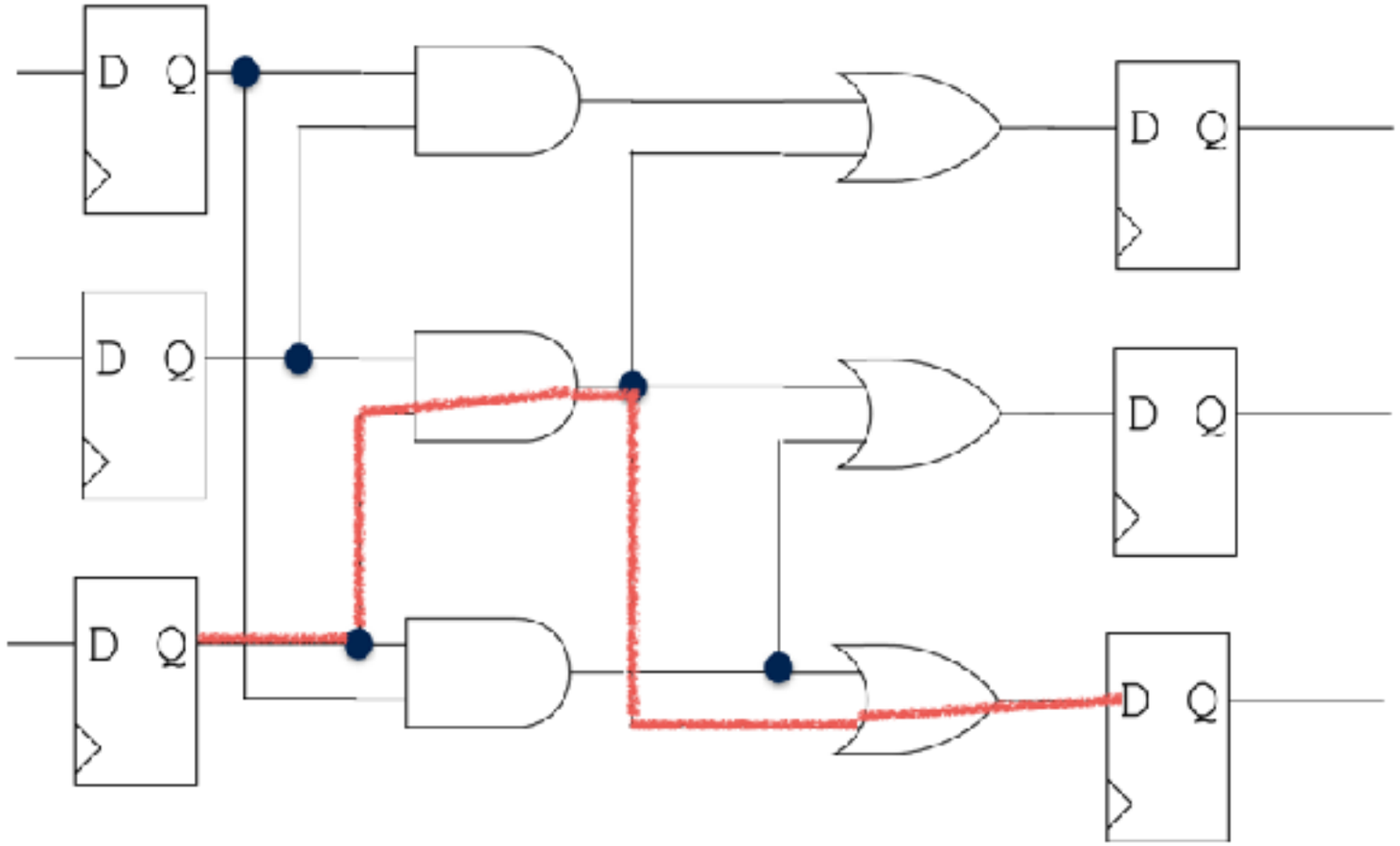
Timing

John Tramel
CSULB CECS 360



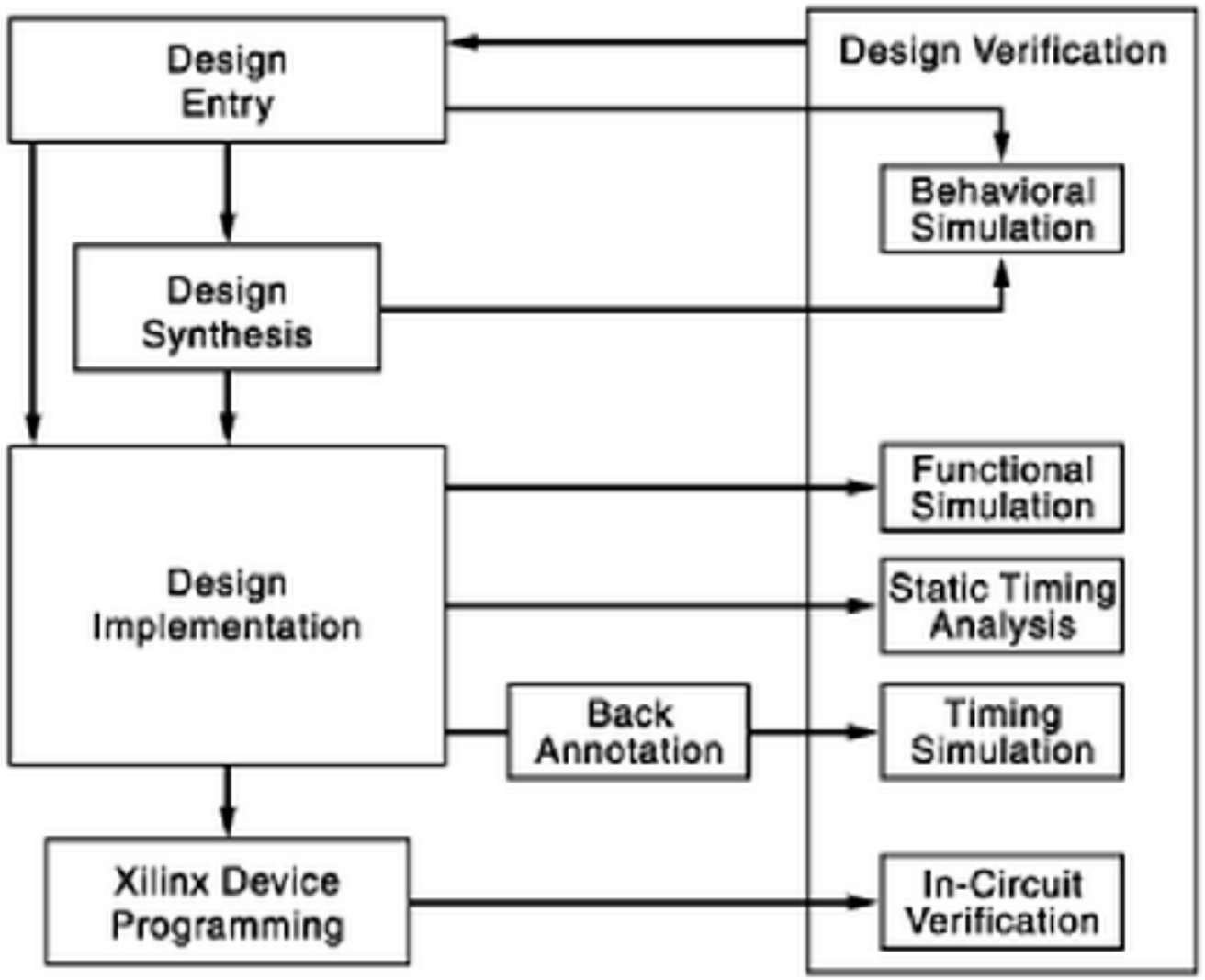
Introduction

- The following discussion was originally directed toward ASIC design. In performing modern digital design with FPGAs the concepts and discussions apply equally well.

- Successful implementation of digital design requires that the design be coded with ‘synthesizable HDL’, that a proper verification of the design be performed using a simulator, and once the design has been translated into gates the design needs to be analyzed in order to ensure that it will work at speed (meet timing).

FPGA Design Flow Overview

The ISE® design flow comprises the following steps: design entry, design synthesis, design implementation, and Xilinx® device programming. Design verification, which includes both functional verification and timing verification, takes places at different points during the design flow. This section describes what to do during each step. For additional details on each design step, click on a link below the following figure.



