



ECE 545—Digital System Design with VHDL

Lecture 9

Timing of Digital Systems,
Advanced Testbenches

11/4/08

1

Outline

- Timing of digital systems
 - Critical path and setup time violations
 - Min-Max-Avg via Synplicity critical path example
 - Clock Skew and Jitter, Hold Violations
 - Post place-and-route timing simulations and SDF
- Timing parameters
 - Throughput, latency, clock frequency, etc.
- Advanced Testbenches
 - Assert
 - File Input/Output

2

Resources

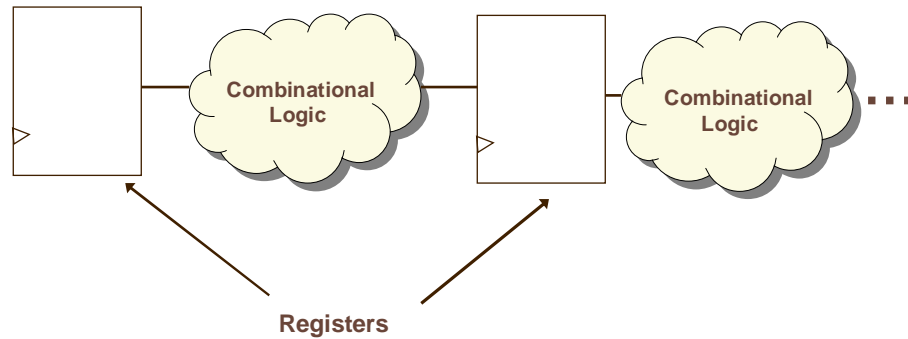
- Handouts on VHDL Simulation (from previous class)

3

Timing of Digital Systems

4

Register Transfer Logic (RTL) Design Description



5

Timing Characteristics of Flip-Flops

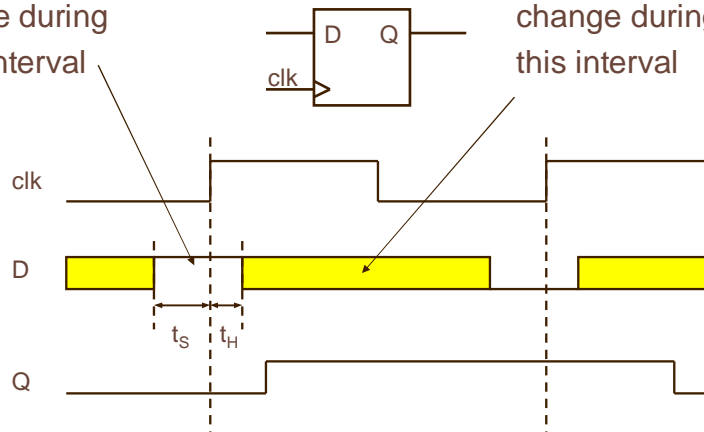
- Timing Features of Flip-flops
 - Setup time t_S – minimum time the input has to be stable before the rising edge of the clock
 - Hold time t_H – minimum time the input has to be stable after the rising edge of the clock
 - Propagation delay t_{CLK2Q} – time to propagate input to output after the rising edge of the clock

6

Setup and Hold Time

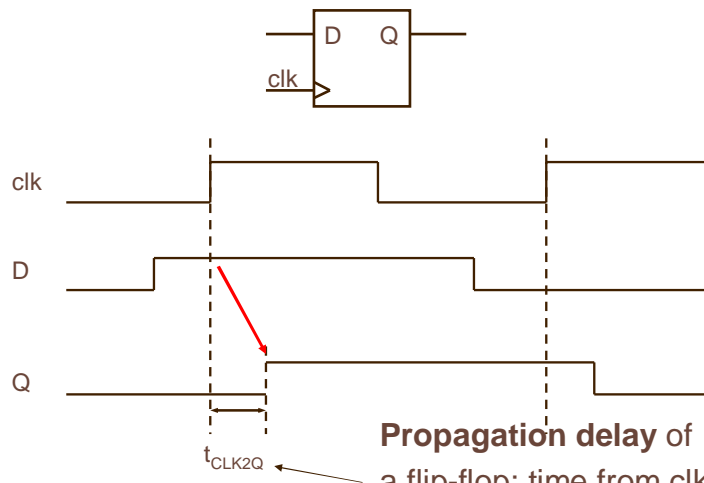
Input D must remain stable during this interval

