C & B;
 B = A | C;
end
```



- Statements are evaluated in order, like a computer program
- Often results in unintentionally long combinational chains
- Note that all registers still change state on the clock edge

Blocking Statements

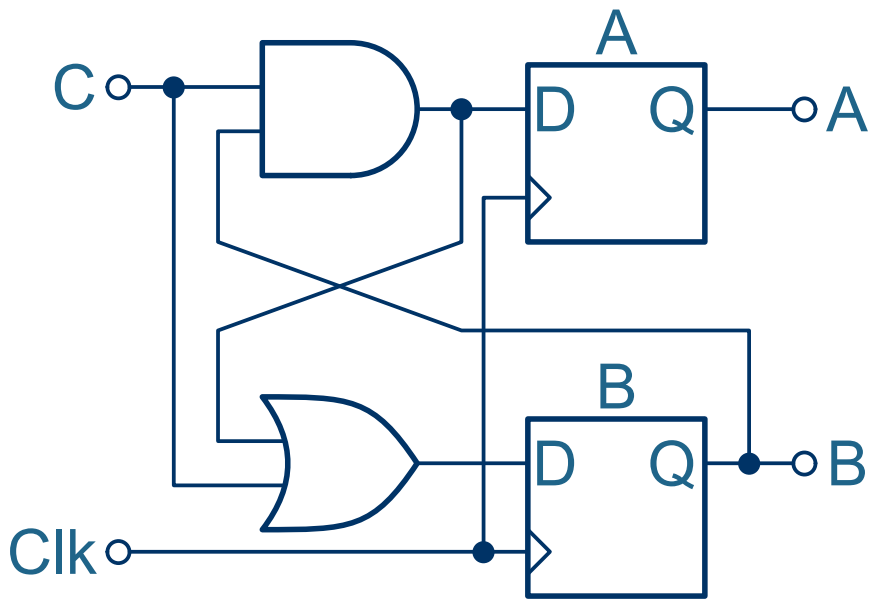

```
reg A, B;

always @(posedge Clk) begin
 A = C & B;
 B = A | C;
end
```



- Statements are evaluated in order, like a computer program
- Often results in unintentionally long combinational chains
- Note that all registers still change state on the clock edge

Blocking Statements

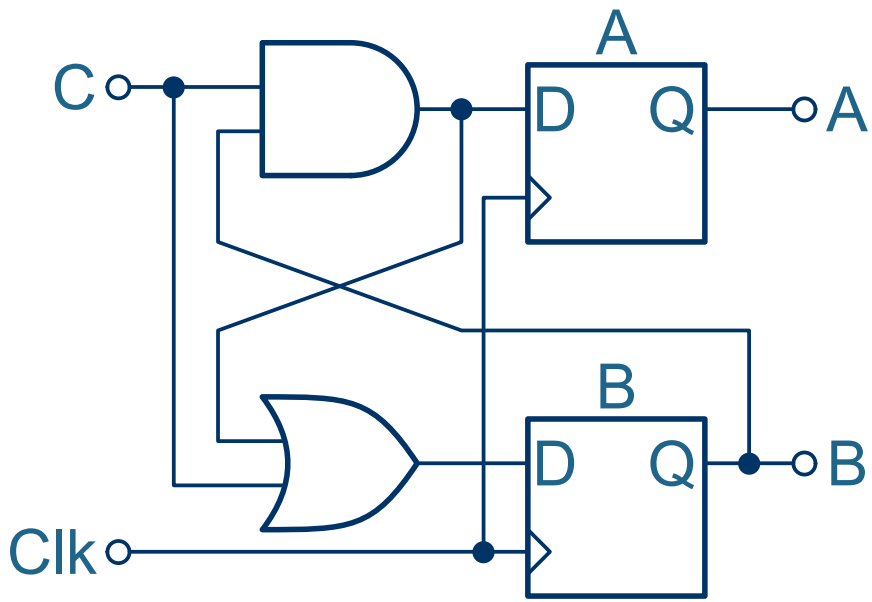

```
reg A, B;

always @(posedge Clk) begin
 A = C & B;
 B = A | C;
end
```



- Statements are evaluated in order, like a computer program
- Often results in unintentionally long combinational chains
- Note that all registers still change state on the clock edge

Blocking Statements

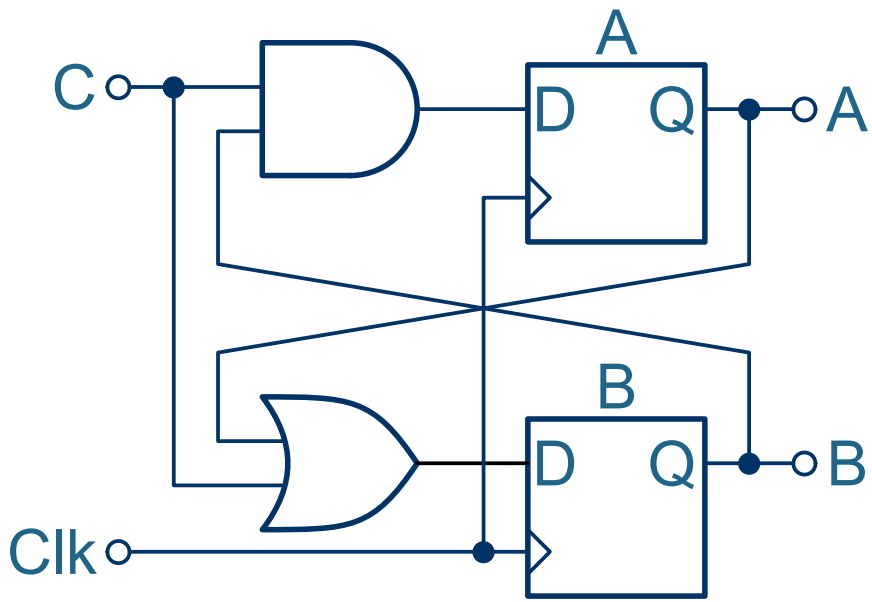

```
reg A, B;

always @(posedge Clk) begin
 A = C & B;
 B = A | C;
end
```



- Statements are evaluated in order, like a computer program
- Often results in unintentionally long combinational chains
- Note that all registers still change state on the clock edge

Non-Blocking Statements

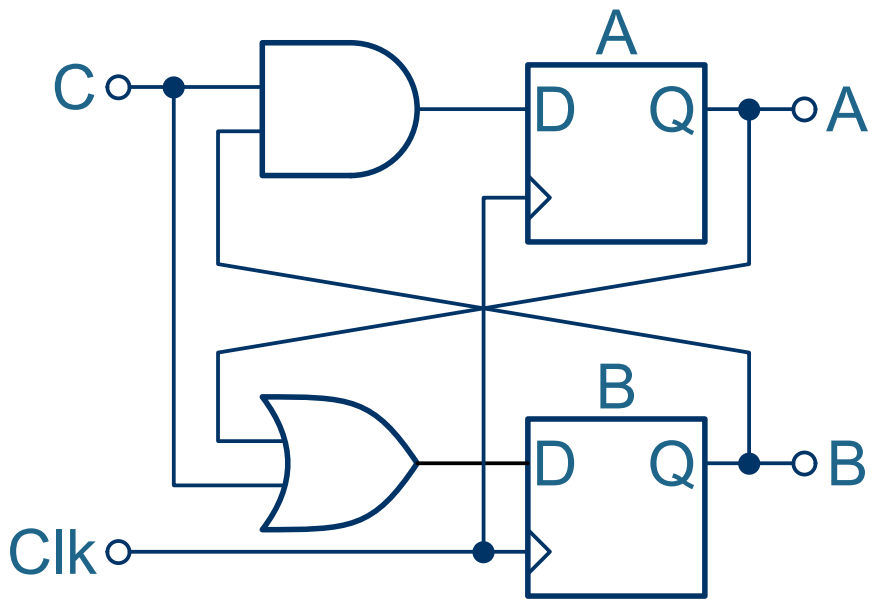

```
reg A, B;

always @(posedge Clk) begin
 A <= C & B;
 B <= A | C;
end
```



- All right-hand-side expressions are evaluated in parallel
- and then assigned to the left-hand-side on the clock edge
- The order of statements makes no difference to the functionality

Non-Blocking Statements

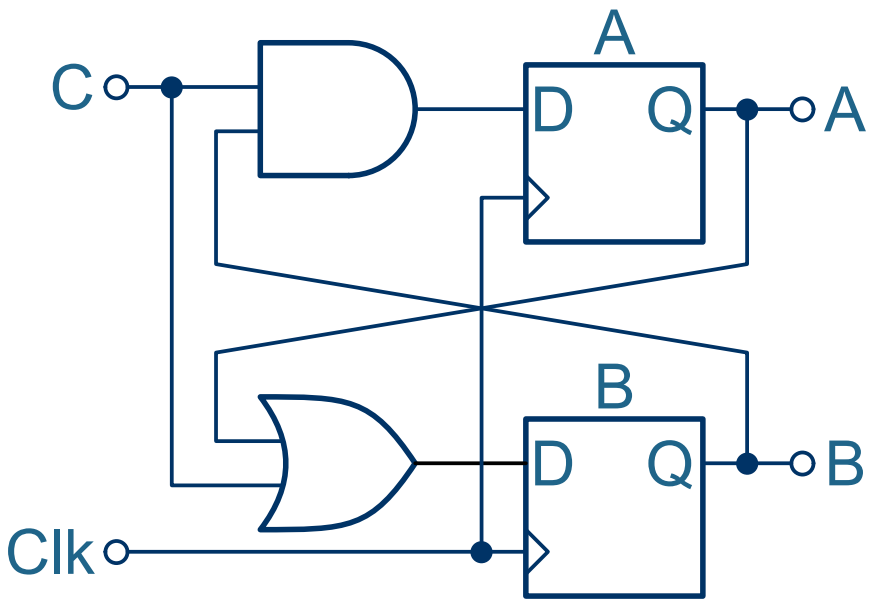

```
reg A, B;

always @(posedge Clk) begin
 A <= C & B;
 B <= A | C;
end
```



- All right-hand-side expressions are evaluated in parallel
- and then assigned to the left-hand-side on the clock edge
- The order of statements makes no difference to the functionality

Non-Blocking Statements

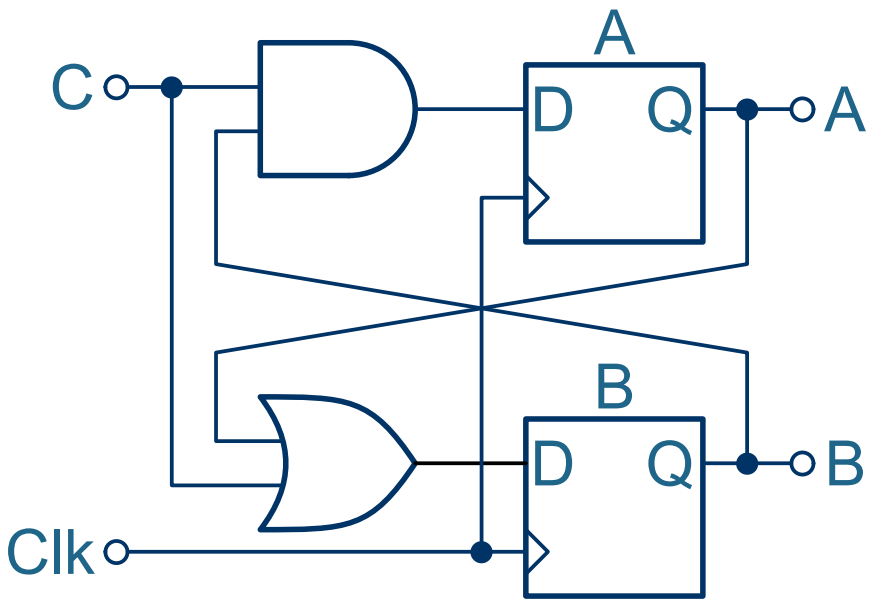

```
reg A, B;

always @(posedge Clk) begin
 A <= C & B;
 B <= A | C;
end
```



- All right-hand-side expressions are evaluated in parallel
- and then assigned to the left-hand-side on the clock edge
- The order of statements makes no difference to the functionality

Non-Blocking Statements


```
reg A, B;

always @(posedge Clk) begin
 A <= C & B;
 B <= A | C;
end
```



- All right-hand-side expressions are evaluated in parallel
- and then assigned to the left-hand-side on the clock edge
- The order of statements makes no difference to the functionality

Mixed Mode

Never mix blocking and non-blocking statements in the same always block

Except inside test-benches, where it is sometimes useful...

Mixed Mode

Never mix blocking and non-blocking statements in the same always block

Except inside test-benches, where it is sometimes useful...

Algorithmic Combinational Blocks

- Use blocking assignments: allows algorithmic descriptions
- If not explicitly assigned a new value, the previous value is "remembered" in an "inferred latch" – to be avoided

```
wire [7:0]Byte;  // Assigned outside always block => wire
reg  [2:0]Count; // Assigned inside  always block => reg
integer   n;     // Non-synthesisable type
                 // (compilation-time only)
always @(*) begin
 Count = 0;
 for(n = 0; n < 8; n = n+1) begin
  Count = Count + Byte[n];
 end
end
```

Algorithmic Combinational Blocks

- Use blocking assignments: allows algorithmic descriptions
- If not explicitly assigned a new value, the previous value is "remembered" in an "inferred latch" – to be avoided

```
wire [7:0]Byte;  // Assigned outside always block => wire
reg  [2:0]Count; // Assigned inside  always block => reg
integer   n;     // Non-synthesisable type
                 // (compilation-time only)
always @(*) begin
 Count = 0;
 for(n = 0; n < 8; n = n+1) begin
  Count = Count + Byte[n];
 end
end
```

Look-up Tables


```
always @(*) begin
 case(BCD)
  4'h0:    SevenSegment = 7'b0111111;
  4'h1:    SevenSegment = 7'b0000110;
  4'h2:    SevenSegment = 7'b1011011;
  4'h3:    SevenSegment = 7'b1001111;
  4'h4:    SevenSegment = 7'b1100110;
  4'h5:    SevenSegment = 7'b1101101;
  4'h6:    SevenSegment = 7'b1111101;
  4'h7:    SevenSegment = 7'b0000111;
  4'h8:    SevenSegment = 7'b1111111;
  4'h9:    SevenSegment = 7'b1101111;
  default:; // This is bad: infers a latch
 endcase
end
```

Look-up Tables


```
always @(*) begin
 case(BCD)
  4'h0:    SevenSegment = 7'b0111111;
  4'h1:    SevenSegment = 7'b0000110;
  4'h2:    SevenSegment = 7'b1011011;
  4'h3:    SevenSegment = 7'b1001111;
  4'h4:    SevenSegment = 7'b1100110;
  4'h5:    SevenSegment = 7'b1101101;
  4'h6:    SevenSegment = 7'b1111101;
  4'h7:    SevenSegment = 7'b0000111;
  4'h8:    SevenSegment = 7'b1111111;
  4'h9:    SevenSegment = 7'b1101111;
  default: SevenSegment = 0; // This is acceptable
 endcase
end
```

Look-up Tables


```
always @(*) begin
 case(BCD)
  4'h0:    SevenSegment = 7'b0111111;
  4'h1:    SevenSegment = 7'b0000110;
  4'h2:    SevenSegment = 7'b1011011;
  4'h3:    SevenSegment = 7'b1001111;
  4'h4:    SevenSegment = 7'b1100110;
  4'h5:    SevenSegment = 7'b1101101;
  4'h6:    SevenSegment = 7'b1111101;
  4'h7:    SevenSegment = 7'b0000111;
  4'h8:    SevenSegment = 7'b1111111;
  4'h9:    SevenSegment = 7'b1101111;
  default: SevenSegment = 7'bXXXXXXX; // This is better
 endcase
end
```

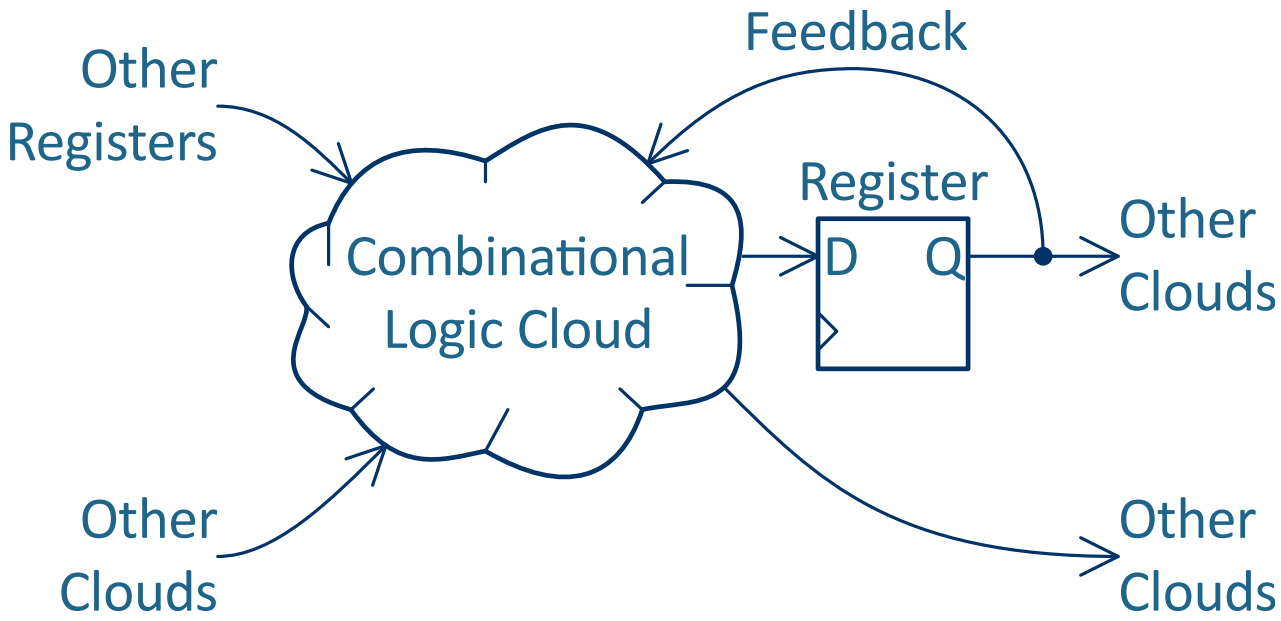
Sparse Case Statements


```
always @(*) begin
 case(Address) // 8-bit address, 16-bit data
  8'h00:   Data = FirmwareVersion;
  8'h01:   Data = BuildDate;
  8'h02:   Data = BuildTime;

  8'h10:   Data = { 6'd0, LED     };
  8'h11:   Data = { 6'd0, Switches};
  8'h12:   Data = {14'd0, Buttons };

  8'h20:   Data = Accelerometer_X;
  8'h21:   Data = Accelerometer_Y;
  8'h22:   Data = Accelerometer_Z;

  default: Data = 0; // This is OK
 endcase
end
```

Sparse Case Statements


```
always @(*) begin
 case(Address) // 8-bit address, 16-bit data
  8'h00:   Data = FirmwareVersion;
  8'h01:   Data = BuildDate;
  8'h02:   Data = BuildTime;

  8'h10:   Data = { 6'd0, LED     };
  8'h11:   Data = { 6'd0, Switches};
  8'h12:   Data = {14'd0, Buttons };

  8'h20:   Data = Accelerometer_X;
  8'h21:   Data = Accelerometer_Y;
  8'h22:   Data = Accelerometer_Z;

  default: Data = 16'hXXXX; // This is better
 endcase
end
```

Register Transfer Logic



Register Transfer Logic

- Use non-blocking assignments:
  easier to relate all calculations to the clock edge

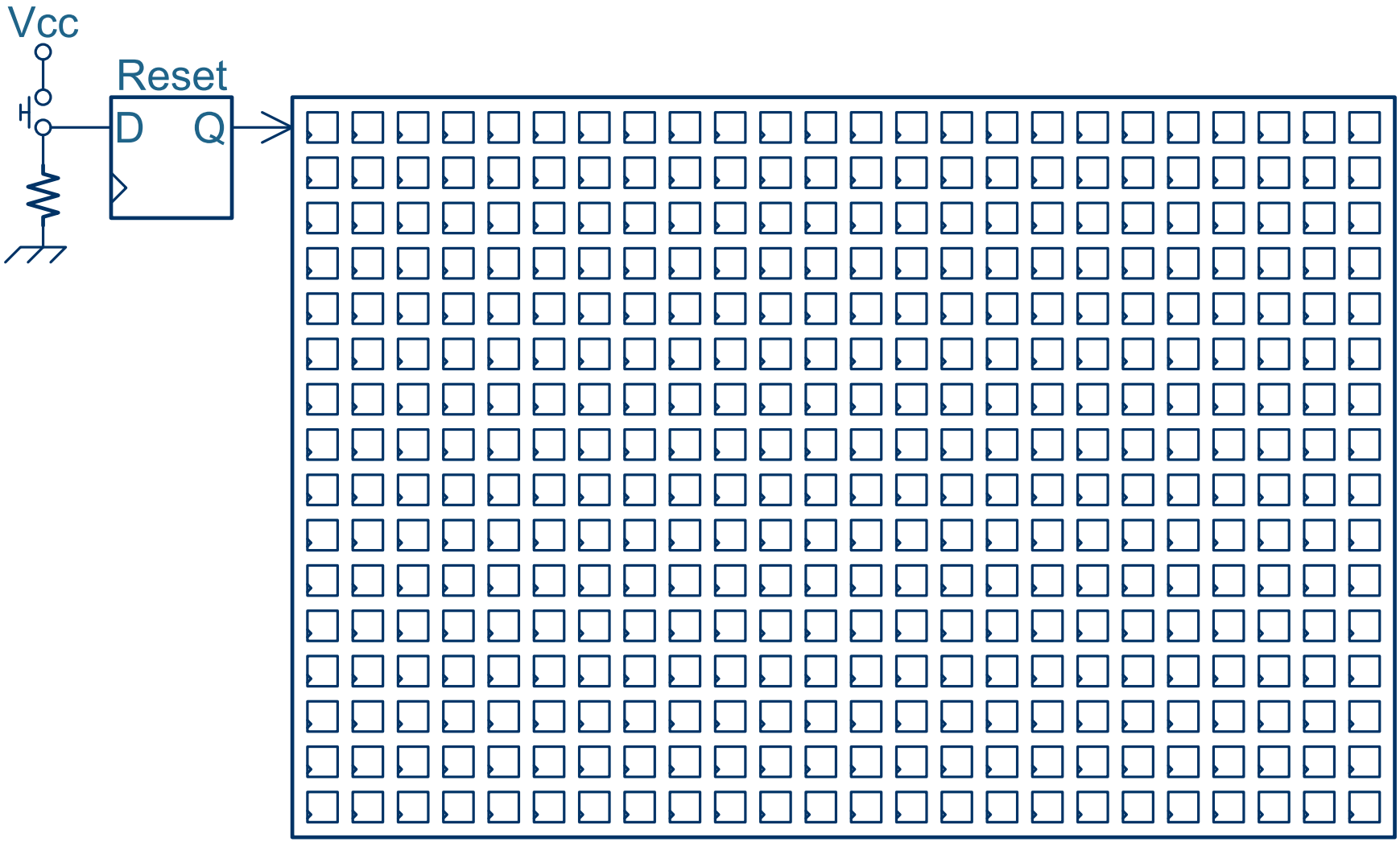
```
reg Local_Reset;

always @(posedge Clk) begin
 Local_Reset <= Reset; // Localise the reset

 if(Local_Reset) begin
  // Reset stuff here

 end else if(Enabled) begin
  // RTL code goes here
 end
end
```

Register Transfer Logic

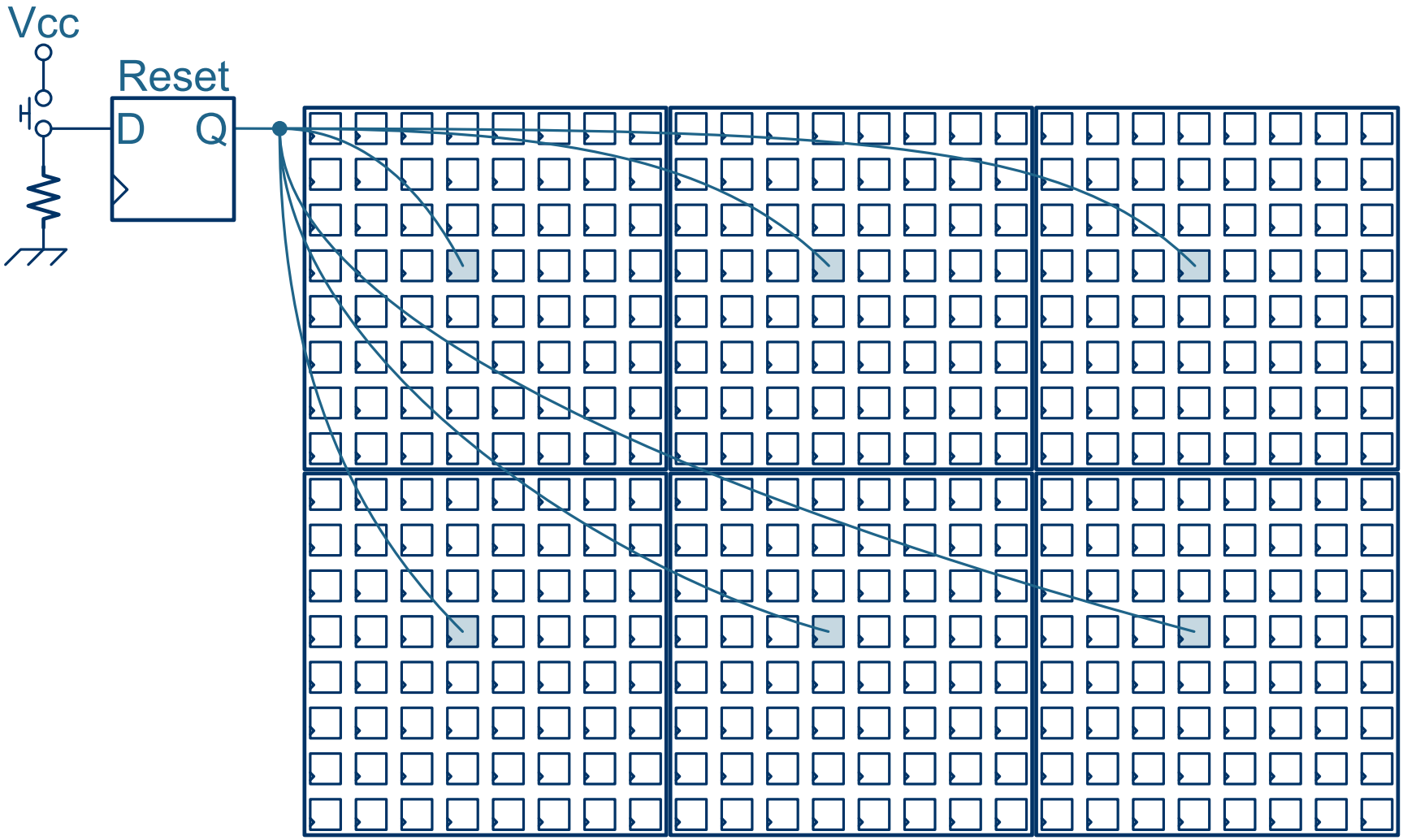
- If not explicitly assigned a new value, the previous value is "remembered" in the register – very useful

```
reg Local_Reset;
reg [11:0]Count;

always @(posedge Clk) begin
 Local_Reset <= Reset; // Localise the reset

 if(Local_Reset) begin
  Count <= 0;

 end else if(Enabled) begin
  Count <= Count + 1'b1;
 end
end
```

Synchronous and Local Resets



Synchronous and Local Resets



Outline

Altera Library

Verilog Processes

Finite State Machines

Timing Constraints

Injection Testing

Simulation

Practical

Appendix – High Resolution PWM

THE RADAR MASTERS COURSE

Finite State Machines

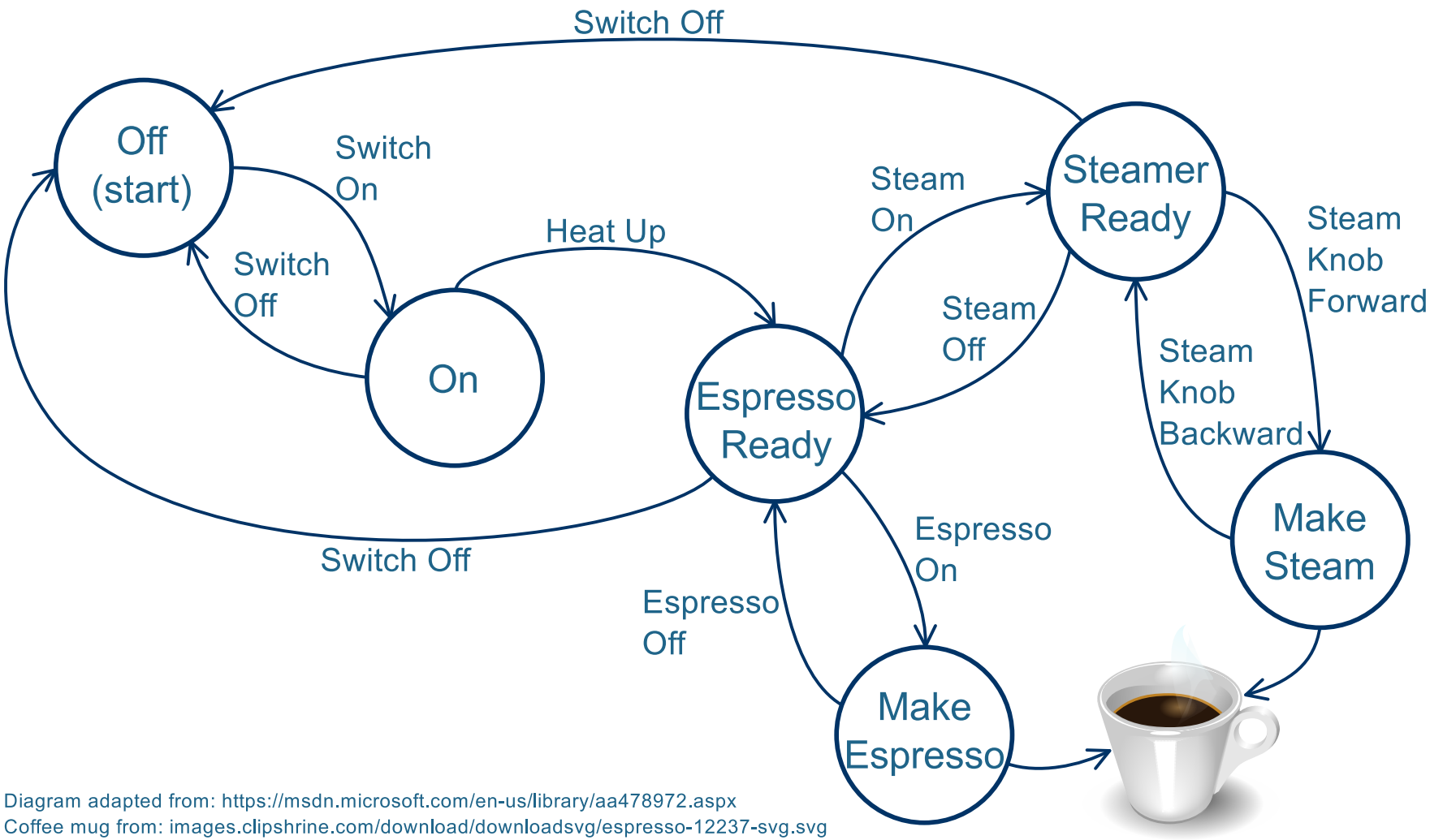
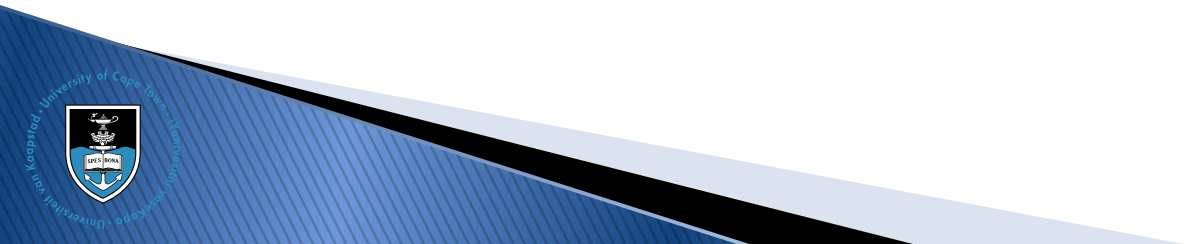


Diagram adapted from: https://msdn.microsoft.com/en-us/library/aa478972.aspx
Coffee mug from: images.clipshrine.com/download/downloadsvg/espresso-12237-svg.svg

Named States

- Example using Gray code:

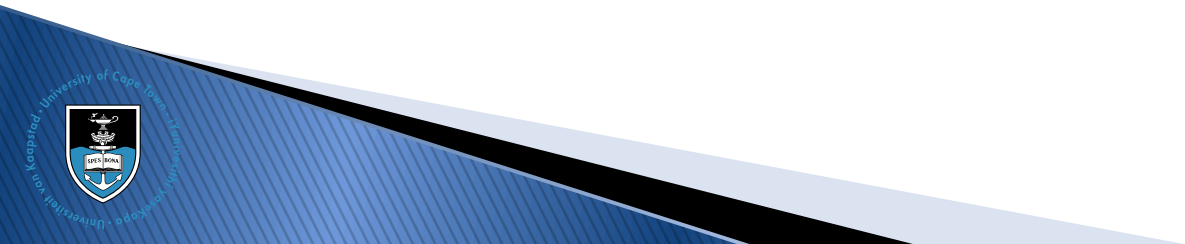
```
reg   [2:0]State;

localparam Off           = 3'b000;
localparam On            = 3'b001;
localparam EspressoReady = 3'b011;
localparam SteamerReady  = 3'b010;
localparam MakeEspresso  = 3'b110;
localparam MakeSteam     = 3'b111;
```



Named States

- Example using one-hot encoding:

```
reg   [5:0]State;

localparam Off           = 6'b000001;
localparam On            = 6'b000010;
localparam EspressoReady = 6'b000100;
localparam SteamerReady  = 6'b001000;
localparam MakeEspresso  = 6'b010000;
localparam MakeSteam     = 6'b100000;
```



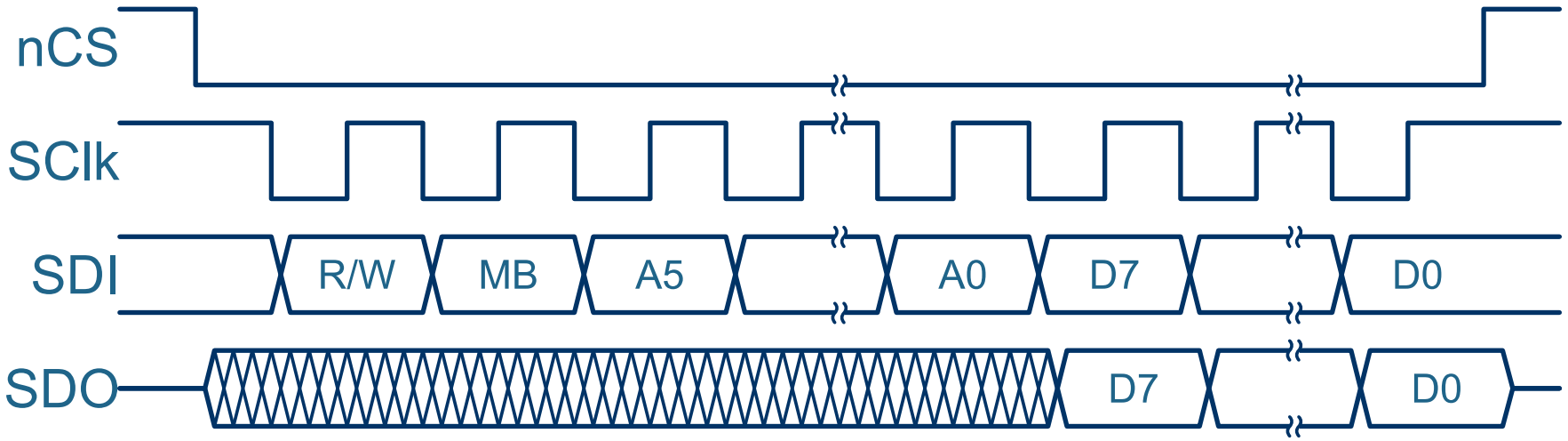
FSM Template


```
always @(posedge Clk) begin
 if(Local_Reset) begin
  State <= Off;

 end else begin
  case(State)
   Off:            begin ... end
   On:             begin ... end
   EspressoReady:  begin ... end
   SteamerReady:   begin ... end
   MakeEspresso:   begin ... end
   MakeSteam:      begin ... end
   default:;
  endcase
 end
end
```

FSM Example


```
                ...
case(State)
 Off: begin
  if(Switch_On) State <= On;
 end

 On: begin
       if(Switch_Off) State <= Off;
  else if(Heat_Up   ) State <= EspressoReady;
 end

 EspressoReady: begin
       if(Switch_Off ) State <= Off;
  else if(Steam_On   ) State <= SteamerReady;
  else if(Espresso_On) State <= MakeEspresso;
 end
                ...
```

SPI Interface



- For the ADXL345 Digital Accelerometer:
  - Maximum SCIk frequency is 5 MHz (200 ns period)
  - SDI sample and hold is 5 ns (sampled on rising edge)
  - SCIk falling edge to SDO delay is 40 ns

The Abstraction


```
module ADXL345 #(
 parameter Clock_Div = 5 // 5 MHz SClk on a 50 MHz Clk
)(
 input Clk, Reset,

 output reg [15:0]X, // 2's Complement
 output reg [15:0]Y,
 output reg [15:0]Z,

 // Physical device interface
 output reg nCS, SClk, SDI,
 input      SDO
);

endmodule
```

General Structure


```
reg      Local_Reset;
reg [3:0]Clock_Count;
wire     Clock_Enable = (Clock_Count == Clock_Div);

always @(posedge Clk) begin
 Local_Reset <= Reset;

 if(Clock_Enable) Clock_Count <= 4'd1;
 else             Clock_Count <= Clock_Count + 1'b1;

 if(Local_Reset) begin
  // Reset the machine here

 end else if(Clock_Enable) begin
  // State machine goes here
 end
end
```

State Machine


```
reg [ 4:0]Count;
reg [15:0]WriteData; // R/W, MB, Address, Byte
reg [15:0]ReadData;  // 2 Bytes

reg [2:0]State;
reg [2:0]RetState; // Used for function calls

localparam Setup       = 3'd0;
localparam ReadX       = 3'd1;
localparam ReadY       = 3'd2;
localparam ReadZ       = 3'd3;
localparam Transaction = 3'd4;
```

State Machine


```
if(Local_Reset) begin
 nCS    <= 1'b1;
 SClk   <= 1'b1;
 SDI    <= 1'b1;
 State  <= Setup;

end else if(Clock_Enable) begin
 case(State)
  Setup: begin
   // SPI 4-wire; Full-res; Right-justify; 4g Range
   WriteData <= {2'b00, 6'h31, 8'b0000_1001};
   Count     <= 5'd16;
   State     <= Transaction;
   RetState  <= ReadX;
  end
```

State Machine


```
ReadX: begin
  Z         <= {ReadData[7:0], ReadData[15:8]};
  WriteData <= {2'b11, 6'h32, 8'd0};
  Count     <= 5'd24;
  State     <= Transaction;
  RetState  <= ReadY;
end

ReadY: begin
  X         <= {ReadData[7:0], ReadData[15:8]};
  WriteData <= {2'b11, 6'h34, 8'd0};
  Count     <= 5'd24;
  State     <= Transaction;
  RetState  <= ReadZ;
end
```

State Machine


```
ReadZ: begin
  Y         <= {ReadData[7:0], ReadData[15:8]};
  WriteData <= {2'b11, 6'h36, 8'd0};
  Count     <= 5'd24;
  State     <= Transaction;
  RetState  <= ReadX;
end
```

Reading Data