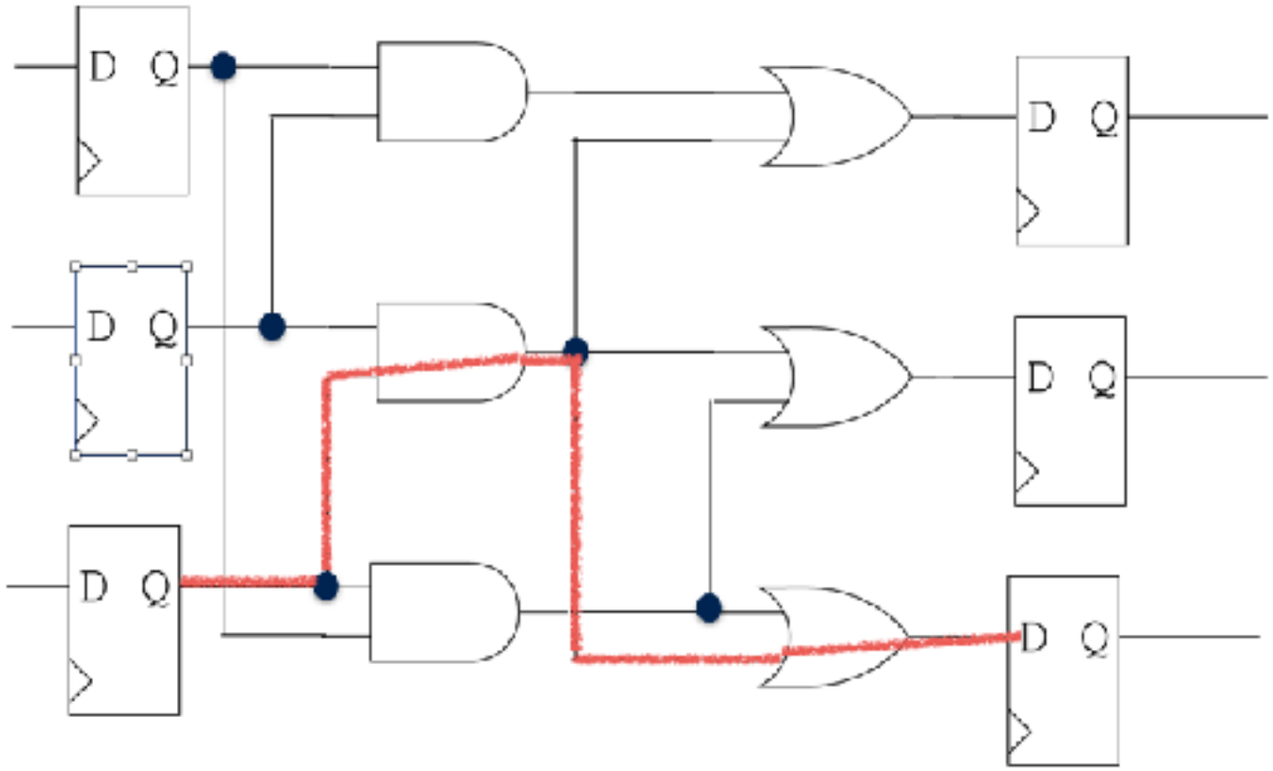
Design Flow Discussion

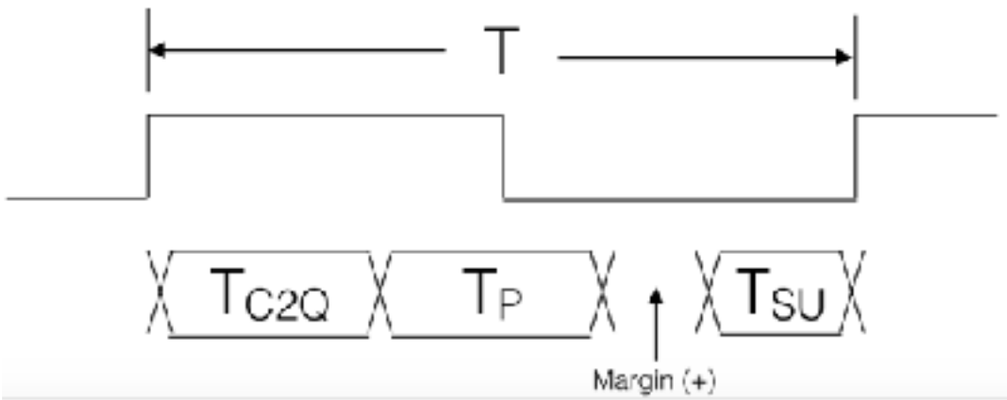
- The design flow documented by Xilinx has several distinct stages - Design, verification, implementation, programming

- Each of these stages requires detailed development in order to master the art of designing modern digital circuits

- The focus of this discussion is the step under design implementation referred to as Static Timing Analysis (STA)

The Need for Timing Analysis





- When every a clock occurs there is a finite amount of time from the clock edge until the Q on the output of the flop is stable. Then all of the Qs feeding the array of gates must propagate through, and then the input must be stable before the setup requirement.

- When the many thousands of paths that are produced within a digital design are considered the need for an automated approach to verifying that the design meets the given timing constraints becomes readily apparent

Timing Models

- When functionally verifying the behavior of a digital circuit using a simulator the designer relies on the tools to provide the logical abstractions, models, of the devices

- The functional models utilized for gates do not include any concept of timing. The flops by definition are only updated on the active edge of the clock so there is the passing of time inferred by waiting for a clock edge

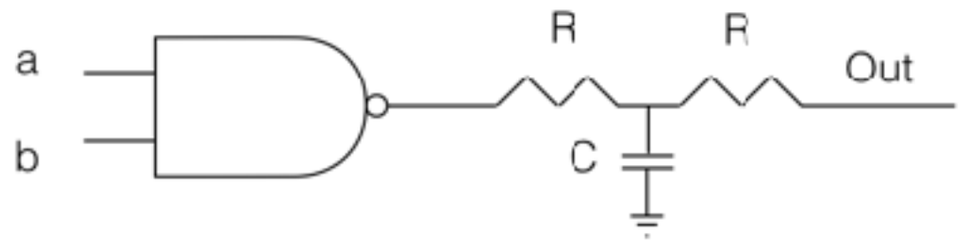
- When verifying the timing of a digital circuit the designer must also rely on the tools for the timing abstractions of the devices used as well as the interconnections between those devices

- The models utilized for timing analysis include the intrinsic delays of the devices themselves but must also provide a means to include the timing delays incurred by the interconnects in the design

Accurate Model Considerations

Out = !(a & b);

Logic Statement
for NAND gate



- Above may be seen the representation of a NAND gate utilized when modeling HDL. On the right is a representation of the resultant circuit that demonstrates that not only will there be a delay through the gate itself but there will be added delay generated by the charging and discharging of the parasitic capacitor through the resisters.

- The accuracy of the timing model is essential to the successful implementation of the final device.