Input D can freely change during this interval



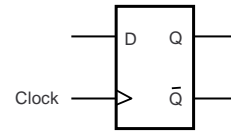
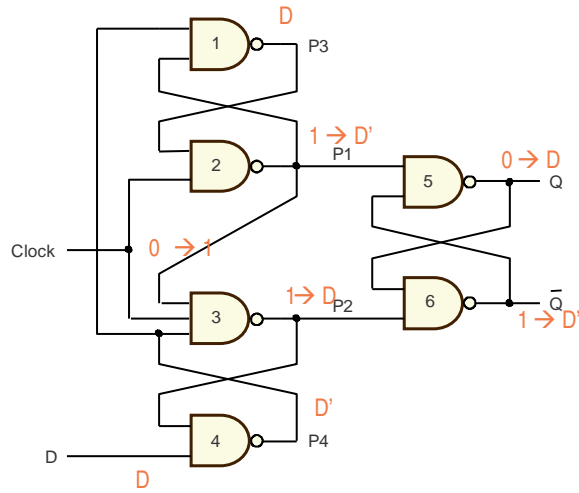
7

Flip-Flop Propagation Delay: t_{CLK2Q}



8

Origin of Setup and Hold Times (positive edge triggered D flip-flop)



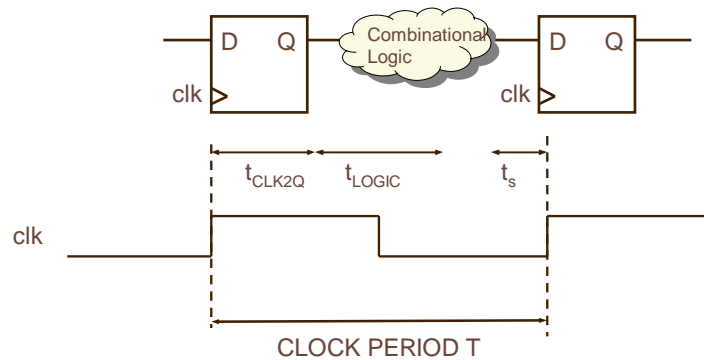
NAND : one input 0, output always 1; all inputs 1 except A, output is A'

9

Critical Path and Setup Violations

10

Combinational Logic Timing



- $t_{\text{CLK2Q}} + t_{\text{LOGIC}} < (T - t_s)$ to avoid setup time violation
- Rewriting the equation: $t_{\text{CLK2Q}} + t_{\text{LOGIC}} + t_s < T$
 $\underbrace{\hspace{10em}}_{t_{\text{path}}}$

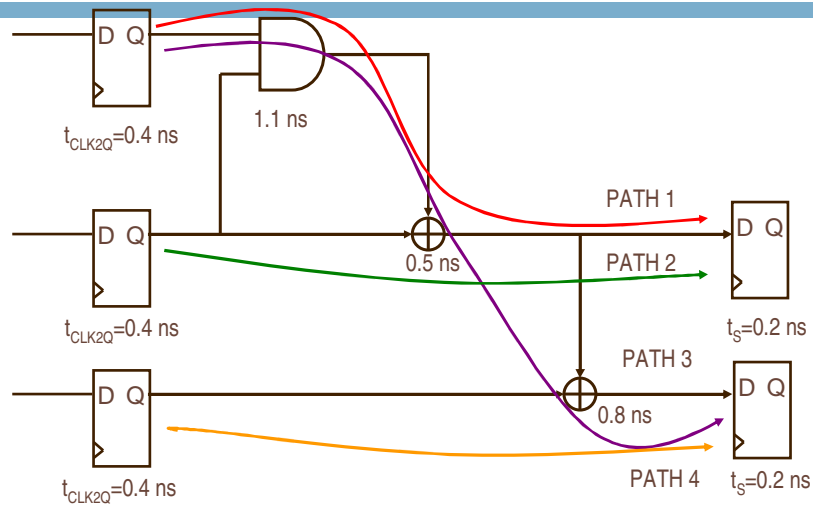
11

Critical Path

- A path is defined as a path from the **output of one flip-flop** to the **input of another flip-flop**
 - Path delay $t_{\text{path}} = t_{\text{CLK2Q}} + t_{\text{LOGIC}} + t_s$
- The largest of all the path delays in a circuit is called the **critical path delay** ($t_{\text{critical_path}}$)
 - The associated path is called the **critical path**
 - There can be millions of paths in a circuit; timing analysis CAD tools help to locate the critical path

12

Critical Path



- Path delays: $t_{path1} = 2.2$ ns, $t_{path2} = 1.1$ ns, $t_{path3} = 3.0$ ns, $t_{path4} = 1.4$ ns
- The **critical path** is path 3; the **critical path delay** is $t_{critical_path} = t_{path3} = 3.0$ ns

13

Minimum Period and Maximum Clock Frequency

- The minimum period a circuit can be clocked is equal to the critical path
 - $T_{min} = t_{critical_path}$
 - Maximum frequency = $1 / T_{min} = 1 / t_{critical_path}$

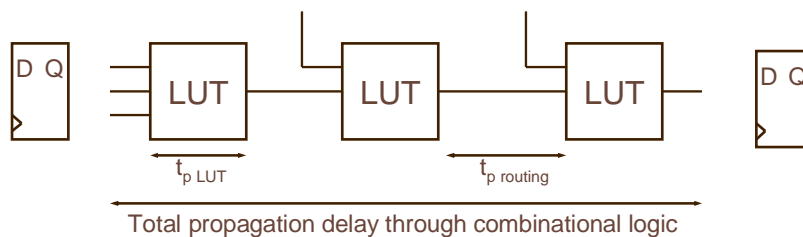
14

What makes up t_{LOGIC} ?