```
Transaction: begin
  if(nCS) begin
    nCS <= 1'b0;

  end else begin
    if(SClk) begin
      if(Count == 0) begin
        nCS <= 1'b1; State <= RetState;
      end else begin
        SClk <= 1'b0;
        {SDI, WriteData[15:1]} <= WriteData;
      end
      Count     <= Count - 1'b1;
      ReadData <= {ReadData[14:0], SDO};

    end else begin
      SClk <= 1'b1;
end end end
```

Coffee Break...

Outline

Altera Library

Verilog Processes

Finite State Machines

Timing Constraints

Injection Testing

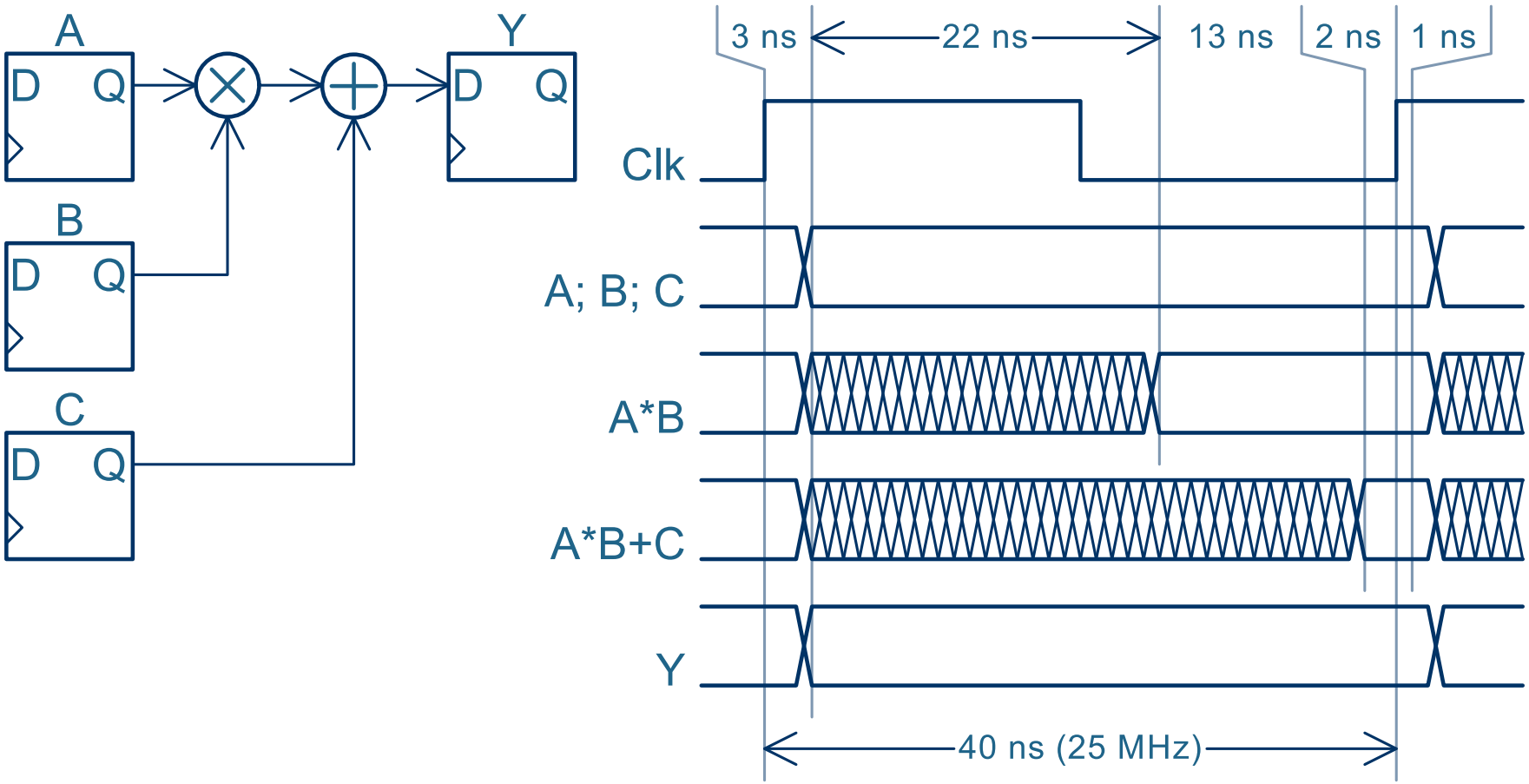
Simulation

Practical

Appendix – High Resolution PWM

THE RADAR MASTERS COURSE

Internal Timing



Synopsis Design Constraints

- De facto industry standard
- TCL based, so one can use TCL scripting within the SDC file
- Only specify what Quartus does not already know:
  - Clock frequencies
  - Asynchronous paths
  - PCB trace delays
  - External device parameters
  - Multi-cycle paths
  - etc.

Synopsis Design Constraints

- De facto industry standard
- TCL based, so one can use TCL scripting within the SDC file
- Only specify what Quartus does not already know:
  - Clock frequencies
  - Asynchronous paths
  - PCB trace delays
  - External device parameters
  - Multi-cycle paths
  - etc.

Synopsis Design Constraints

- De facto industry standard
- TCL based, so one can use TCL scripting within the SDC file
- Only specify what Quartus does not already know:
  - Clock frequencies
  - Asynchronous paths
  - PCB trace delays
  - External device parameters
  - Multi-cycle paths
  - etc.

Synopsis Design Constraints

- De facto industry standard
- TCL based, so one can use TCL scripting within the SDC file
- Only specify what Quartus does not already know:
  - Clock frequencies
  - Asynchronous paths
  - PCB trace delays
  - External device parameters
  - Multi-cycle paths
  - etc.

Synopsis Design Constraints

- De facto industry standard
- TCL based, so one can use TCL scripting within the SDC file
- Only specify what Quartus does not already know:
  - Clock frequencies
  - Asynchronous paths
  - PCB trace delays
  - External device parameters
  - Multi-cycle paths
  - etc.

Synopsis Design Constraints

- De facto industry standard
- TCL based, so one can use TCL scripting within the SDC file
- Only specify what Quartus does not already know:
  - Clock frequencies
  - Asynchronous paths
  - PCB trace delays
  - External device parameters
  - Multi-cycle paths
  - etc.

Synopsis Design Constraints

- De facto industry standard
- TCL based, so one can use TCL scripting within the SDC file
- Only specify what Quartus does not already know:
  - Clock frequencies
  - Asynchronous paths
  - PCB trace delays
  - External device parameters
  - Multi-cycle paths
  - etc.

Synopsis Design Constraints

- De facto industry standard
- TCL based, so one can use TCL scripting within the SDC file
- Only specify what Quartus does not already know:
  - Clock frequencies
  - Asynchronous paths
  - PCB trace delays
  - External device parameters
  - Multi-cycle paths
  - etc.

Synopsis Design Constraints

- De facto industry standard
- TCL based, so one can use TCL scripting within the SDC file
- Only specify what Quartus does not already know:
  - Clock frequencies
  - Asynchronous paths
  - PCB trace delays
  - External device parameters
  - Multi-cycle paths
  - etc.

Asynchronous Pins

- Pins such as LEDs, buttons, RS-232 signals, etc. does not belong to a clock domain
- Quartus must not try to meet timing on these:

```
set_false_path -from * -to   [get_ports LED*]
set_false_path -to   * -from [get_ports Switch*]
set_false_path -to   * -from [get_ports Button*]
```

Asynchronous Pins

- Pins such as LEDs, buttons, RS-232 signals, etc. does not belong to a clock domain
- Quartus must not try to meet timing on these:

```
set_false_path -from * -to   [get_ports LED*]
set_false_path -to   * -from [get_ports Switch*]
set_false_path -to   * -from [get_ports Button*]
```

Clock Specification


```
create_clock -period 100 [get_ports ADC_CLK_10]
create_clock -period  20 [get_ports MAX10_CLK1_50]
create_clock -period  20 [get_ports MAX10_CLK2_50]

derive_pll_clocks -create_base_clocks -use_net_name
derive_clock_uncertainty

# Reference the SDRAM clock to the pin
create_generated_clock \
  -source [get_pins {MyQsys_Inst|altpll_0|sd1|pll7|clk[1]}] \
  -name DRAM_CLK [get_ports {CLK_DRAM}]
```

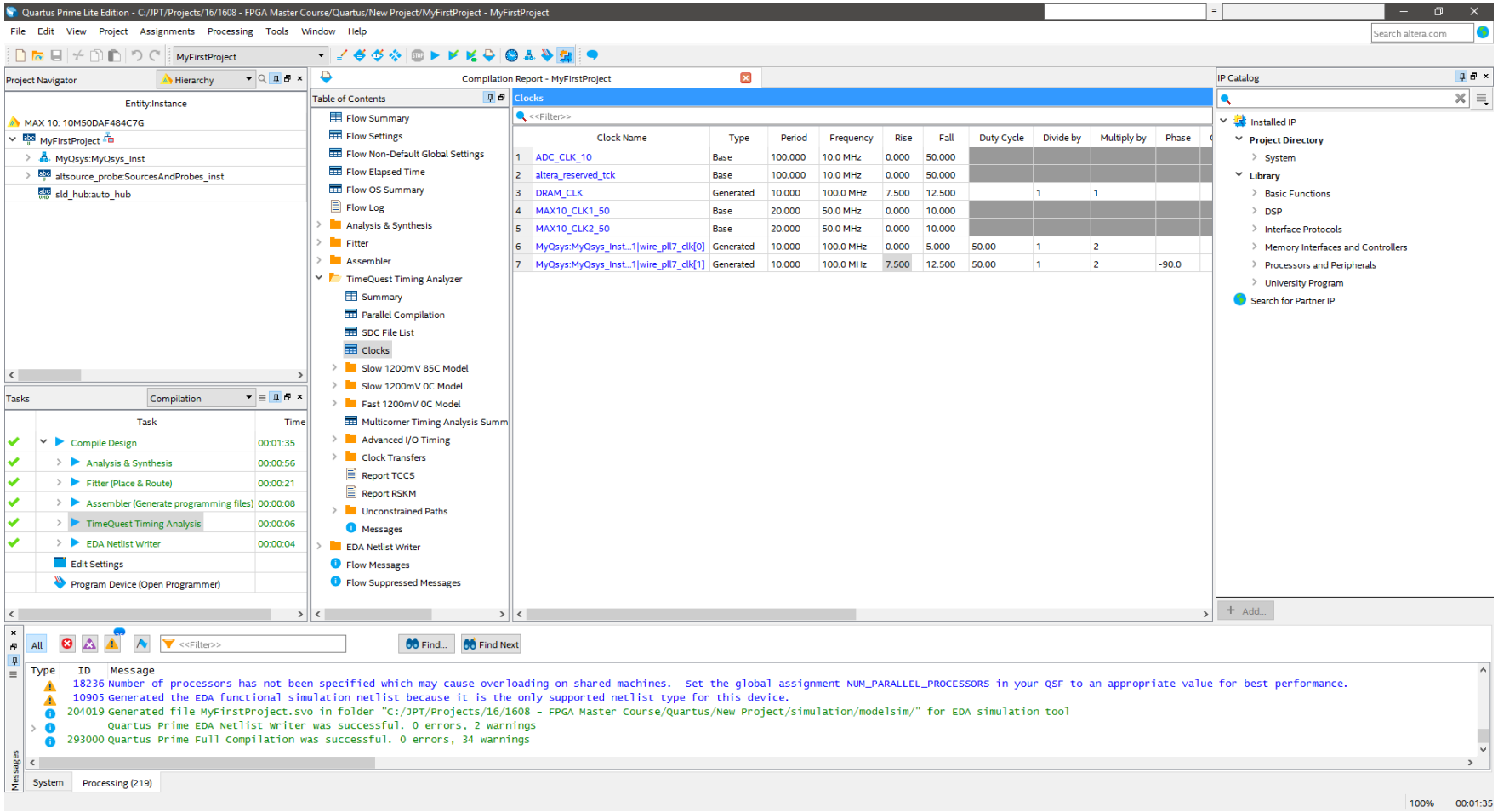
To get the node names, use the “Clocks” report of the TimeQuest Timing Analyser and copy the name, or use the TimeQuest GUI to search for it.

Clock Specification


```
create_clock -period 100 [get_ports ADC_CLK_10]
create_clock -period  20 [get_ports MAX10_CLK1_50]
create_clock -period  20 [get_ports MAX10_CLK2_50]

derive_pll_clocks -create_base_clocks -use_net_name
derive_clock_uncertainty

# Reference the SDRAM clock to the pin
create_generated_clock \
  -source [get_pins {MyQsys_Inst|altpll_0|sd1|pll7|clk[1]}] \
  -name DRAM_CLK [get_ports {CLK_DRAM}]
```

To get the node names, use the “Clocks” report of the TimeQuest Timing Analyser and copy the name, or use the TimeQuest GUI to search for it.

Clock Report



Clock Groups

- Unless specified otherwise, Quartus assumes that all clocks are related and in the same clock domain
- When clocks are unrelated, all paths between them must be marked as “false paths”
- Do this with clock groups:

```
set_clock_groups -logically_exclusive \
  -group [get_clocks ADC_CLK_10]      \
  -group [get_clocks MAX10_CLK1_50]   \
  -group [get_clocks MAX10_CLK2_50]   \
  -group [get_clocks {DRAM_CLK *altpll_0*}]
```

Clock Groups

- Unless specified otherwise, Quartus assumes that all clocks are related and in the same clock domain
- When clocks are unrelated, all paths between them must be marked as “false paths”
- Do this with clock groups:

```
set_clock_groups -logically_exclusive \
  -group [get_clocks ADC_CLK_10]      \
  -group [get_clocks MAX10_CLK1_50]   \
  -group [get_clocks MAX10_CLK2_50]   \
  -group [get_clocks {DRAM_CLK *altpll_0*}]
```

Clock Groups

- Unless specified otherwise, Quartus assumes that all clocks are related and in the same clock domain
- When clocks are unrelated, all paths between them must be marked as “false paths”
- Do this with clock groups:

```
set_clock_groups -logically_exclusive \
  -group [get_clocks ADC_CLK_10]      \
  -group [get_clocks MAX10_CLK1_50]   \
  -group [get_clocks MAX10_CLK2_50]   \
  -group [get_clocks {DRAM_CLK *altpll_0*}]
```

External Timing Requirements

- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

External Timing Requirements

- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

External Timing Requirements

- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

External Timing Requirements

- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

External Timing Requirements

- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

External Timing Requirements

- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

External Timing Requirements

- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

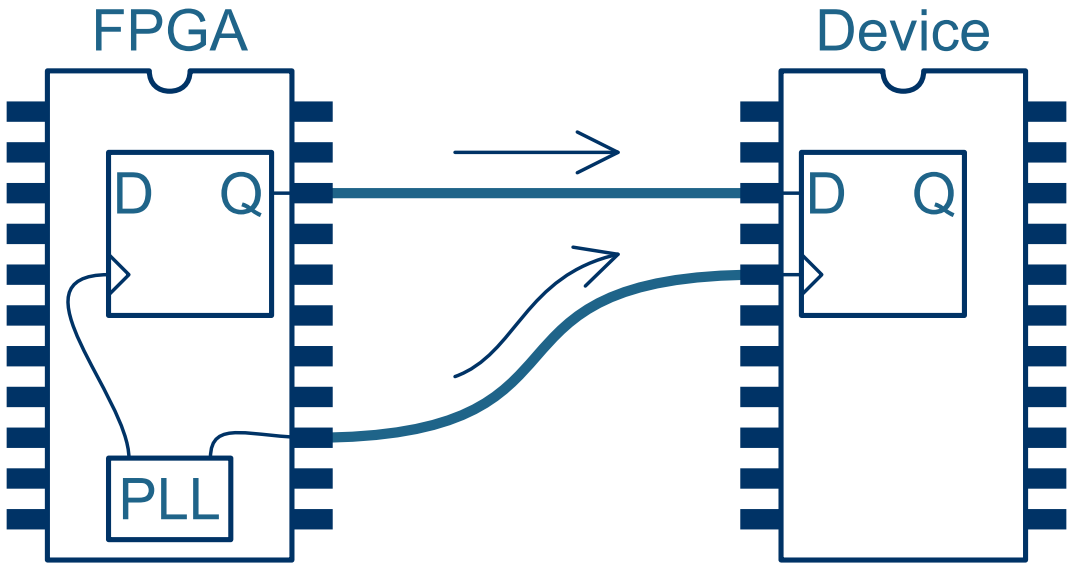
External Timing Requirements

- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

External Timing Requirements

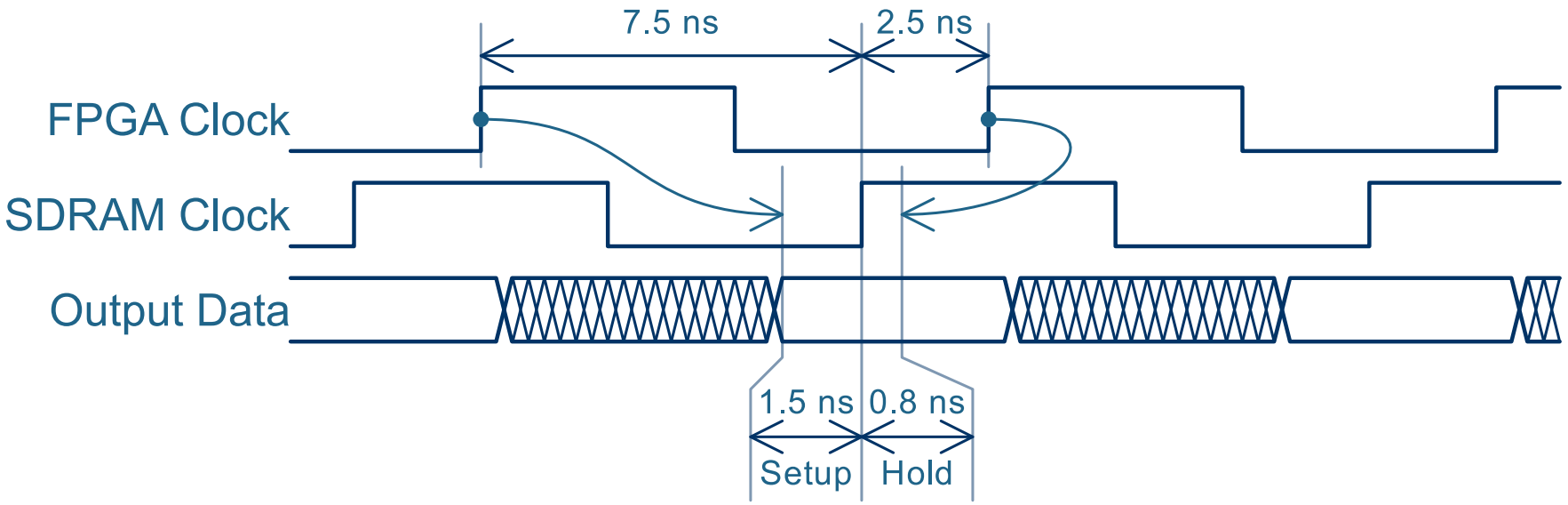
- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

External Timing – Output



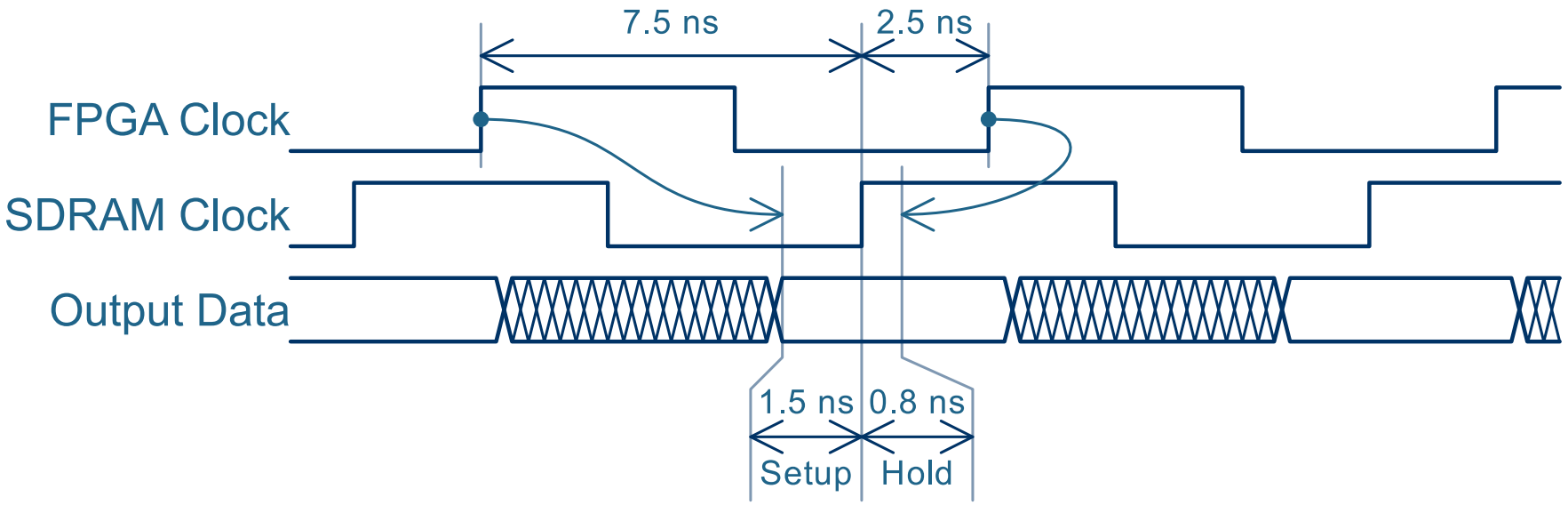
- FPGA internal delays and PCB trace delays cancel
- The important parameters are:
  - Setup and hold times of the external device
  - Clock jitter and other uncertainties
- Shift the external clock to ease the hold margin

External Timing – Output



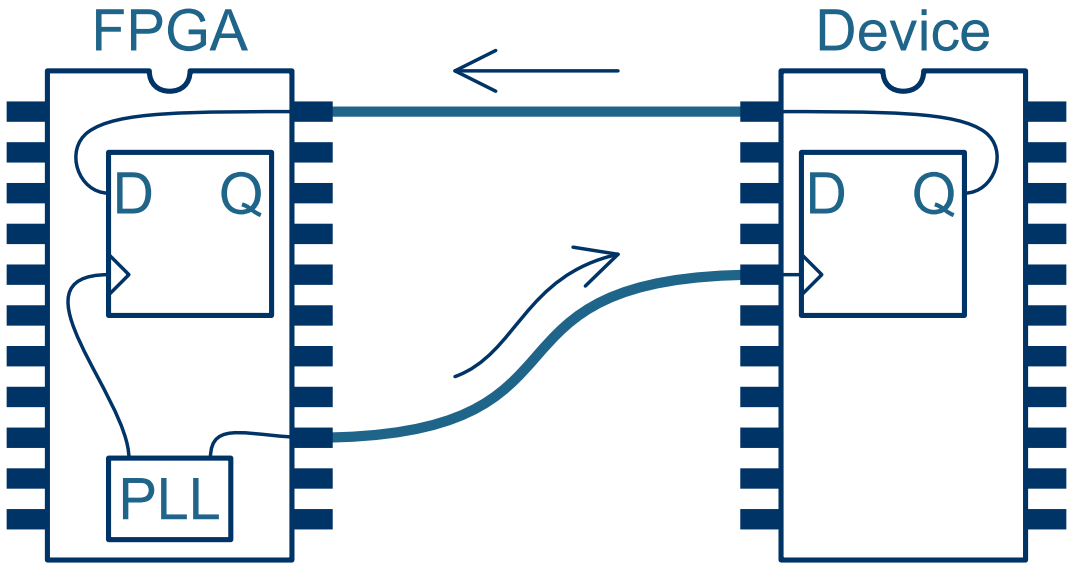
- FPGA internal delays and PCB trace delays cancel
- The important parameters are:
  - Setup and hold times of the external device
  - Clock jitter and other uncertainties
- Shift the external clock to ease the hold margin

External Timing – Output



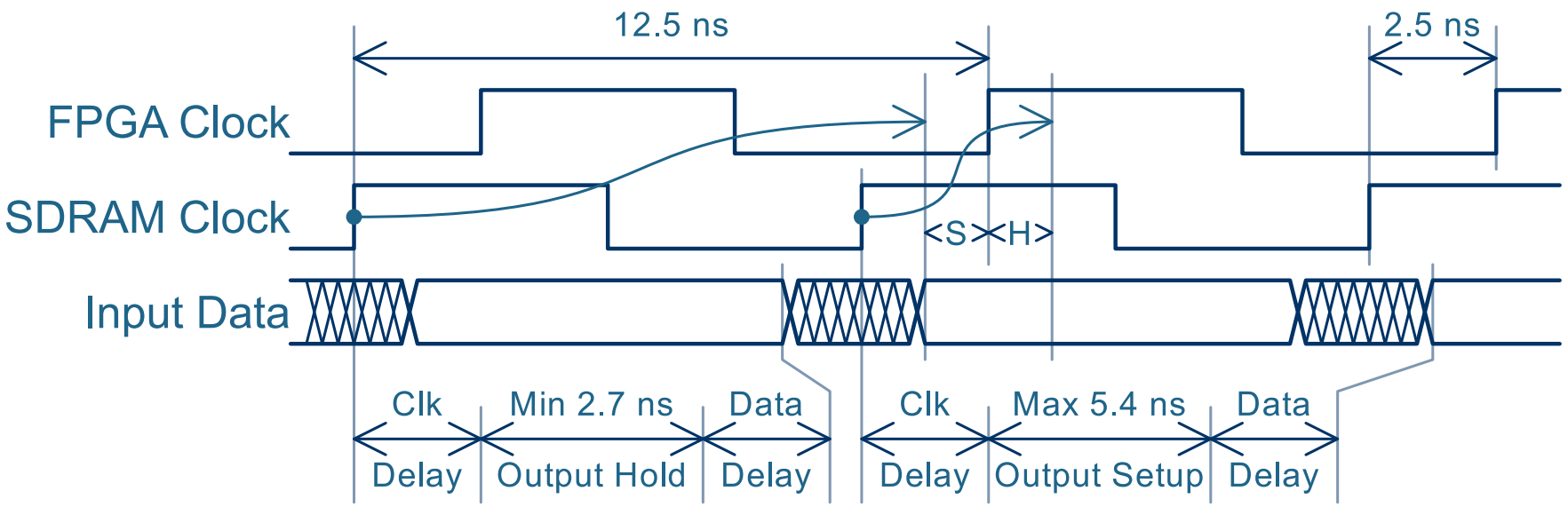

```
# Suppose 100 ps uncertainty
set_output_delay -max -clock DRAM_CLK  1.6 [get_ports OP_DRAM*]
set_output_delay -min -clock DRAM_CLK -0.9 [get_ports OP_DRAM*]
set_output_delay -max -clock DRAM_CLK  1.6 [get_ports BP_DRAM*]
set_output_delay -min -clock DRAM_CLK -0.9 [get_ports BP_DRAM*]
```

External Timing – Input



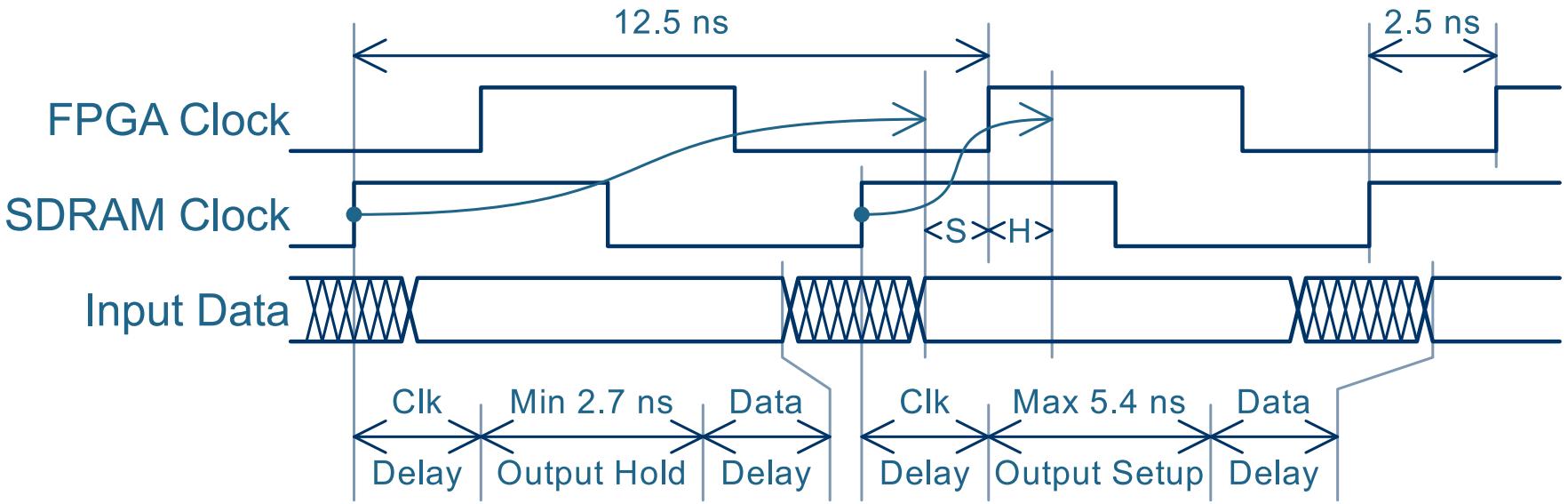
- Large FPGA internal delays and PCB trace delays
- Shift the external clock to ease the setup margin
- Multi-cycle requirement on the setup path (otherwise Quartus uses the 2.5 ns path) – the minimum propagation delay of 2.7 ns makes the 2.5 ns clock shift safe

External Timing – Input



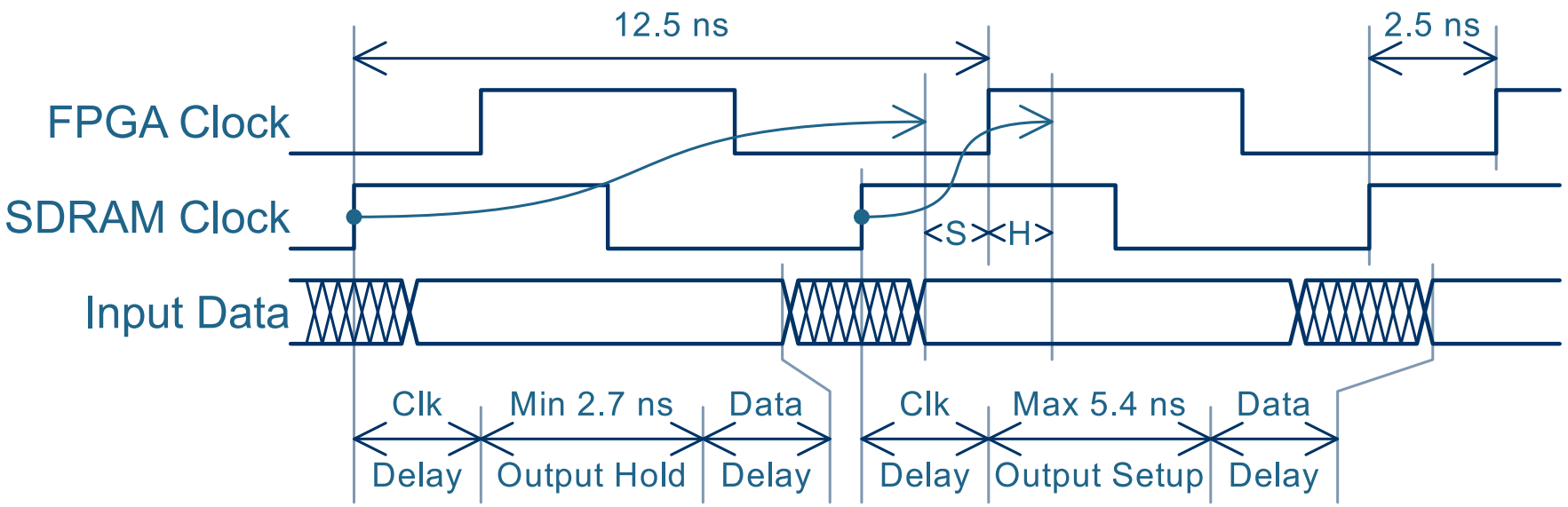
- Large FPGA internal delays and PCB trace delays
- Shift the external clock to ease the setup margin
- Multi-cycle requirement on the setup path (otherwise Quartus uses the 2.5 ns path) – the minimum propagation delay of 2.7 ns makes the 2.5 ns clock shift safe

External Timing – Input



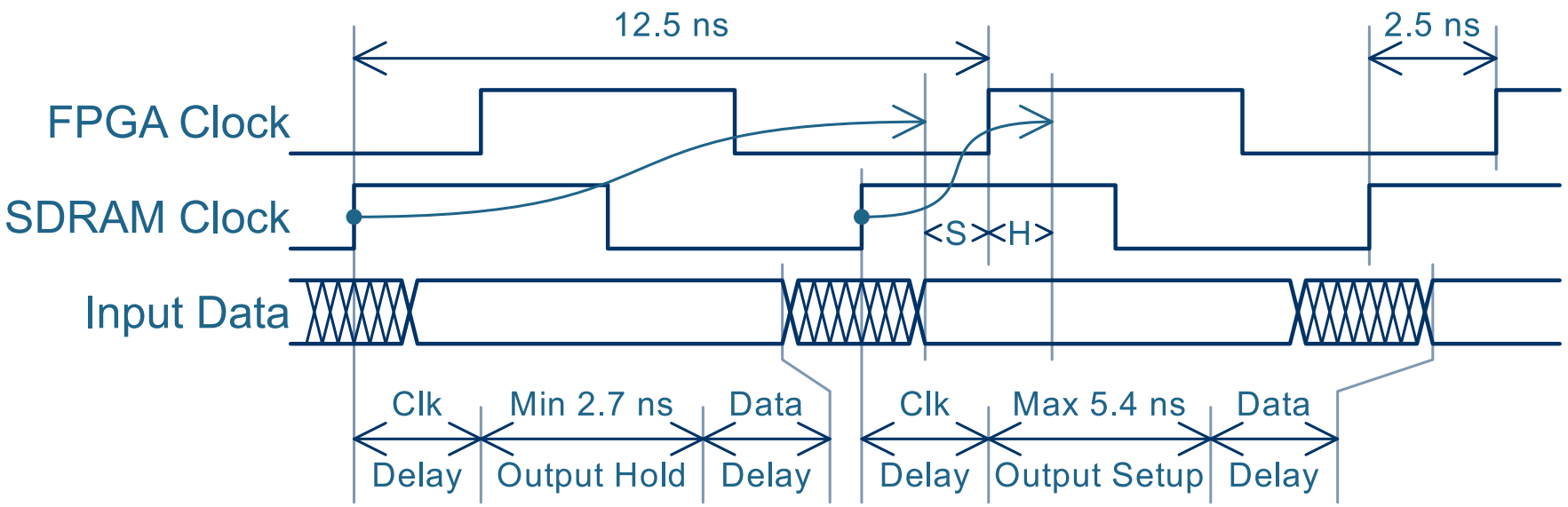
- Large FPGA internal delays and PCB trace delays
- Shift the external clock to ease the setup margin
- Multi-cycle requirement on the setup path (otherwise Quartus uses the 2.5 ns path) – the minimum propagation delay of 2.7 ns makes the 2.5 ns clock shift safe

External Timing – Input



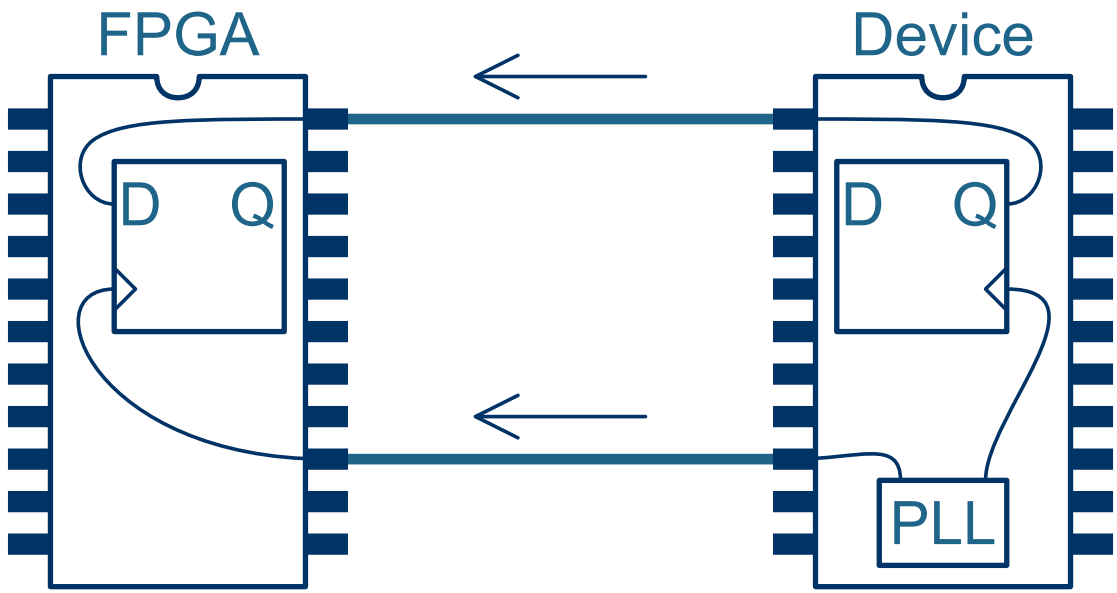

```
set_multicycle_path                                \
  -from [get_clocks {DRAM_CLK}]                    \
  -to   [get_clocks {*altpll_0*clk[0]}]            \
  -setup 2
```

External Timing – Input



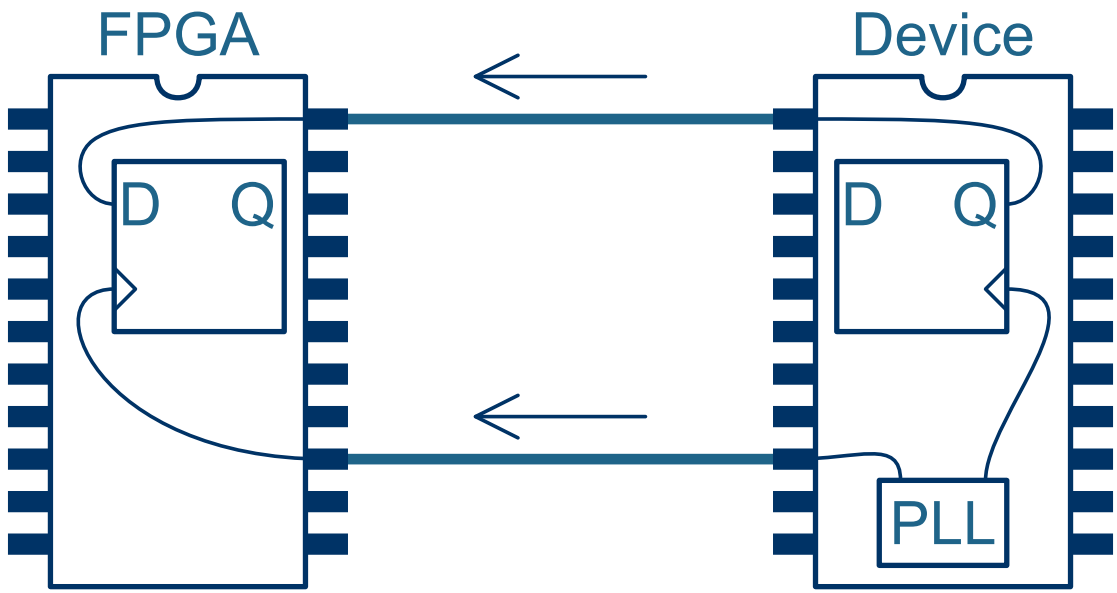