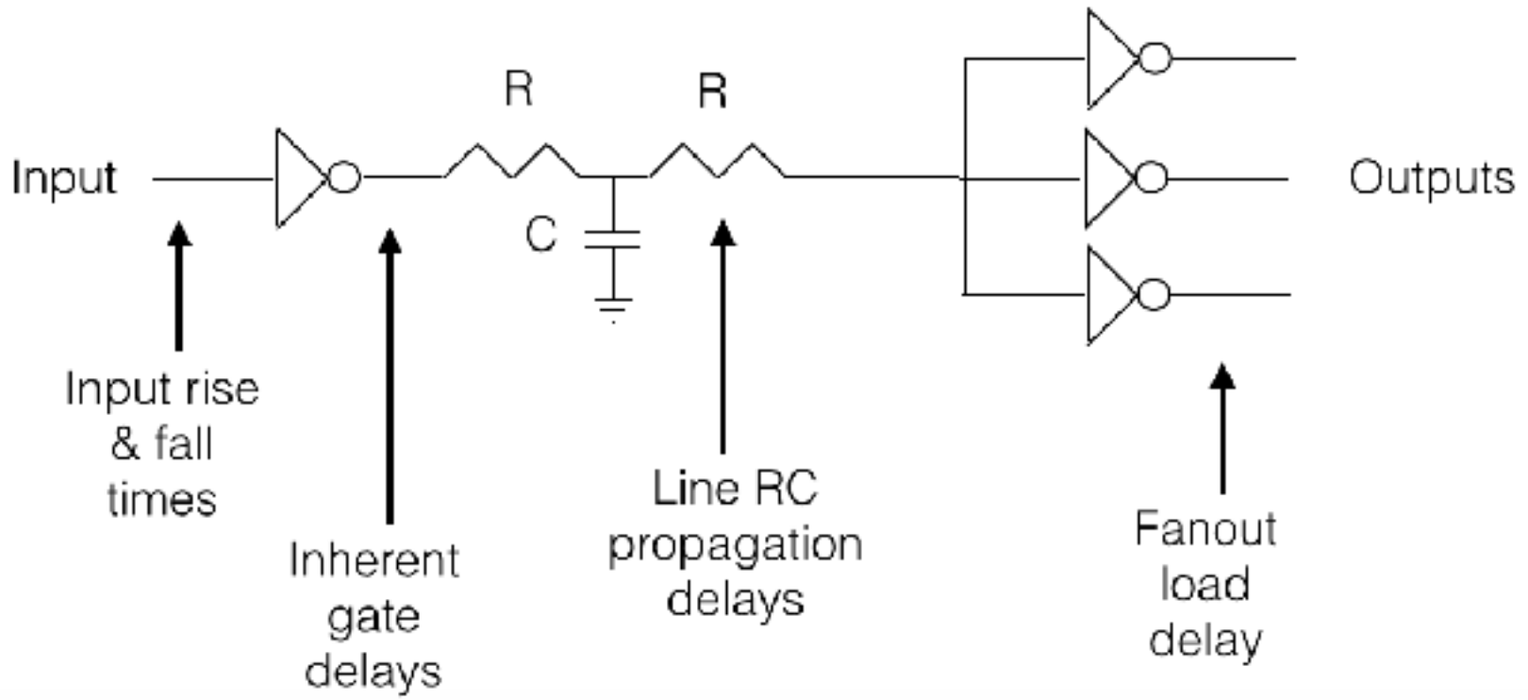
Timing Models Accuracy

- In order for all contributors to be considered the final timing analysis should be done on the design that has not only been synthesized (mapped to gates) but has also been placed and routed.

- The ‘post-map’ analysis performed by Xilinx tools is still an approximation of the final timing. This is because all of the elements that comprise the design have yet to be placed onto the FPGA itself and interconnected (Placed And Routed (PAR))

- PAR may be performed considering timing (timing-driven PAR) or ignoring timing (non-timing-driven PAR)

Components of Circuit Delay



- The switching delay begins with the output driving the circuit. It will require time to reach a zero or a one. The gate delays are characterized by the manufacturer. The trace (path) delay is most accurate at post layout. The arrival of the desired value on the output is also affected by the number of loads on the outputs (fanout).

Pre-layout Timing

- HDL modeling may be done at two different levels: RTL and gate level

- As mentioned earlier, RTL models are written to implement the desired logical functionality with no regard to the physical delays through them

- Gate level models, also known as library models, are written so that the functionality is defined with Verilog primitives (built-in and user defined).

- The library models have the ‘hooks’ to add physical delays to the models (specify blocks)

RTL Level Modeling

```
reg out;

always @(*)
    out = (~I1 & ~I2);
```

- When writing HDL at the RTL level the only consideration of time is the clock period

- This is brought into consideration because the flops that hold the state in the design are always updated on the same edge of the clock (ie. posedge clk)

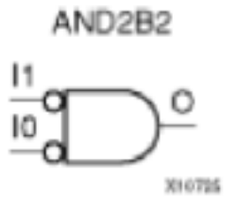
- The functions defined (gates) never have any reference to time are are assumed (for the time being) to be ideal circuits (no propagation delay)

Gate Level Modeling

XILINX

AND2B2

Primitive: 2-Input AND Gate with Inverted Inputs




```
module test (A,B,Y);

input  A, B;
output Y;

AND2B2 and2b2_1 (
   .I1(A),
   .I2(B),

   .O(Y)
   );

endmodule
```

- Within the Spartan 3E library there are many models that may be instantiated to build a design
- In 201 you learned how to create a schematic by placing the symbols on a page and then interconnecting them as desired
- Verilog allows you to essentially create the same low-level design by instantiating the library cells directly into the Verilog model
- When the synthesis tool generates the netlist it will produce a design with the same logic functionality that is composed of interconnected library model instantiations

Timing in RTL Code

- For modeling purposes it is possible to assign delays to RTL code

- Examples of RTL code that may have delays are Bus Models (BFMs), Memory Models, I/O

- This type of modeling is performed when it is necessary to interface to a block that is not modeled in RTL. An example is the external SRAM on the Diligent board that the FPGA is capable of communicating with.

- The designer wants to know if the interface they have designed to communicate to the memory will work before the FPGA is programmed and the interface is tested.