- Path delay $t_{\text{path}} = t_{\text{CLK2Q}} + t_{\text{LOGIC}} + t_{\text{S}}$
 - Usually t_{CLK2Q} and t_{S} are about the same for all flip-flops in the circuit
 - Thus t_{LOGIC} usually determines critical path in a circuit
- t_{LOGIC} is the **propagation delay** of the combination logic between registers in the circuit
 - t_{LOGIC} composed of two elements:
 - 1) propagation delay through logic components (LUTs)
 - 2) propagation delay through routing (wires)

15

Timing Characteristics of Combinational Circuits



16

Components of t_{LOGIC}

- Total propagation delay through logic components depends on:
 - 1) the number of logic levels
 - number of logic levels is the number of logic components (gates, LUTs) the signal propagates through
 - 2) the propagation delay of each gate
 - in ASICs, a NAND2 gate is faster than an XNOR2
- Total propagation delay through routing delays depends on:
 - 1) length of interconnects
 - Longer wires take more delay
 - 2) fanout
 - Larger fanout takes more delay

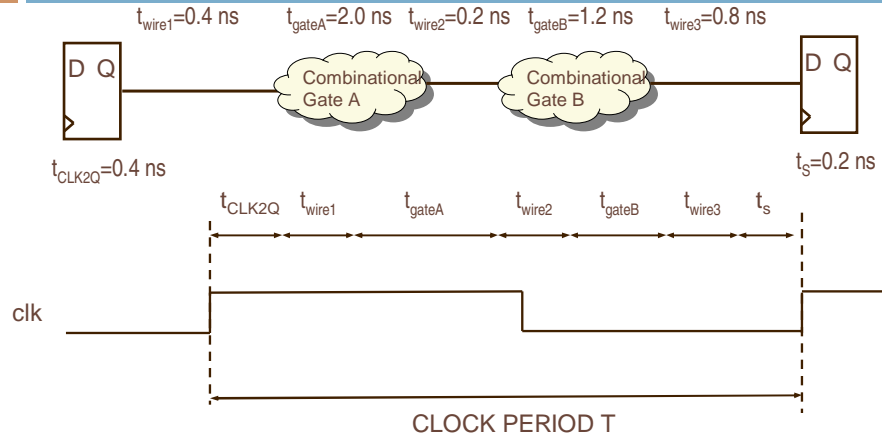
17

Routing Delays

- In Current Technologies Routing Delays Make **45-65%** of the Total Propagation Delays

18

Setup Time Violation (a.k.a Critical Path Violation)



- Critical path delay = $t_{\text{critical_path}} = 5.2 \text{ ns}$
- The minimum period for this circuit to work is $T_{\text{min}} = 5.2 \text{ ns}$
 - Maximum clock frequency = $1/T_{\text{min}} = 192 \text{ MHz}$
- If the clock period is smaller than T_{min} , you will get a timing violation and circuit will not operate correctly!!
 - This kind of timing violation is called a "setup time" violation (also known as critical path violation)

19

Min-Max-Avg Critical Path Example using Synplify Pro

20

Synplify Pro Synthesis Report

```
##### START OF TIMING REPORT #####
# Timing Report written on Wed Oct 31 16:05:49 2007
#
```

```
Top view: minmaxavg
Requested Frequency: 159.2 MHz
Wire load mode: top
Paths requested: 5
Constraint File(s):
@N: MT195 |This timing report estimates place and route data. Please look at the place and route timing report for final timing..

@N: MT197 |Clock constraints cover only FF-to-FF paths associated with the clock..
```

using auto-constrain to estimate, before synthesis
Synplicity thinks it can approximately achieve this

```
Performance Summary
*****
```

```
Worst slack in design: -1.765
```