```
# Suppose 100 ps uncertainty and 200 ps PCB delay (each way)
set_input_delay -max -clock DRAM_CLK 5.9 [get_ports BP_DRAM*]
set_input_delay -min -clock DRAM_CLK 3.0 [get_ports BP_DRAM*]
```

External Timing – DDR ADC



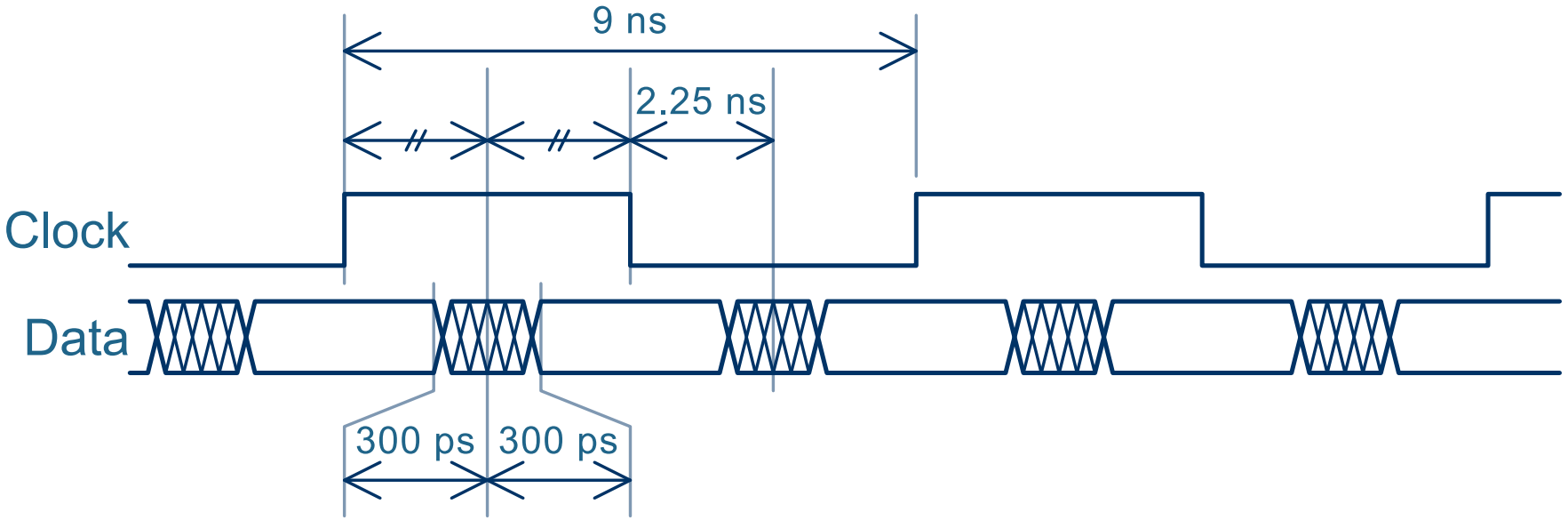
- The clock is sourced by the external device
- The PCB trace delays cancel
- The data is centre-aligned
- Quartus automatically does input port alignment; Xilinx must be manually tuned: use the IDELAY primitive

External Timing – DDR ADC



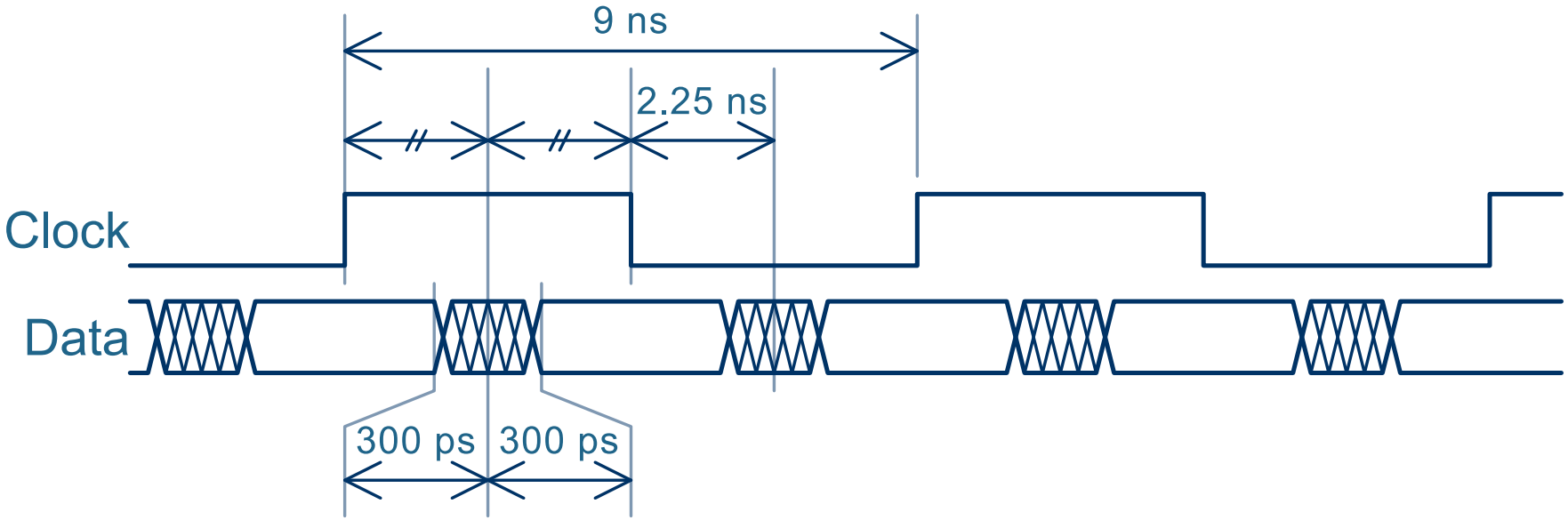
- The clock is sourced by the external device
- The PCB trace delays cancel
- The data is centre-aligned
- Quartus automatically does input port alignment; Xilinx must be manually tuned: use the IDELAY primitive

External Timing – DDR ADC



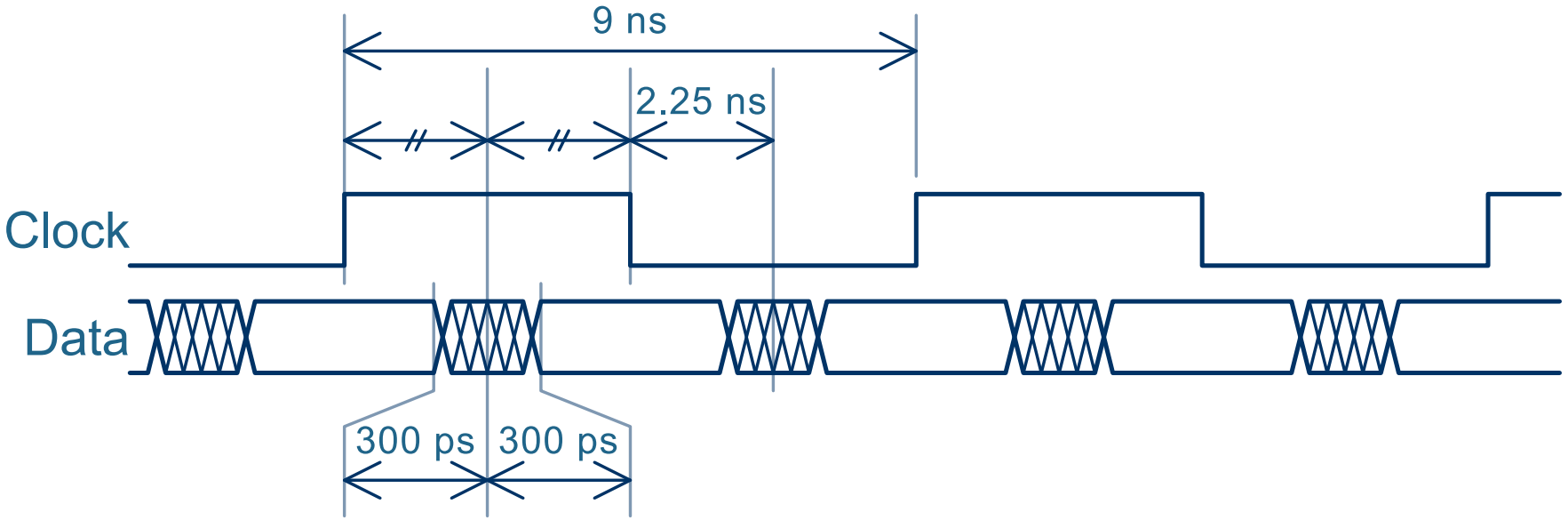
- The clock is sourced by the external device
- The PCB trace delays cancel
- The data is centre-aligned
- Quartus automatically does input port alignment;
  Xilinx must be manually tuned: use the IDELAY primitive

External Timing – DDR ADC



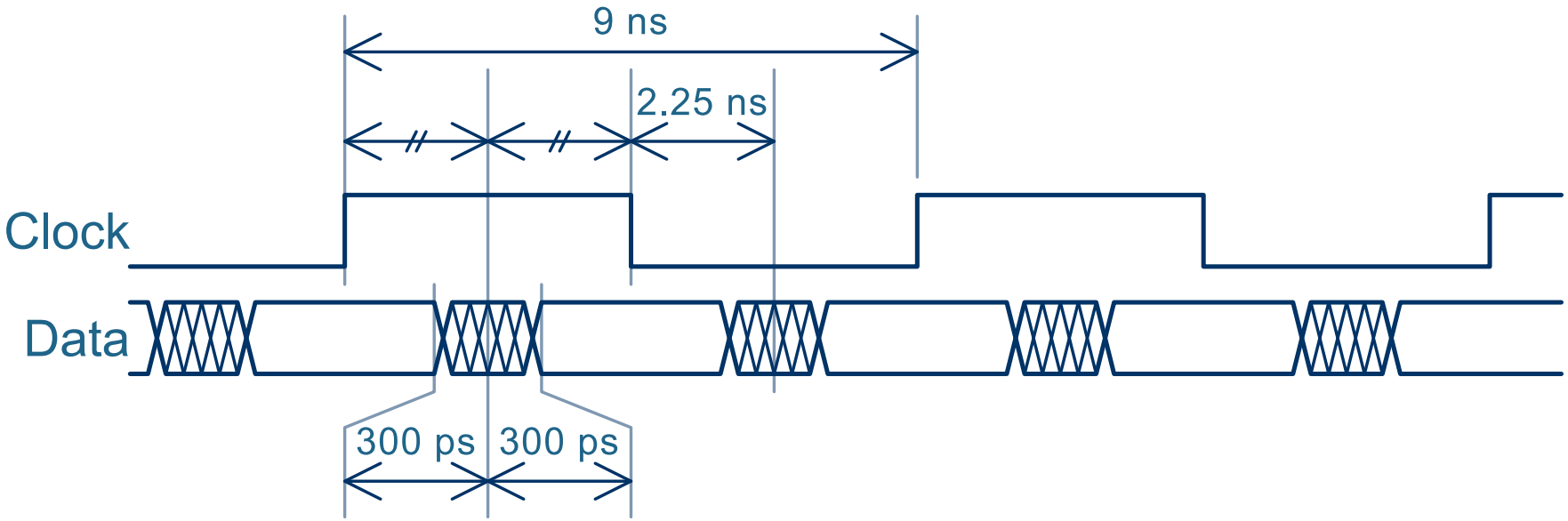
- The clock is sourced by the external device
- The PCB trace delays cancel
- The data is centre-aligned
- Quartus automatically does input port alignment;
  Xilinx must be manually tuned: use the IDELAY primitive

External Timing – DDR ADC




```
# Suppose 100 ps uncertainty
create_clock     -name  ADC_DCO \
  -period 9 [get_ports IP_ADC_DCLK_P]
set_input_delay -clock ADC_DCO \
  -min 1.85 [get_ports IP_ADC_CH*]
set_input_delay -clock ADC_DCO \
  -max 2.65 [get_ports IP_ADC_CH*]
```

External Timing – DDR ADC




```
set_input_delay -clock ADC_DCO -clock_fall \
  -min 1.85 [get_ports IP_ADC_CH*] -add_delay
set_input_delay -clock ADC_DCO -clock_fall \
  -max 2.65 [get_ports IP_ADC_CH*] -add_delay
```

Timing-related Pin Attributes

- Use the correct I/O standard
- Set the correct output current
- Set the pin capacitance

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" \
  -to OP_DRAM*
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" \
  -to BP_DRAM*
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" \
  -to CLK_DRAM
```

Timing-related Pin Attributes

- Use the correct I/O standard
- Set the correct output current
- Set the pin capacitance

```
set_instance_assignment -name CURRENT_STRENGTH_NEW 8MA \
  -to OP_DRAM*
set_instance_assignment -name CURRENT_STRENGTH_NEW 8MA \
  -to BP_DRAM*
set_instance_assignment -name CURRENT_STRENGTH_NEW 8MA \
  -to CLK_DRAM
```

Timing-related Pin Attributes

- Use the correct I/O standard
- Set the correct output current
- Set the pin capacitance

```
set_instance_assignment -name BOARD_MODEL_NEAR_C 3.8P \
  -to OP_DRAM*
set_instance_assignment -name BOARD_MODEL_NEAR_C 6.0P \
  -to BP_DRAM_DQ*
set_instance_assignment -name BOARD_MODEL_NEAR_C 3.5P \
  -to CLK_DRAM
```

JTAG Timing Specification

- Standard across all Altera FPGAs
- Copied from Altera examples:

```
# Don't specify the altera_reserved_tck frequency,
# Quartus knows what it is.
set_clock_groups -exclusive \
  -group [get_clocks {altera_reserved_tck}]

set_input_delay  -clock altera_reserved_tck 20 \
  [get_ports {altera_reserved_tdi}]
set_input_delay  -clock altera_reserved_tck 20 \
  [get_ports {altera_reserved_tms}]
set_output_delay -clock altera_reserved_tck 20 \
  [get_ports {altera_reserved_tdo}]
```

Coffee Break...

Outline

Altera Library

Verilog Processes

Finite State Machines

Timing Constraints

Injection Testing

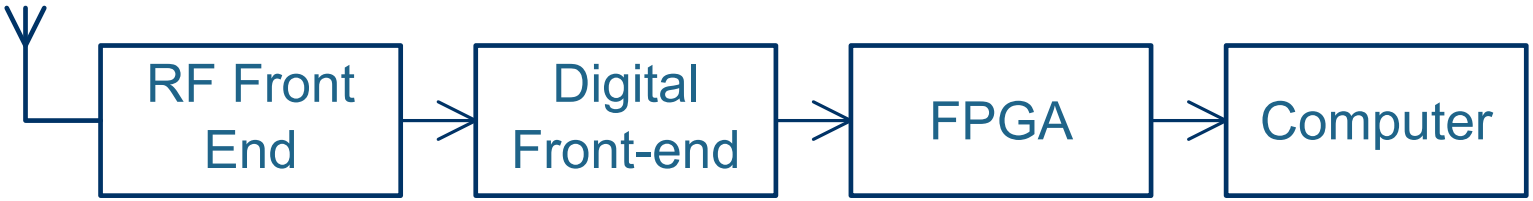
Simulation

Practical

Appendix – High Resolution PWM

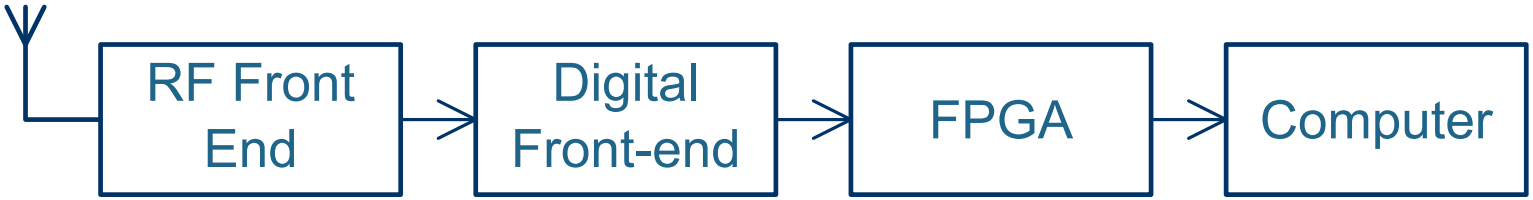
THE RADAR MASTERS COURSE

Typical Development Challenges



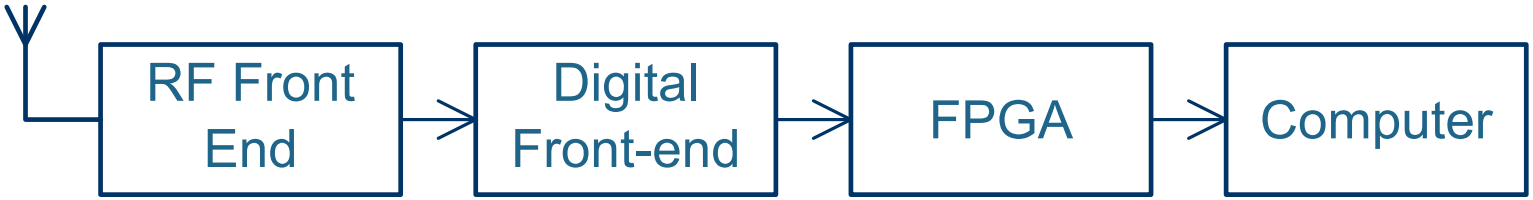
- The hardware (RF front-end and DFE) is not available at the time of FPGA firmware development
- There is no real-world data available
- Lab-generated signals are not suitable for DSP-chain testing
- Field tests are short-term experiments and expensive to run
- The DSP algorithms are experimental and uncertain

Typical Development Challenges



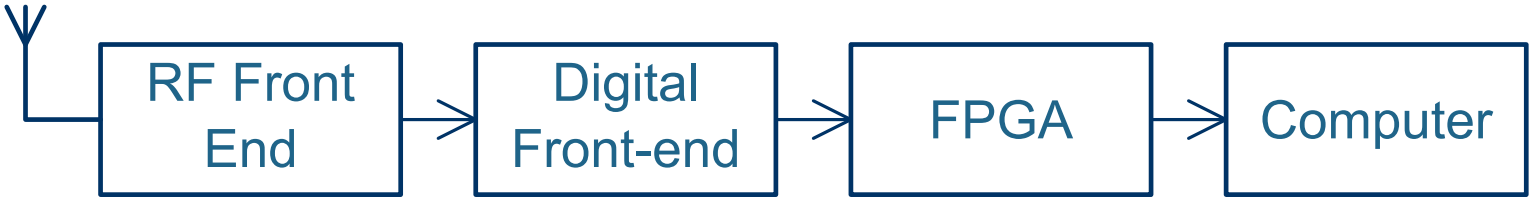
- The hardware (RF front-end and DFE) is not available at the time of FPGA firmware development
- There is no real-world data available
- Lab-generated signals are not suitable for DSP-chain testing
- Field tests are short-term experiments and expensive to run
- The DSP algorithms are experimental and uncertain

Typical Development Challenges



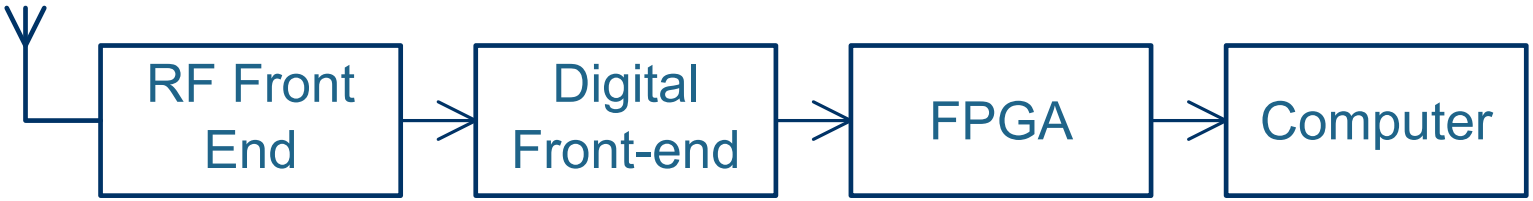
- The hardware (RF front-end and DFE) is not available at the time of FPGA firmware development
- There is no real-world data available
- Lab-generated signals are not suitable for DSP-chain testing
- Field tests are short-term experiments and expensive to run
- The DSP algorithms are experimental and uncertain

Typical Development Challenges



- The hardware (RF front-end and DFE) is not available at the time of FPGA firmware development
- There is no real-world data available
- Lab-generated signals are not suitable for DSP-chain testing
- Field tests are short-term experiments and expensive to run
- The DSP algorithms are experimental and uncertain

Typical Development Challenges



- The hardware (RF front-end and DFE) is not available at the time of FPGA firmware development
- There is no real-world data available
- Lab-generated signals are not suitable for DSP-chain testing
- Field tests are short-term experiments and expensive to run
- The DSP algorithms are experimental and uncertain

Early Development

- The system is simulated
- The simulated ADC data is injected into the FPGA
- The DSP-chain is developed on this simulated data
- At first, the computer-based DSP does most, if not all, of the processing (FPGA firmware consist only of supporting infrastructure)
- As the algorithms mature, the functionality is moved into the FPGA

Early Development

- The system is simulated
- The simulated ADC data is injected into the FPGA
- The DSP-chain is developed on this simulated data
- At first, the computer-based DSP does most, if not all, of the processing (FPGA firmware consist only of supporting infrastructure)
- As the algorithms mature, the functionality is moved into the FPGA

Early Development

- The system is simulated
- The simulated ADC data is injected into the FPGA
- The DSP-chain is developed on this simulated data
- At first, the computer-based DSP does most, if not all, of the processing (FPGA firmware consist only of supporting infrastructure)
- As the algorithms mature, the functionality is moved into the FPGA

Early Development

- The system is simulated
- The simulated ADC data is injected into the FPGA
- The DSP-chain is developed on this simulated data
- At first, the computer-based DSP does most, if not all, of the processing (FPGA firmware consist only of supporting infrastructure)
- As the algorithms mature, the functionality is moved into the FPGA

Early Development

- The system is simulated
- The simulated ADC data is injected into the FPGA
- The DSP-chain is developed on this simulated data
- At first, the computer-based DSP does most, if not all, of the processing (FPGA firmware consist only of supporting infrastructure)
- As the algorithms mature, the functionality is moved into the FPGA

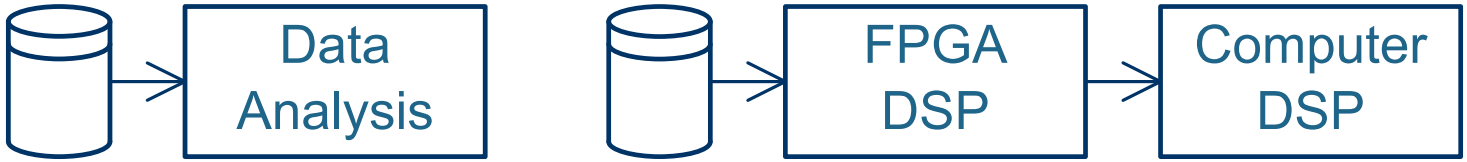
Late Development

- The front-end hardware is built, but must be tested
- An FPGA-based logger is implemented that saves raw ADC data from a field trial
- This recorded data is analysed in order to test the hardware and injected into the FPGA to test the DSP-chain
- This same recorded data can be used in simulation...

Late Development

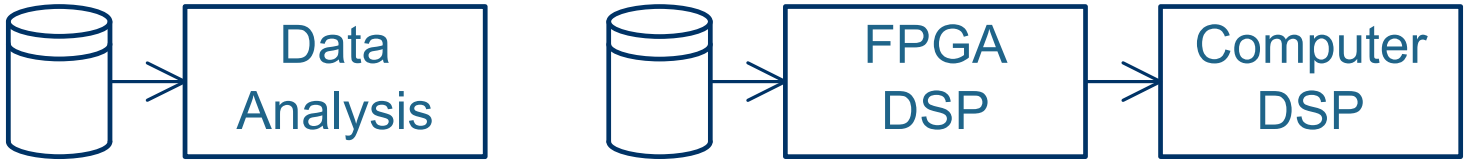
- The front-end hardware is built, but must be tested
- An FPGA-based logger is implemented that saves raw ADC data from a field trial
- This recorded data is analysed in order to test the hardware and injected into the FPGA to test the DSP-chain
- This same recorded data can be used in simulation...

Late Development



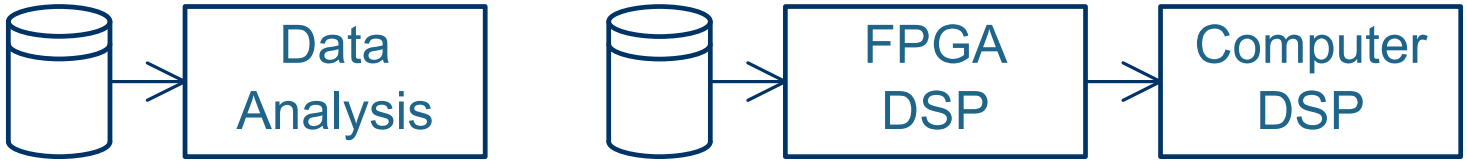
- The front-end hardware is built, but must be tested
- An FPGA-based logger is implemented that saves raw ADC data from a field trial
- This recorded data is analysed in order to test the hardware and injected into the FPGA to test the DSP-chain
- This same recorded data can be used in simulation...

Late Development



- The front-end hardware is built, but must be tested
- An FPGA-based logger is implemented that saves raw ADC data from a field trial
- This recorded data is analysed in order to test the hardware and injected into the FPGA to test the DSP-chain
- This same recorded data can be used in simulation...

Late Development



- The front-end hardware is built, but must be tested
- An FPGA-based logger is implemented that saves raw ADC data from a field trial
- This recorded data is analysed in order to test the hardware and injected into the FPGA to test the DSP-chain
- This same recorded data can be used in simulation...

Outline

Altera Library

Verilog Processes

Finite State Machines

Timing Constraints

Injection Testing

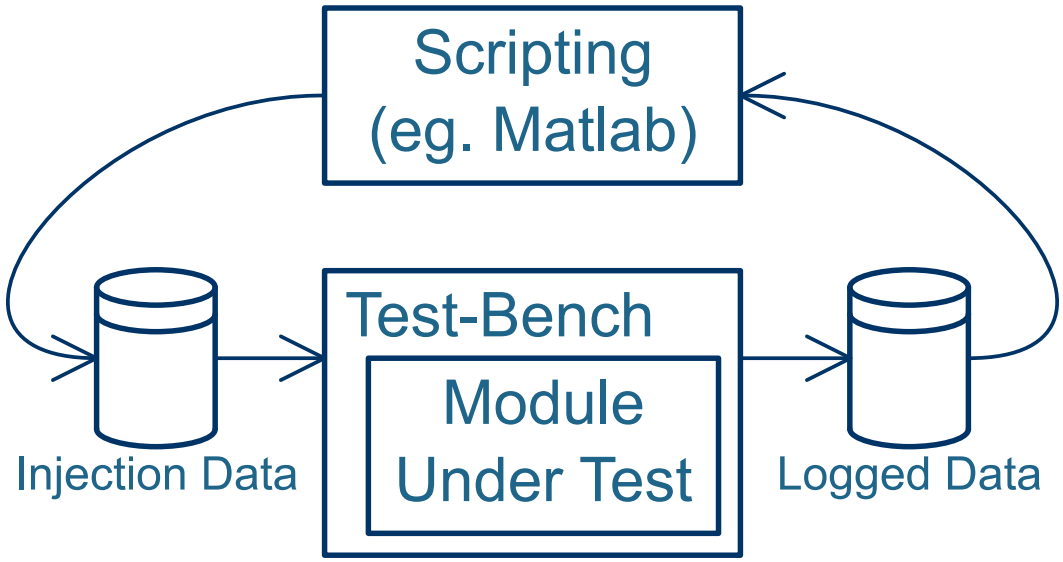
Simulation

Practical

Appendix – High Resolution PWM

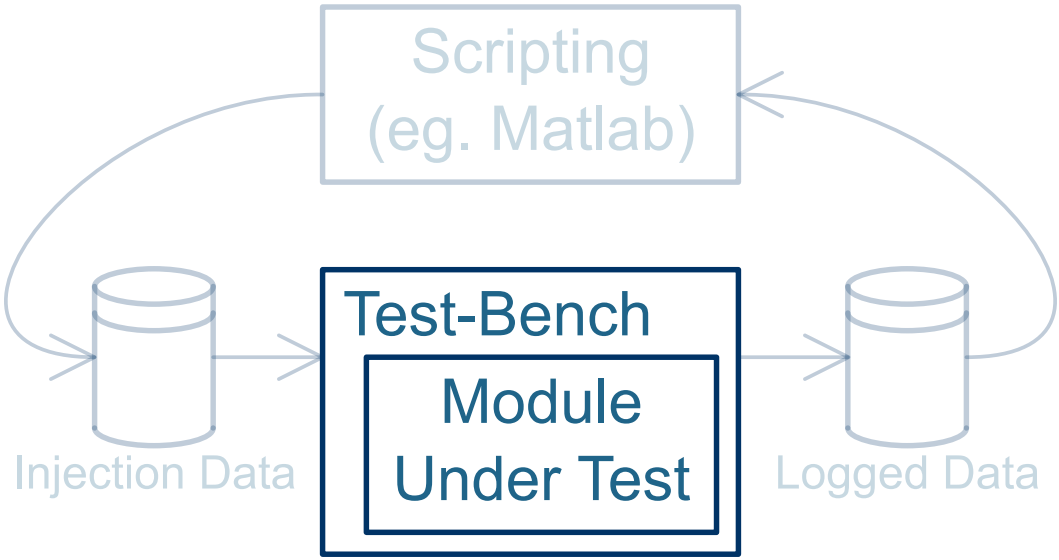
THE RADAR MASTERS COURSE

Simulation Basics



Today we only focus on the test-bench and simulation tool

Simulation Basics



Today we only focus on the test-bench and simulation tool

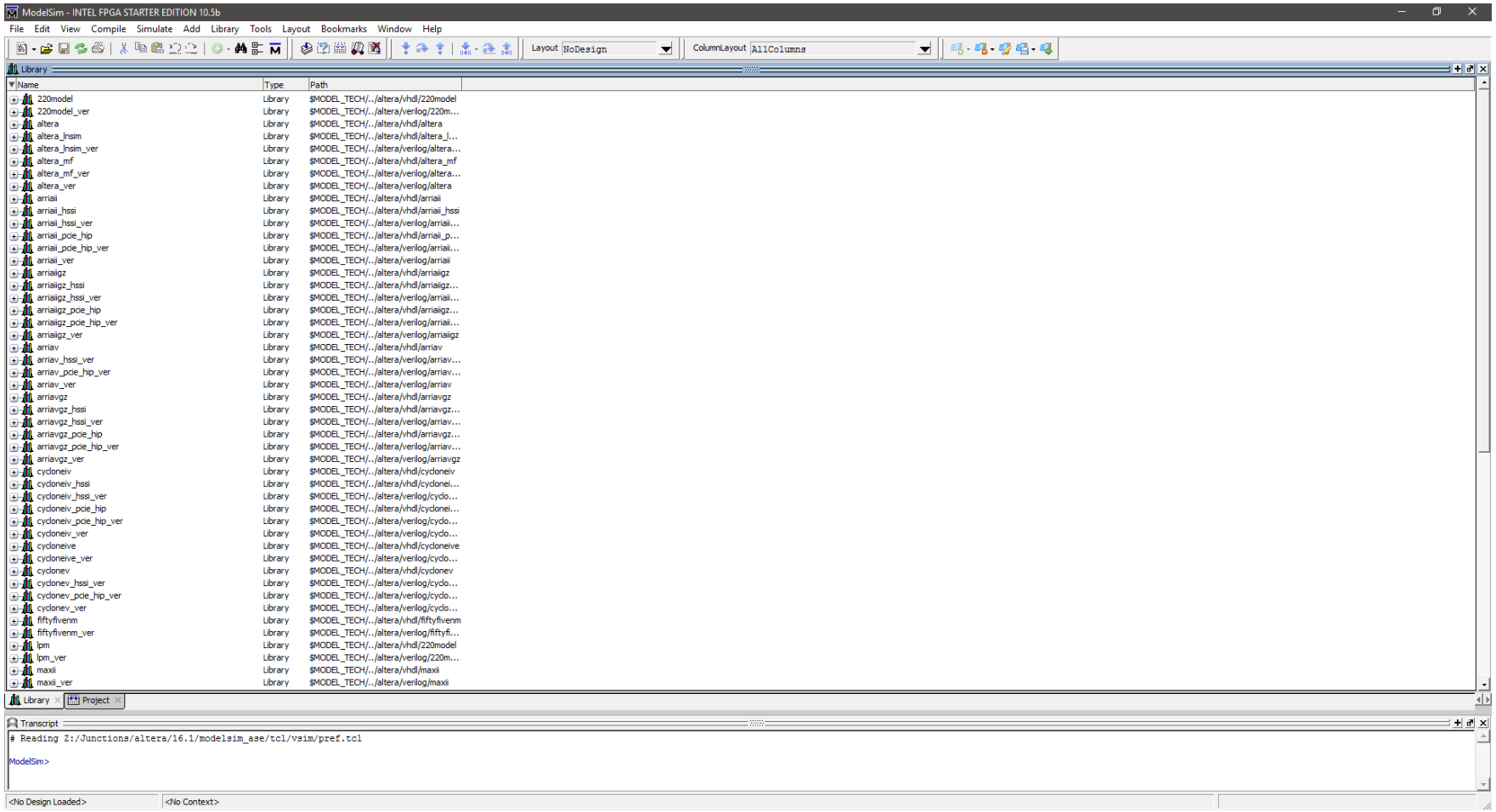
Test Bench Basics


```
`timescale 1ns/1ps
module ADXL345_TB;

// Clock
reg Clk_100M = 0;
always #5 Clk_100M <= ~Clk_100M;

// Reset
reg Reset = 1;
initial #20 Reset <= 0;
```

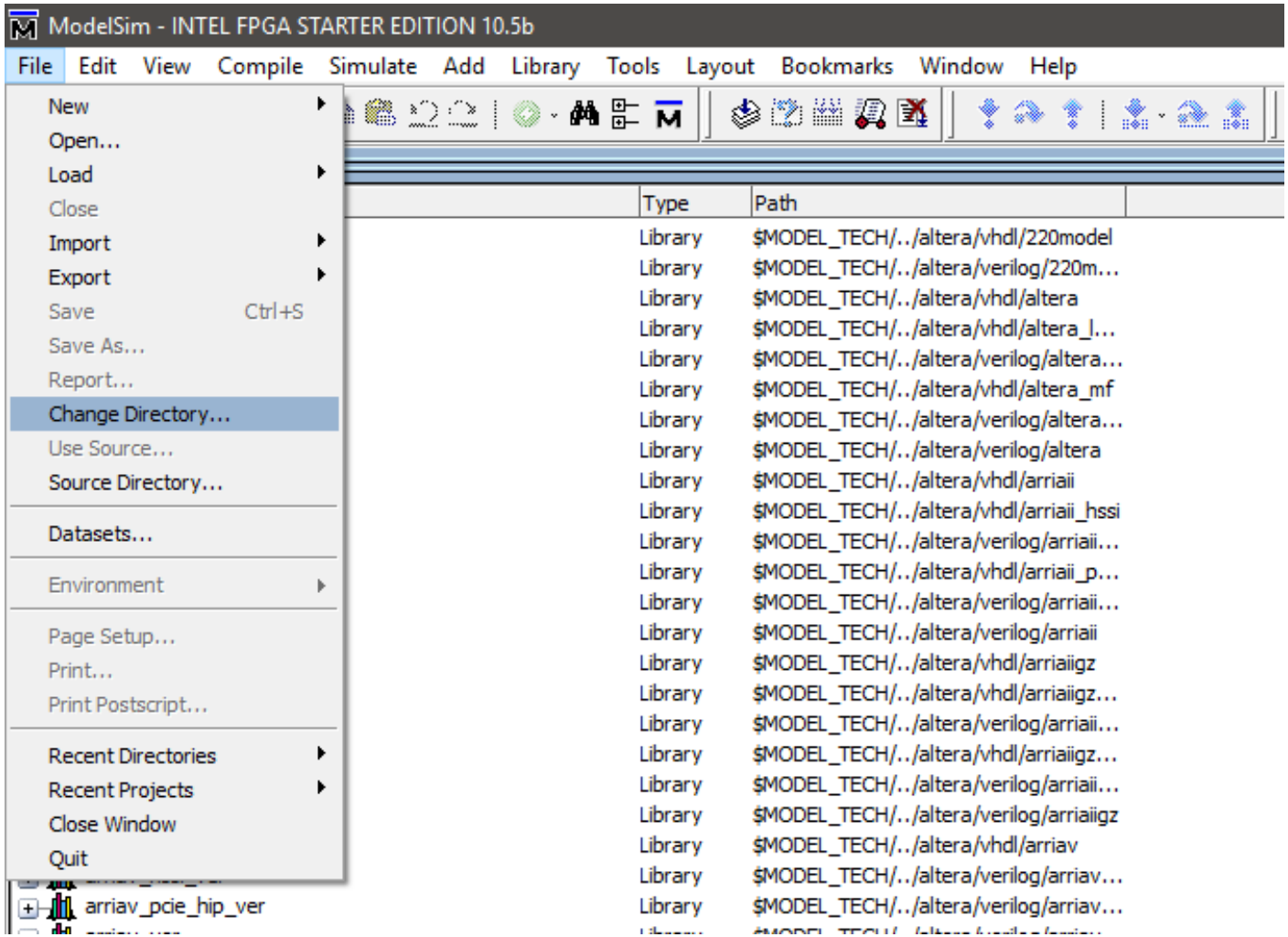
Test Bench Basics


```
// DUT
wire [15:0]X, Y, Z;
wire nCS, SClk, SDI;
reg  SDO = 0;

ADXL345 #(10) Accelerometer( // Set parameter for 100 MHz
 Clk_100M, Reset,
 X, Y, Z,
 nCS, SClk, SDI, SDO
);

endmodule
```