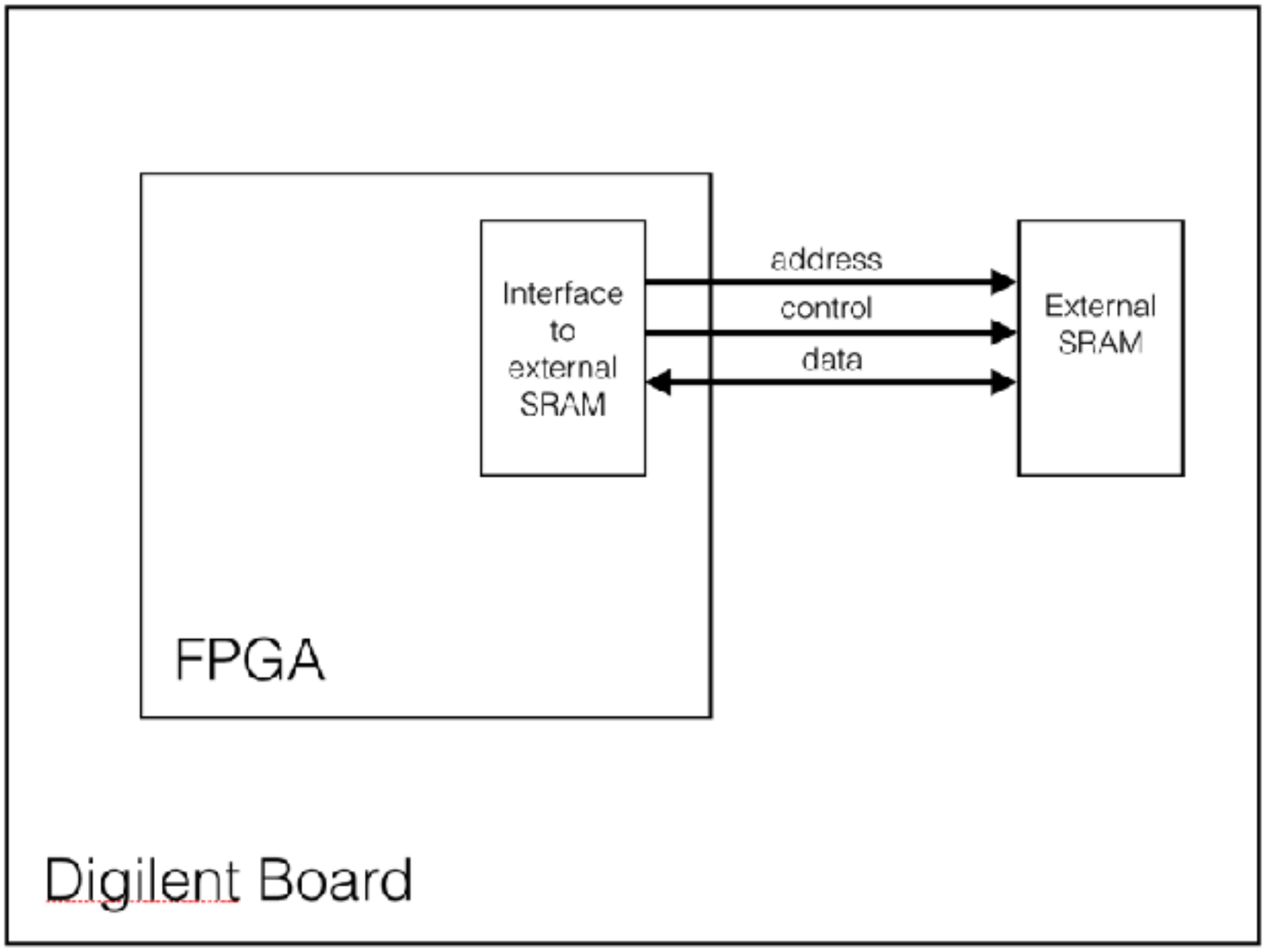
Bus Functional Model (BFM)

```
//
//  File Name:  cellram.v
//      Model:  BUS Functional
//  Simulator:  Model Technology
//
// Dependencies:  cellram_parameters.vh
//
//     Author:  Micron Technology, Inc.
//      Email:  modelsupport@micron.com
//
// Description:  Micron 128Mb CellularRAM 2.0 (Async / Burst)
//
//  Disclaimer: This software code and all associated documentation, comments
//              or other information (collectively "Software") is provided
//              "AS IS" without warranty of any kind. MICRON TECHNOLOGY, INC.
//              ("MTI") EXPRESSLY DISCLAIMS ALL WARRANTIES EXPRESS OR IMPLIED,
//              INCLUDING BUT NOT LIMITED TO, NONINFRINGEMENT OF THIRD PARTY
//              RIGHTS, AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS
//              FOR ANY PARTICULAR PURPOSE. MTI DOES NOT WARRANT THAT THE
//              SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF
//              THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE. FURTHERMORE,
//              MTI DOES NOT MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE
//              RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS,
//              ACCURACY, RELIABILITY, OR OTHERWISE. THE ENTIRE RISK ARISING OUT
//              OF USE OR PERFORMANCE OF THE SOFTWARE REMAINS WITH YOU. IN NO
//              EVENT SHALL MTI, ITS AFFILIATED COMPANIES OR THEIR SUPPLIERS BE
//              LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL, OR
//              SPECIAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS
//              OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION)
//              ARISING OUT OF YOUR USE OF OR INABILITY TO USE THE SOFTWARE,
//              EVEN IF MTI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
//              Because some jurisdictions prohibit the exclusion or limitation
//              of liability for consequential or incidental damages, the above
//              limitation may not apply to you.
//
//              Copyright 2006 Micron Technology, Inc. All rights reserved.
//
// Rev  Author         Date          Changes
// ---  -------------------------  ------------------------------------------
// 1.0  SU             05/10/2001  - Initial release
// 1.1  MH             05/15/2004  - Fixed problem with row crossing
//                                   functionality.
// 1.2  PF             10/20/2004  - Changed default value of
//                                   operating_mode to match datasheet.
//                                   Conditioned sync operation on setting
//                                   operating_mode
// 1.3  dritz          01/20/2005  - Updated parameters, parameterized the model, corrected a few bugs.
//                                     populated BCR, RCR
//                                     added BIOR feature
//                                     fixed burst interrupt on ADV#
//                                     added CR1.0 1.5 features
// 1.4  dritz          10/04/2005  - Fixed sync 2 async transition
// 1.5  dritz          11/21/2005  - o_wait on async corrected.
// 1.6  dritz          02/15/2005  - Row boundary crossing fixed.
// 1.7  dritz          01/21/2005  - Fixed burst read issue (data 1 clk too late with 0 setup)
// 1.8  jmk            05/30/2005  - Added CR2.0 features
// 1.9  jmk            05/07/2005  - Added tracing checks.
//                                 - Added delays on output signals.
//                                 - Added associative memory array.
// 2.1  jmk            05/14/2005  - Added cellram_partdelay module to model
//                                     delays on the read data path.
//                                 - DIOR[19:16] returns 2'b01 during reads.
//                                 - Added tCEM checking on WE#
// 2.2  jmk            05/29/2005  - Added support for x16 part width.
// 3.0  jmk            11/03/2005  - Added ASYNC/PAGE support and ZZ# pin.
//                                 - Added error message for ADV# low for multiple clocks.
//                                 - Entering PAR will corrupt memory.
//                                 - Added tAVZ and tALZ for Numonyx parts.
// 3.2  jmk            03/30/2007  - tHZ, tOHZ, tWHZ, tCHZ minimum = 0
//                                 - tCEW_MIN and tCEW_MAX are checked
//                                 - BYS input port replaced with LB#, UB#
//                                 - Added data corruption when CE# is deasserted prior to the first data transfer.
//                                 - Added checking for illegal burst interruption.
//                                 - Fixed WAIT assertion at end of row during burst write for Cellram 1.5
//                                 - Fixed tAVW checking on ADDR and CRE.
//                                 - Changed the way tCW was checked.
//                                 - Added checking for illegal page mode rmode and writes.
// 3.3  jmk            02/09/2007  - Added Cellram 1.0 features.
// 3.4  jmk            04/13/2007  - Removed tCO, tAADV, and tBA requirement during variable latency mode.
// 3.5  jmk            05/30/2007  - Added tVPH checking.
// 3.6  jmk            10/05/2007  - Added support for 8Mb devices (ADDR_BITS=19)
// 3.7  jmk            01/30/2005  - Fixed message for operating mode
// 3.8  jmk            02/29/2005  - Fixed Cellram 1.0 LOW/LOW operation during reads.
//////////////////////////////////////////////////////////////////////////////

// DO NOT CHANGE THE TIMESCALE
// MAKE SURE YOUR SIMULATOR USE "PS" RESOLUTION
`timescale 1ns / 1ps
```

- The BFM header shown here is for the memory found on the Nexys 3 board from Digilent
- The point of this slide is to demonstrate from the comments the emphasis put upon proper switching characteristics of the model
- The BFM should behave functionally (limited) and exhibit the proper switching characteristics of the device it is modeling

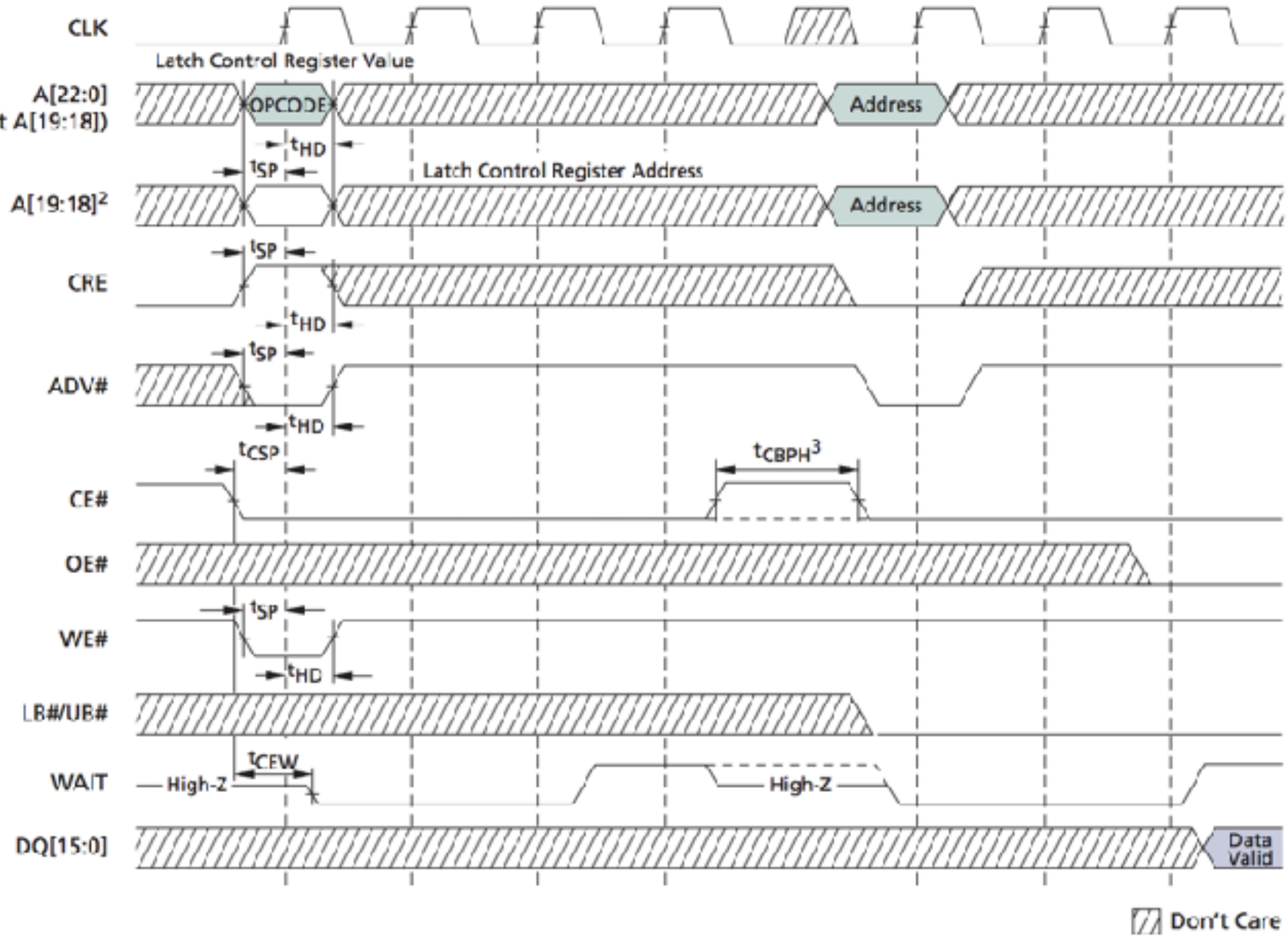
Timing in RTL Code



- During simulation the behavior of the external SRAM is captured in a Bus Functional Model for simulation