Starting Clock	Requested Frequency	Estimated Frequency	Requested Period	Estimated Period	Slack	Clock Type	Clock Group
minmaxavg clk	159.2 MHz	124.3 MHz	6.282	8.046	-1.765	inferred	
Autoconstr_clkgroup_0							

after synthesis, Synplicity thinks it can achieve this

Synplify Pro Synthesis Report cont'd

```
Worst Path Information
*****
```

```
Path information for path number 1:
```

```
Requested Period: 6.282
- Setup time: 0.176
= Required time: 6.106
```

```
- Propagation time: 7.870
= Slack (critical) : -1.765
```

```
Number of logic level(s): 38
```

```
Starting point:
```

```
Ending point:
```

```
The start point is clocked by
```

```
The end point is clocked by
```

using auto-constrain, before synthesis
Synplicity tries to achieve this clock period

setup time t_s

after synthesis, realizes it can only
achieve this (clock period - t_s)

negative slack means could not reach goal

Instance / Net Name	Type	Pin Name	Pin Dir	Delay	t_{CLK2Q} Arrival Time	No. of Fan Out(s)
Cl.pr_state_fast[1]	FDRS	Q	Out	0.626	0.626	-
pr_state_fast[1]	Net	-	-	0.727	-	4
D1.ADDR_0[0]	LUT3	I0	In	-	1.353	-
D1.ADDR_0[0]	LUT3	O	Out	0.504	1.857	-
ADDR[0]	Net	-	-	0.737	-	8
D1.Memory.mem.I_5	RAM16X4S	A0	In	-	2.593	-
D1.Memory.mem.I_5	RAM16X4S	O0	Out	0.504	3.097	-
ADATA[0]	Net	-	-	0.727	-	6
D1.un1_new_sum_axb_0	LUT2	I0	In	-	3.824	-
D1.un1_new_sum_axb_0	LUT2	O	Out	0.504	4.328	-

Synplify Pro Synthesis Report cont'd

un1_new_sum_axb_0	Net	-	-	0.000	-	2
D1.un1_new_sum_cry_0	MUXCY_L	S	In	-	4.328	-
D1.un1_new_sum_cry_0	MUXCY_L	LO	Out	0.419	4.747	-
un1_new_sum_cry_0	Net	-	-	0.000	-	2
. . .						
D1.un1_new_sum_cry_33	MUXCY_L	CI	In	-	6.411	-
D1.un1_new_sum_cry_33	MUXCY_L	LO	Out	0.052	6.463	-
un1_new_sum_cry_33	Net	-	-	0.000	-	1
D1.un1_new_sum_s_34	XORCY	CI	In	-	6.463	-
D1.un1_new_sum_s_34	XORCY	O	Out	0.768	7.231	-
un1_new_sum_s_34	Net	-	-	0.639	-	1
D1.RegSUM.Q[34]	FDRE	D	In	-	7.870	-

 Total path delay (propagation time + setup) of 8.046 is 5.217(64.8%) logic and 2.829(35.2%) route.

shows percentage logic versus routing delay

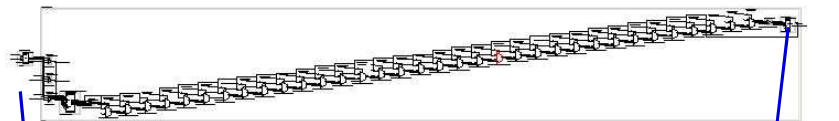
$$t_{critical_path} = t_{CLK2Q} + t_{LOGIC} + t_s = 8.046 \text{ ns}$$

Minimum clock period $T_{min} = 8.046 \text{ ns}$

Maximum clock frequency = 123.4 MHz

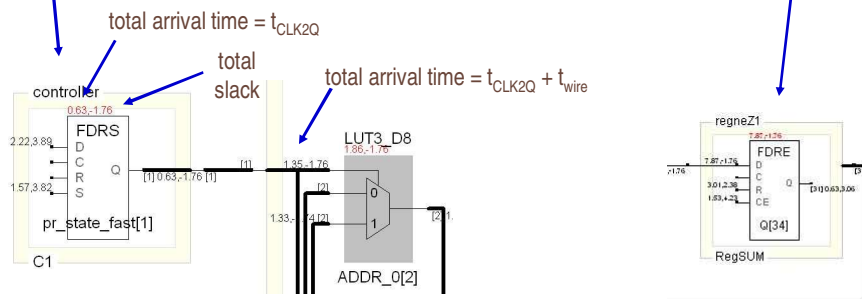
23

Synplify Pro Technology View Critical Path (BOLD BLACK LINE IS CRITICAL PATH)