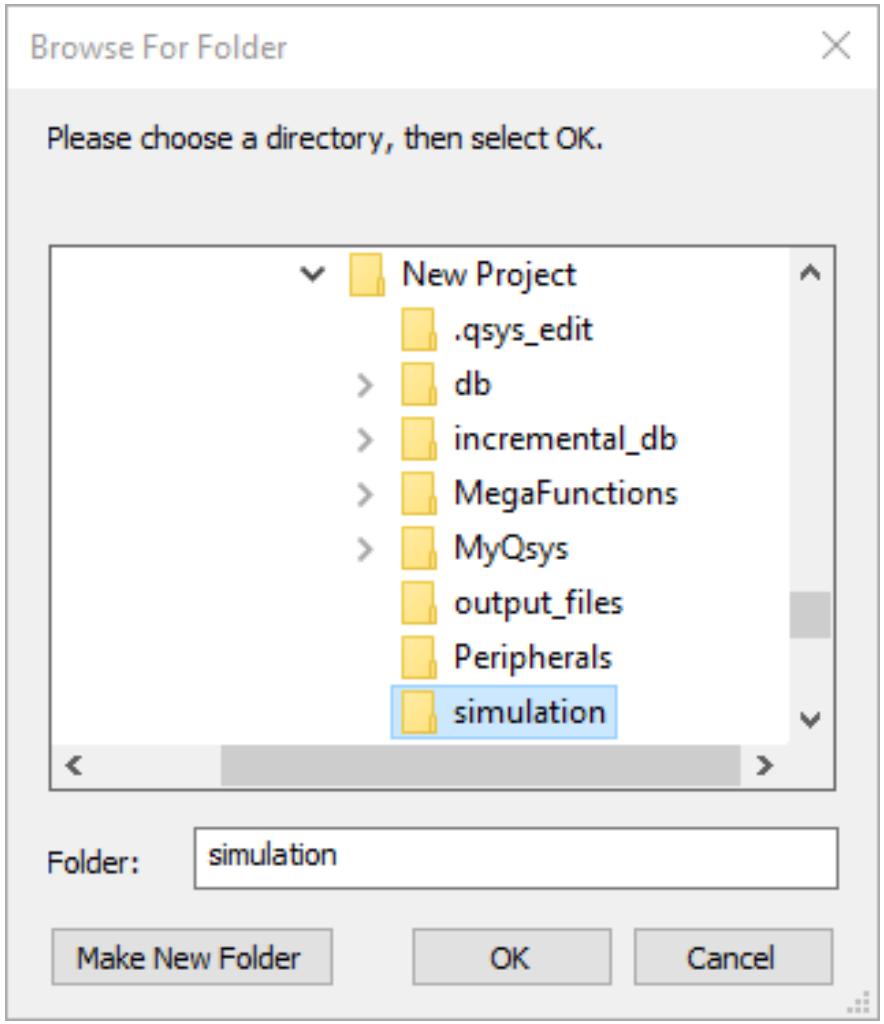
Modelsim


ModelSim - INTEL FPGA STARTER EDITION 10.5b
File Edit View Compile Simulate Add Library Tools Layout Bookmarks Window Help
Layout NoDesign
ColumnLayout AllColumns
Library
Name Type Path
220model Library $MODEL_TECH/../altera/vhdl/220model
220model_ver Library $MODEL_TECH/../altera/verilog/220m...
altera Library $MODEL_TECH/../altera/vhdl/altera
altera_lnsim Library $MODEL_TECH/../altera/vhdl/altera_l...
altera_lnsim_ver Library $MODEL_TECH/../altera/verilog/altera...
altera_mf Library $MODEL_TECH/../altera/vhdl/altera_mf
altera_mf_ver Library $MODEL_TECH/../altera/verilog/altera...
altera_ver Library $MODEL_TECH/../altera/verilog/altera
arriaii Library $MODEL_TECH/../altera/vhdl/arriaii
arriaii_hssi Library $MODEL_TECH/../altera/vhdl/arriaii_hssi
arriaii_hssi_ver Library $MODEL_TECH/../altera/verilog/arriaii...
arriaii_pcie_hip Library $MODEL_TECH/../altera/vhdl/arriaii_p...
arriaii_pcie_hip_ver Library $MODEL_TECH/../altera/verilog/arriaii...
arriaii_ver Library $MODEL_TECH/../altera/verilog/arriaii
arriaiigz Library $MODEL_TECH/../altera/vhdl/arriaiigz
arriaiigz_hssi Library $MODEL_TECH/../altera/vhdl/arriaiigz...
arriaiigz_hssi_ver Library $MODEL_TECH/../altera/verilog/arriaii...
arriaiigz_pcie_hip Library $MODEL_TECH/../altera/vhdl/arriaiigz...
arriaiigz_pcie_hip_ver Library $MODEL_TECH/../altera/verilog/arriaii...
arriaiigz_ver Library $MODEL_TECH/../altera/verilog/arriaiigz
arriav Library $MODEL_TECH/../altera/vhdl/arriav
arriav_hssi_ver Library $MODEL_TECH/../altera/verilog/arriav...
arriav_pcie_hip_ver Library $MODEL_TECH/../altera/verilog/arriav...
arriav_ver Library $MODEL_TECH/../altera/verilog/arriav
arriavgz Library $MODEL_TECH/../altera/vhdl/arriavgz
arriavgz_hssi Library $MODEL_TECH/../altera/vhdl/arriavgz...
arriavgz_hssi_ver Library $MODEL_TECH/../altera/verilog/arriav...
arriavgz_pcie_hip Library $MODEL_TECH/../altera/vhdl/arriavgz...
arriavgz_pcie_hip_ver Library $MODEL_TECH/../altera/verilog/arriav...
arriavgz_ver Library $MODEL_TECH/../altera/verilog/arriavgz
cycloneiv Library $MODEL_TECH/../altera/vhdl/cycloneiv
cycloneiv_hssi Library $MODEL_TECH/../altera/vhdl/cycloneiv...
cycloneiv_hssi_ver Library $MODEL_TECH/../altera/verilog/cyclo...
cycloneiv_pcie_hip Library $MODEL_TECH/../altera/vhdl/cycloneiv...
cycloneiv_pcie_hip_ver Library $MODEL_TECH/../altera/verilog/cyclo...
cycloneiv_ver Library $MODEL_TECH/../altera/verilog/cyclo...
cycloneive Library $MODEL_TECH/../altera/vhdl/cycloneive
cycloneive_ver Library $MODEL_TECH/../altera/verilog/cyclo...
cyclonev Library $MODEL_TECH/../altera/vhdl/cyclonev
cyclonev_hssi_ver Library $MODEL_TECH/../altera/verilog/cyclo...
cyclonev_pcie_hip_ver Library $MODEL_TECH/../altera/verilog/cyclo...
cyclonev_ver Library $MODEL_TECH/../altera/verilog/cyclo...
fiftyfivenm Library $MODEL_TECH/../altera/vhdl/fiftyfivenm
fiftyfivenm_ver Library $MODEL_TECH/../altera/verilog/fiftyfi...
lpm Library $MODEL_TECH/../altera/vhdl/220model
lpm_ver Library $MODEL_TECH/../altera/verilog/220m...
maxii Library $MODEL_TECH/../altera/vhdl/maxii
maxii_ver Library $MODEL_TECH/../altera/verilog/maxii
Library
Project
Transcript
Reading Z:/Junctions/altera/16.1/modelsim_ase/tcl/vsim/pref.tcl
ModelSim>
<No Design Loaded>
<No Context>

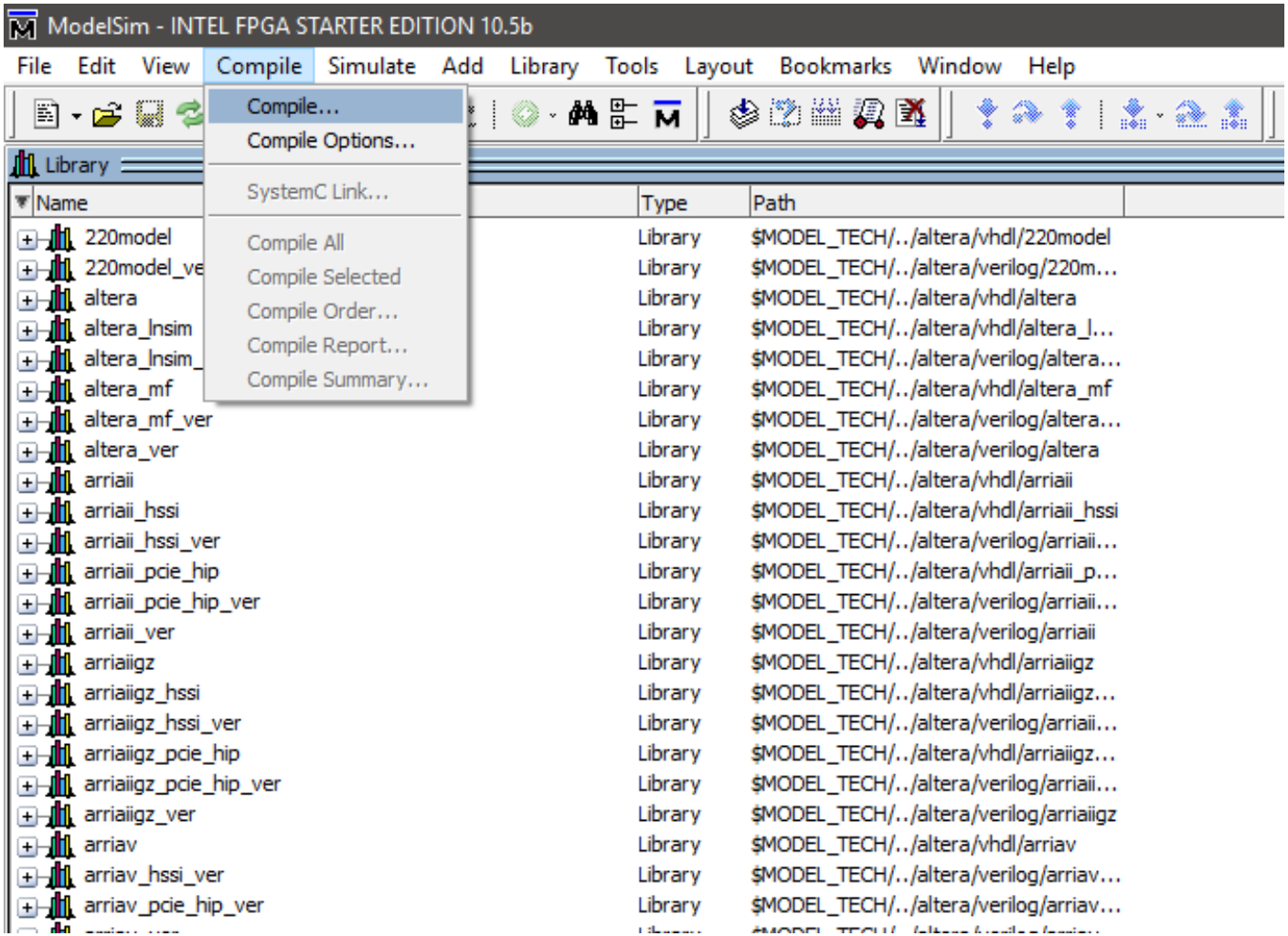
Change Directory...



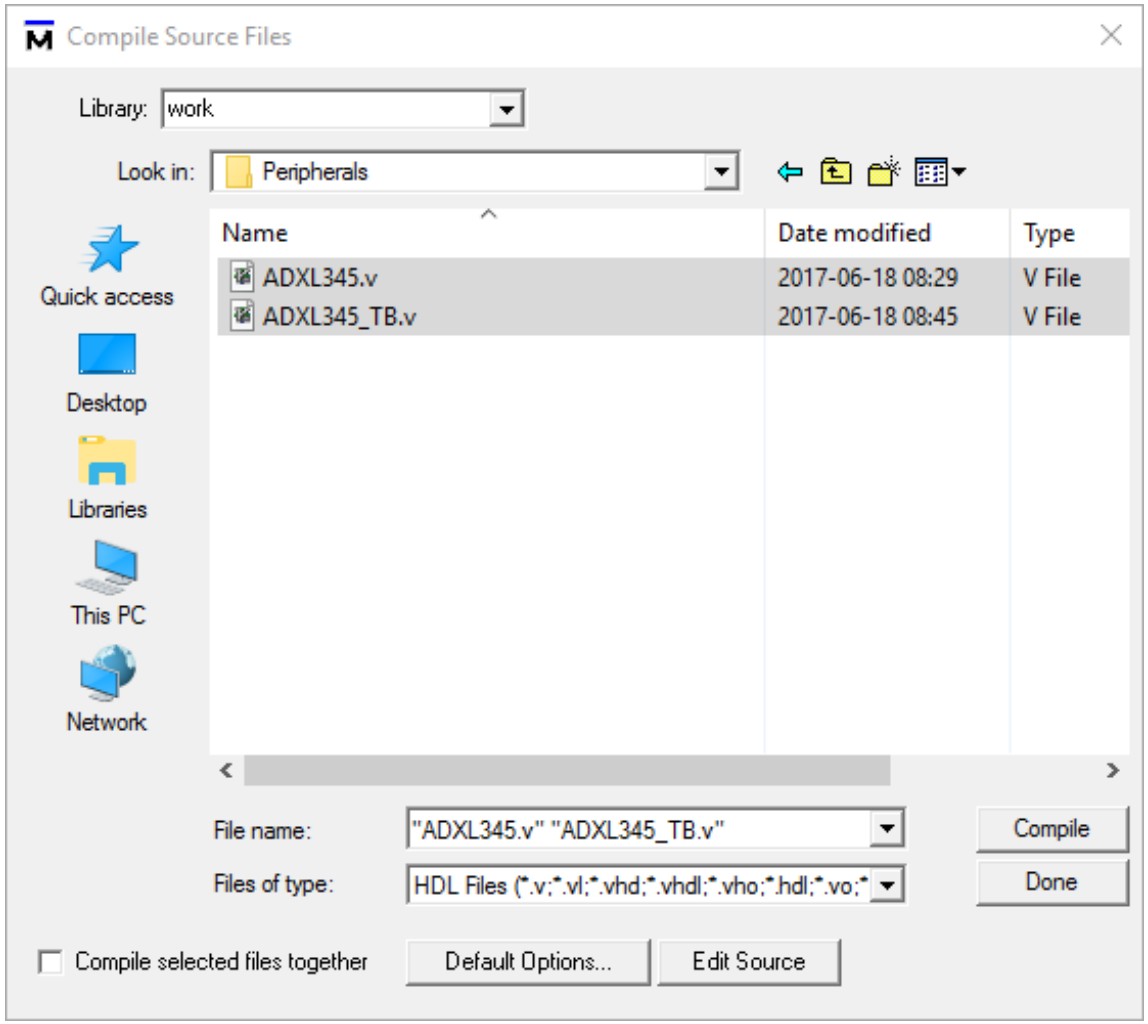
to the Local Project



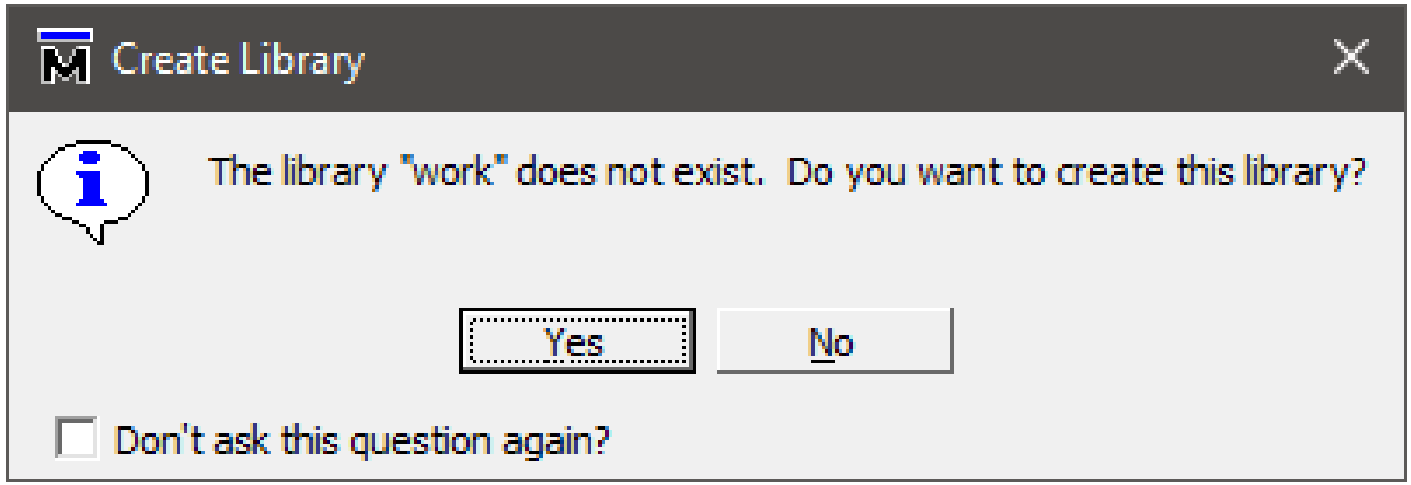
Compile


ModelSim - INTEL FPGA STARTER EDITION 10.5b
File Edit View Compile Simulate Add Library Tools Layout Bookmarks Window Help
Compile...
Compile Options...
SystemC Link...
Compile All
Compile Selected
Compile Order...
Compile Report...
Compile Summary...
Library
Name Type Path
220model Library $MODEL_TECH/../altera/vhdl/220model
220model_ve Library $MODEL_TECH/../altera/verilog/220m...
altera Library $MODEL_TECH/../altera/vhdl/altera
altera_lnsim Library $MODEL_TECH/../altera/vhdl/altera_l...
altera_lnsim_ Library $MODEL_TECH/../altera/verilog/altera...
altera_mf Library $MODEL_TECH/../altera/vhdl/altera_mf
altera_mf_ver Library $MODEL_TECH/../altera/verilog/altera...
altera_ver Library $MODEL_TECH/../altera/verilog/altera
arriaii Library $MODEL_TECH/../altera/vhdl/arriaii
arriaii_hssi Library $MODEL_TECH/../altera/vhdl/arriaii_hssi
arriaii_hssi_ver Library $MODEL_TECH/../altera/verilog/arriaii...
arriaii_pcie_hip Library $MODEL_TECH/../altera/vhdl/arriaii_p...
arriaii_pcie_hip_ver Library $MODEL_TECH/../altera/verilog/arriaii...
arriaii_ver Library $MODEL_TECH/../altera/verilog/arriaii
arriaiigz Library $MODEL_TECH/../altera/vhdl/arriaiigz
arriaiigz_hssi Library $MODEL_TECH/../altera/vhdl/arriaiigz...
arriaiigz_hssi_ver Library $MODEL_TECH/../altera/verilog/arriaii...
arriaiigz_pcie_hip Library $MODEL_TECH/../altera/vhdl/arriaiigz...
arriaiigz_pcie_hip_ver Library $MODEL_TECH/../altera/verilog/arriaii...
arriaiigz_ver Library $MODEL_TECH/../altera/verilog/arriaiigz
arriav Library $MODEL_TECH/../altera/vhdl/arriav
arriav_hssi_ver Library $MODEL_TECH/../altera/verilog/arriav...
arriav_pcie_hip_ver Library $MODEL_TECH/../altera/verilog/arriav...