Bus Functional Model Requirement

Figure 13: Configuration Register WRITE, Synchronous Mode Followed by READ ARRAY Operation



- External interfaces have complex timing that must be met if the designer wants their FPGA to interface properly

Bus Functional Model Design

```
`ifdef RDIMM
    initial if (DEBUG) $display("%m: RDIMM");

    input                   ck     ;
    input                   ck_n   ;
    input             [3:0] s_n    ;
    inout             [7:0] cb     ;
    input             [2:0] sa     ; // no connect

    wire              [1:0] rck    = {2{ck}};
    wire              [1:0] rck_n  = {2{ck_n}};
    reg               [3:0] rs_n   ;
    reg                     rras_n ;
    reg                     rcas_n ;
    reg                     rwe_n  ;
    reg               [2:0] rba    ;
    reg              [15:0] raddr  ;
    reg               [3:0] rcke   ;
    reg               [3:0] rodt   ;

always @(negedge reset_n or posedge ck) begin
    if (!reset_n) begin
        rs_n   <= #(500) 0;
        rras_n <= #(500) 0;
        rcas_n <= #(500) 0;
        rwe_n  <= #(500) 0;
        rba    <= #(500) 0;
        raddr  <= #(500) 0;
        rcke   <= #(500) 0;
        rodt   <= #(500) 0;
    end else begin
        rs_n   <= #(500) s_n  ;
        rras_n <= #(500) ras_n;
        rcas_n <= #(500) cas_n;
        rwe_n  <= #(500) we_n ;
```

- The Verilog for a BFM looks similar to the HDL that we design but closer inspection reveals that there are added delays included along with timing constraint checks that will notify the user if any of the interface signals (from FPGA design) are switching improperly

- This approach makes integration with the hardware much easier

Verilog Language Capabilities Include Specifying Delays

- Always remember that Verilog is a general purpose digital modeling language

- There are many capabilities of the language that are not utilized when designing RTL

- When creating test environments, including BFMs, it is acceptable to utilize all of the capabilities of the language

- One example of such a capability is including delays in the definition of the Verilog code

Specifying Delays

- When modeling for synthesis we will never model a physical delay with a continuous assignment
- When writing test benches or BFMs it is perfectly acceptable to do so
- Delays may be specified using both continuous assignments and procedural assignments as well
- Delays may be a set period of time - ‘#10’ or it may be a delay for an event ‘@(posedge acknowledge);

Specifying Delays - Continuous Assignments

```
1   `timescale 1ns/10ps
2
3   module doit (a, b, c, y);
4
5   input a, b, c;
6
7   output y;
8
9   wire y;
10
11  assign #5 y = a | b | c;
12
13  endmodule
```

- Delays in a continuous assignment may be interpreted as “when any of the inputs switch, wait 5ns and update the output”

Specifying Delays - Continuous Assignments Considering the Behavior when Simulating

```
module doit_tb;

   // Inputs
   reg a;
   reg b;
   reg c;

   // Outputs
   wire y;

   // Instantiate the Unit Under Test (UUT)
   doit uut (
      .a(a),
      .b(b),
      .c(c),
      .y(y)
   );

   initial begin
      // Initialize Inputs
      a = 0;
      b = 0;
      c = 0;

      // Wait 100 ns for global reset to finish
      #50 a = 1;
      #3  a = 0;

      #50 b = 1;
      #3  b = 0;

      #50 c = 1;
      #3  c = 0;

      #50 a = 1;
          b = 1;
          c = 1;


   end

endmodule
```

- The test bench to the left will switch a, b and c for only 3 ns and then later sets them and leaves them set
- Can you explain the behavior of the simulation?

Explanation

- How the simulator works should be understood and may be controlled to behave the way the designer needs it to

- In the preceding example the inputs switch 0-1-0 in 3 ns which is less than the 5 ns delay imposed by the continuous assignment

- **Inertial delay** models only propagate signals to an output after the input signals have remained unchanged (been stable) for a time period equal to or greater than the propagation delay of the model. If the time between two input changes is shorter than a procedural assignment delay, a continuous assignment delay, or gate delay, a previously scheduled but unrealized output event is replaced with a newly scheduled output event.

- **Transport delay** models propagate all signals to an output after any input signals change. Scheduled output value changes are queued for transport delay models.

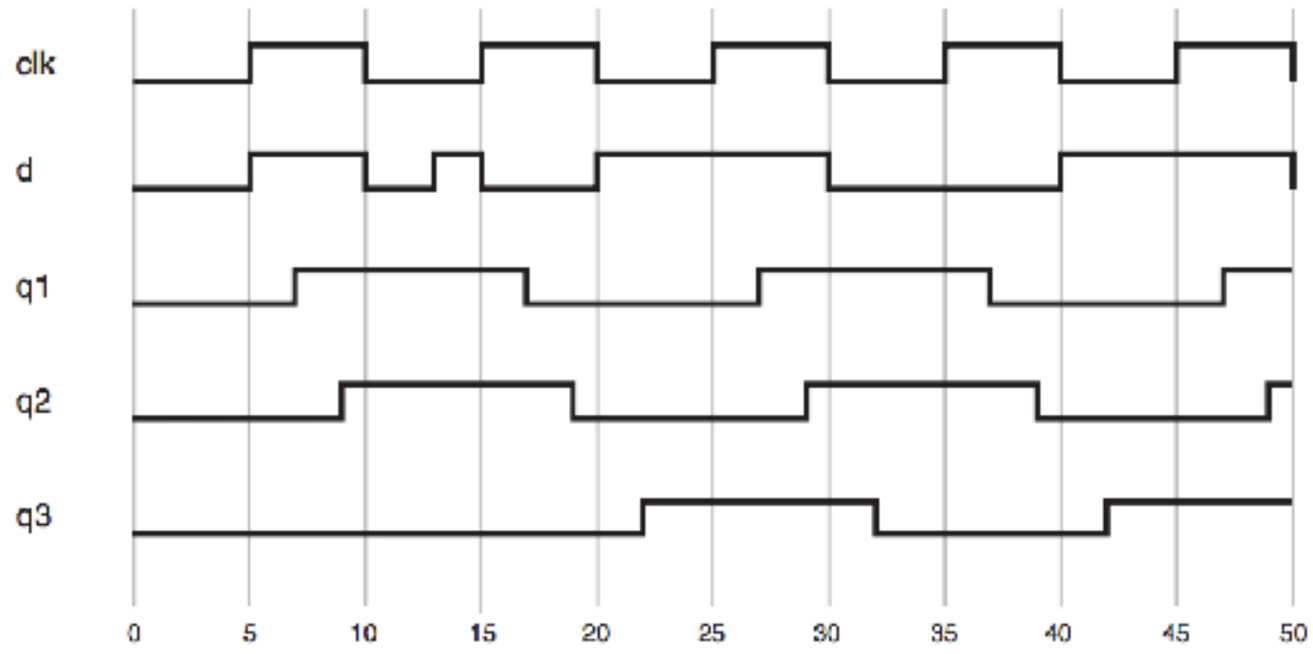
Specifying Delays - Procedural Assignments

- Within a procedural block, designated by ‘always’ or ‘initial’, there are two types of assignment operators that are dealt with differently by the simulator

- A ‘blocking assignment operator (=)’ means that the simulator is blocked from moving on to any subsequent statements until the present assignment is completed