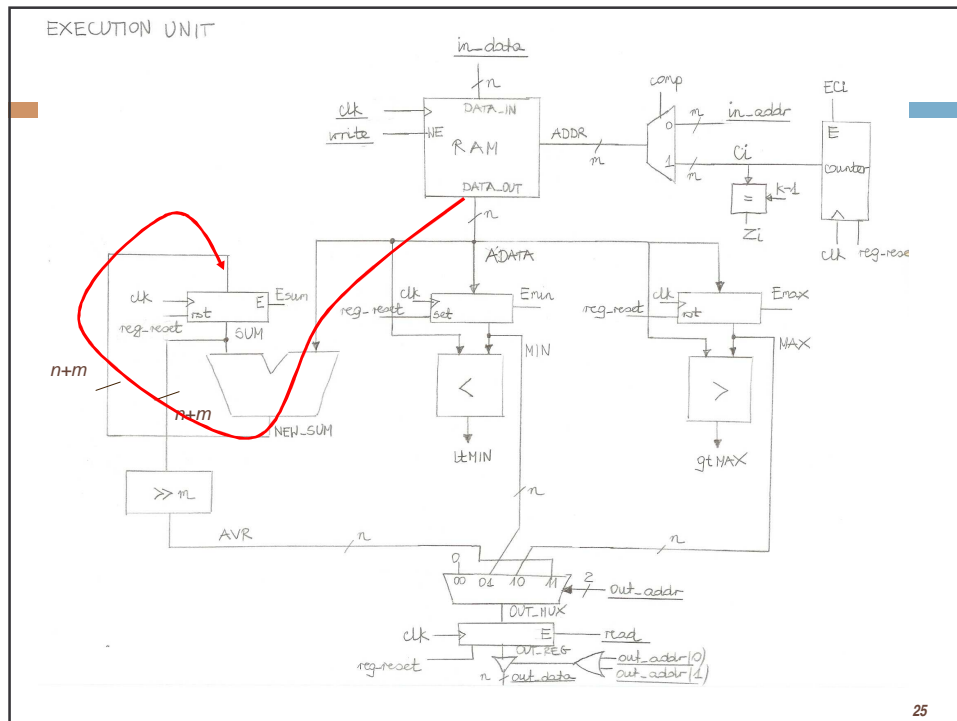


zoom at starting flip-flop

zoom at ending flip-flop



24



Note about synthesis

- Sometimes it is difficult in Synplify Pro technology view to find critical path's relation to block diagram
- Solution: look at the signal names and entity names
- Synplify Pro synthesis creates two files:
 - [your_entity_name].vhm → a single netlist for your entire design
 - Used for post-synthesis simulation
 - [your_entity_name].edif → a single netlist for your entire design in EDIF format (a different kind of format than .vhm)
 - Used by FPGA implementation tool
- XST synthesis
 - [your_entity_name].vhd → a single netlist for your entire design
 - [your_entity_name].ngc → a single netlist for your entire design in NGC format (binary file not in ASCII format)
 - Used by FPGA implementation tool

minmaxavg.vhm from Synplify Pro

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library synplify;
use synplify.components.all;
library UNISIM;
use UNISIM.VCOMPONENTS.all;

entity RAM_16Xn_DISTRIBUTED is
port(
  in_data_c : in std_logic_vector (31 downto 0);
  ADDR : in std_logic_vector (2 downto 0);
  ADATA : out std_logic_vector (31 downto 0);
  write_c : in std_logic;
  clk_c : in std_logic;
end RAM_16Xn_DISTRIBUTED;

architecture beh of RAM_16Xn_DISTRIBUTED is
  signal NN_1 : std_logic ;
  signal NN_2 : std_logic ;
begin
  \II_mem.I_14\ : RAM16X4S port map (
    O0 => ADATA(20),
    O1 => ADATA(21),
    O2 => ADATA(22),
    O3 => ADATA(23),
    A0 => ADDR(0),
    A1 => ADDR(1),
    A2 => ADDR(2),
```

UNISIM is a Xilinx post-synthesis library of gates/LUTs

Xilinx RAM cell (distributed)

27

minmaxavg.edf