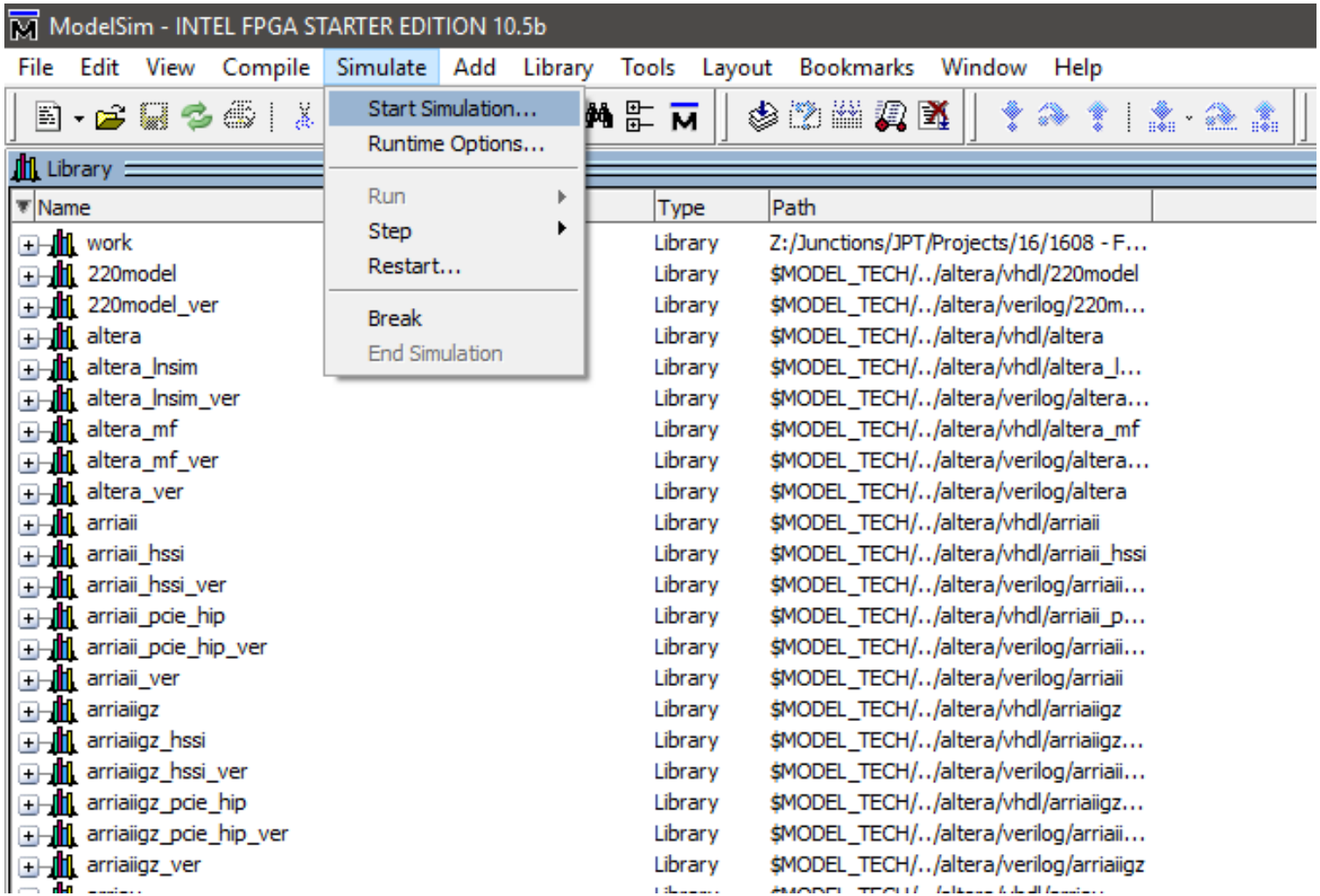
Select Files



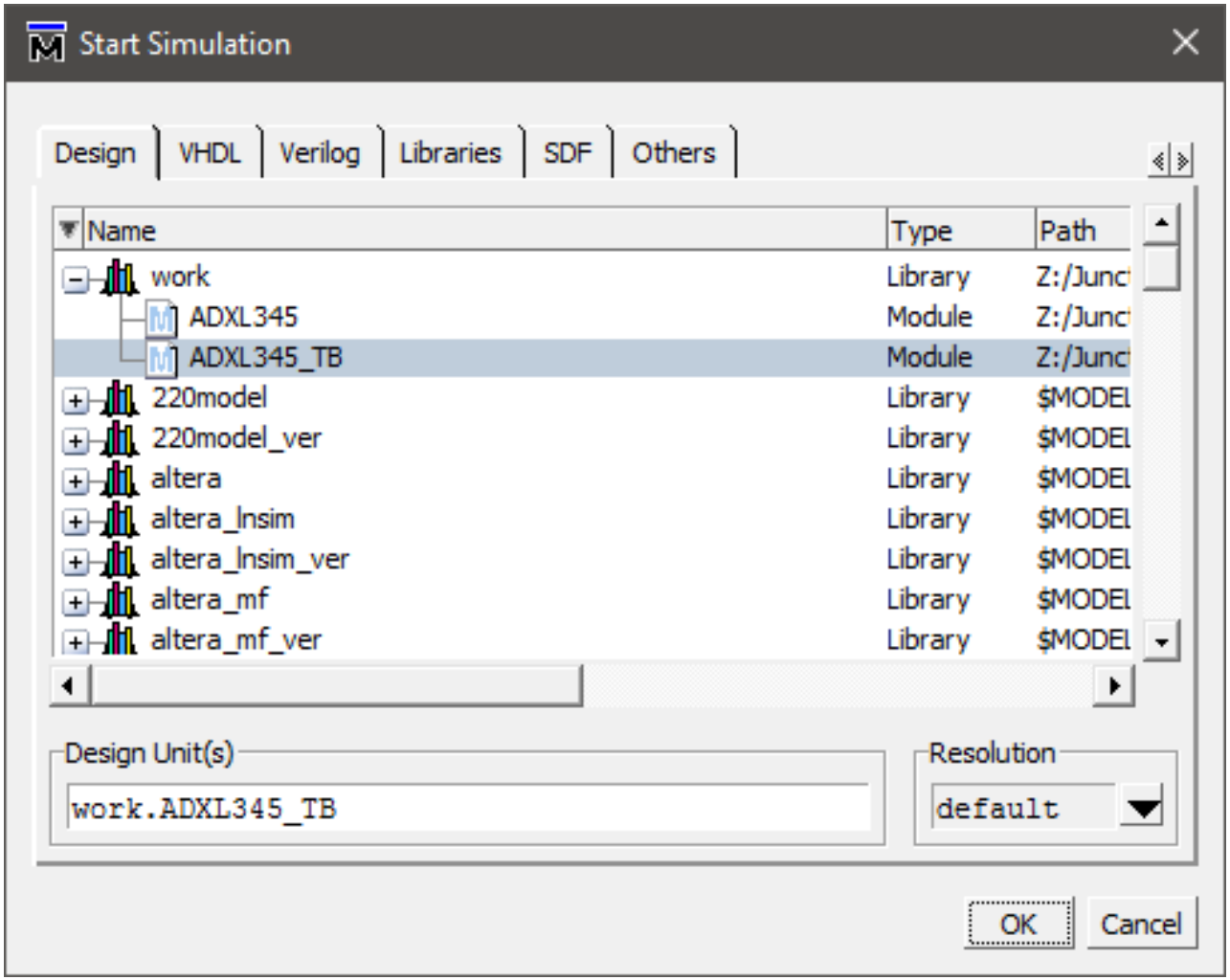
Create Work Library



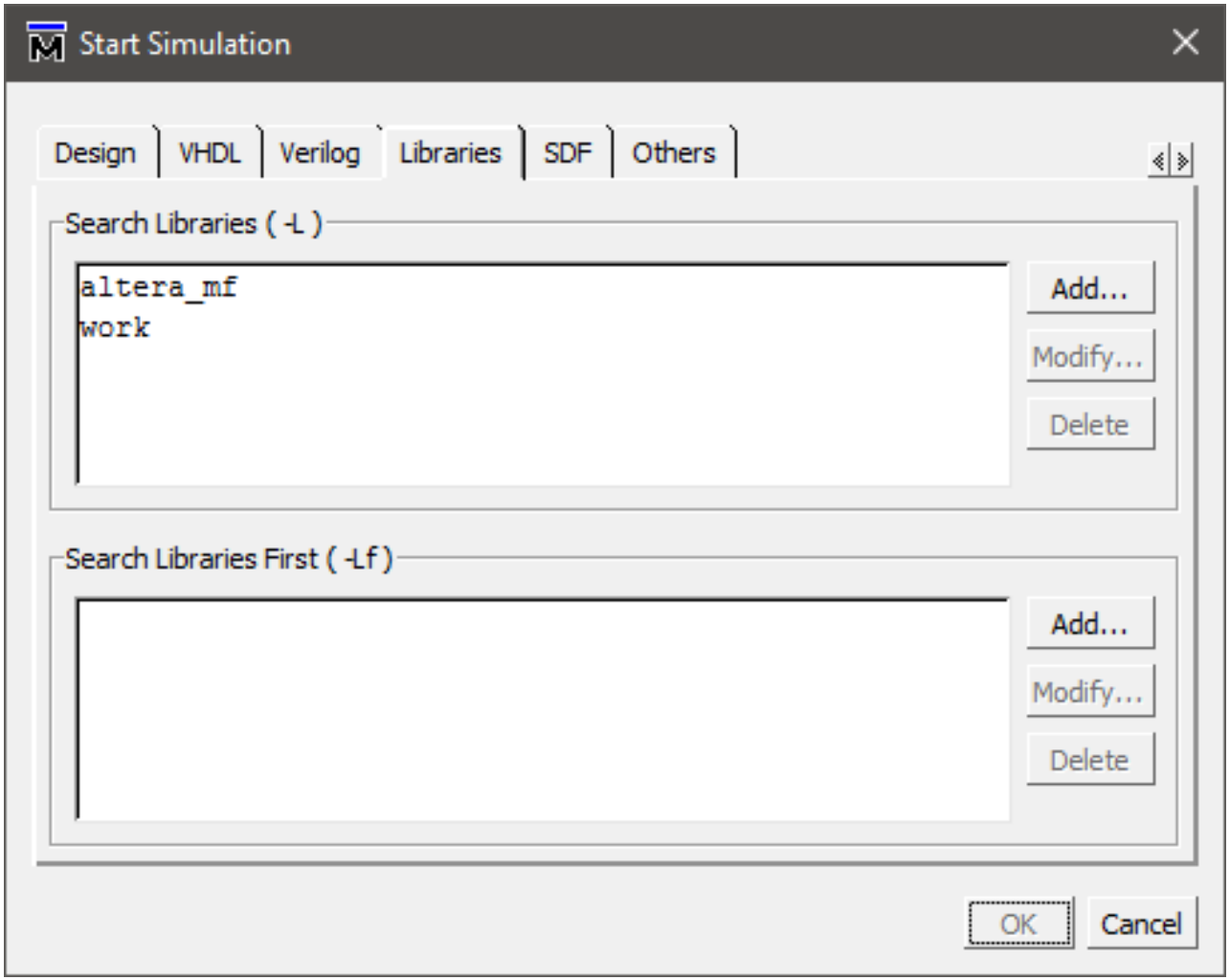
Start Simulation



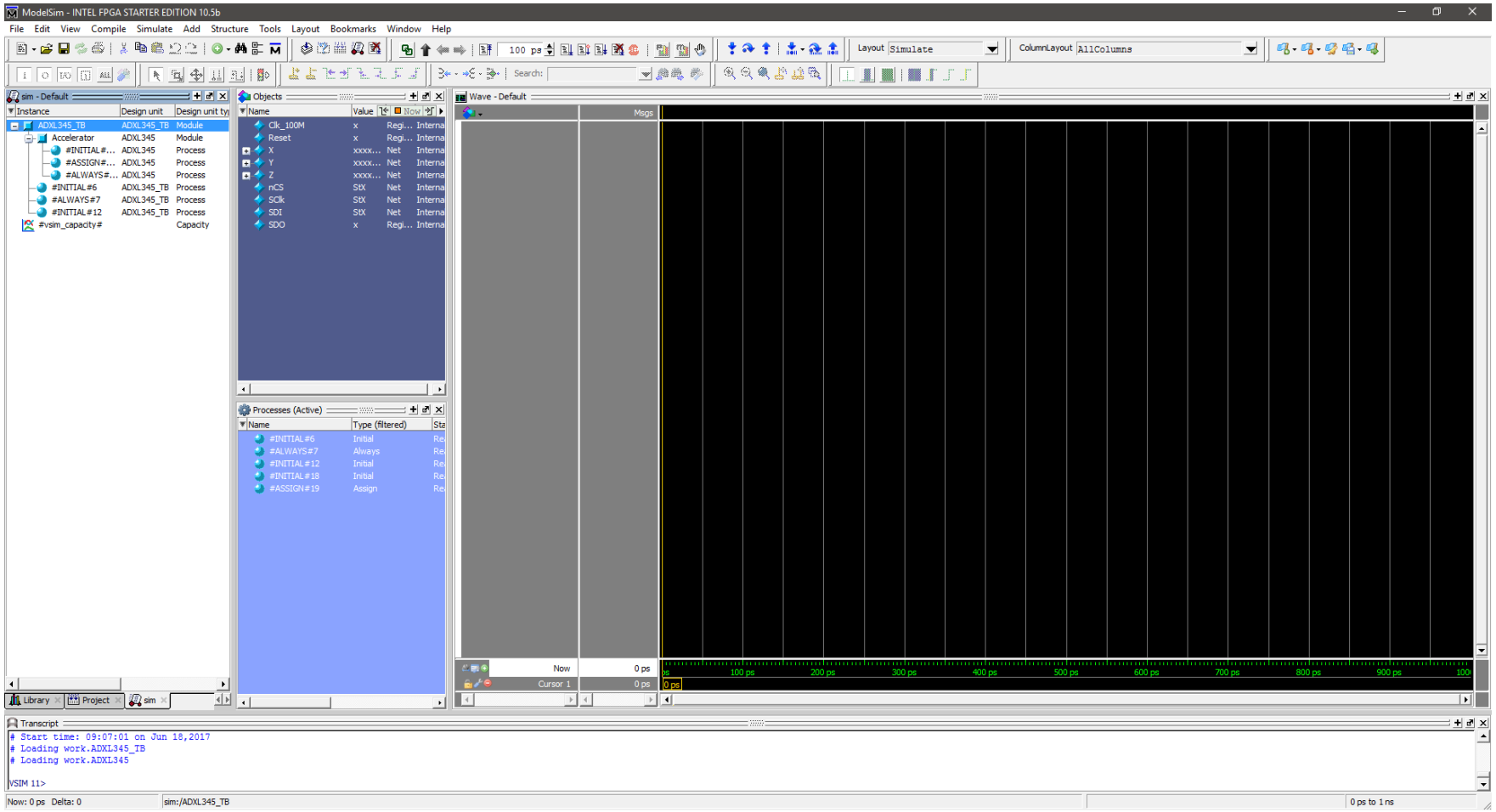
Choose the Test Bench



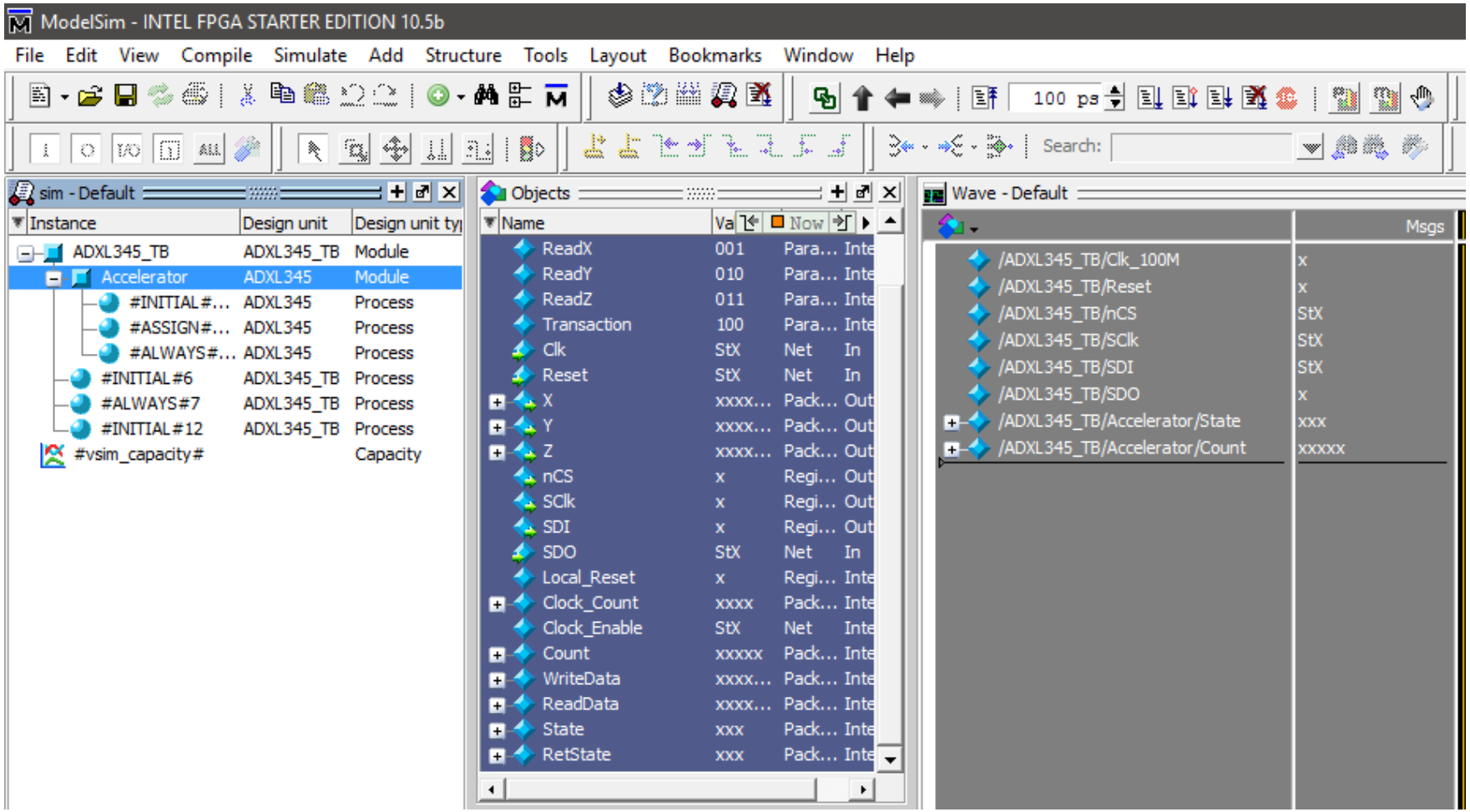
Add Libraries



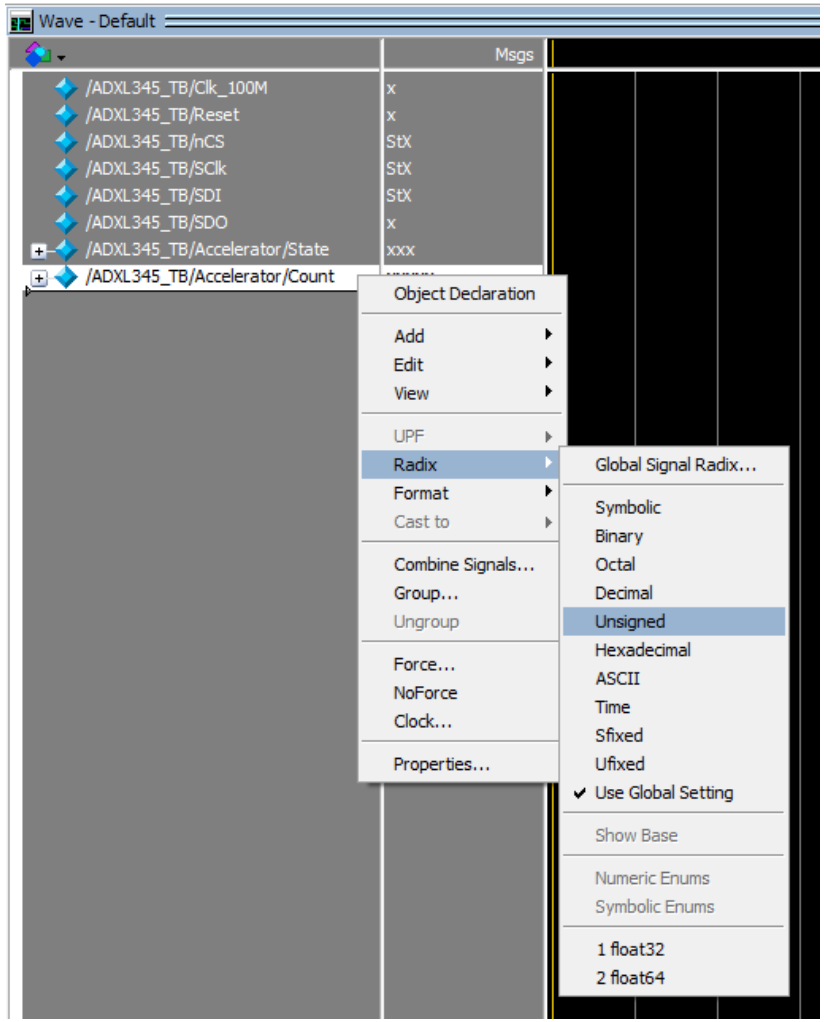
Simulation View



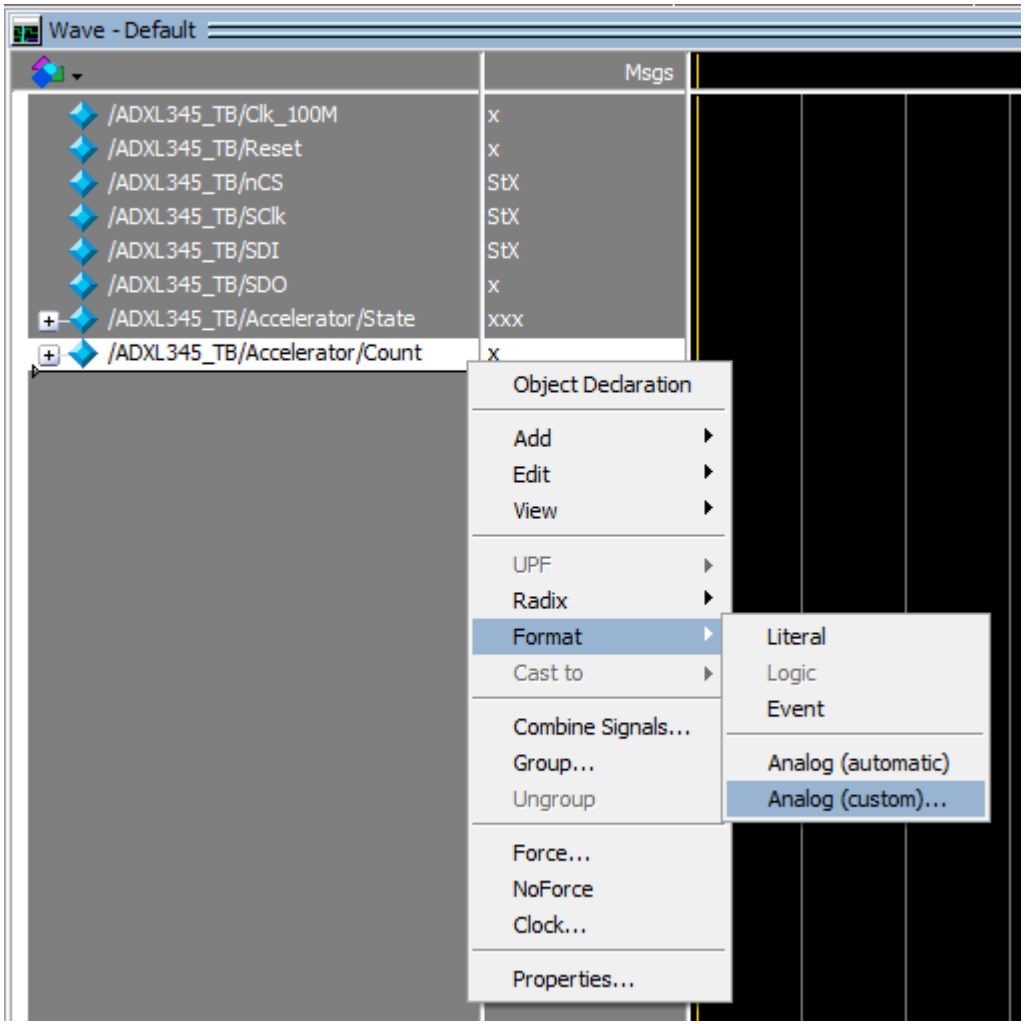
Drag-and-Drop Signals



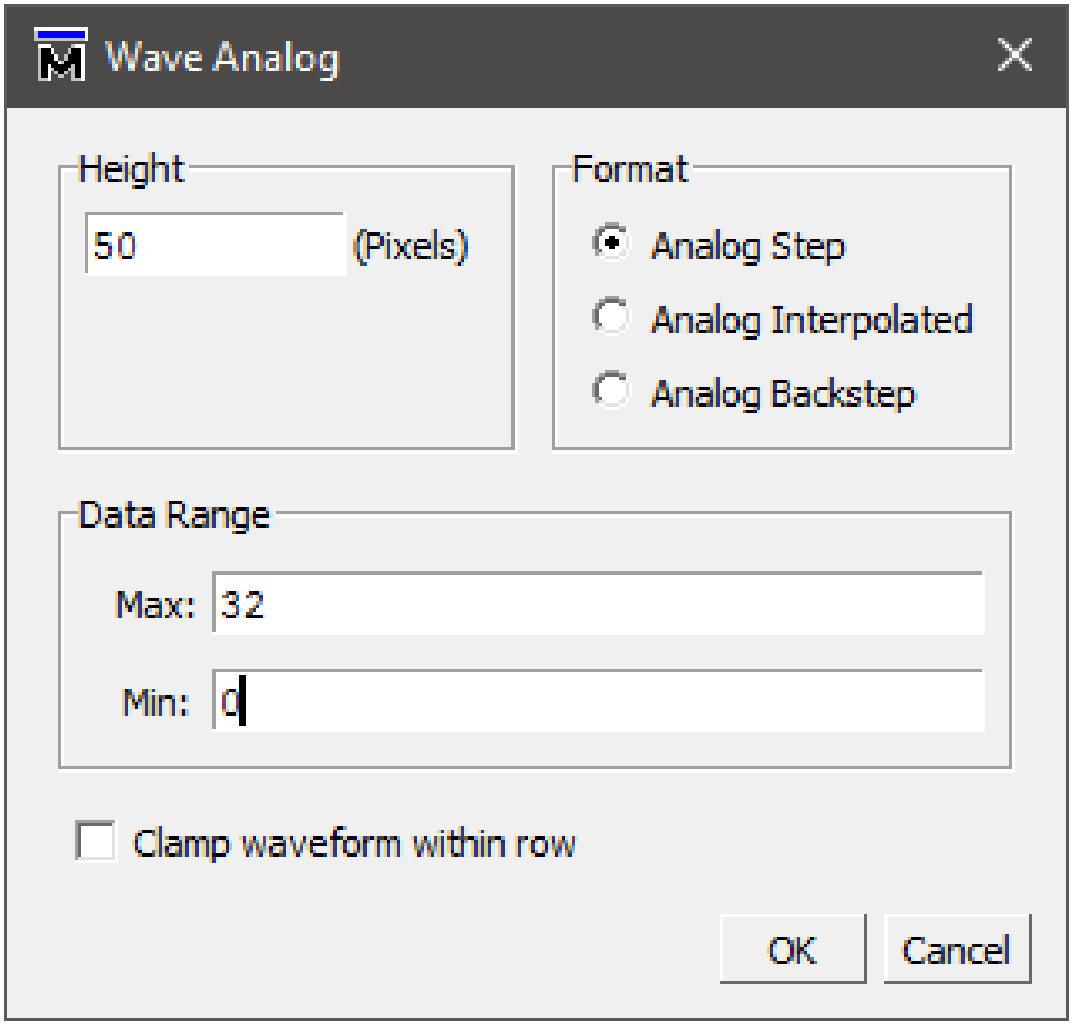
Set the Radix



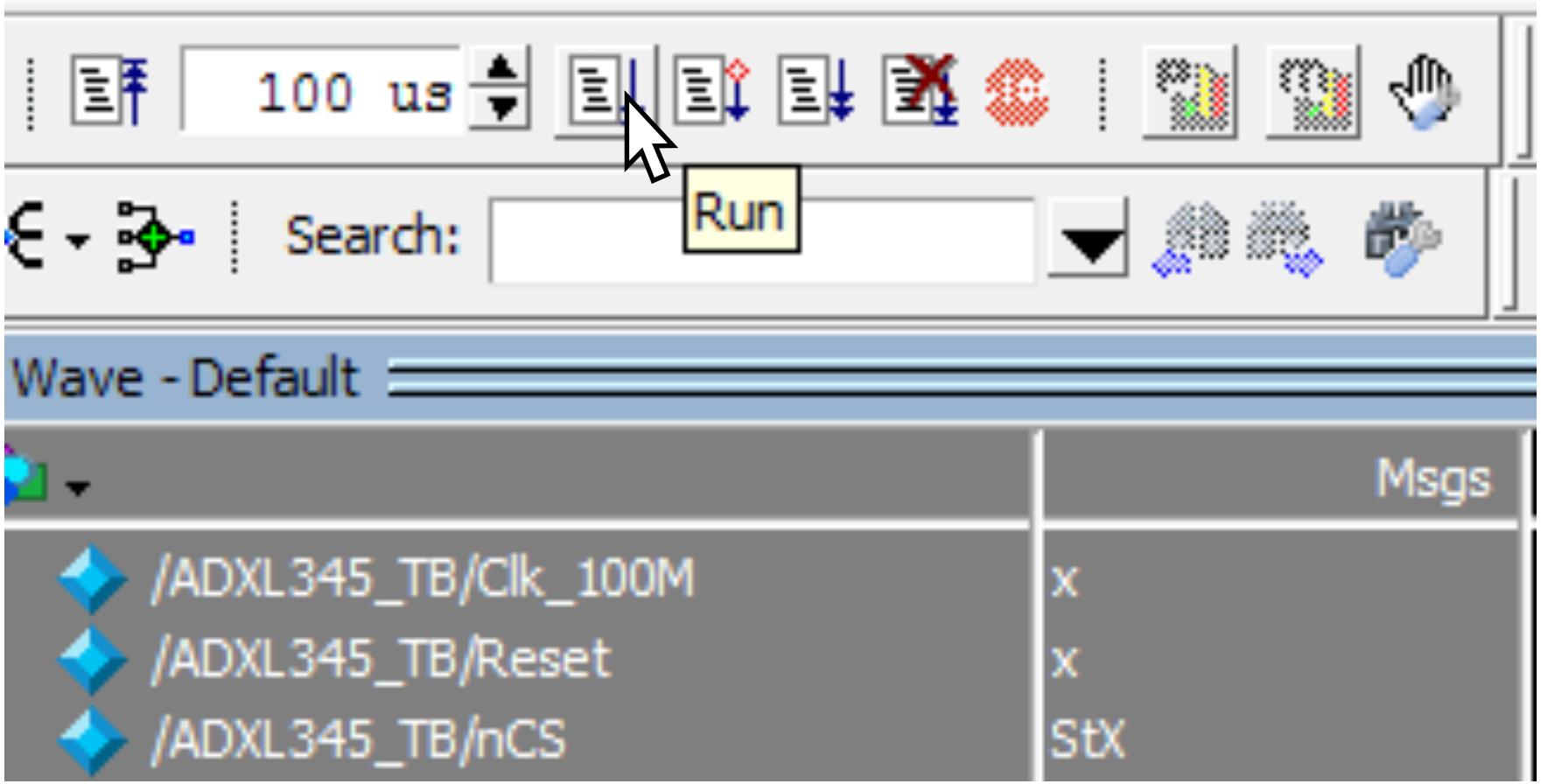
Set the Format



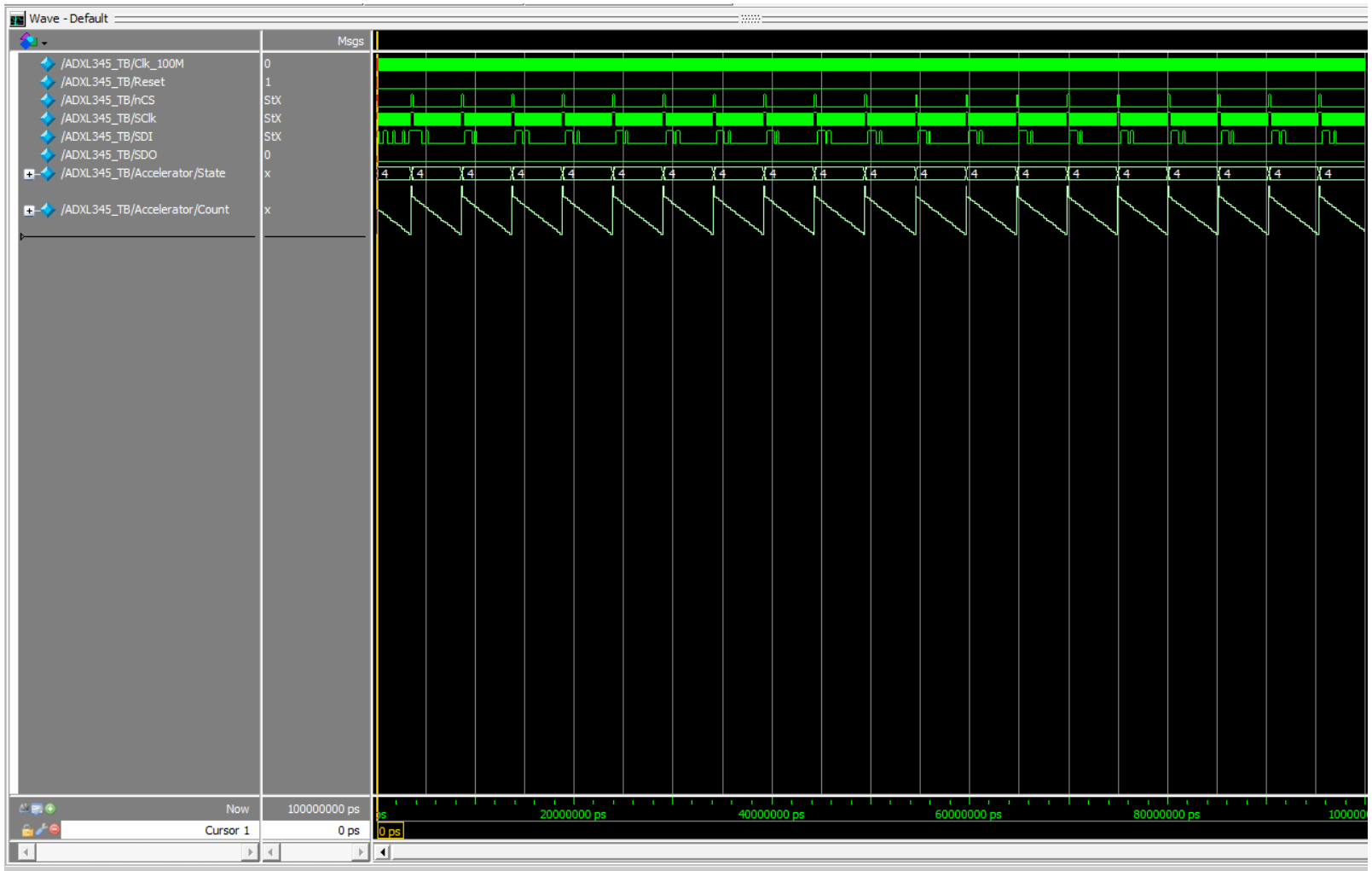
Set the Format



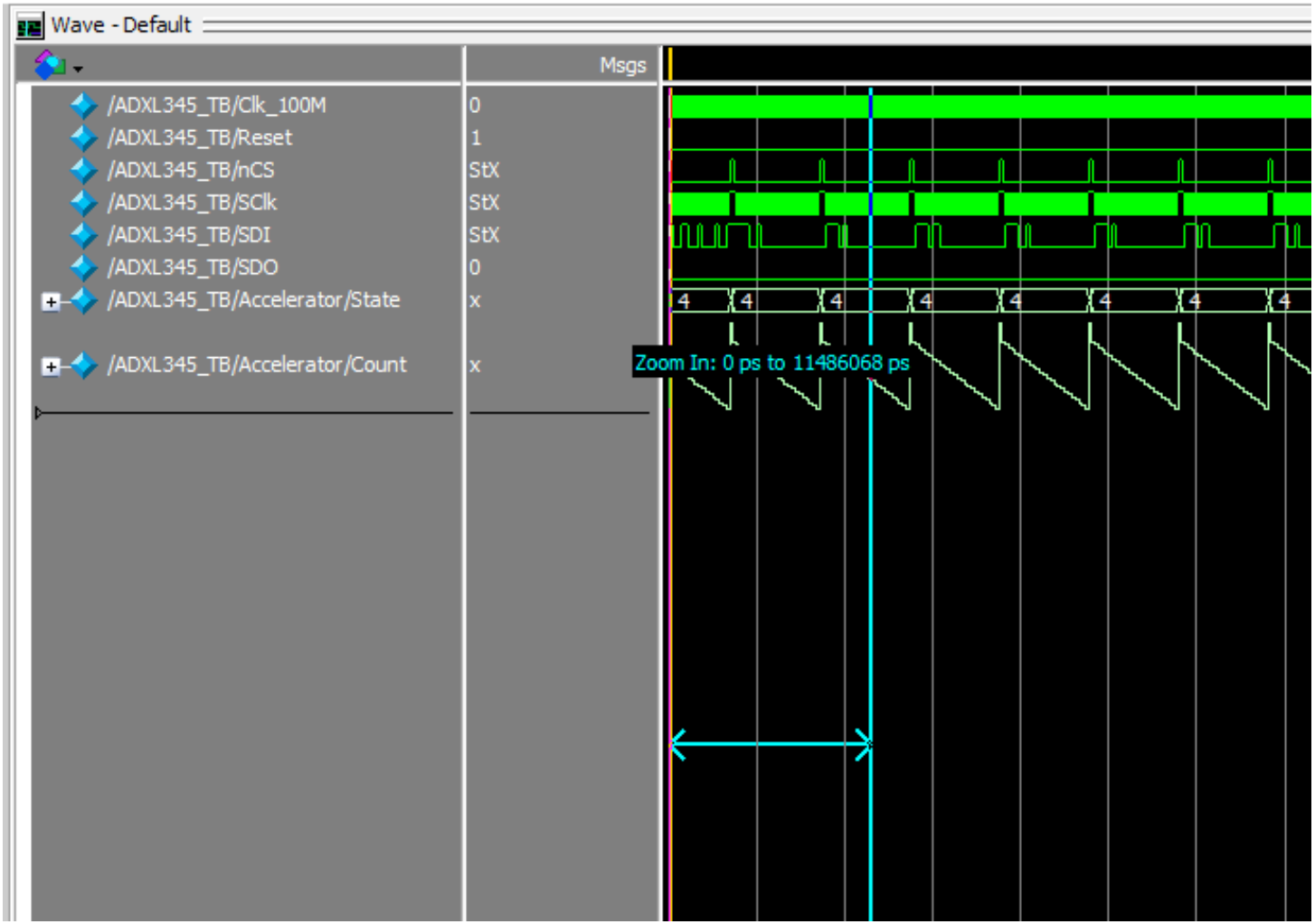
Run the Simulation



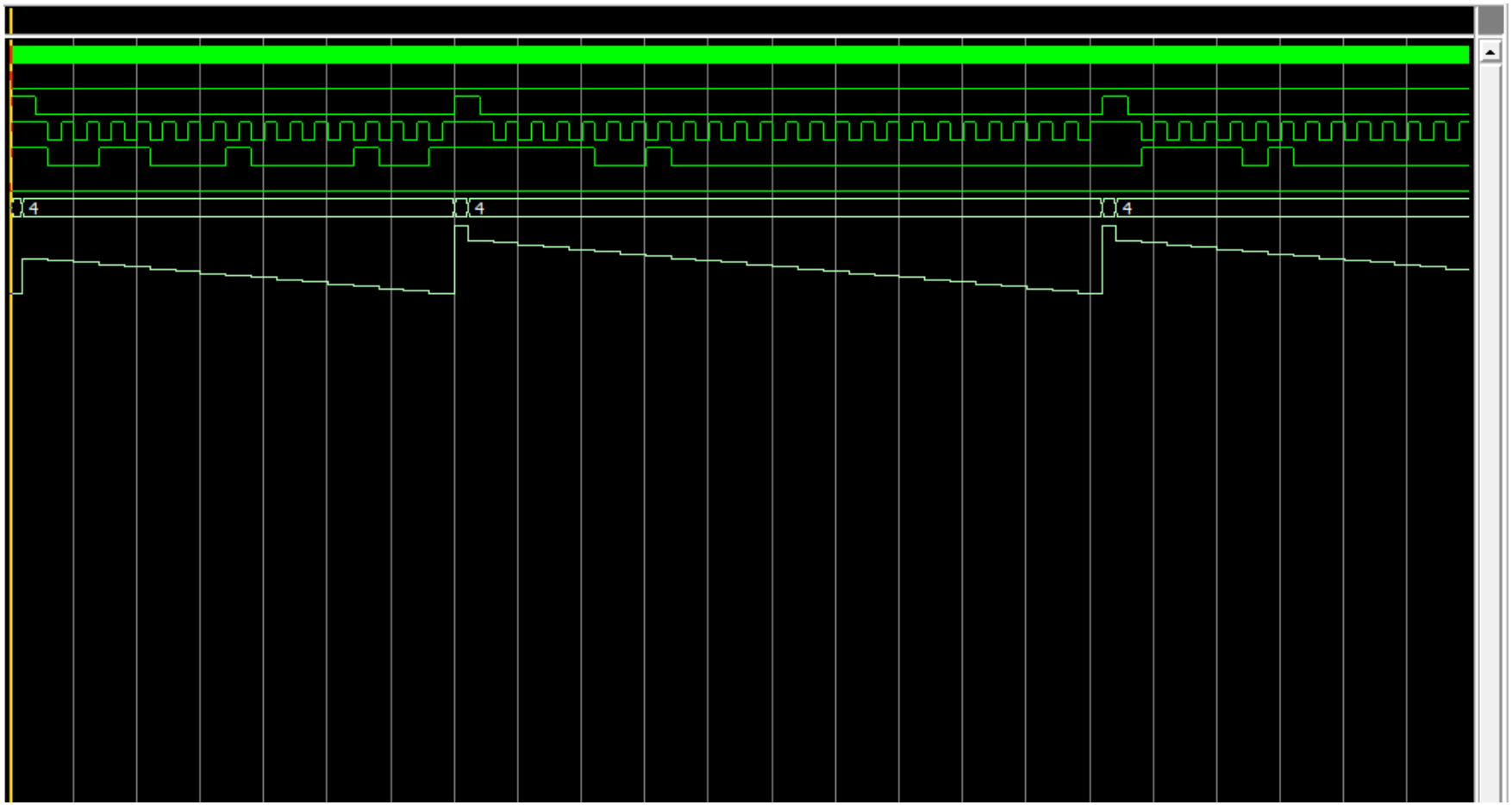
Zoom to Fit


Wave - Default
Msgs
/ADXL345_TB/Clk_100M
/ADXL345_TB/Reset
/ADXL345_TB/nCS
/ADXL345_TB/SClk
/ADXL345_TB/SDI
/ADXL345_TB/SDO
/ADXL345_TB/Accelerator/State
/ADXL345_TB/Accelerator/Count
0
1
StX
StX
StX
0
x
x
Now
100000000 ps
Cursor 1
0 ps
20000000 ps
40000000 ps
60000000 ps
80000000 ps
0 ps

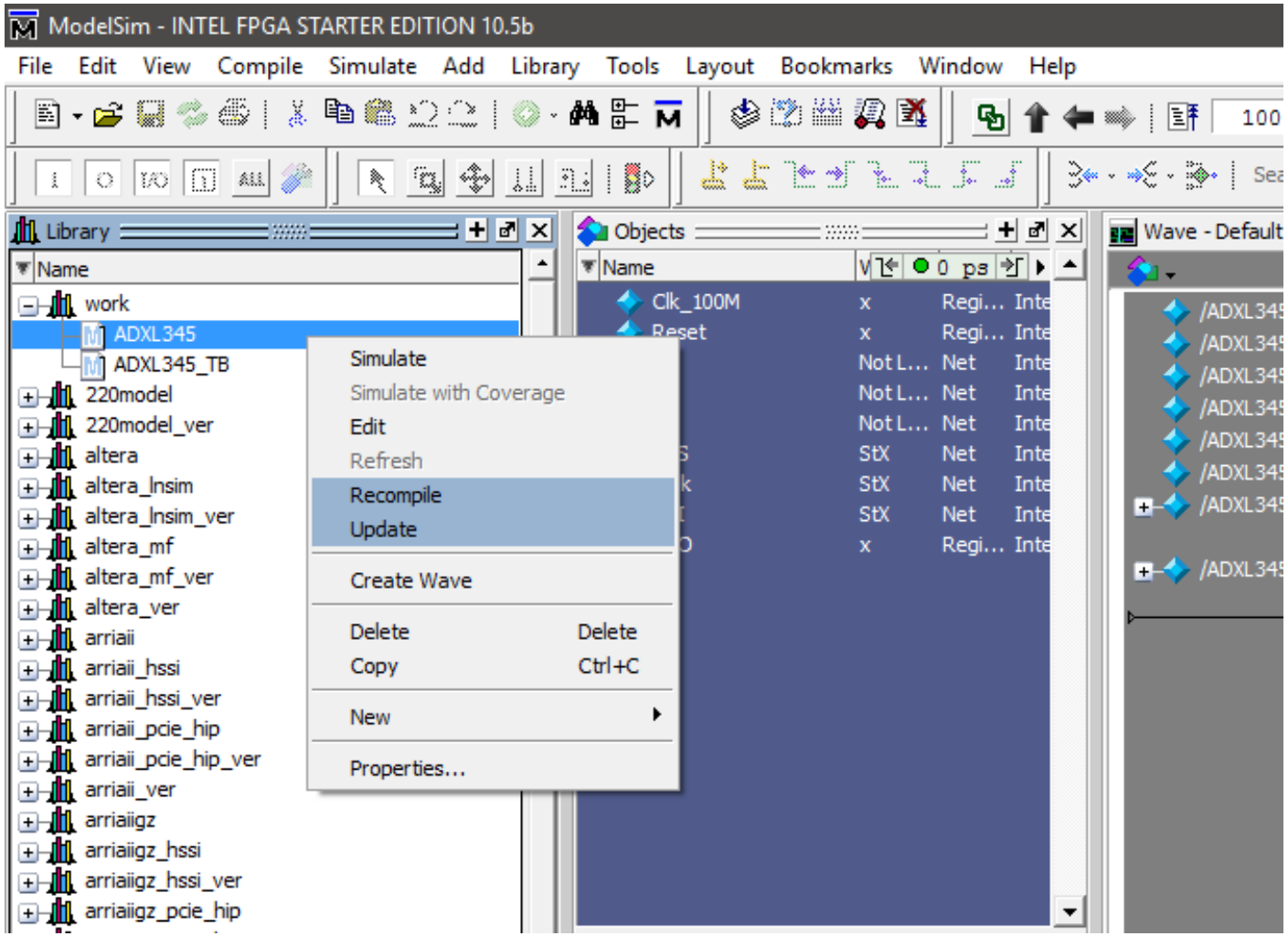
Middle Mouse to Zoom



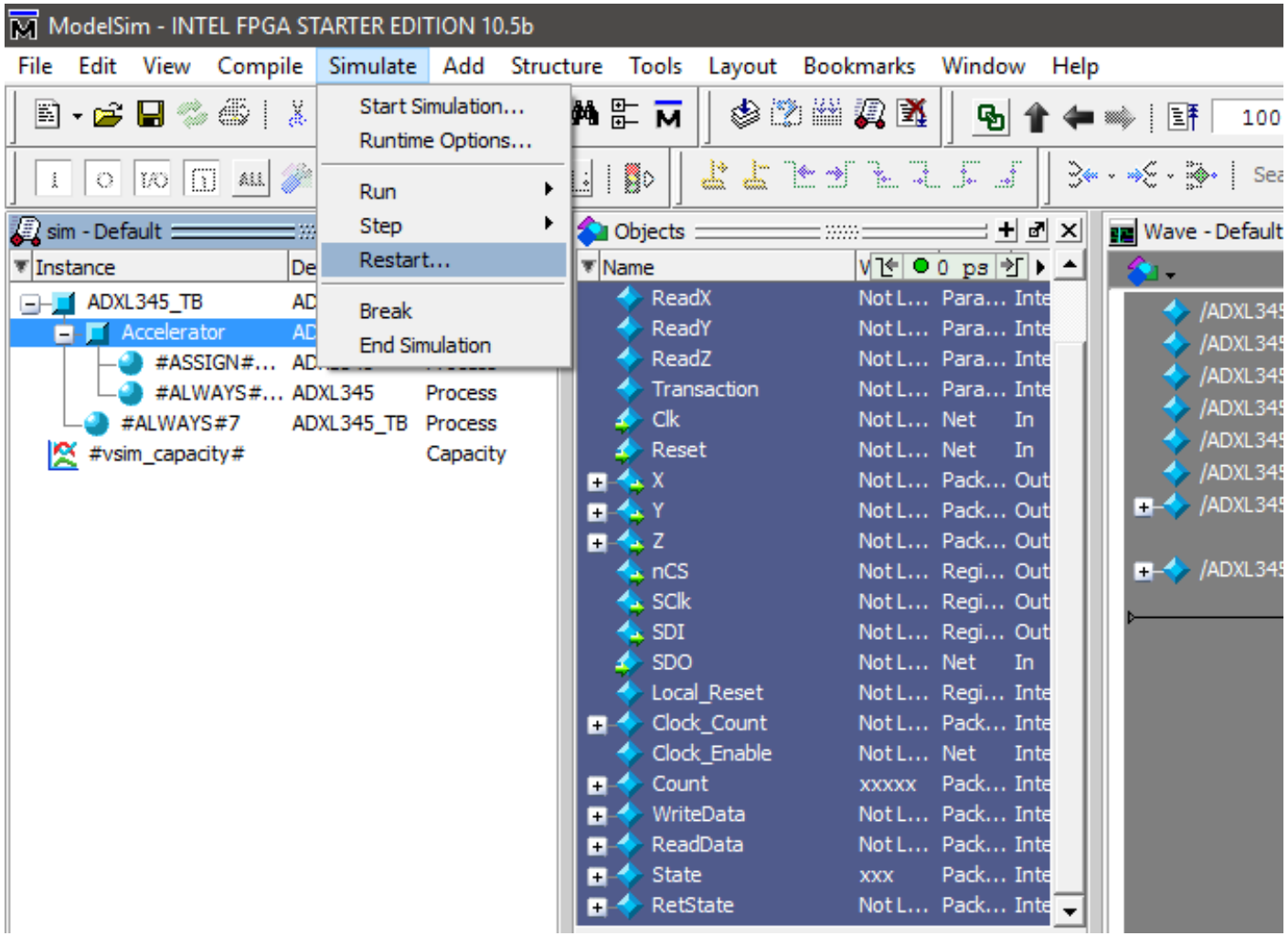
Middle Mouse to Zoom



Recompile and Update Source



Restart Simulation



Outline

Altera Library

Verilog Processes

Finite State Machines

Timing Constraints

Injection Testing

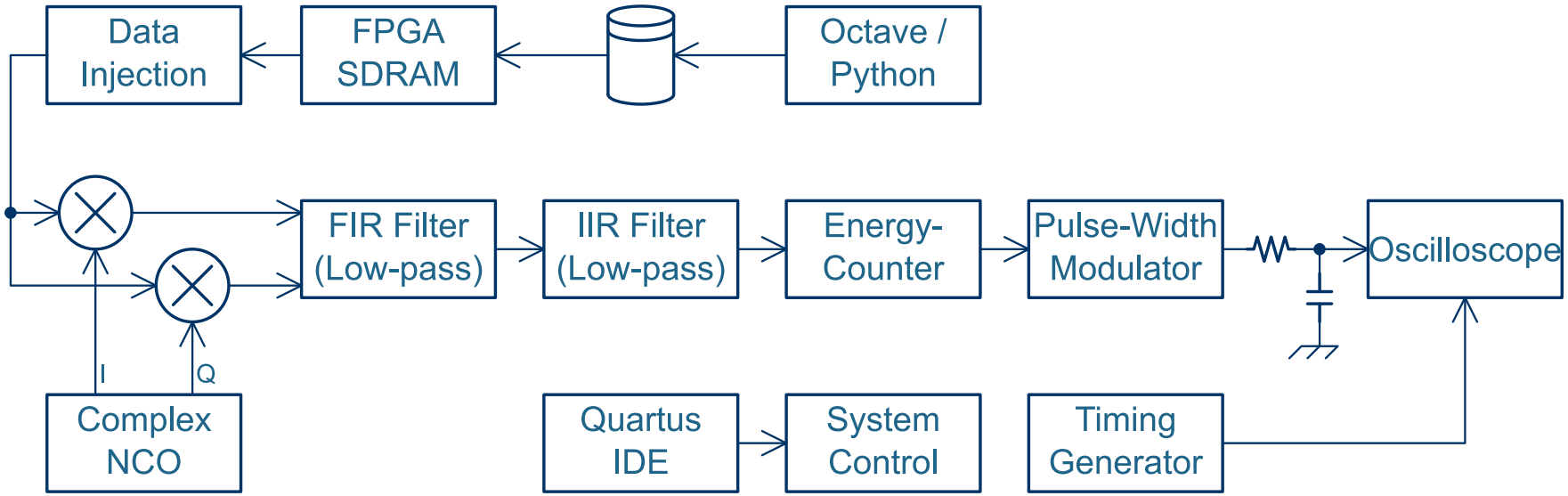
Simulation

Practical

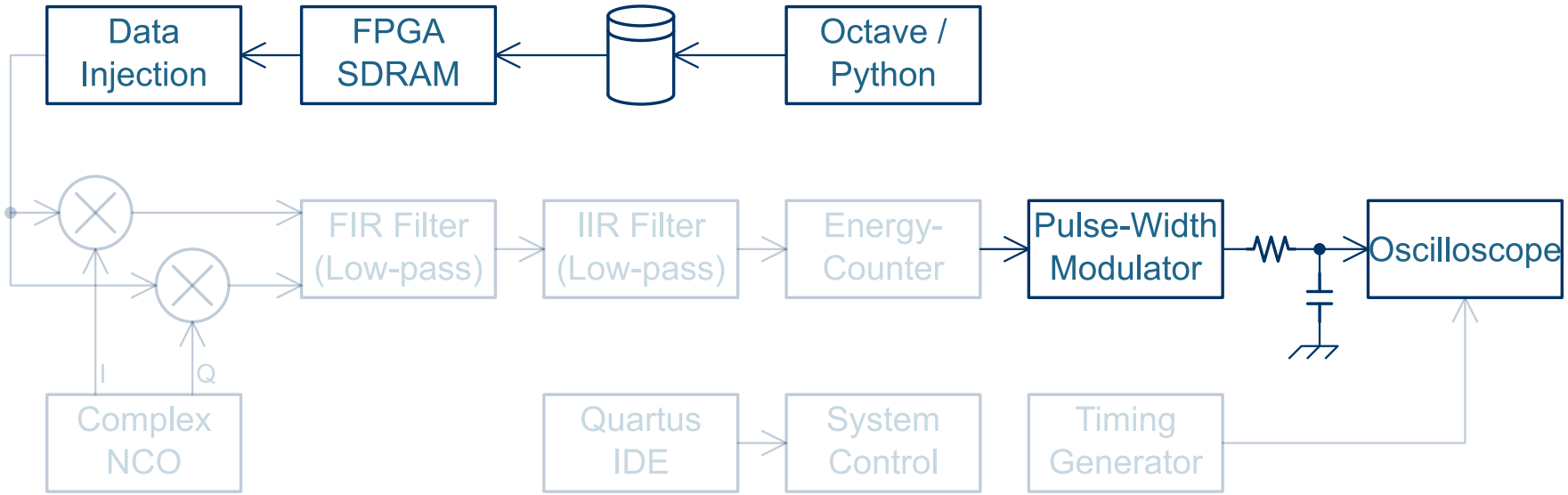
Appendix – High Resolution PWM

THE RADAR MASTERS COURSE

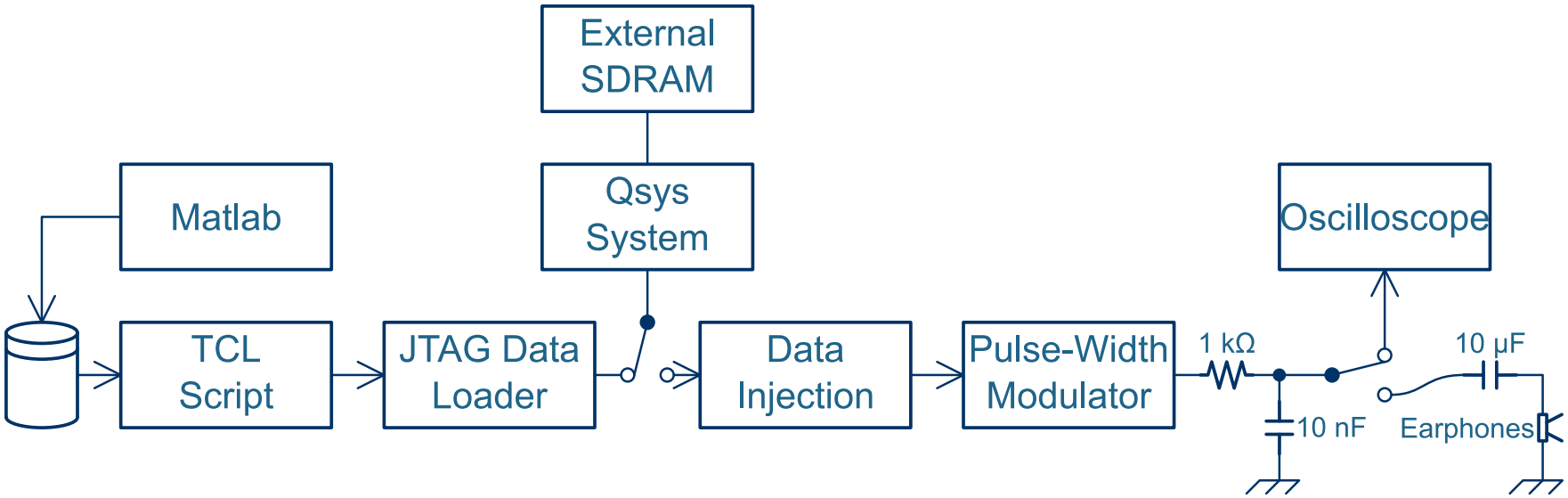
Practical


Data Injection
FPGA SDRAM
Octave / Python
FIR Filter (Low-pass)
IIR Filter (Low-pass)
Energy-Counter
Pulse-Width Modulator
Oscilloscope
I
Q
Complex NCO
Quartus IDE
System Control
Timing Generator

Practical


Data Injection
FPGA SDRAM
Octave / Python
FIR Filter (Low-pass)
IIR Filter (Low-pass)
Energy-Counter
Pulse-Width Modulator
Oscilloscope
I
Q
Complex NCO
Quartus IDE
System Control
Timing Generator

Audio Player



Agenda

1. PWM Module
   - Write a PWM module (optionally with noise shaping)
     Use a 100 MHz master clock and 390.625 kHz PWM (8-bit)
   - Simulate the PWM module to verify functionality
2. Simple Injection Module
3. SDRAM Injection Module

Agenda

1. PWM Module
   - Write a PWM module (optionally with noise shaping)
     Use a 100 MHz master clock and 390.625 kHz PWM (8-bit)
   - Simulate the PWM module to verify functionality
2. Simple Injection Module
3. SDRAM Injection Module

Agenda

1. PWM Module
   - Write a PWM module (optionally with noise shaping)
     Use a 100 MHz master clock and 390.625 kHz PWM (8-bit)
   - Simulate the PWM module to verify functionality
2. Simple Injection Module
3. SDRAM Injection Module

Agenda

1. PWM Module
   - Write a PWM module (optionally with noise shaping)
     Use a 100 MHz master clock and 390.625 kHz PWM (8-bit)
   - Simulate the PWM module to verify functionality
2. Simple Injection Module
   - Pre-load a RAM block (1024 $\times$ 8-bit) with a MIF file
     (Generated by Python / Octave / Matlab / Excel / etc.)
   - Play back the RAM block data through the PWM module
     Inject at 48.828 125 kSps
3. SDRAM Injection Module

Agenda

1. PWM Module
   - Write a PWM module (optionally with noise shaping)
     Use a 100 MHz master clock and 390.625 kHz PWM (8-bit)
   - Simulate the PWM module to verify functionality
2. Simple Injection Module
   - Pre-load a RAM block (1024 $\times$ 8-bit) with a MIF file
     (Generated by Python / Octave / Matlab / Excel / etc.)
   - Play back the RAM block data through the PWM module
     Inject at 48.828 125 kSps
3. SDRAM Injection Module

Agenda

1. PWM Module
   - Write a PWM module (optionally with noise shaping)
     Use a 100 MHz master clock and 390.625 kHz PWM (8-bit)
   - Simulate the PWM module to verify functionality
2. Simple Injection Module
   - Pre-load a RAM block (1024 $\times$ 8-bit) with a MIF file
     (Generated by Python / Octave / Matlab / Excel / etc.)
   - Play back the RAM block data through the PWM module
     Inject at 48.828 125 kSps
3. SDRAM Injection Module

Agenda

1. PWM Module
   - Write a PWM module (optionally with noise shaping)
     Use a 100 MHz master clock and 390.625 kHz PWM (8-bit)
   - Simulate the PWM module to verify functionality
2. Simple Injection Module
   - Pre-load a RAM block (1024 $\times$ 8-bit) with a MIF file
     (Generated by Python / Octave / Matlab / Excel / etc.)
   - Play back the RAM block data through the PWM module
     Inject at 48.828 125 kSps
3. SDRAM Injection Module
   - Move data injection to the external memory
   - Use the RAM block as a FIFO cache
   - Use a virtual JTAG interface to load data into the memory
   - Use the buttons, switches and LEDs to control injection