- In the following example the simulator will wait for the positive edge of the clock, then wait 2 time units, evaluate d, and assign q1, wait 2 more time units, evaluate d, and assign q2, and then wait 3 time units, evaluate d, and assign q3.

```
always @(posedge clk)
  begin
  #2 q1 = d;
  #2 q2 = d;
  #3 q3 = d;
  end
```



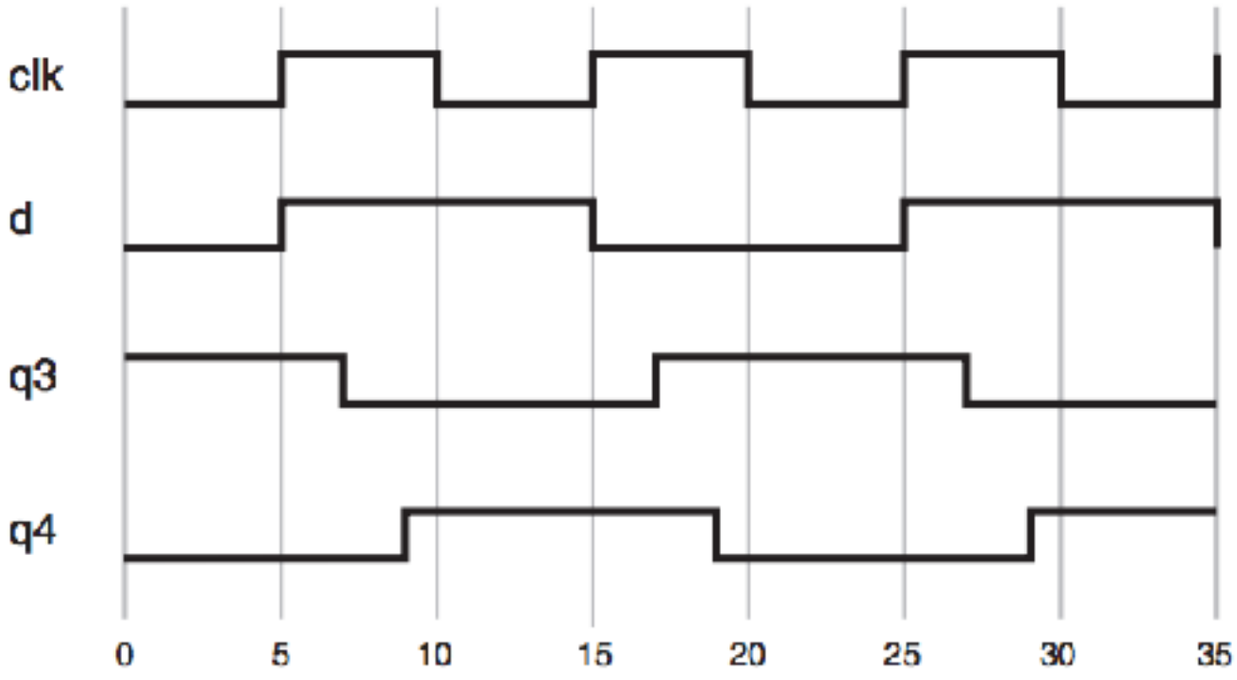
Specifying Delays - Procedural Assignments - Intra-Assignment Delays

- An intra-assignment delay will, upon execution, immediately evaluate the input signals and determine the value of the output signal. It will then wait the specified time and update the LHS

- The behavior is not intuitive and should be avoided unless thoroughly understood

```
always @(posedge clk)
begin
    q3 = #2 d;
    q4 = #2 d;
end

always @(posedge clk)
begin
    d <= ~d;
end
```



Specifying Delays - Procedural Assignments - Nonblocking Assignments

- When modeling sequential logic, flops, you should always utilize the nonblocking procedural assignment operator ‘<=‘

- This causes the simulator to evaluate the RHS of all expressions and then assigning the results to the LHS

- This allows the designer to model the digital circuit with the assurance that the synthesized gate-level representation of the design will behave the same as the simulation. This is a **major goal** when performing digital simulation - we want the simulation to behave exactly the way that the hardware will behave

The Verilog Specify Block

- In modeling digital circuits with Verilog we have stressed the importance of the **module**. Within the module we have the name, port list, input/output declarations, and the definition of variables. I refer to this as the **header** portion of the module.

- The second (and last so far) section of the module has been the declaration of the functionality of the module. We have seen that the functionality may be 1) directly instantiated modules, 2) continuous assignments, or 3) procedural blocks. I refer to this as the **functionality** portion of the module.

- The third available option for a module is the specify block (**specify/endspecify)** where the switching characteristics of the module are defined.

- Specify blocks are found in libraries and Bus Functional Models.

Specify Block Delays

- Delays in a specify block are defined with respect to module inputs and module outputs
- The specify block allows the definition of delays both as a strict propagation delay or as delays dependent on the values of other inputs
- The specify block is also the home for Verilog timing constraints that enforce timing relationships (setup/hold for example) and will notify the user when one of the timing constraints has been violated.

Specify Block Example

- Assume a synchronous memory model is to be written
- The memory model will have to deal with two issues: 1) the correct functionality of the memory and 2) ensuring that the interface timing to the memory has been implemented properly
- The functionality will be defined using the complete shopping list of Verilog constructs (not synthesized)
- The interface timing will be enforced through the specify block included in the memory model

Memory Description

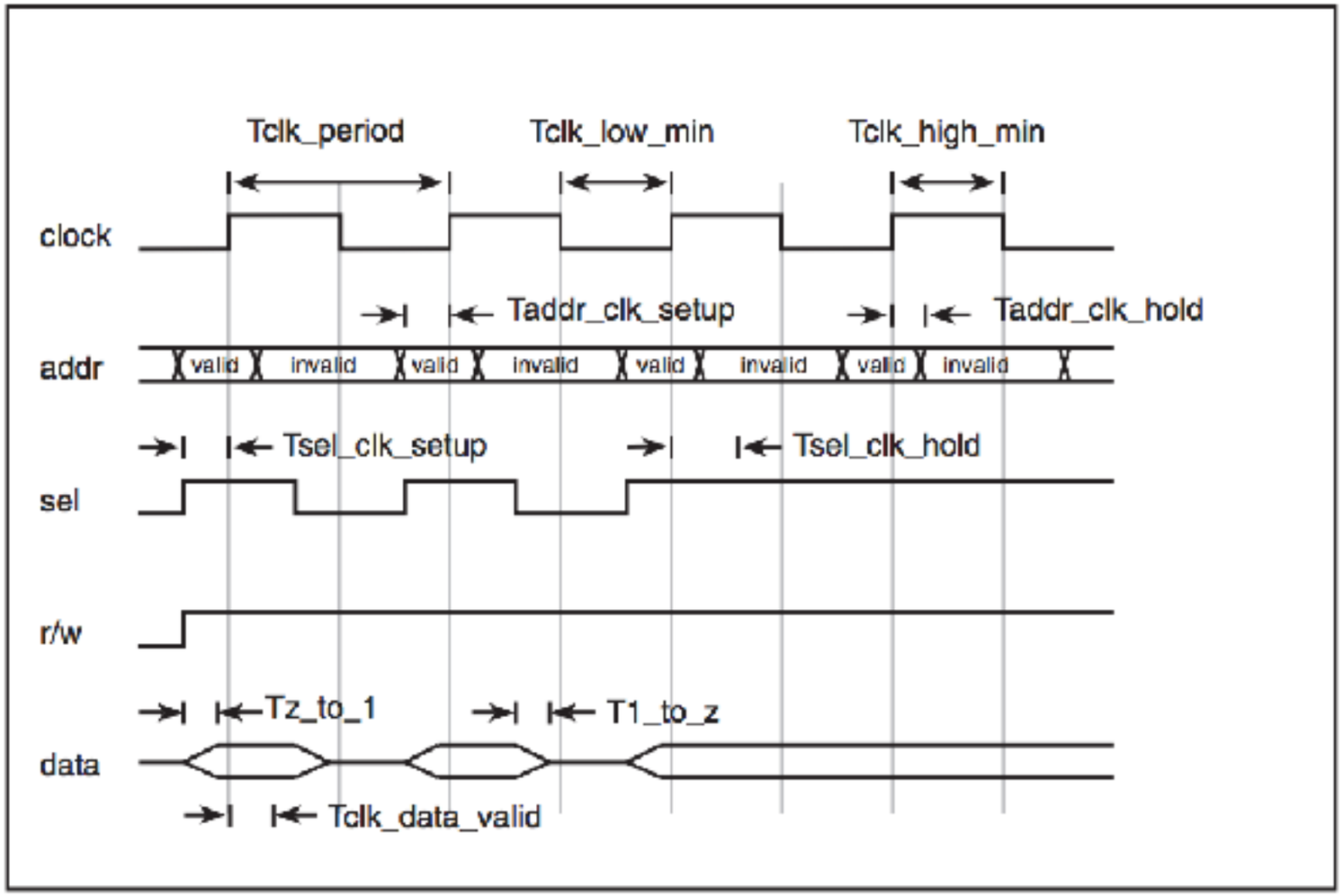


Fig. 3.11 Partial Timing Diagram for a Synchronous RAM

- The timing diagram documents how the interface signals need to switch in order for the memory to function properly