```
(edif minmaxavg
(edifVersion 2 0 0)
(edifLevel 0)
(keywordMap (keywordLevel 0))
(status
  (written
    (timeStamp 2007 10 31 16 5 49)
    (author "Synplicity, Inc.")
    (program "Synplify Pro" (version "Version 8.6.2, mapper 8.6.2, Build 027R"))
  )
)
(library VIRTEX
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell RAM16X1S (cellType GENERIC)
    (view PRIM (viewType NETLIST)
      (interface
        (port O (direction OUTPUT))
        (port A0 (direction INPUT))
        (port A1 (direction INPUT))
        (port A2 (direction INPUT))
        (port A3 (direction INPUT))
        (port D (direction INPUT))
        (port WCLK (direction INPUT))
      )
    )
    (port WE (direction INPUT))
  )
)
```

28

Clock Skew and Jitter, Hold Violations

29

Clock Jitter

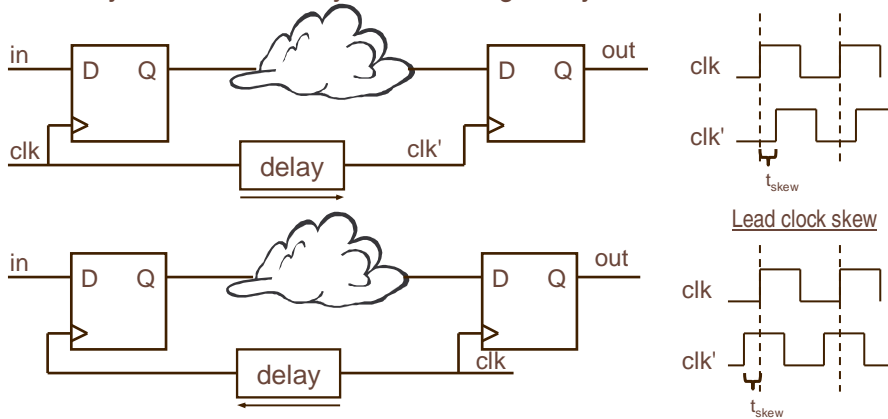
- Rising Edge of The Clock Does Not Occur **Precisely** Periodically
 - May cause faults in the circuit
 - Can sometimes be lumped together with skew



30

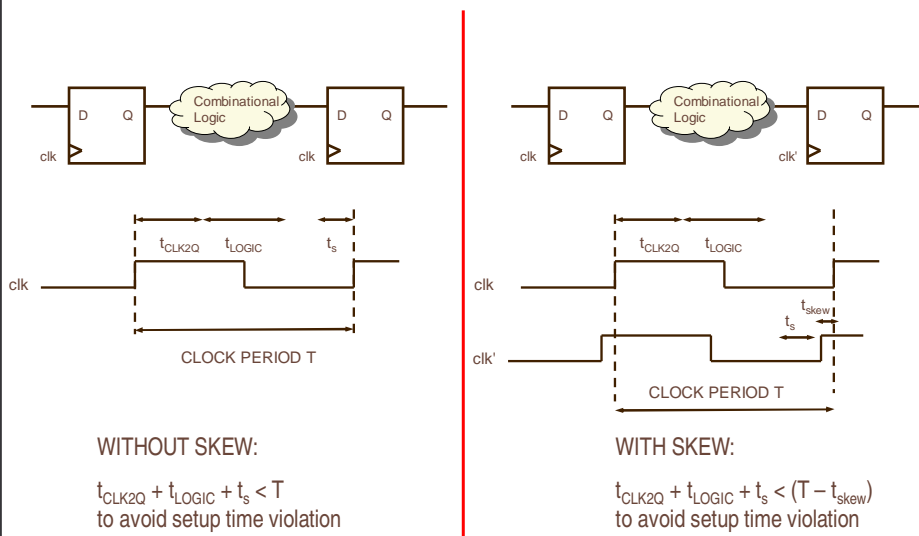
Clock Skew

- Rising Edge of the Clock Does Not Arrive at Clock Inputs of All Flip-flops at The Same Time
- Delay often caused by wire routing delay



31

Lead clock skew is bad because it may cause setup time violations



WITHOUT SKEW:

$$t_{\text{CLK2Q}} + t_{\text{LOGIC}} + t_s < T$$

to avoid setup time violation

WITH SKEW:

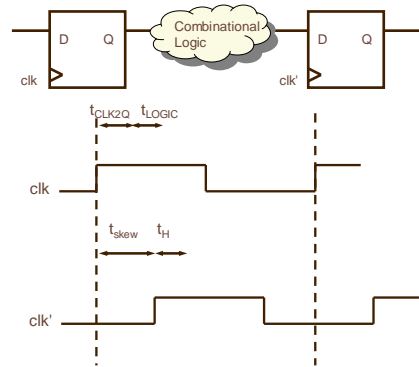
$$t_{\text{CLK2Q}} + t_{\text{LOGIC}} + t_s < (T - t_{\text{skew}})$$

to avoid setup time violation

→ less time to perform logic than you normally would

32

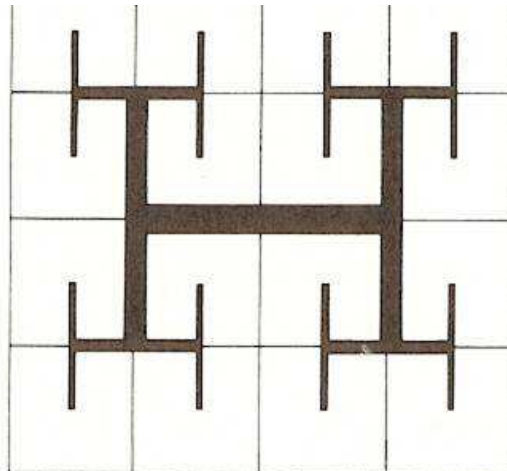
Lag clock skew is bad because it may cause hold time violations



- $t_{CLK2Q} + t_{LOGIC} > (t_{skew} + t_H)$ to avoid hold time violation
- If this is violated, get data feedthrough (data gets fed into the next register one cycle too early)
- **There is no clock period (T) in the equation; changing clock period cannot help this problem!**

33

H-clock tree used to minimize clock skew



34

Dealing With Clock Problems

- Use only dedicated clock nets for clock signals
- Do not put any logic in clock nets
- Clock skew harder and harder to predict in deep submicron technologies due to process variation
 - Can only be sure of skew after fabrication
 - **Usually design certain to tolerate t_{skew_max}**
 - CAD tools make sure leading t_{skew_max} does not cause setup time violations
 - CAD tools make sure lagging t_{skew_max} does not cause hold time violations

35

Setup and Hold Violations in the Lab

- Setup time violation
 - $t_{CLK2Q} + t_{LOGIC} + t_s < (T - t_{skew})$
 - Rewriting: $t_{LOGIC} < (T - t_{skew} - t_{CLK2Q} - t_s) \rightarrow t_{LOGIC}$ cannot be too big!
 - If your circuit is not meeting timing in the lab, just increase the clock period (i.e. make T bigger)
 - Set up time violation also called **critical path violation** or **max delay violation**
 - This is what we usually think of as a timing violation
- Hold time violation
 - $t_{CLK2Q} + t_{LOGIC} > (t_{skew} + t_H)$
 - Rewriting: $t_{LOGIC} > (t_{skew} + t_H - t_{CLK2Q}) \rightarrow t_{LOGIC}$ cannot be too small!
 - **If your circuit is making hold time violations in the lab, you generally cannot do anything. Catastrophic!!**
 - Hold time violation also called **feedthrough violation** or **min delay violation**
 - Typically not a big problem in FPGAs; a bigger problem in ASICs

36



Post Place and Route Timing Simulations and SDF

37

Timing vs. Functional Simulation

- Simulation before synthesis is used to verify circuit functionality and may differ from the one after synthesis and implementation
- Implementation tool generates SDF (Standard Delay Format) as a standard delay file and the netlist for synthesized VHDL code with delays.
- Generated netlist contains many component instantiation statements with library references

38

Implementation

- Aldec flow (Active-HDL 7.2 SP2 using Xilinx ISE 9.1 SP3) produces a number of files including:
 - TIME_SIM.vhd: for post-place-and-route simulation
 - TIME_SIM.SDF: timing information for TIME_SIM.vhd
- Xilinx ISE 9.1 SP3 flow
 - [entity_name]_timesim.vhd: for post-place-and-route simulation
 - [entity_name]_timesim.sdf: timing information for [entity_name]_timesim.vhd

39

SDF file TIME_SIM.SDF