Agenda

1. PWM Module
   - Write a PWM module (optionally with noise shaping)
     Use a 100 MHz master clock and 390.625 kHz PWM (8-bit)
   - Simulate the PWM module to verify functionality
2. Simple Injection Module
   - Pre-load a RAM block (1024 $\times$ 8-bit) with a MIF file
     (Generated by Python / Octave / Matlab / Excel / etc.)
   - Play back the RAM block data through the PWM module
     Inject at 48.828 125 kSps
3. SDRAM Injection Module
   - Move data injection to the external memory
   - Use the RAM block as a FIFO cache
   - Use a virtual JTAG interface to load data into the memory
   - Use the buttons, switches and LEDs to control injection

Agenda

1. PWM Module
   - Write a PWM module (optionally with noise shaping)
     Use a 100 MHz master clock and 390.625 kHz PWM (8-bit)
   - Simulate the PWM module to verify functionality
2. Simple Injection Module
   - Pre-load a RAM block (1024 $\times$ 8-bit) with a MIF file
     (Generated by Python / Octave / Matlab / Excel / etc.)
   - Play back the RAM block data through the PWM module
     Inject at 48.828 125 kSps
3. SDRAM Injection Module
   - Move data injection to the external memory
   - Use the RAM block as a FIFO cache
   - Use a virtual JTAG interface to load data into the memory
   - Use the buttons, switches and LEDs to control injection

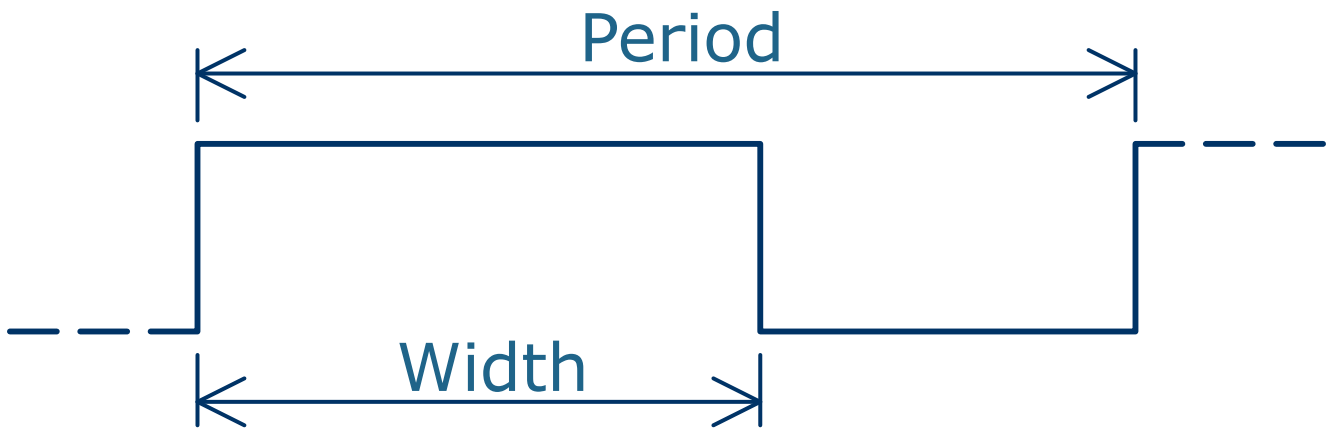
Agenda

1. PWM Module
   - Write a PWM module (optionally with noise shaping)
     Use a 100 MHz master clock and 390.625 kHz PWM (8-bit)
   - Simulate the PWM module to verify functionality
2. Simple Injection Module
   - Pre-load a RAM block (1024 $\times$ 8-bit) with a MIF file
     (Generated by Python / Octave / Matlab / Excel / etc.)
   - Play back the RAM block data through the PWM module
     Inject at 48.828 125 kSps
3. SDRAM Injection Module
   - Move data injection to the external memory
   - Use the RAM block as a FIFO cache
   - Use a virtual JTAG interface to load data into the memory
   - Use the buttons, switches and LEDs to control injection

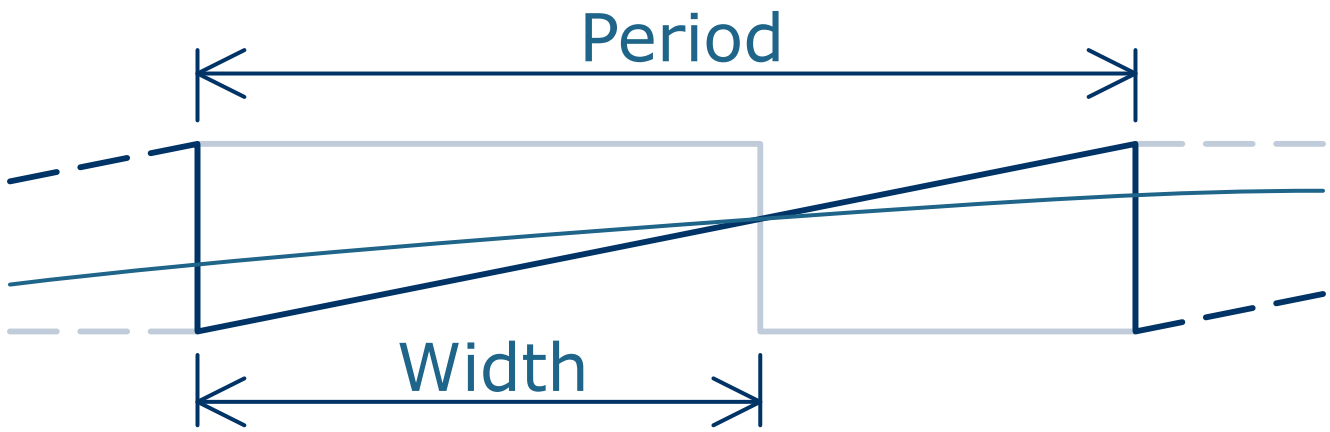
Agenda

1. PWM Module
   - Write a PWM module (optionally with noise shaping)
     Use a 100 MHz master clock and 390.625 kHz PWM (8-bit)
   - Simulate the PWM module to verify functionality
2. Simple Injection Module
   - Pre-load a RAM block (1024 $\times$ 8-bit) with a MIF file
     (Generated by Python / Octave / Matlab / Excel / etc.)
   - Play back the RAM block data through the PWM module
     Inject at 48.828 125 kSps
3. SDRAM Injection Module
   - Move data injection to the external memory
   - Use the RAM block as a FIFO cache
   - Use a virtual JTAG interface to load data into the memory
   - Use the buttons, switches and LEDs to control injection

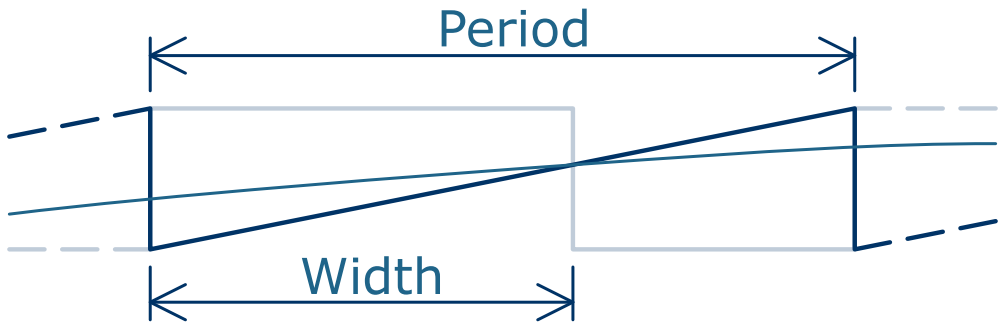
Pulse-width Modulation



Pulse-width Modulation

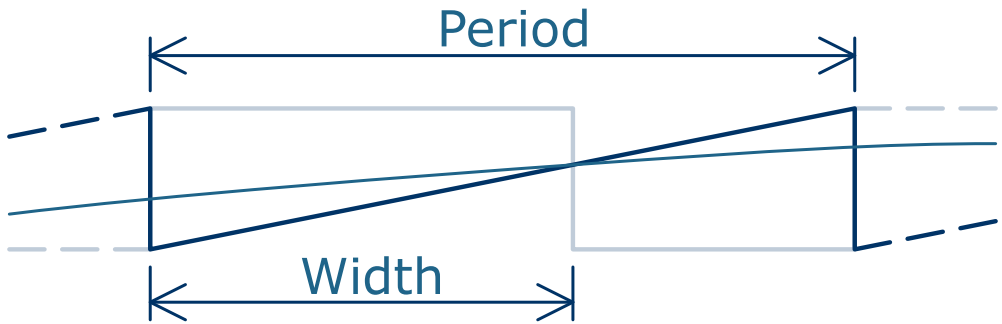


Pulse-width Modulation



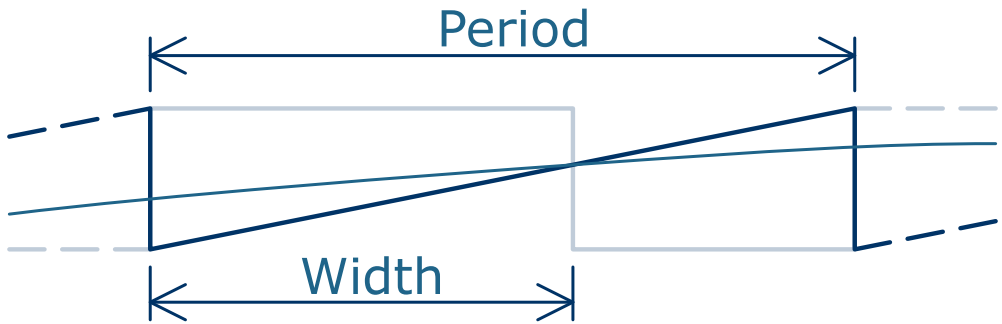
- Typically, the signal is in 2's complement
- The PWM module requires an unsigned input from 0 to $(2^N - 1)$ – also known as offset-binary
- To convert from one to the other, invert the most-significant bit

Pulse-width Modulation



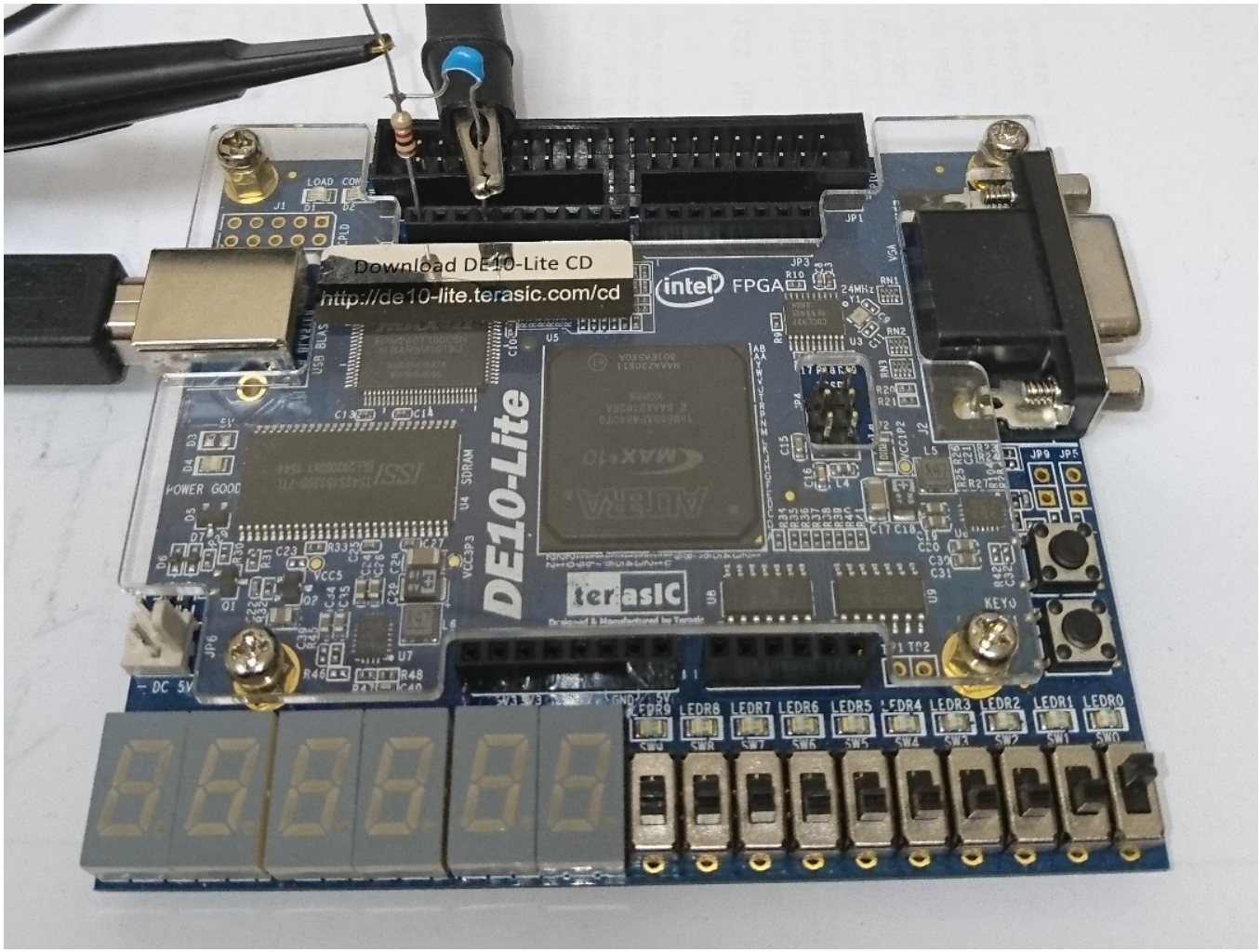
- Typically, the signal is in 2's complement
- The PWM module requires an unsigned input from 0 to $(2^N - 1)$ – also known as offset-binary
- To convert from one to the other, invert the most-significant bit

Pulse-width Modulation



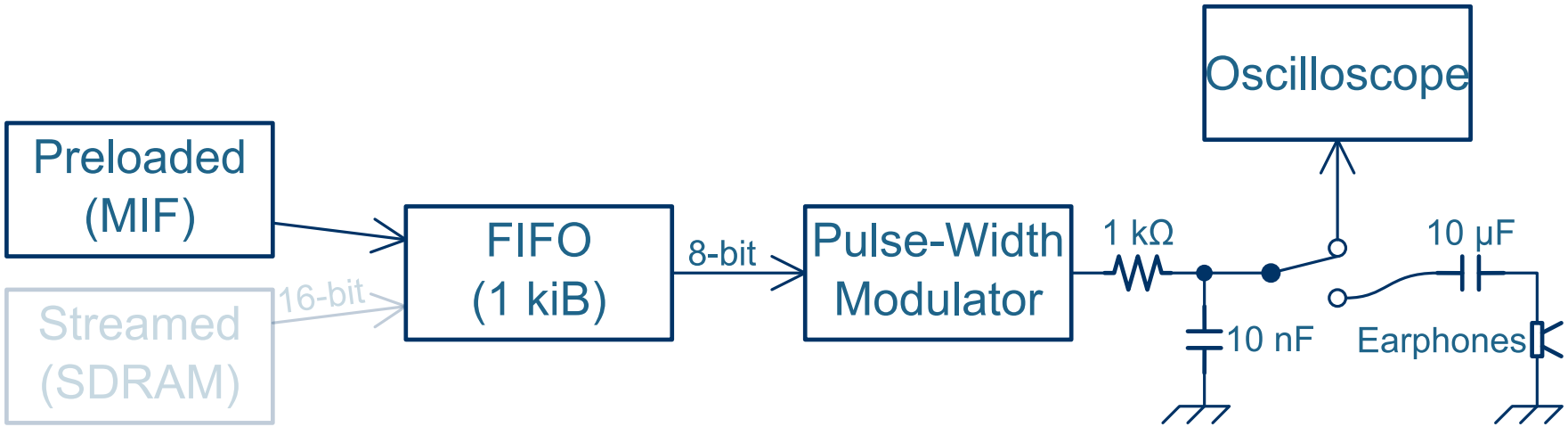
- Typically, the signal is in 2's complement
- The PWM module requires an unsigned input from 0 to $(2^N - 1)$ – also known as offset-binary
- To convert from one to the other, invert the most-significant bit

Pulse-width Modulation



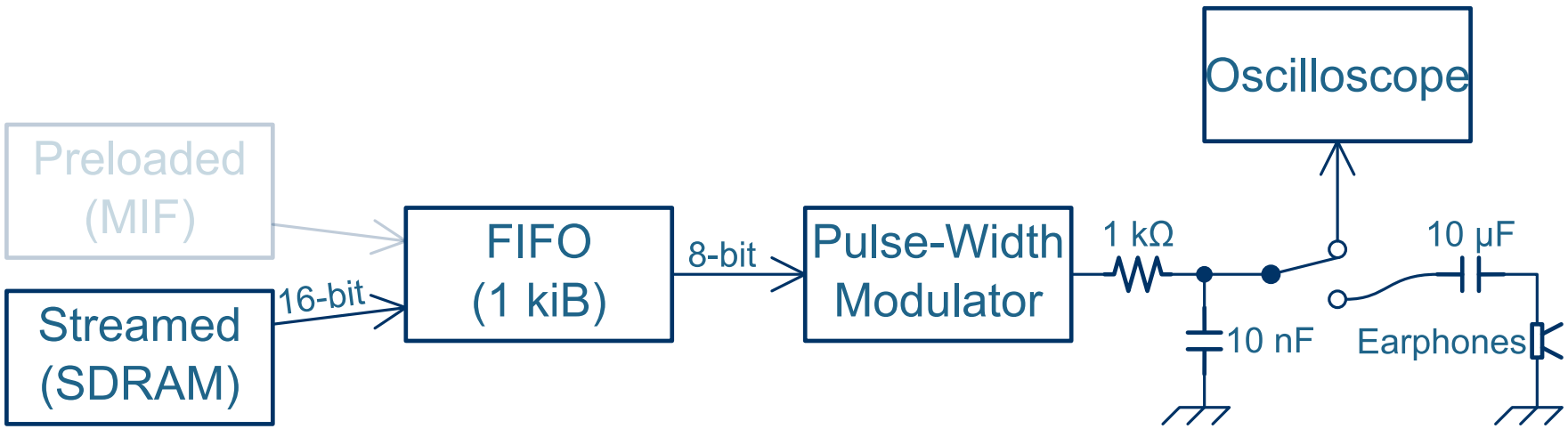
Simple Injection

Before continuing,
get the PWM and simple injection modules working...
(You don't have a 100 MHz clock yet, so use the 50 MHz for the time being)

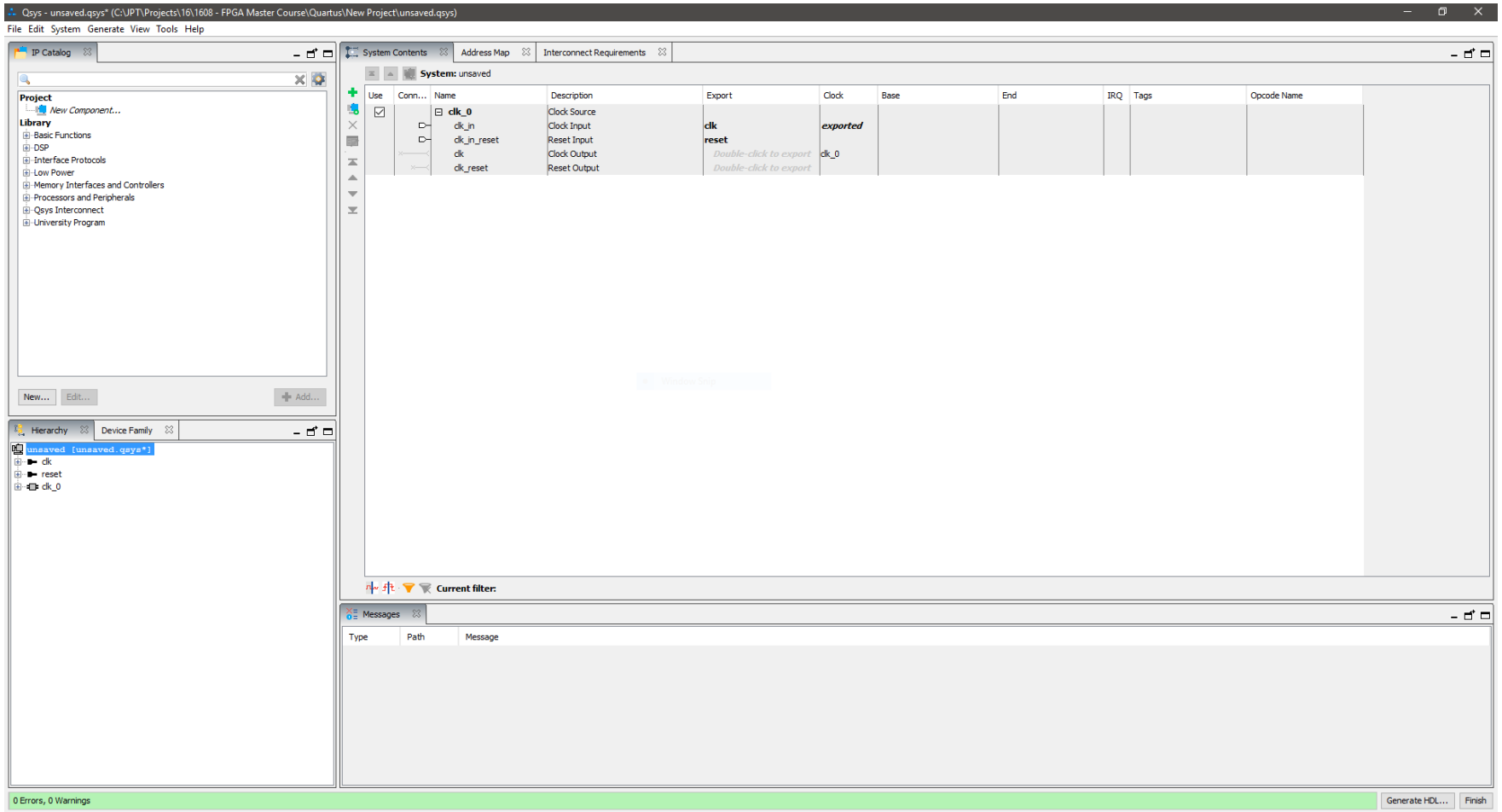


SDRAM Injection

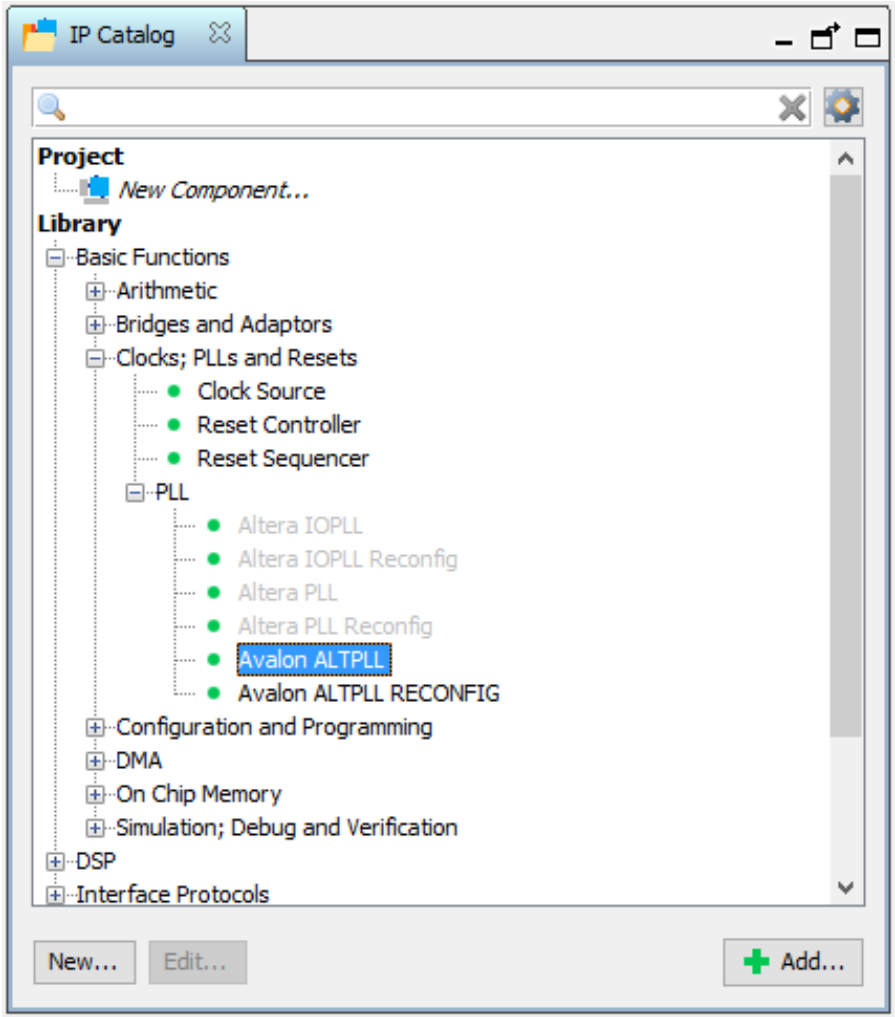
Keep in mind that the same FIFO-based injection module must interface with the 16-bit SDRAM...