- The timing diagram is full of timing dependencies that serve as requirements to the implementation of the interface in order for it to function properly

Memory Description

Table 3.1 Synchronous Memory Timing Parameters

Parameter	Time (ns)	Parameter	Time (ns)
Tclk_period	20	Tclk_data_valid	9
Tclk_high_min	9	T0_to_z	0.1
Tclk_low_min	7	Tz_to_1	0.3
Taddr_clk_setup	4	T1_to_z	0.1
Taddr_clk_hold	3	Tz_to_0	0.2
Tsel_clk_setup	4	Trise	0.5
Tsel_clk_hold	12	Tfall	0.3

- Each of the timing dependencies is defined as parameters and those parameters are given a numerical value (in real design most likely in terms of minimum/maximum)

Memory Description Explanation

- The timing diagram documents how the interface signals need to switch in order for the memory to function properly

- The timing diagram is full of timing dependencies that serve as requirements to the implementation of the interface in order for it to function properly

- Each of the timing dependencies is defined as parameters and those parameters are given a numerical value (in real design most likely in terms of minimum/maximum)

Memory Model - Functionality

```
 1. module RAM (clk, addr, data, sel, rw);
 2.
 3. input            clk;
 4. input [15:0]     addr;
 5. inout [15:0]     data;
 6. input            sel, rw;
 7.
 8. reg [15:0]       mem_array [0:65536], data_internal;
 9. reg              tprob;
10.
11. // data bus tri-state. Bi-directional.
12. assign data = (sel) ? data_internal : 16'bz;
13.
14. // Always statement that does the actual read and write
15. always @ (posedge clk)
16.    begin
17.    // read memory
18.    if ((rw === 1'b1) && (sel === 1'b1))
19.       data_internal = mem_array [addr[15:0]];
20.
21.    // write memory
22.    if ((rw === 1'b0) && (sel === 1'b1))
23.       mem_array [addr[15:0]] = data;
24.    end
25.
```

Memory Model - Timing

```
26. // The specify block where all the timing and verification is
       placed.
27. /*****************************************************/
28. specify
29. // define timing parameters
30.    specparam Tclk_period = 20;
31.    specparam Tclk_high_min = 9;
32.    specparam Tclk_low_min = 7;
33.    specparam Taddr_clk_setup = 4;
34.    specparam Taddr_clk_hold = 3;
35.    specparam Tsel_clk_setup = 4;
36.    specparam Tsel_clk_hold = 12;
37.    specparam Tclk_data_valid = 9;
38.    specparam Trise = 0.5;
39.    specparam Tfall = 0.3;
40.    specparam T0_to_z = 0.1
41.    specparam Tz_to_1 = 0.3
42.    specparam T1_to_z = 0.1
43.    specparam Tz_to_0 = 0.2
44.
45.    // declare module path and apply delay
46.    (if sel) (posedge clk *> data) = (Tclk_data_valid + Trise,
47.                                      Tclk_data_valid + Tfall, 0, 0, 0, 0);
48.
49.    (negedge sel *> data) = (0, 0, T0_to_z, 0, T1_to_z, 0);
50.    (posedge sel *> data) = (0, 0, 0, Tz_to_1, 0, Tz_to_0);
51.
52. // do timing verification like set & hold, etc.
53.    $period (posedge clk, Tclk_period, tprob);
54.    $width (posedge clk, Tclk_high_min, 0, tprob);
55.    $width (negedge clk, Tclk_low_min, 0, tprob);
56.
57.    $setup (addr, posedge clk, Taddr_clk_setup, tprob);
58.    $hold (addr, posedge clk, Taddr_clk_hold, tprob);
59.
60.    $setuphold (sel, posedge clk, Tsel_clk_setup, Tsel_clk_hold,
       tprob);
61.
62. endspecify
63.
64.    // Report the time of every timing violation
65.    always @ (tprob)
66.    begin
67.       $display (%0d: "Timing violation found", $time);
68.    end
69.
70. end module
```

Timing checks are performed in the specify block. They verify the period of the clock, the width HIGH/LOW (duty cycle) along with setup and hold. If violated they send an error message to the log file.

Timing in Gate-Level Code

Gate Level Code Timing

- HDL languages can simulate a design on the ‘gate level’ where every gate is instantiated in a net list
- Most designs don’t start at the gate level but at the RTL level and then go through the synthesis process to produce the gate level setlist
- Gate level simulations are used to verify the logical correctness of the new representation and also provides a means for verifying the timing capabilities of the design

Synthesis and Timing Constraints

- The goal of synthesis is to produce a logically correct circuit from the RTL that will meet the timing requirements

- The logical implementation is checked by simulation but more efficiently by Logical Equivalence Checking

- The basic concern with the resultant net list is whether or not it will meet the system timing requirements

- Inputs to the synthesis process that help to ensure the results are referred to as ‘timing constraints’

Synthesis Priorities

- Different designs may have different design goals: low-power, high-speed, small-area
- Design Rule Constraints (DRCs) impose rules on the synthesis process by the physical limitations of the technology library chosen to implement the design
- Examples of DRCs are: max fanout per gate, max transition time of a signal, max capacitance per net
- The designer specifies optimization constraints to control speed, area and power

Design Rule Constraints

- The synthesis tool attempts to meet the user constraints and the DRCs. When it is close the tool will enforce the DRCs over the user constraints

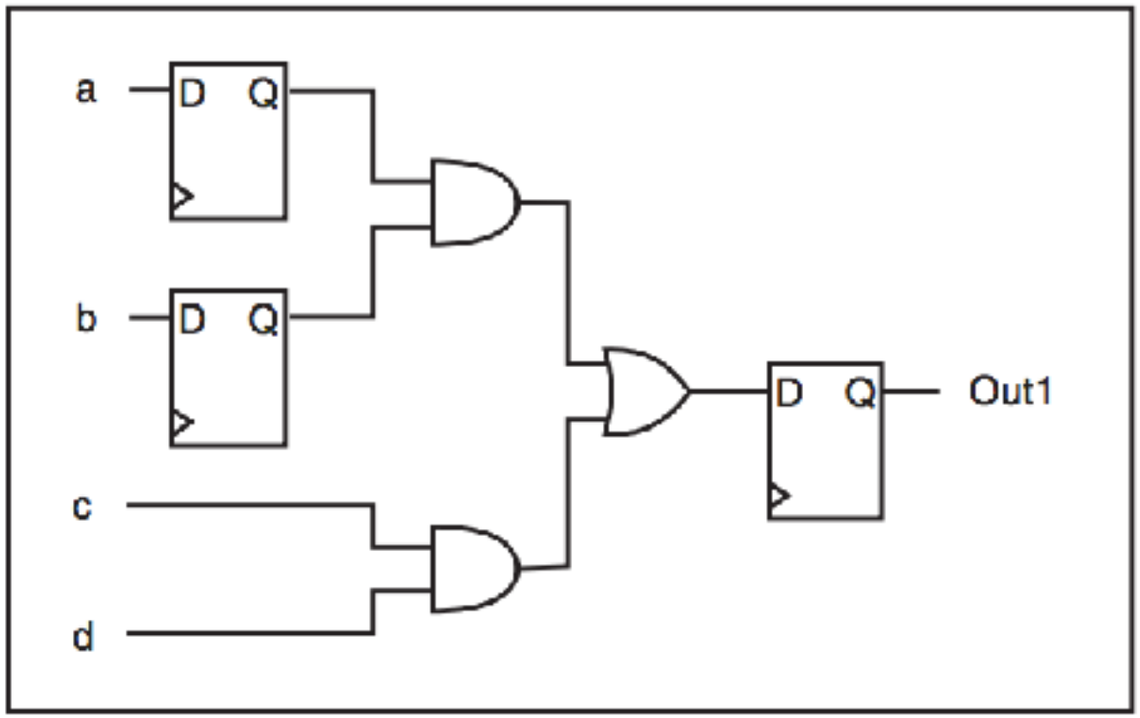
- If DRCs can not be met then the designer must change the target technology or redesign

- Many companies will have the designers build in their own margins when defining the constraints. This is intended to add a level of confidence to the resultant design.