```
(CELL (CELLTYPE "X_LUT4")
(INSTANCE D1_OUT_MUX_3_0_11_Q)
(DELAY
(ABSOLUTE
(PORT ADR0 (409:409:512))
(PORT ADR1 (2082:2082:2602))
(PORT ADR2 (1585:1585:1981))
(PORT ADR3 (2017:2017:2521))
(IOPATH ADR0 O (415:519:519)(415:519:519))
(IOPATH ADR1 O (415:519:519)(415:519:519))
(IOPATH ADR2 O (415:519:519)(415:519:519))
(IOPATH ADR3 O (415:519:519)(415:519:519))
)
)
)
(CELL (CELLTYPE "X_SFF")
(INSTANCE D1_RegOUT_Q_11_Q)
(DELAY
(ABSOLUTE
(IOPATH CLK O (500:626:626))
(IOPATH SET O (500:626:626))
(IOPATH RST O (500:626:626))
)
)
)
)
(TIMINGCHECK
(SETUPHOLD (posedge I) (posedge CLK) (56:70:81)(264:331:331))
(SETUPHOLD (negedge I) (posedge CLK) (56:70:81)(264:331:331))
(SETUPHOLD (posedge CE) (posedge CLK) (419:524:524)(0))
(SETUPHOLD (negedge CE) (posedge CLK) (419:524:524)(0))
(PERIOD (posedge CLK) (1092:1366:1366))
(SETUPHOLD (posedge SRST) (posedge CLK) (688:861:861)(0))
(SETUPHOLD (negedge SRST) (posedge CLK) (688:861:861)(0))
)
)
```

A part of the SDF file is shown.
It indicates input to output gate delays
high to low
low to high
of (minimum: typical: maximum) timing
-maximum timing is called worst-case timing
-there are three ranges due to process
variation in fabrication of the FPGA

40

VHDL file TIME_SIM.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library SIMPRIM;
use SIMPRIM.VCOMPONENTS.ALL;
use SIMPRIM.VPACKAGE.ALL;

entity minmaxavg is
port (
  START : in STD_LOGIC := 'X';
  reset : in STD_LOGIC := 'X';
  read : in STD_LOGIC := 'X';
  write : in STD_LOGIC := 'X';
  DONE : out STD_LOGIC;
  clk : in STD_LOGIC := 'X';
  out_addr : in STD_LOGIC_VECTOR ( 1 downto 0 );
  out_data : out STD_LOGIC_VECTOR ( 31 downto 0 );
  in_addr : in STD_LOGIC_VECTOR ( 2 downto 0 );
  in_data : in STD_LOGIC_VECTOR ( 31 downto 0 )
);
end minmaxavg;

architecture Structure of minmaxavg is
signal GLOBAL_LOGIC0 : STD_LOGIC;
signal D1_ADATA_0_0 : STD_LOGIC;
signal D1_ADATA_1_0 : STD_LOGIC;
. . .

D1_ltmin_cry_7_0_CVMUXFAST : X_MUX2
  generic map(
    LOC => "SLICE_X4Y11"
  )
  port map (
    IA => D1_ltmin_cry_7_0_CVMUXG2_29,
    IB => D1_ltmin_cry_7_0_FASTCARRY_28,
    SEL => D1_ltmin_cry_7_0_CYAND_27,
    O => D1_ltmin_cry_7_0_CVMUXFAST_26
  );
```