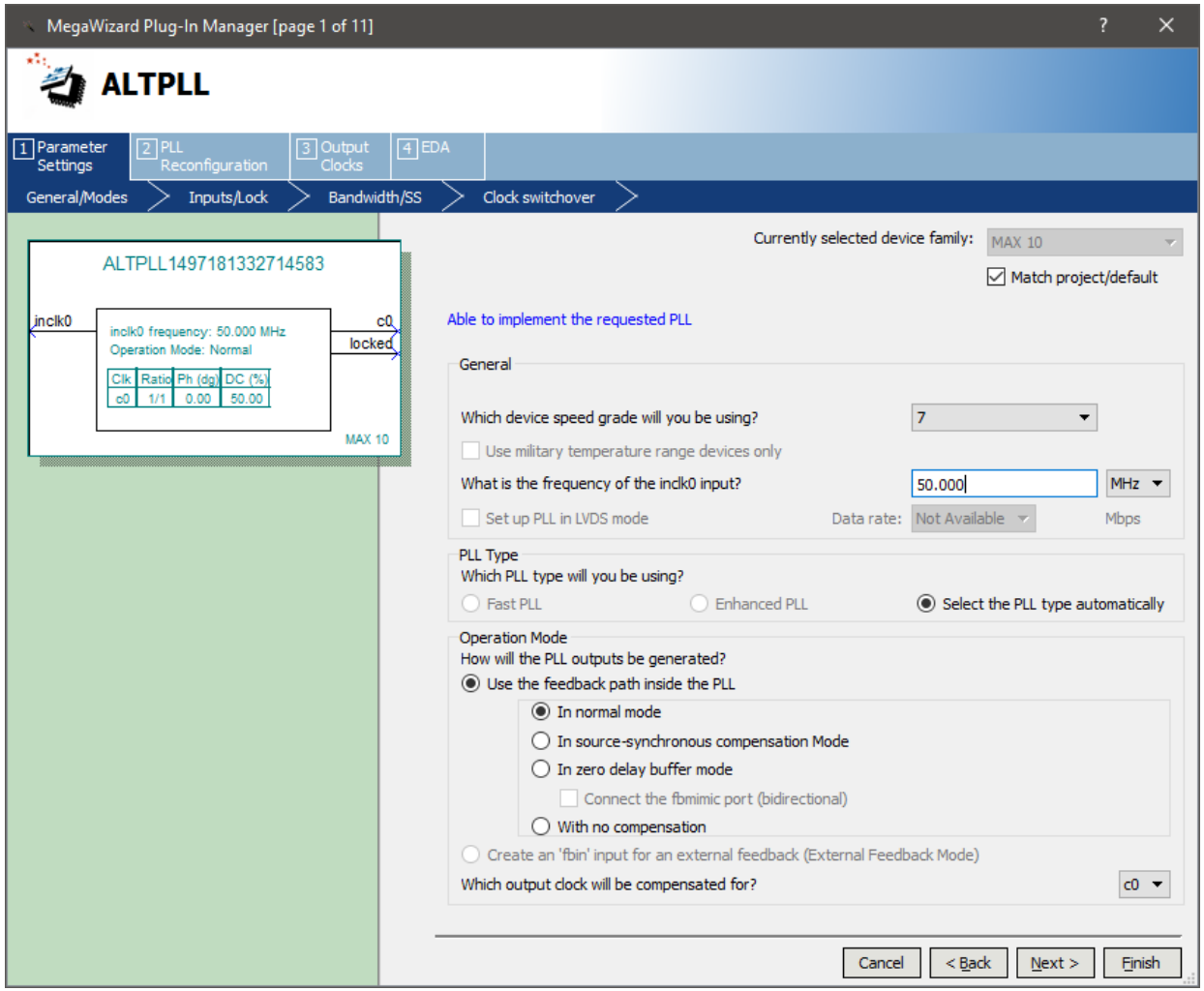
Qsys



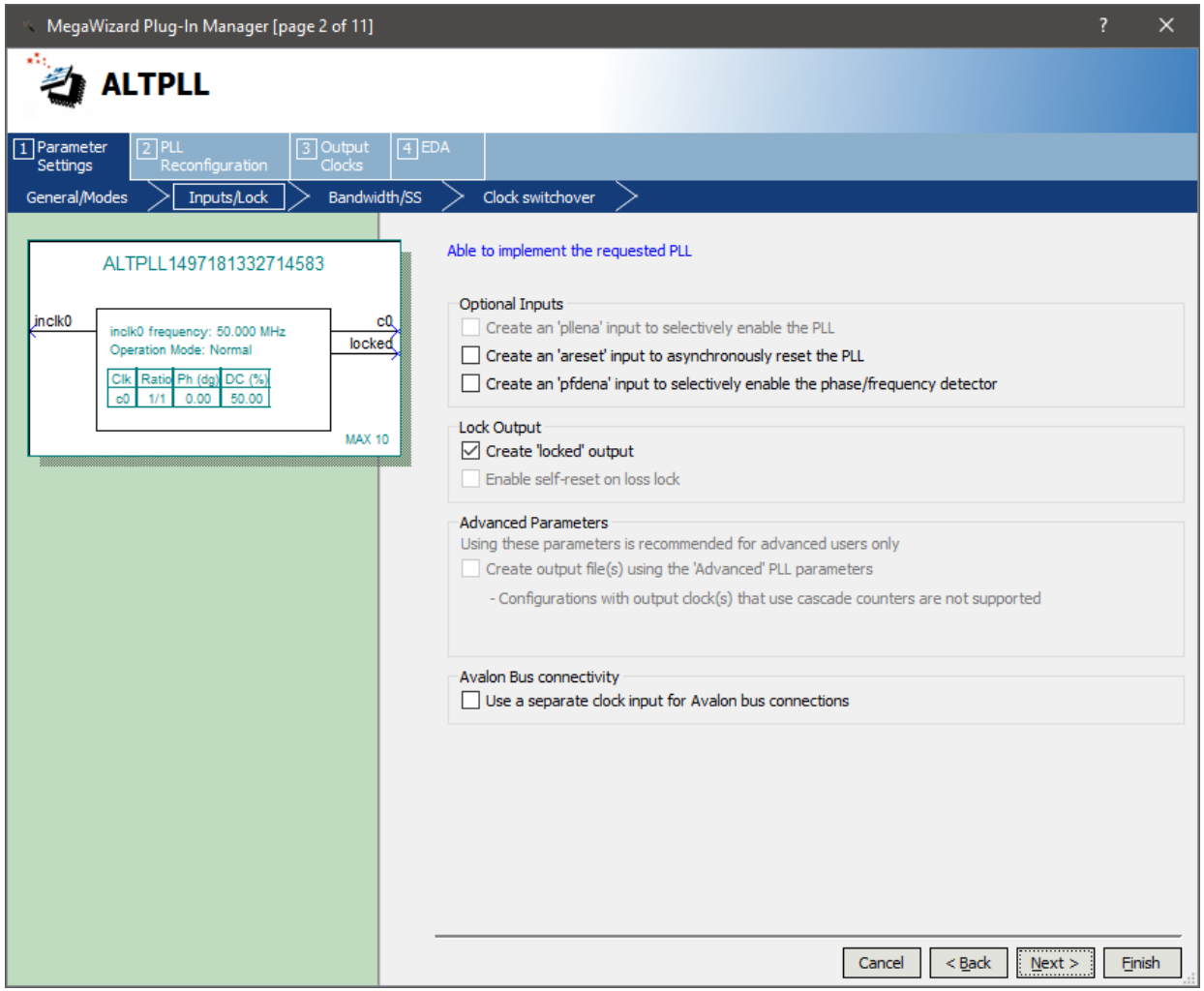
Create a New PLL



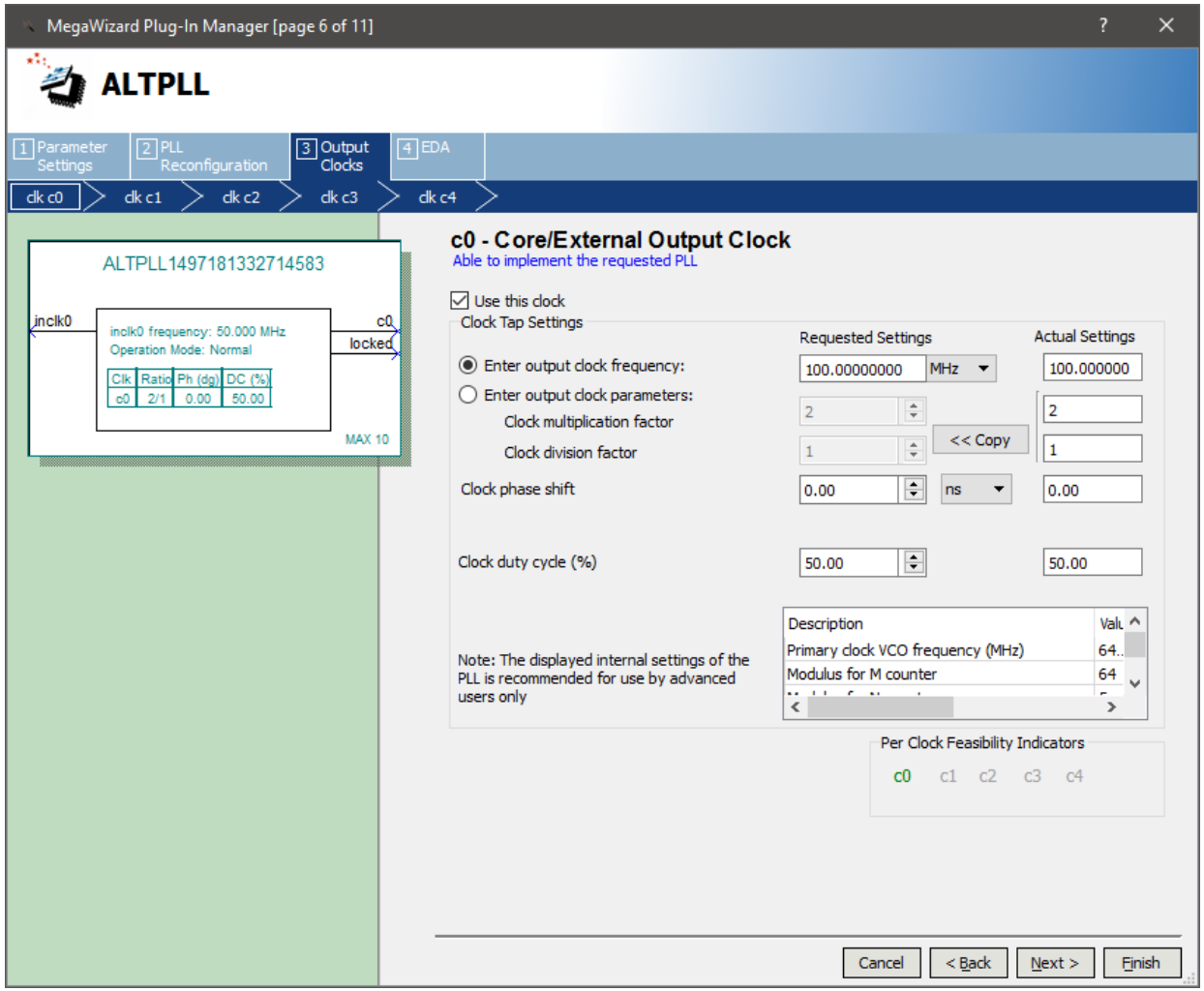
Create a New PLL



Create a New PLL

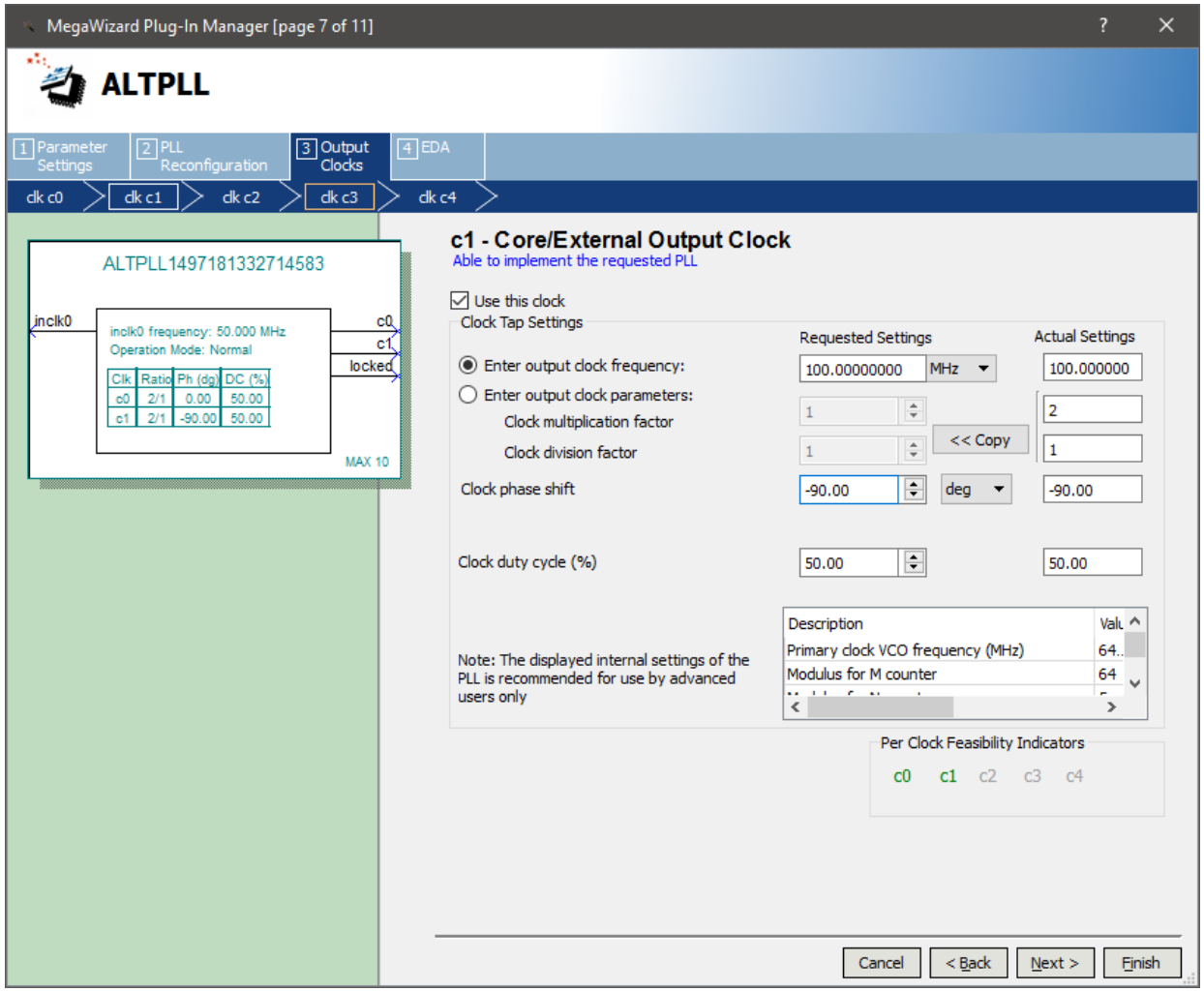


Create a New PLL

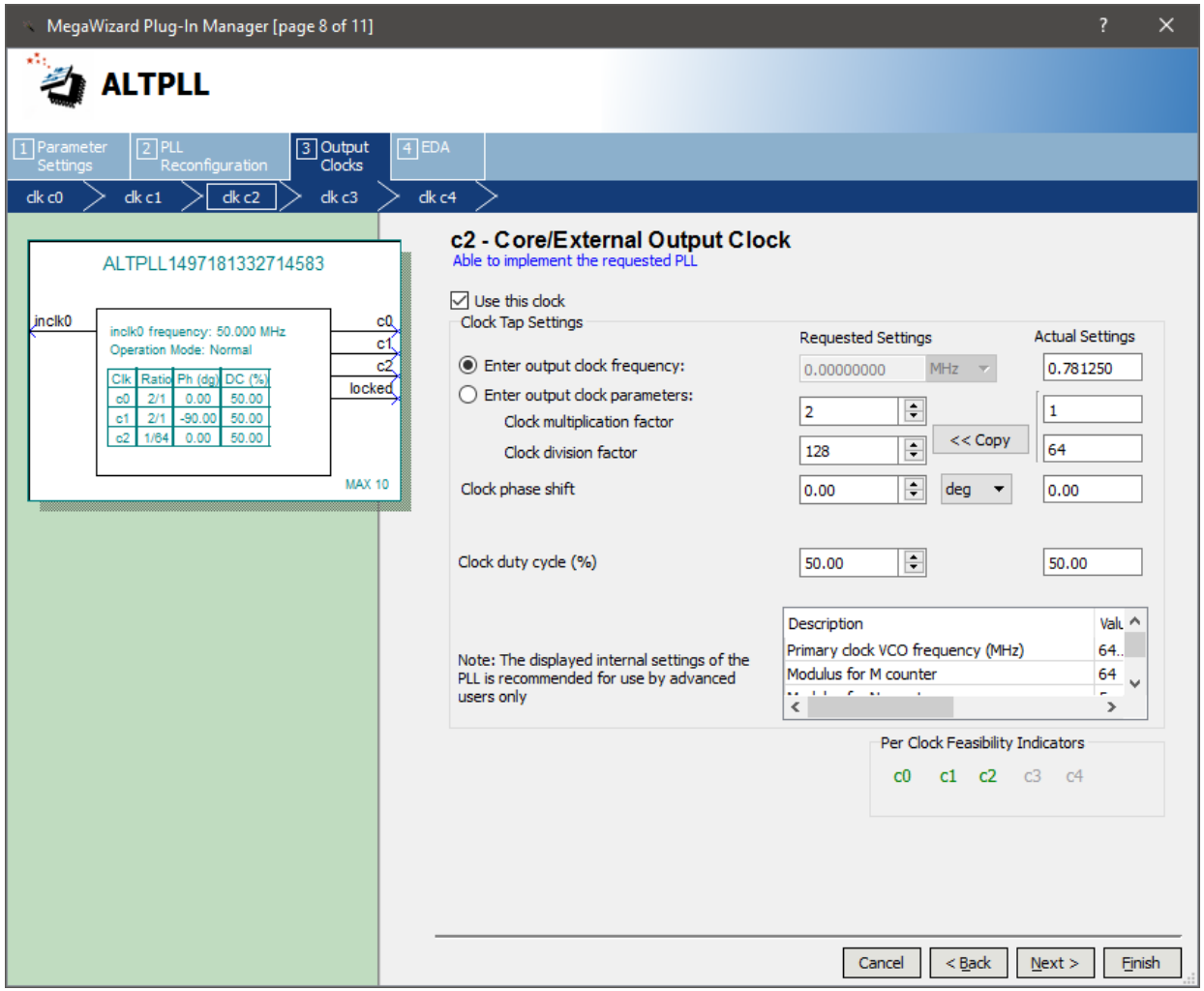


THE RADAR MASTERS COURSE

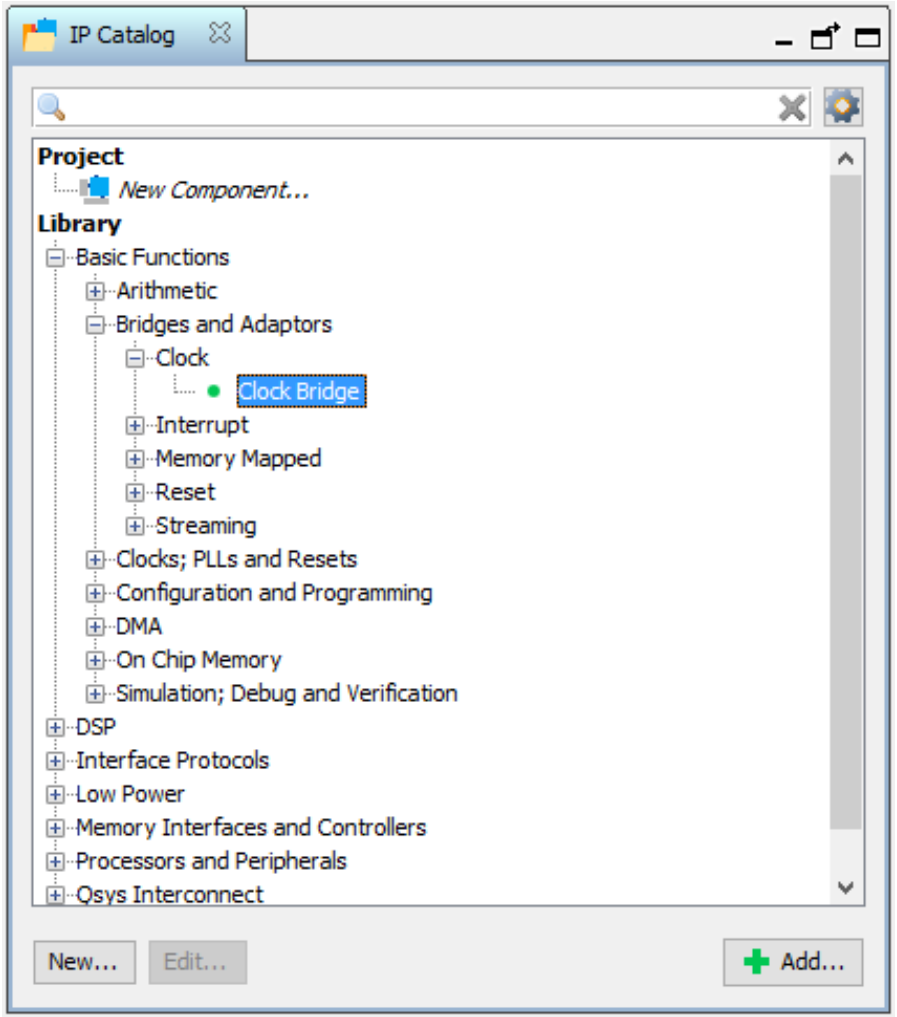
Create a New PLL



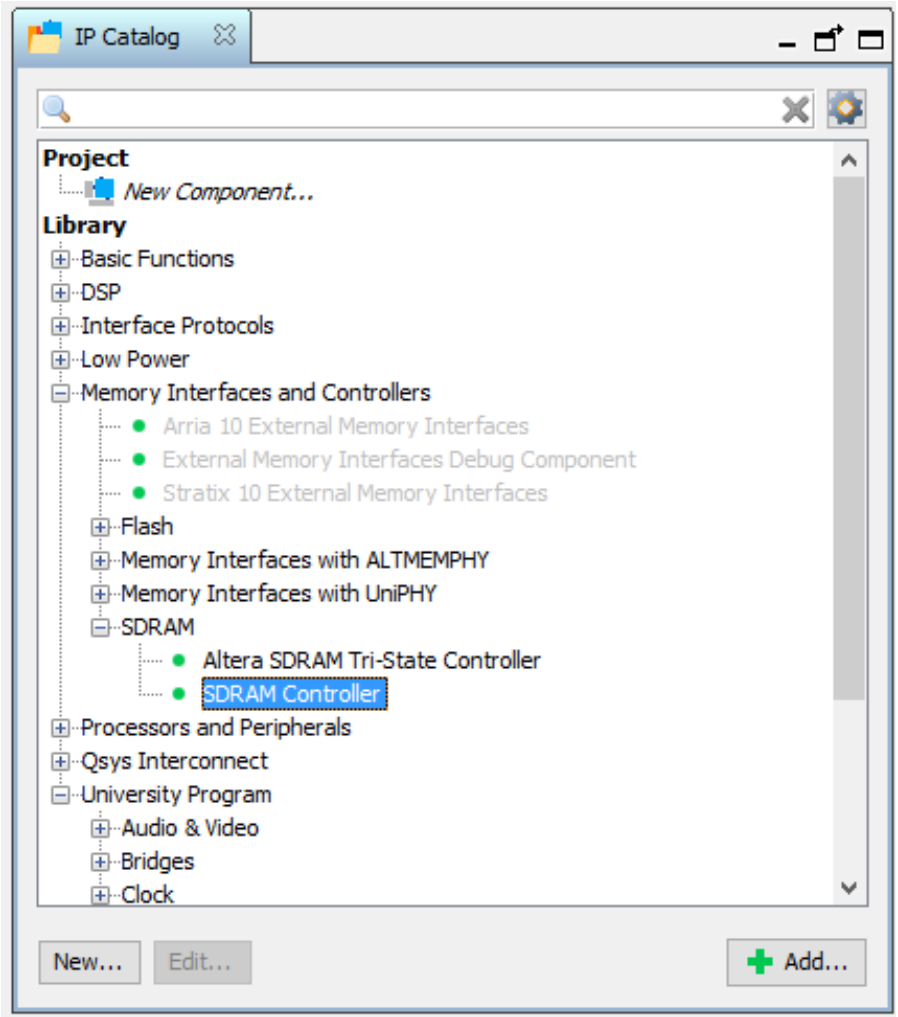
Create a New PLL



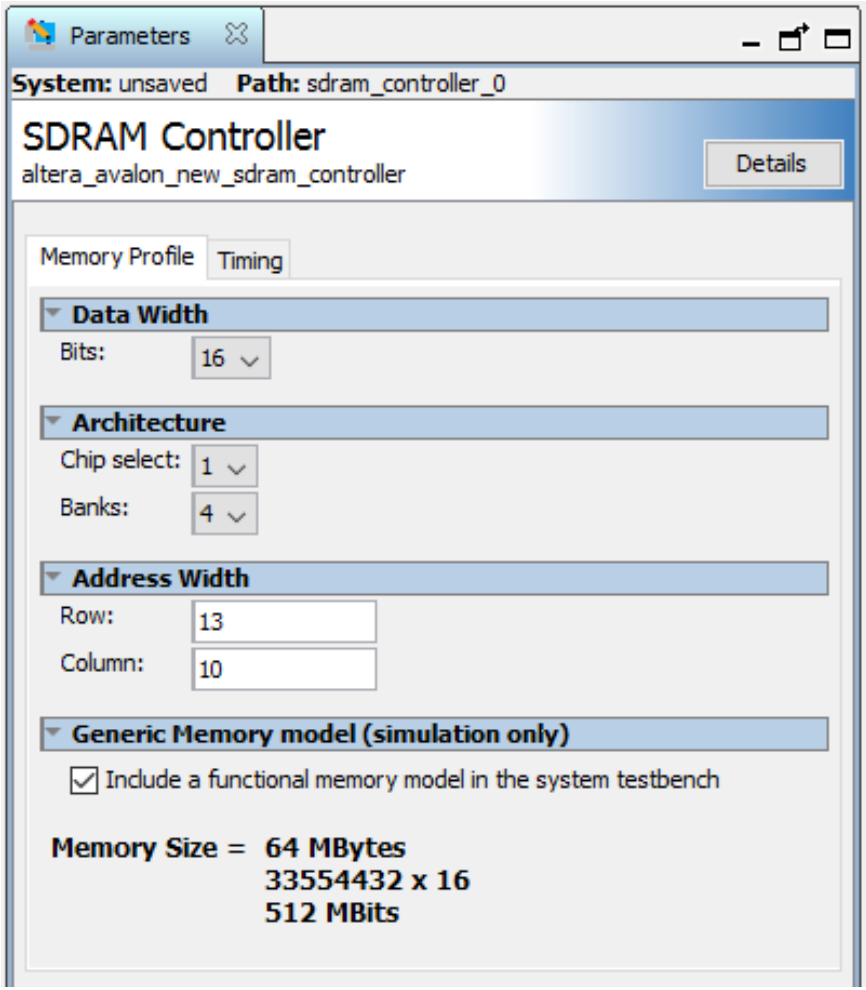
Clock Bridge

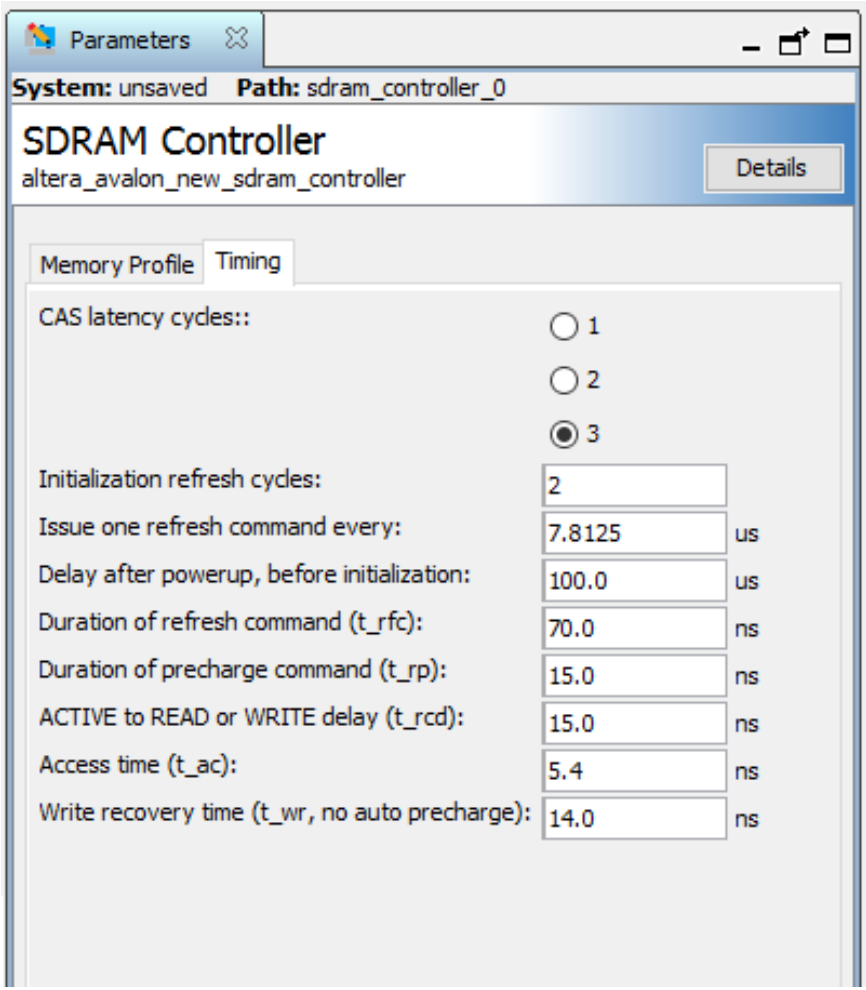


Create a New SDRAM Controller

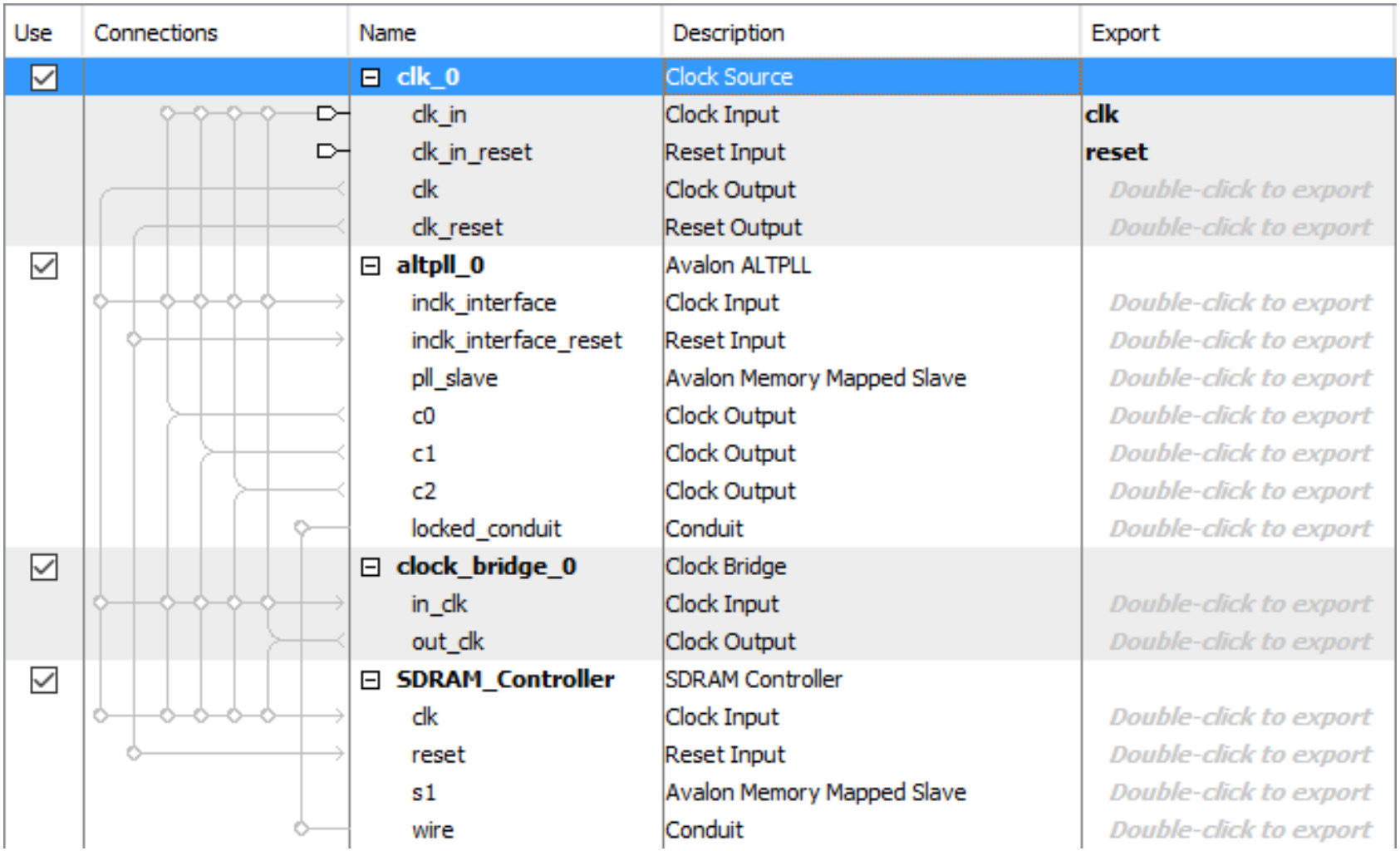


Create a New SDRAM Controller



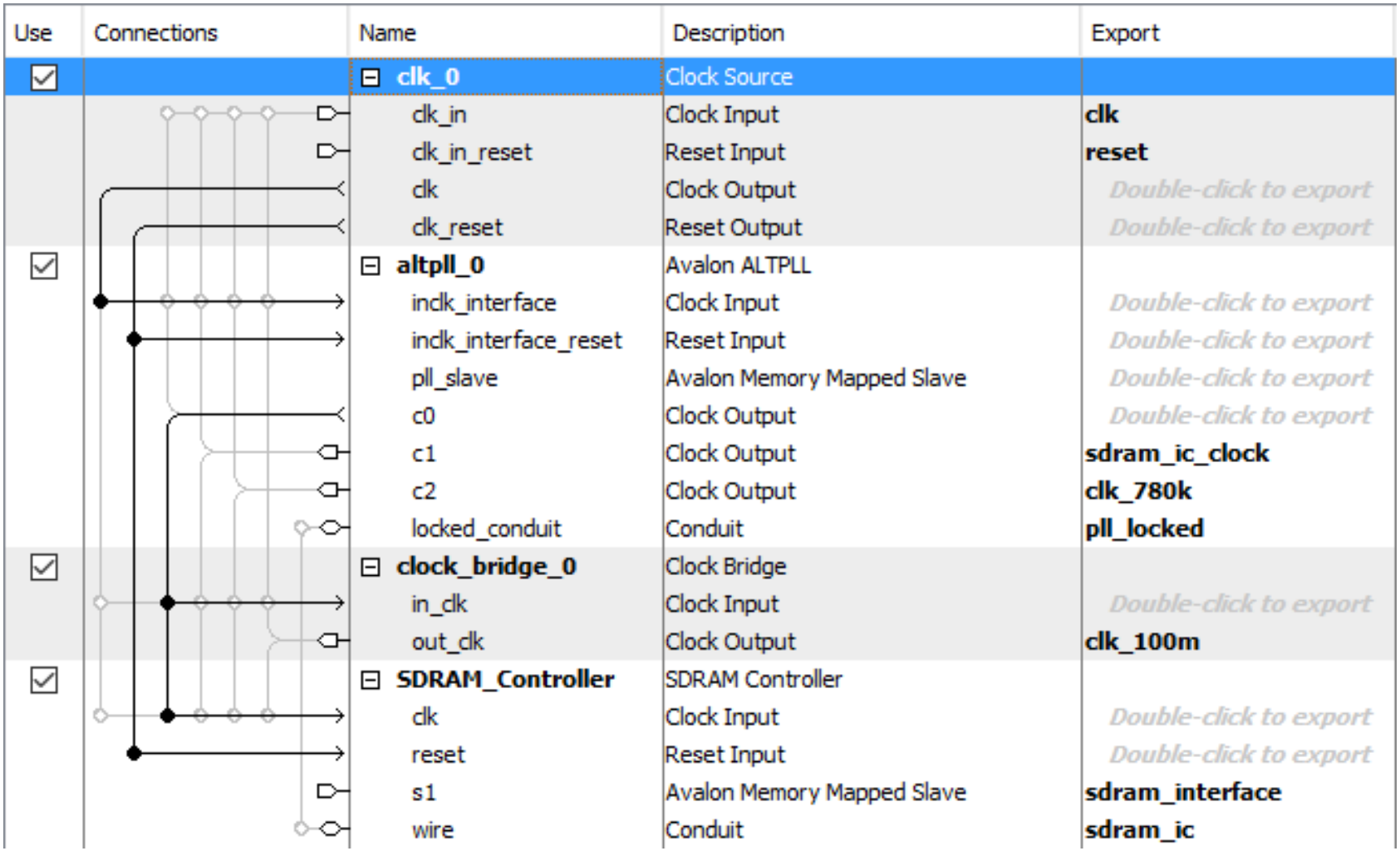


Unconnected System



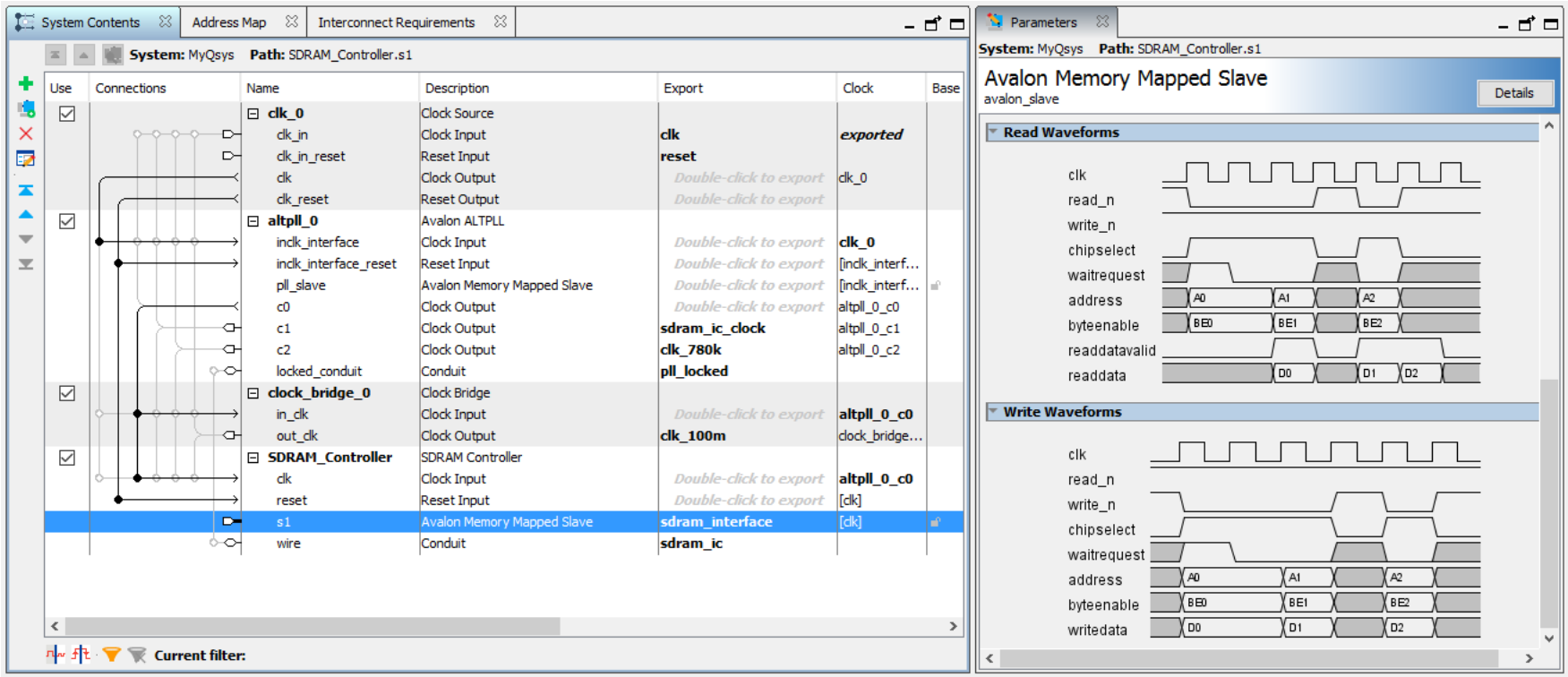
Use	Connections	Name	Description	Export
☑		**clk_0**	Clock Source	
		clk_in	Clock Input	**clk**
		clk_in_reset	Reset Input	**reset**
		clk	Clock Output	Double-click to export
		clk_reset	Reset Output	Double-click to export
☑		**altpll_0**	Avalon ALTPLL	
		inclk_interface	Clock Input	Double-click to export
		inclk_interface_reset	Reset Input	Double-click to export
		pll_slave	Avalon Memory Mapped Slave	Double-click to export
		c0	Clock Output	Double-click to export
		c1	Clock Output	Double-click to export
		c2	Clock Output	Double-click to export
		locked_conduit	Conduit	Double-click to export
☑		**clock_bridge_0**	Clock Bridge	
		in_clk	Clock Input	Double-click to export
		out_clk	Clock Output	Double-click to export
☑		**SDRAM_Controller**	SDRAM Controller	
		clk	Clock Input	Double-click to export
		reset	Reset Input	Double-click to export
		s1	Avalon Memory Mapped Slave	Double-click to export
		wire	Conduit	Double-click to export

Connected System

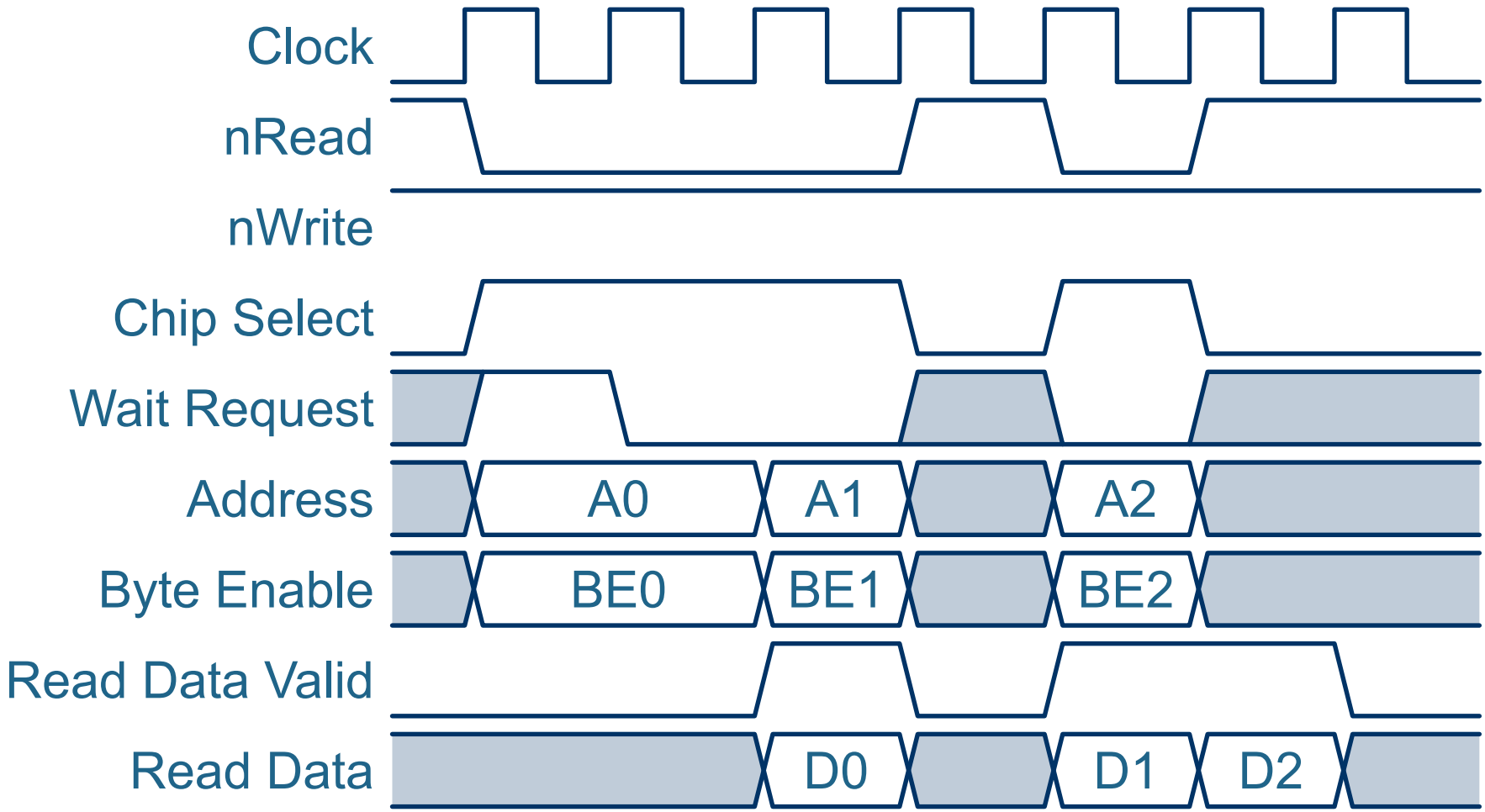


Use	Connections	Name	Description	Export
☑		**clk_0**	Clock Source	
		clk_in	Clock Input	**clk**
		clk_in_reset	Reset Input	**reset**
		clk	Clock Output	*Double-click to export*
		clk_reset	Reset Output	*Double-click to export*
☑		**altpll_0**	Avalon ALTPLL	
		inclk_interface	Clock Input	*Double-click to export*
		inclk_interface_reset	Reset Input	*Double-click to export*
		pll_slave	Avalon Memory Mapped Slave	*Double-click to export*
		c0	Clock Output	*Double-click to export*
		c1	Clock Output	**sdram_ic_clock**
		c2	Clock Output	**clk_780k**
		locked_conduit	Conduit	**pll_locked**
☑		**clock_bridge_0**	Clock Bridge	
		in_clk	Clock Input	*Double-click to export*
		out_clk	Clock Output	**clk_100m**
☑		**SDRAM_Controller**	SDRAM Controller	
		clk	Clock Input	*Double-click to export*
		reset	Reset Input	*Double-click to export*
		s1	Avalon Memory Mapped Slave	**sdram_interface**
		wire	Conduit	**sdram_ic**

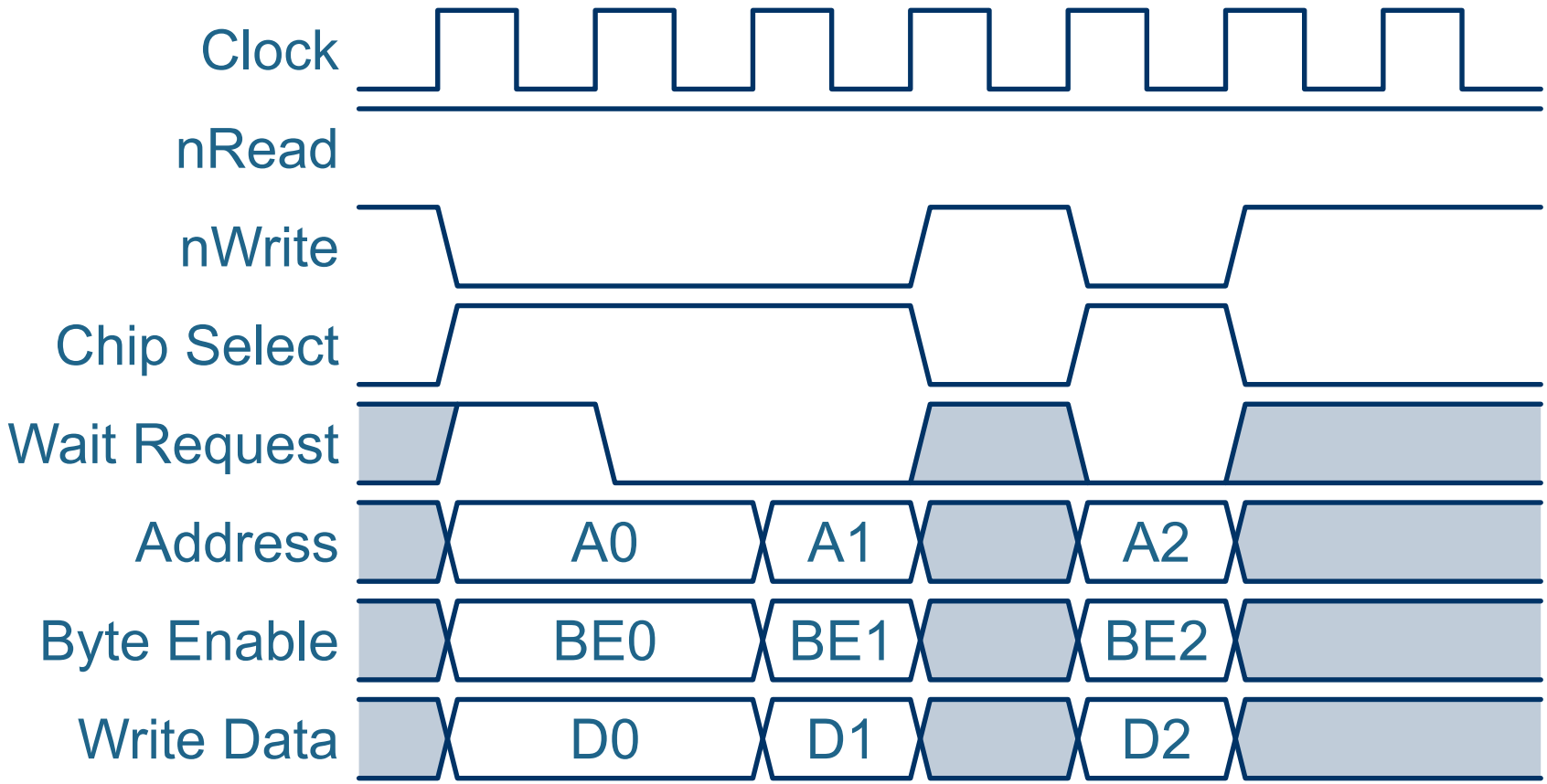
Interface Timing Diagrams



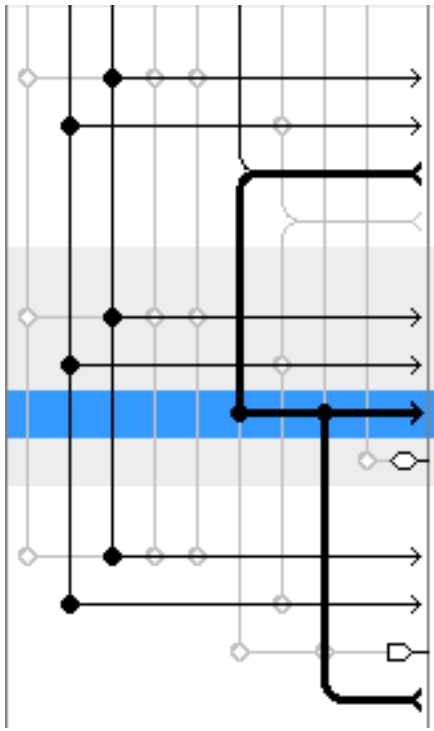
Interface Timing Diagrams



Interface Timing Diagrams



Two Masters Example



JTAG_MM	JTAG to Avalon Master Bridge	
clk	Clock Input	*Double-click to export*
clk_reset	Reset Input	*Double-click to export*
master	Avalon Memory Mapped Master	*Double-click to export*
master_reset	Reset Output	*Double-click to export*
SDRAM_Controller	SDRAM Controller	
clk	Clock Input	*Double-click to export*
reset	Reset Input	*Double-click to export*
s1	Avalon Memory Mapped Slave	*Double-click to export*
wire	Conduit	**sdram_ic**
MM_Bridge	Avalon-MM Pipeline Bridge	
clk	Clock Input	*Double-click to export*
reset	Reset Input	*Double-click to export*
s0	Avalon Memory Mapped Slave	**sdram_interface**
m0	Avalon Memory Mapped Master	*Double-click to export*

Note: This is just an example – for today's practical, export the slave interface directly and use an on-board slide-switch to manually choose which master (JTAG loader vs. Injection) is connected to the interface.

Resulting Wrapper Module

- Save the system as “MyQsys” and generate the HDL...

```
// Contents of the Quartus-generated module:

module MyQsys(
   input          clk_clk,
   input          reset_reset_n,

   output         clk_780k_clk,
   output         clk_100m_clk,

   output         pll_locked_export,
   .
   .
   .
```

Resulting Wrapper Module


```
.
.
.
output        sdram_ic_clock_clk,
output [12:0]sdram_ic_addr,
output [ 1:0]sdram_ic_ba,
output        sdram_ic_cas_n,
output        sdram_ic_cke,
output        sdram_ic_cs_n,
inout  [15:0]sdram_ic_dq,
output [ 1:0]sdram_ic_dqm,
output        sdram_ic_ras_n,
output        sdram_ic_we_n,
.
.
.
```

Resulting Wrapper Module


```
    .
    .
    .
    input   [24:0]sdram_interface_address,
    input   [ 1:0]sdram_interface_byteenable_n,
    input         sdram_interface_chipselect,
    input   [15:0]sdram_interface_writedata,
    input         sdram_interface_read_n,
    input         sdram_interface_write_n,
    output  [15:0]sdram_interface_readdata,
    output        sdram_interface_readdatavalid,
    output        sdram_interface_waitrequest
);
```

SDRAM Connections

1. Create an instance of the MyQsys module in the top-level module
2. Add the SDRAM ports to the top-level module
3. Allocate the correct pin numbers, logic standard, output current rating and pin capacitance to the SDRAM ports
4. Add timing constraints for the SDRAM ports
5. Use the “PLL-Locked” signal to reset the rest of the FPGA circuit

Note: Consult previous slides on how to do this...