Compiler Commands for DRC Limits

- **set_max_fanout**: limits the number of loads on output
- **set_max_transition**: limits the RC delays on nets
- **set_max_capacitance**: limits interconnect capacitance
- Once all DRCs are met the synthesis tool works to optimize the design. Most typically speed. The synthesis tool uses a timing analyzer to perform this

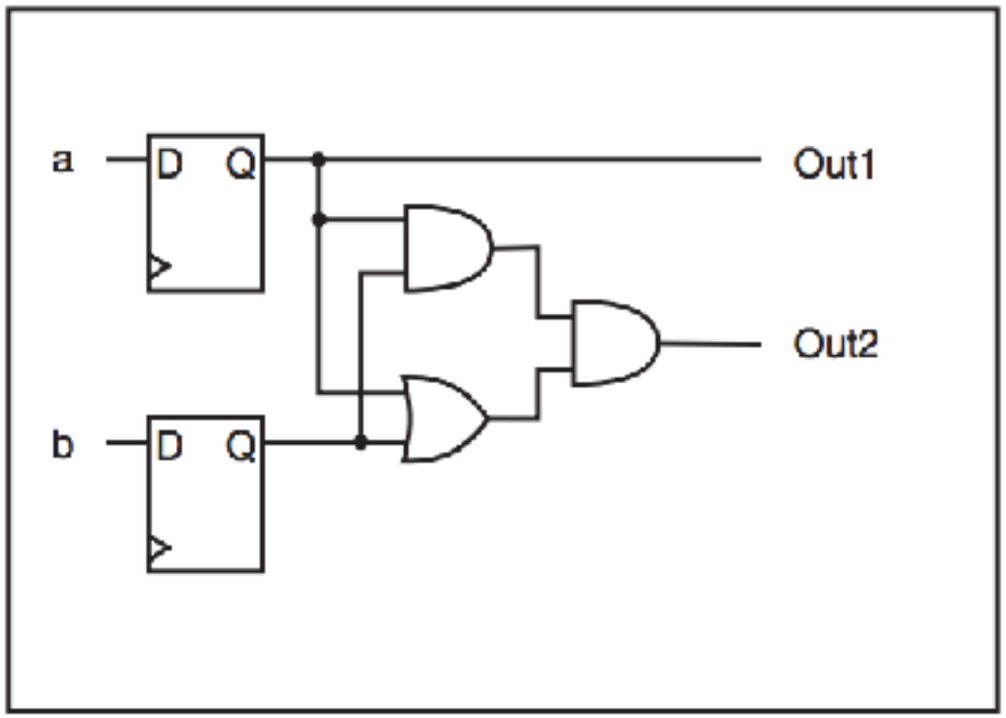
Optimization Constraints - Input Delays



- Assign input delay to 2 ns will assign the delay to inputs a, b, c & d

- This process is intended to be applied to synchronous designs.
- The clock tree at this stage is considered to be ideal
- **create_clock**: sets clock period and duty cycle
- **set_input_delay**: inputs must be characterized in order to meet internal timing. This command helps synthesis meet timing.

Optimization Constraints - Output Delays



- Assign output delay to 5 ns will assign the delay to outputs Out1 & Out2

- **set_output_delay**: Delay out increased by the amount defined.

- **set_max_delay**: Timing constraints set on combinational paths

- **set_max_area**: area of gates and interconnects (wires) limited

Gate and Wire-Load Models

- Synthesis tool calculates delays using the models of the gates and the timing model of the interconnect - ‘wire-load’ model

- The gate models are found in the target library

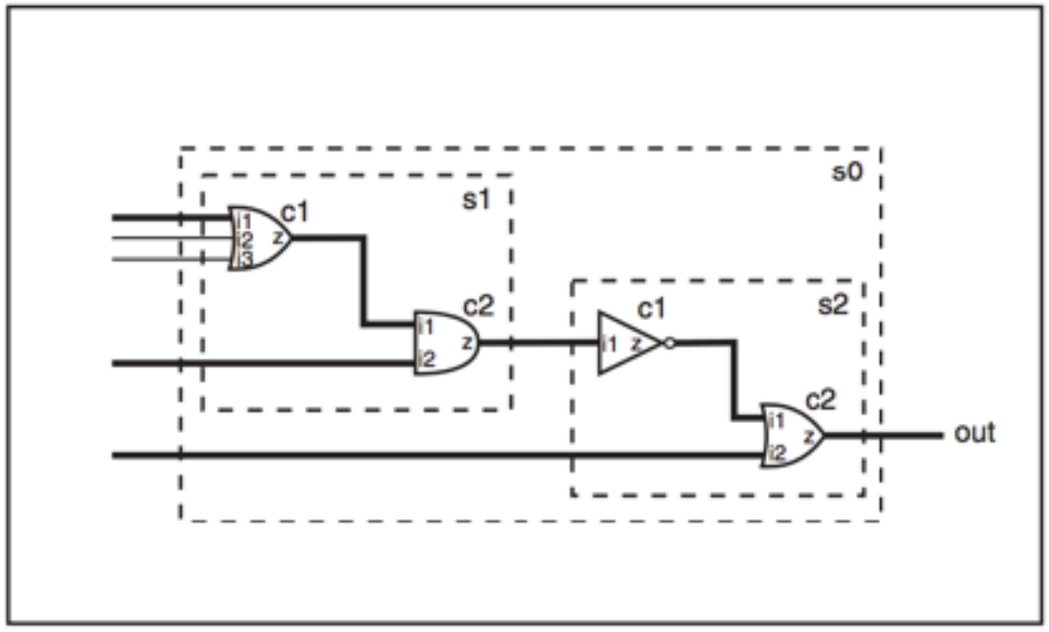
- The wire-load models are estimates based on the anticipated length of an interconnect

- As designs progress through the design process the accuracy of the wire-load models improve

Standard Delay Format (SDF) File

- The standard delay format file specifies delays
- Tools use the SDF file to pass timing on from one stage to another
- The command to read in an SDF file is $sdf_annotate
- The library components of the synthesized net list become the structure that is populated by the data contained in the SDF file

SDF Example




```
   (DELAYFILE
// Start of the sdf header. This file contains all typical data
   (SDFVERSION "1.0")
   (DESIGN "test")
   (DATE "Monday January 30 08:30:33 PST 1999")
   (VENDOR "Intrinsix Corp.")
   (PROGRAM "delay_find_forward")
   (VERSION "3.6")
   (DIVIDER /)
   (VOLTAGE 5.0:5.0:5.0)
   (PROCESS "typical")
   (TEMPERATURE 85:85:85)
   (TIMESCALE 1ns)
// The intrinsic delays of each cell used in the design. Equivalent
       to gate delays.
(CELL
   (CELLTYPE "test")
   (INSTANCE s0)
(TIMINGCHECK
(PATHCONSTRAINT   s1/c1/i1                  // start node
         s1/c1/z                            // intermediate nodes
         s1/c2/z
         s2/c1/z
         s2/c2/z                                  // end node
         (3.76:3.76:3.76) (3.44:3.44:3.44))       // times
(PATHCONSTRAINT   s1/c2/i2                        // start node
         s2/c1/z
         s2/c2/z                                  // end node
         (2.65:2.65:2.65) (2.52:2.52:2.52))       // times
(PATHCONSTRAINT   s2/c2/i2                        // start node
         s2/c2/z                                  // end node
         (1.54:1.54:1.54) (1.33:1.33:1.33))       // times
   ))
```

Synthesis Tips

- Are the constraints realistic?
- Have false paths and multi cycle paths been identified?
- Have minimum and maximum delays been set for all asynchronous paths?
- Do the timing violations require a change in architecture?
- Can minor timing problems be solved with incremental compiles? (each iteration performs an optimization which may allow the tool to converge on a solution)