SDRAM Connections

1. Create an instance of the MyQsys module in the top-level module
2. Add the SDRAM ports to the top-level module
3. Allocate the correct pin numbers, logic standard, output current rating and pin capacitance to the SDRAM ports
4. Add timing constraints for the SDRAM ports
5. Use the “PLL-Locked” signal to reset the rest of the FPGA circuit

Note: Consult previous slides on how to do this...

SDRAM Connections

1. Create an instance of the MyQsys module in the top-level module
2. Add the SDRAM ports to the top-level module
3. Allocate the correct pin numbers, logic standard, output current rating and pin capacitance to the SDRAM ports
4. Add timing constraints for the SDRAM ports
5. Use the “PLL-Locked” signal to reset the rest of the FPGA circuit

Note: Consult previous slides on how to do this...

SDRAM Connections

1. Create an instance of the MyQsys module in the top-level module
2. Add the SDRAM ports to the top-level module
3. Allocate the correct pin numbers, logic standard, output current rating and pin capacitance to the SDRAM ports
4. Add timing constraints for the SDRAM ports
5. Use the “PLL-Locked” signal to reset the rest of the FPGA circuit

Note: Consult previous slides on how to do this...

SDRAM Connections

1. Create an instance of the MyQsys module in the top-level module
2. Add the SDRAM ports to the top-level module
3. Allocate the correct pin numbers, logic standard, output current rating and pin capacitance to the SDRAM ports
4. Add timing constraints for the SDRAM ports
5. Use the “PLL-Locked” signal to reset the rest of the FPGA circuit

Note: Consult previous slides on how to do this...

Loading Data into SDRAM

1. Add the provided “VirtualJTAG_MM_Write.v” module to the project and create an instance in the top-level entity
2. Connect the Avalon interface of this module to the MyQsys instance
3. Note the polarity of the signals: the JTAG interface only uses positive-logic, whereas the Avalon slave interface of the Qsys system has some negative-logic signals...
4. Use the provided “CreateData.m” Matlab script to create test data files
5. Use the provided “WriteData.tcl” TCL script to load data into the memory (the transfer takes between 2 and 5 minutes)
6. Read the source code of these 3 parts and make sure you understand what each does – ask if you don’t

Loading Data into SDRAM

1. Add the provided “VirtualJTAG_MM_Write.v” module to the project and create an instance in the top-level entity
2. Connect the Avalon interface of this module to the MyQsys instance
3. Note the polarity of the signals: the JTAG interface only uses positive-logic, whereas the Avalon slave interface of the Qsys system has some negative-logic signals...
4. Use the provided “CreateData.m” Matlab script to create test data files
5. Use the provided “WriteData.tcl” TCL script to load data into the memory (the transfer takes between 2 and 5 minutes)
6. Read the source code of these 3 parts and make sure you understand what each does – ask if you don’t

Loading Data into SDRAM

1. Add the provided “VirtualJTAG_MM_Write.v” module to the project and create an instance in the top-level entity
2. Connect the Avalon interface of this module to the MyQsys instance
3. Note the polarity of the signals: the JTAG interface only uses positive-logic, whereas the Avalon slave interface of the Qsys system has some negative-logic signals...
4. Use the provided “CreateData.m” Matlab script to create test data files
5. Use the provided “WriteData.tcl” TCL script to load data into the memory (the transfer takes between 2 and 5 minutes)
6. Read the source code of these 3 parts and make sure you understand what each does – ask if you don’t

Loading Data into SDRAM

1. Add the provided “VirtualJTAG_MM_Write.v” module to the project and create an instance in the top-level entity
2. Connect the Avalon interface of this module to the MyQsys instance
3. Note the polarity of the signals: the JTAG interface only uses positive-logic, whereas the Avalon slave interface of the Qsys system has some negative-logic signals...
4. Use the provided “CreateData.m” Matlab script to create test data files
5. Use the provided “WriteData.tcl” TCL script to load data into the memory (the transfer takes between 2 and 5 minutes)
6. Read the source code of these 3 parts and make sure you understand what each does – ask if you don’t

Loading Data into SDRAM

1. Add the provided “VirtualJTAG_MM_Write.v” module to the project and create an instance in the top-level entity
2. Connect the Avalon interface of this module to the MyQsys instance
3. Note the polarity of the signals: the JTAG interface only uses positive-logic, whereas the Avalon slave interface of the Qsys system has some negative-logic signals...
4. Use the provided “CreateData.m” Matlab script to create test data files
5. Use the provided “WriteData.tcl” TCL script to load data into the memory (the transfer takes between 2 and 5 minutes)
6. Read the source code of these 3 parts and make sure you understand what each does – ask if you don’t

Loading Data into SDRAM

1. Add the provided “VirtualJTAG_MM_Write.v” module to the project and create an instance in the top-level entity
2. Connect the Avalon interface of this module to the MyQsys instance
3. Note the polarity of the signals: the JTAG interface only uses positive-logic, whereas the Avalon slave interface of the Qsys system has some negative-logic signals...
4. Use the provided “CreateData.m” Matlab script to create test data files
5. Use the provided “WriteData.tcl” TCL script to load data into the memory (the transfer takes between 2 and 5 minutes)
6. Read the source code of these 3 parts and make sure you understand what each does – ask if you don’t

Injecting Data from SDRAM

- Write an injection module that connects to the Avalon interface of the MyQsys instance and injects data into your PWM module at 48.828 125 kSps
- Make use of the existing FIFO queue from the previous steps
- Use one of the on-board slide-switches to switch the Avalon interface between the VirtualJTAG_MM_Write module and your injection module
- If you used the Matlab and TCL scripts provided, the data is 8-bit unsigned integers, but the SDRAM provides them two bytes at a time
- When the switch is in the “loading” position, the injection module must reset it’s address to 0

Injecting Data from SDRAM

- Write an injection module that connects to the Avalon interface of the MyQsys instance and injects data into your PWM module at 48.828 125 kSps
- Make use of the existing FIFO queue from the previous steps
- Use one of the on-board slide-switches to switch the Avalon interface between the VirtualJTAG_MM_Write module and your injection module
- If you used the Matlab and TCL scripts provided, the data is 8-bit unsigned integers, but the SDRAM provides them two bytes at a time
- When the switch is in the “loading” position, the injection module must reset it’s address to 0

Injecting Data from SDRAM

- Write an injection module that connects to the Avalon interface of the MyQsys instance and injects data into your PWM module at 48.828 125 kSps
- Make use of the existing FIFO queue from the previous steps
- Use one of the on-board slide-switches to switch the Avalon interface between the VirtualJTAG_MM_Write module and your injection module
- If you used the Matlab and TCL scripts provided, the data is 8-bit unsigned integers, but the SDRAM provides them two bytes at a time
- When the switch is in the “loading” position, the injection module must reset it’s address to 0

Injecting Data from SDRAM

- Write an injection module that connects to the Avalon interface of the MyQsys instance and injects data into your PWM module at 48.828 125 kSps
- Make use of the existing FIFO queue from the previous steps
- Use one of the on-board slide-switches to switch the Avalon interface between the VirtualJTAG_MM_Write module and your injection module
- If you used the Matlab and TCL scripts provided, the data is 8-bit unsigned integers, but the SDRAM provides them two bytes at a time
- When the switch is in the “loading” position, the injection module must reset it’s address to 0

Injecting Data from SDRAM

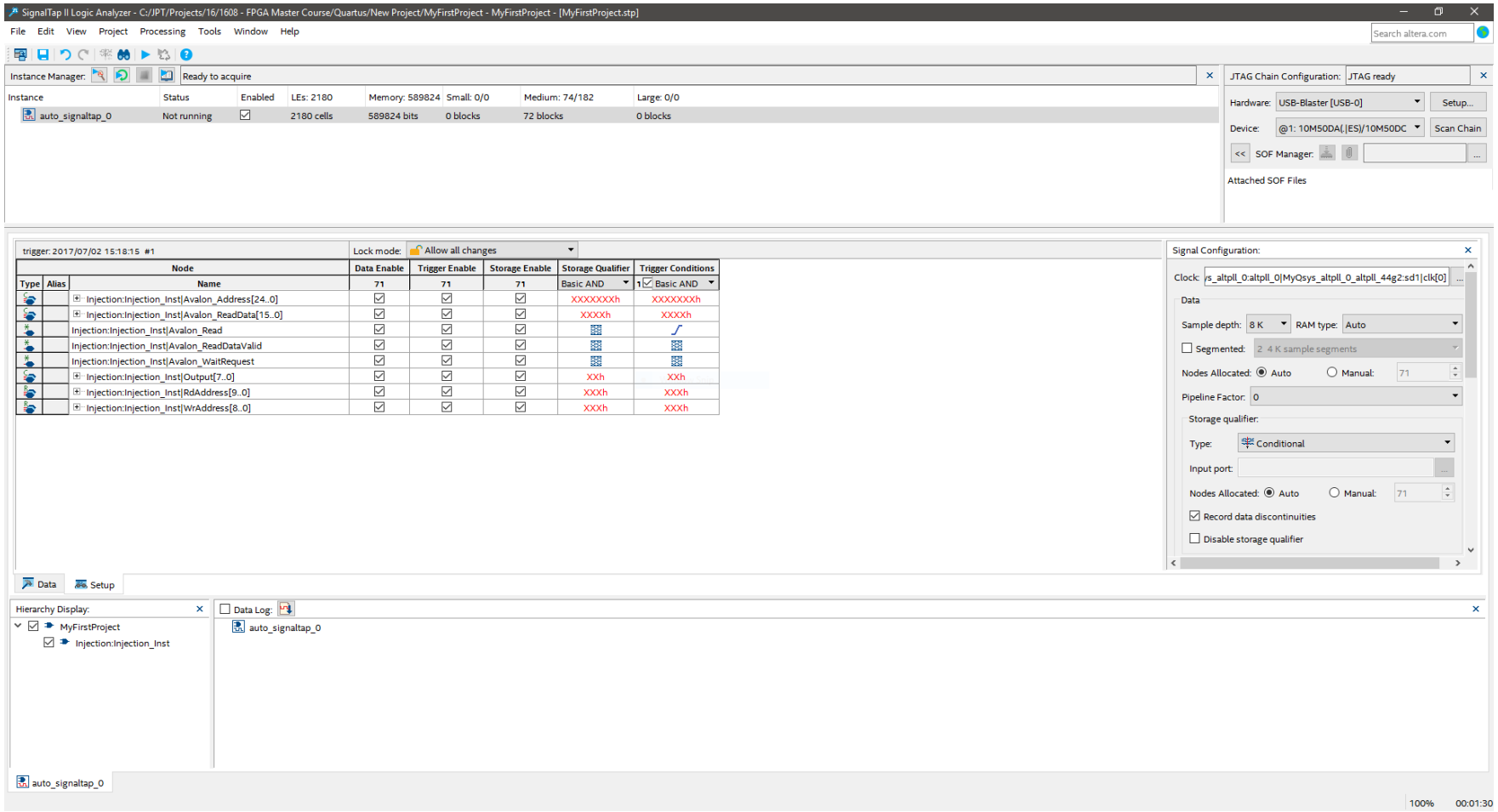
- Write an injection module that connects to the Avalon interface of the MyQsys instance and injects data into your PWM module at 48.828 125 kSps
- Make use of the existing FIFO queue from the previous steps
- Use one of the on-board slide-switches to switch the Avalon interface between the VirtualJTAG_MM_Write module and your injection module
- If you used the Matlab and TCL scripts provided, the data is 8-bit unsigned integers, but the SDRAM provides them two bytes at a time
- When the switch is in the “loading” position, the injection module must reset it’s address to 0

Signal-tap Logic Analyser

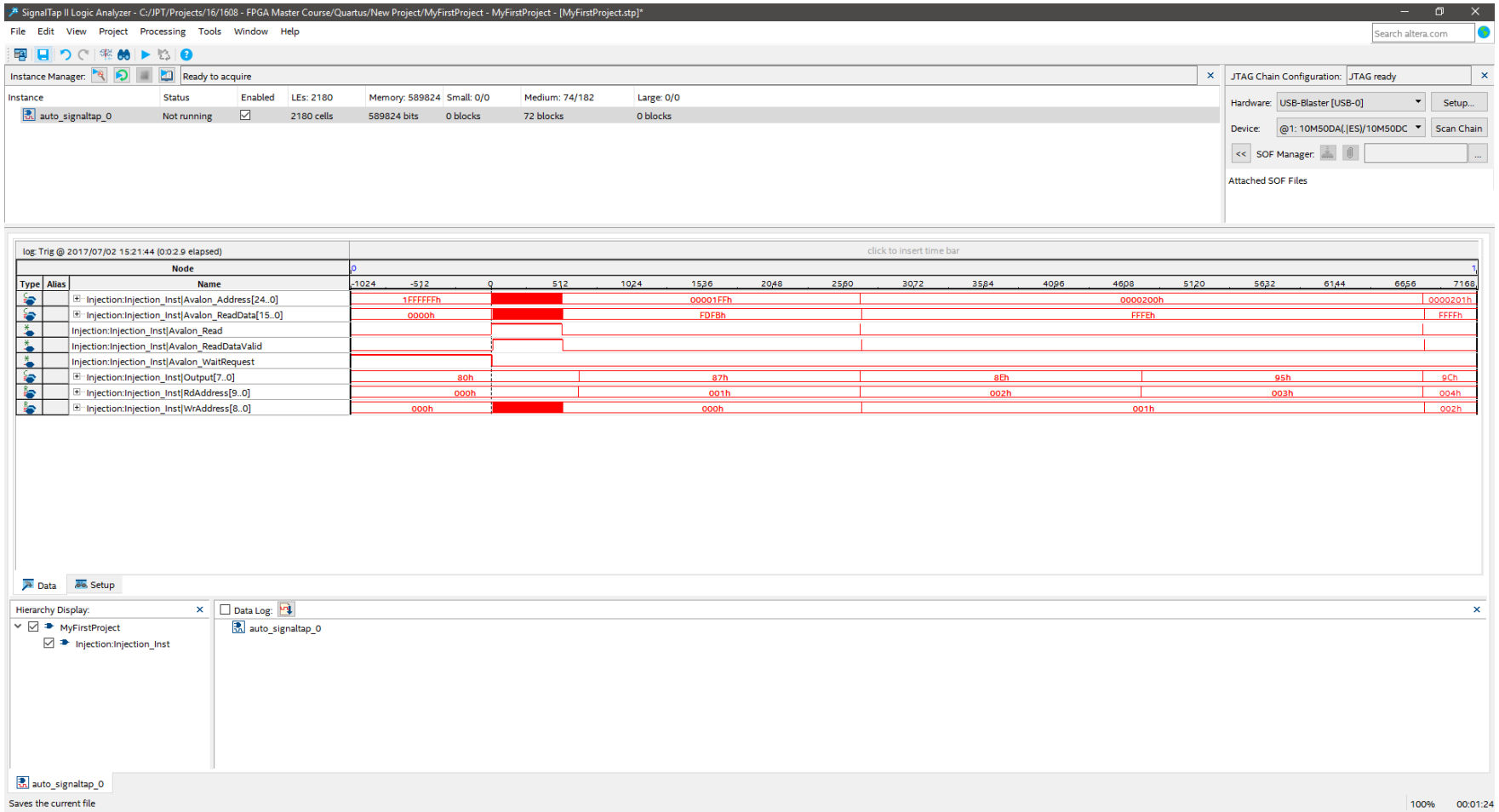
This is a good time to try out the Signal-tap logic analyser...

https://www.youtube.com/watch?v=vhkzxCEXuaA

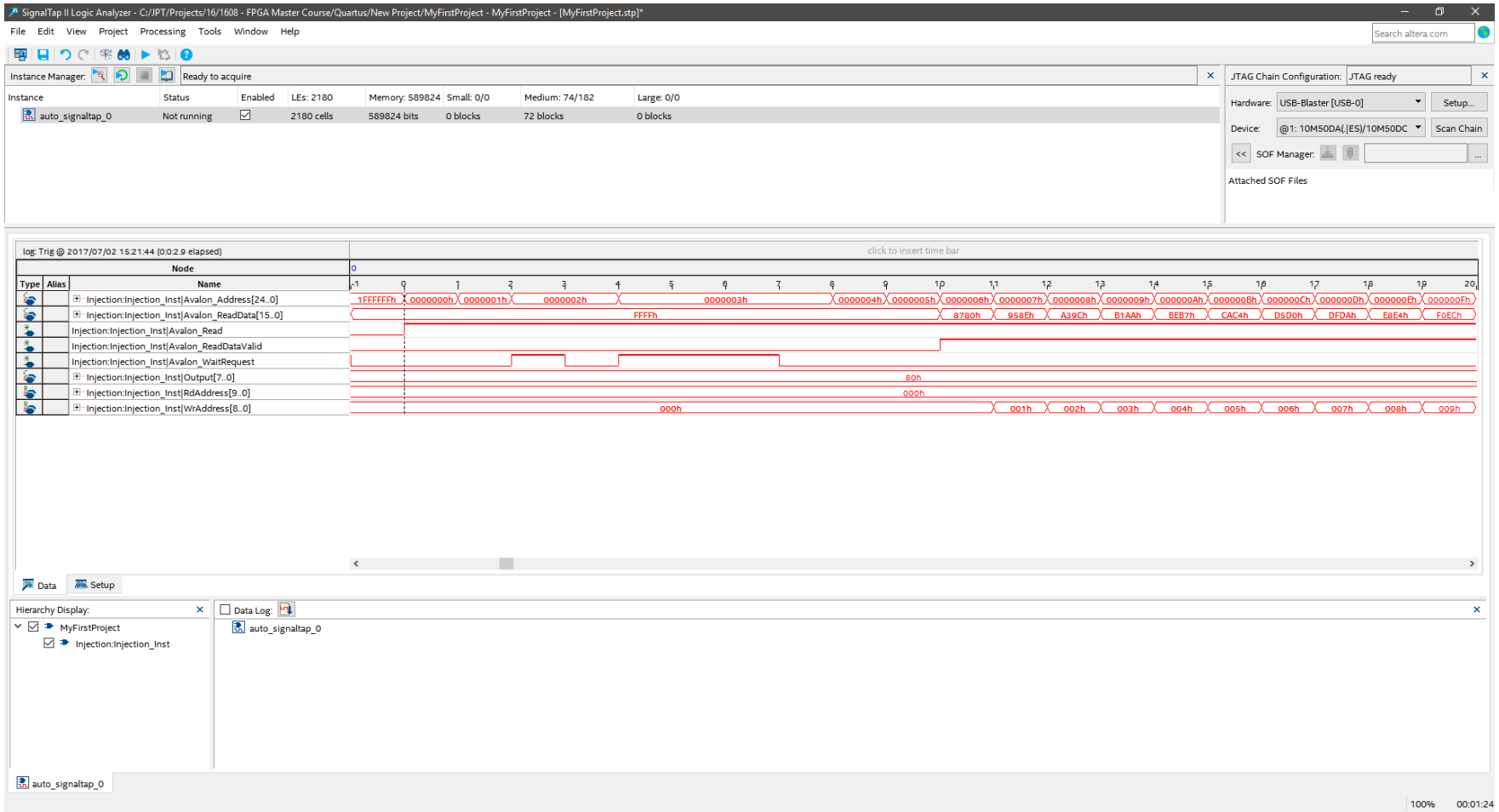
Signal-tap Logic Analyser



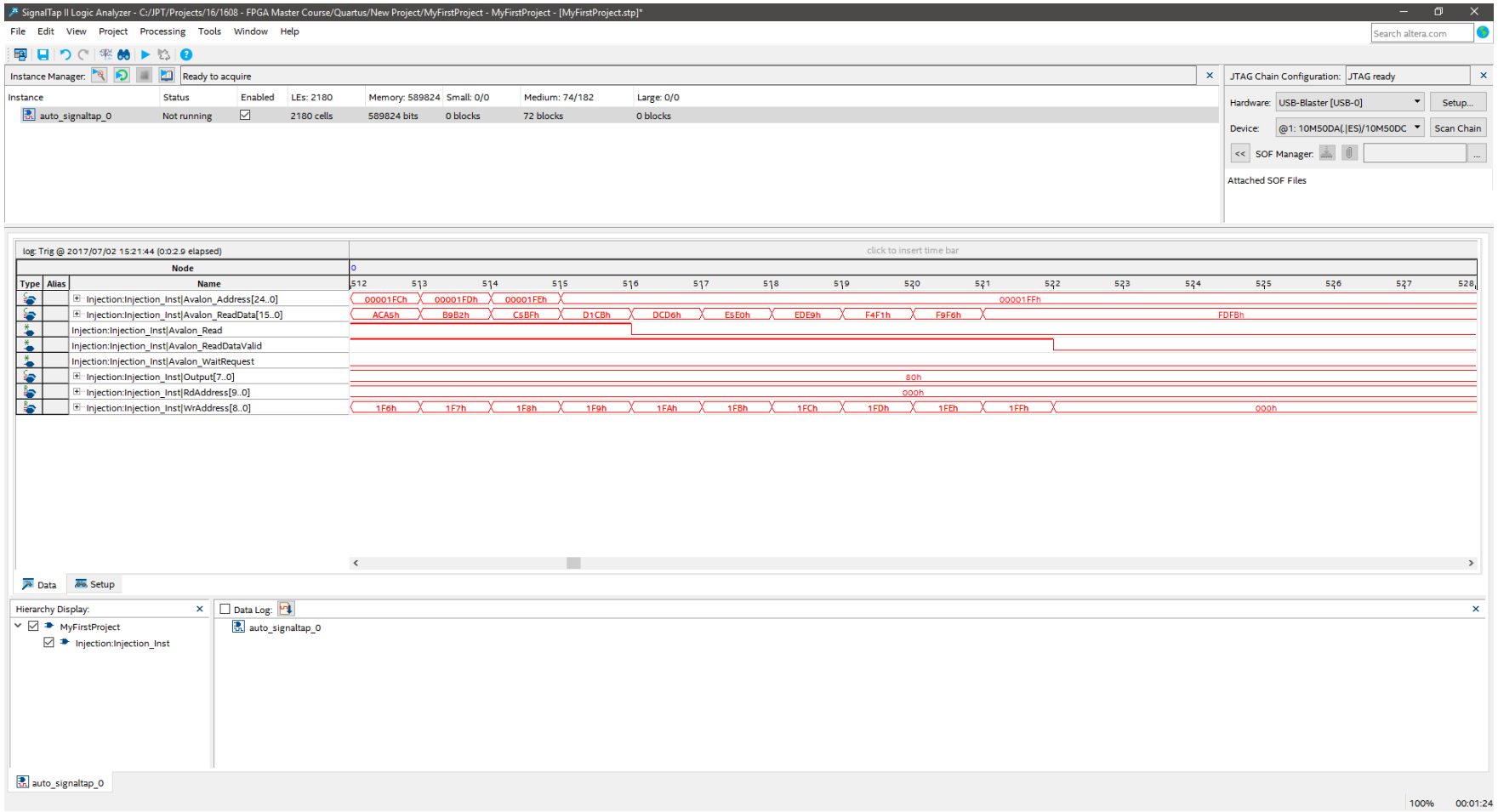
Signal-tap Logic Analyser



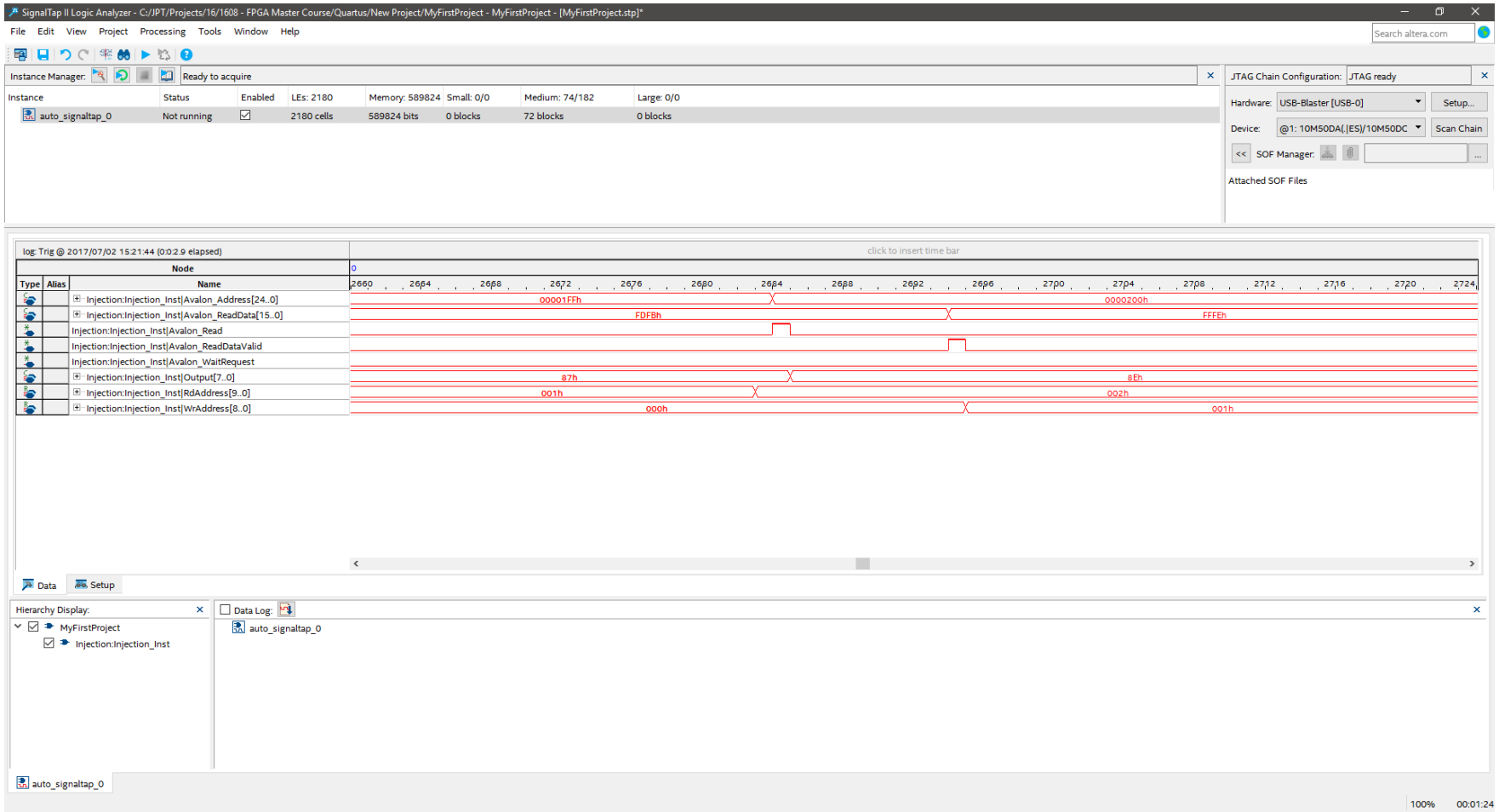
Signal-tap Logic Analyser



Signal-tap Logic Analyser



Signal-tap Logic Analyser



Outline

Altera Library

Verilog Processes

Finite State Machines

Timing Constraints

Injection Testing

Simulation

Practical

Appendix – High Resolution PWM

THE RADAR MASTERS COURSE

High Resolution PWM

- Assume that your signal is audio (20 kHz bandwidth)
- You want the PWM frequency at least 10 times the signal bandwidth $\Rightarrow$ 200 kHz
- Say you want 16-bit output resolution...
- You have to run the saw-tooth counter at 200 kHz $\times$ $2^{16} = 13.1072$ GHz
- That is a FAST clock!
- The solution...

High Resolution PWM

- Assume that your signal is audio (20 kHz bandwidth)
- You want the PWM frequency at least 10 times the signal bandwidth $\Rightarrow$ 200 kHz
- Say you want 16-bit output resolution...
- You have to run the saw-tooth counter at 200 kHz $\times$ $2^{16} = 13.1072$ GHz
- That is a FAST clock!
- The solution...

High Resolution PWM

- Assume that your signal is audio (20 kHz bandwidth)
- You want the PWM frequency at least 10 times the signal bandwidth $\Rightarrow$ 200 kHz
- Say you want 16-bit output resolution...
- You have to run the saw-tooth counter at 200 kHz $\times$ $2^{16} = 13.1072$ GHz
- That is a FAST clock!
- The solution...

High Resolution PWM

- Assume that your signal is audio (20 kHz bandwidth)
- You want the PWM frequency at least 10 times the signal bandwidth $\Rightarrow$ 200 kHz
- Say you want 16-bit output resolution...
- You have to run the saw-tooth counter at 200 kHz $\times$ $2^{16} = 13.1072$ GHz
- That is a FAST clock!
- The solution...

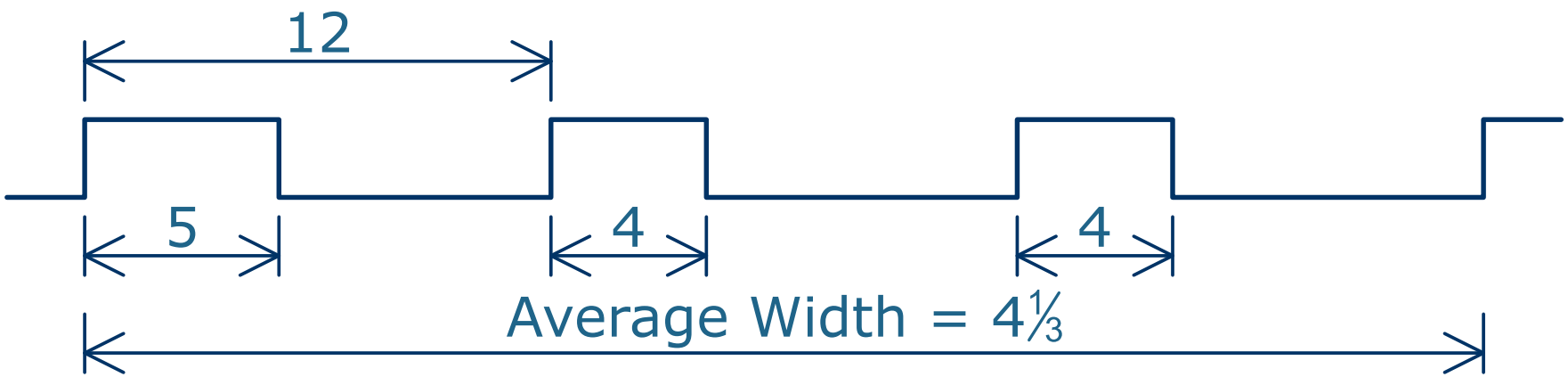
High Resolution PWM

- Assume that your signal is audio (20 kHz bandwidth)
- You want the PWM frequency at least 10 times the signal bandwidth $\Rightarrow$ 200 kHz
- Say you want 16-bit output resolution...
- You have to run the saw-tooth counter at 200 kHz $\times$ $2^{16} = 13.1072$ GHz
- That is a FAST clock!
- The solution...

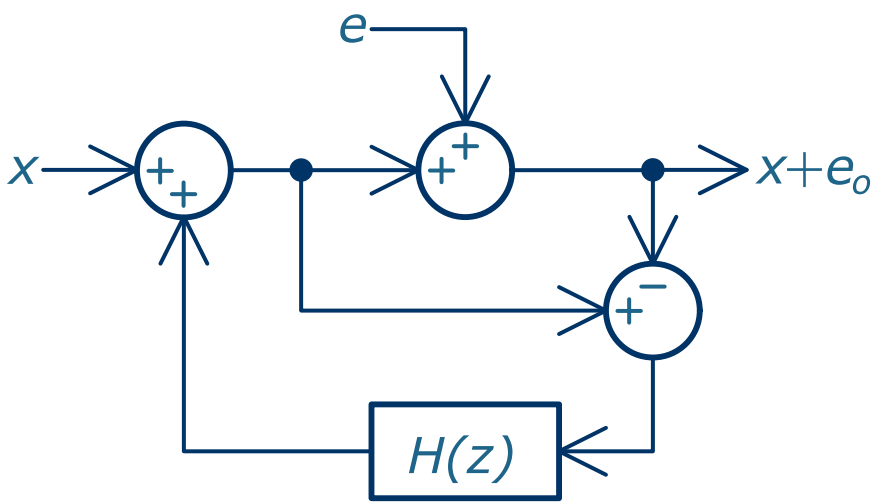
High Resolution PWM

- Assume that your signal is audio (20 kHz bandwidth)
- You want the PWM frequency at least 10 times the signal bandwidth $\Rightarrow$ 200 kHz
- Say you want 16-bit output resolution...
- You have to run the saw-tooth counter at 200 kHz $\times$ $2^{16} = 13.1072$ GHz
- That is a FAST clock!
- The solution...

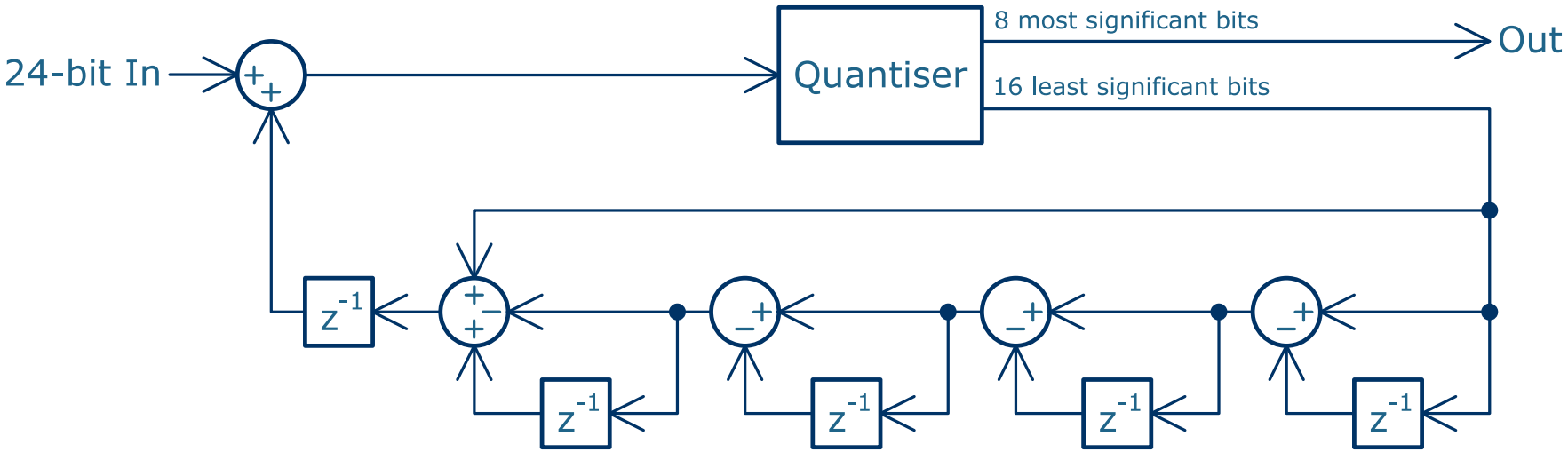
Noise Shaping



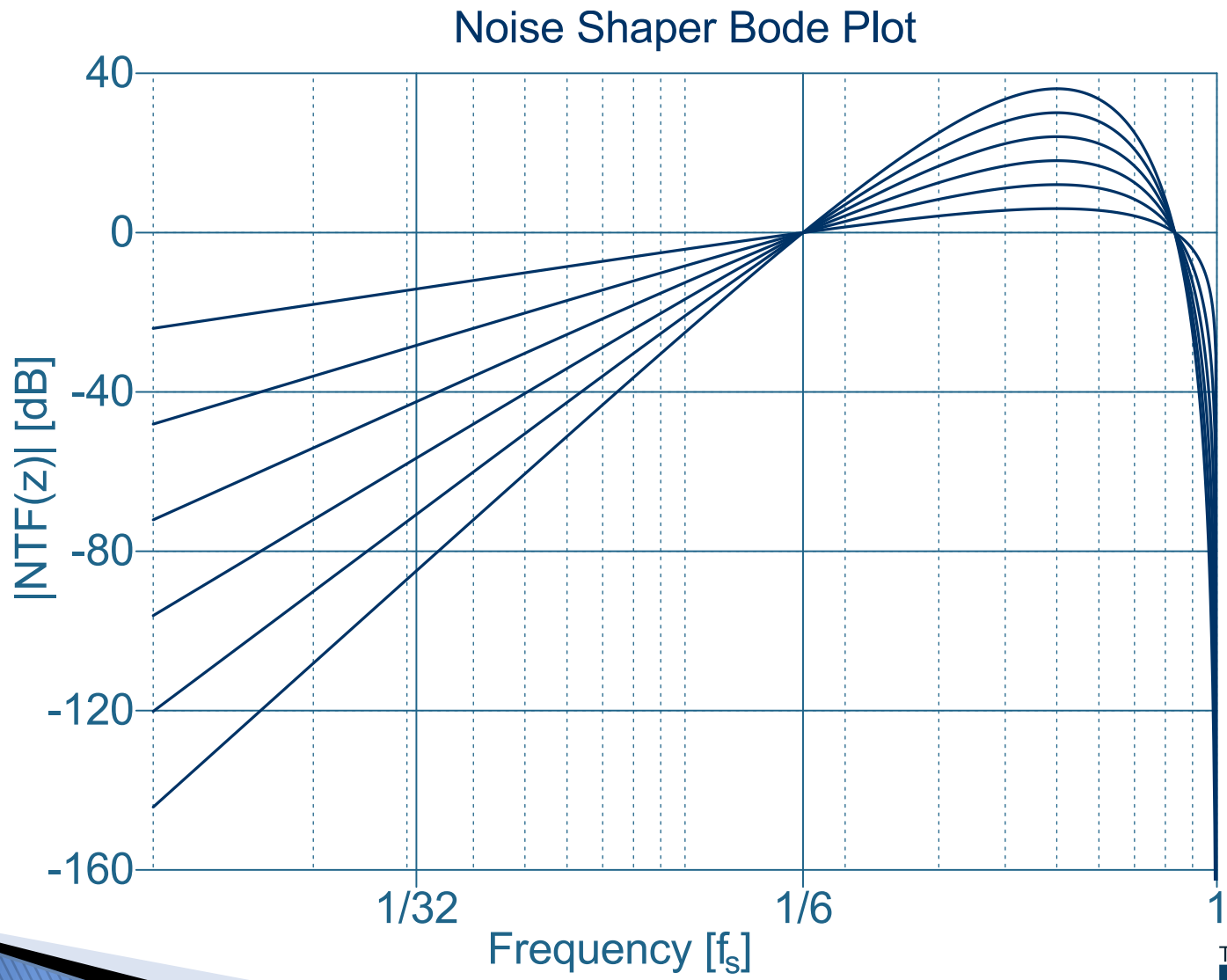
Noise Shaping



Noise Shaping



Noise Shaping



Select References

Stephen Brown and Zvonko Vranesic
Fundamentals of Digital Logic with Verilog Design, 2nd Edition
ISBN 978-0-07-721164-6

Merrill L Skolnik
Introduction to RADAR Systems
ISBN 978-0-07-288138-7

Mark A. Richards and James A. Scheer
Principles of Modern Radar: Basic Principles
ISBN 978-1-89-112152-4

Deepak Kumar Tala
World of ASIC
http://www.asic-world.com/

Jean P. Nicolle
FPGA 4 Fun
http://www.fpga4fun.com/

FPGA Development for Radar, Radio-Astronomy and Communications

THE RADAR MASTERS COURSE

Dept. Electrical Engineering, University of Cape Town
Private Bag, Rondebosch, 7701, South Africa

http://www.rrsg.uct.ac.za

Presented by John-Philip Taylor

Convened by Prof Daniel O'Hagan

Tutored by Stephen Paine and Randy Cheng

Day 2 – 18 July 2017