SIMPRIM is a Xilinx post-place-and-route library of gates/LUTs

41

Timing Parameters

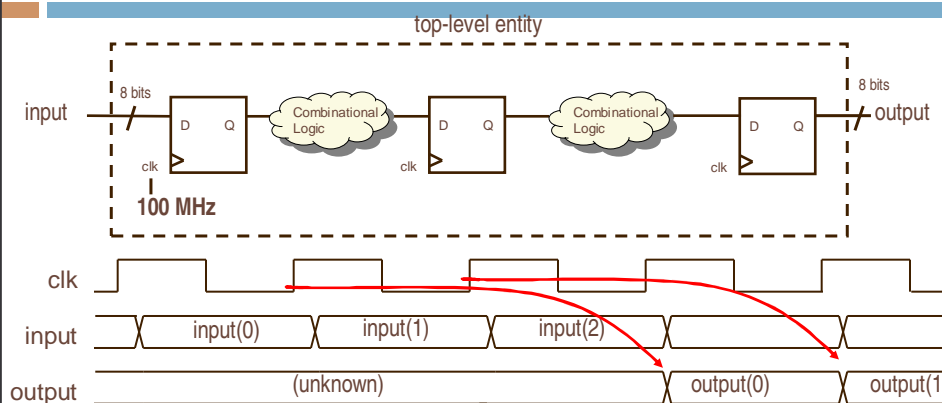
42

Timing Parameters

	definition	units
delay	time from point→point	ns
clock period T	rising edge →rising edge of clock	ns
clock frequency	$\frac{1}{\text{clock period}}$	MHz
latency	time from input→output	ns
throughput	#output bits/time unit	Mbits/s

43

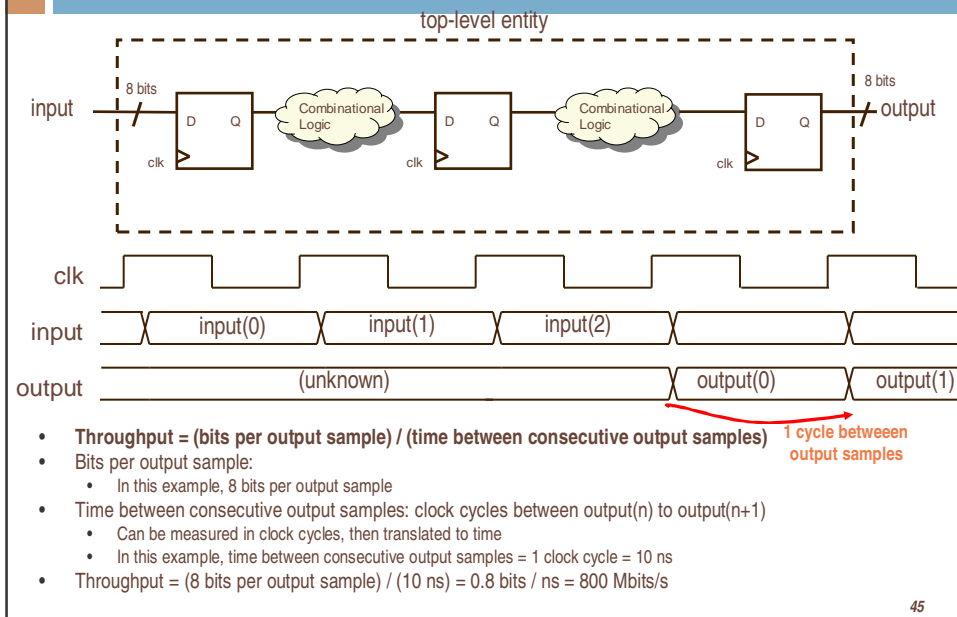
Latency



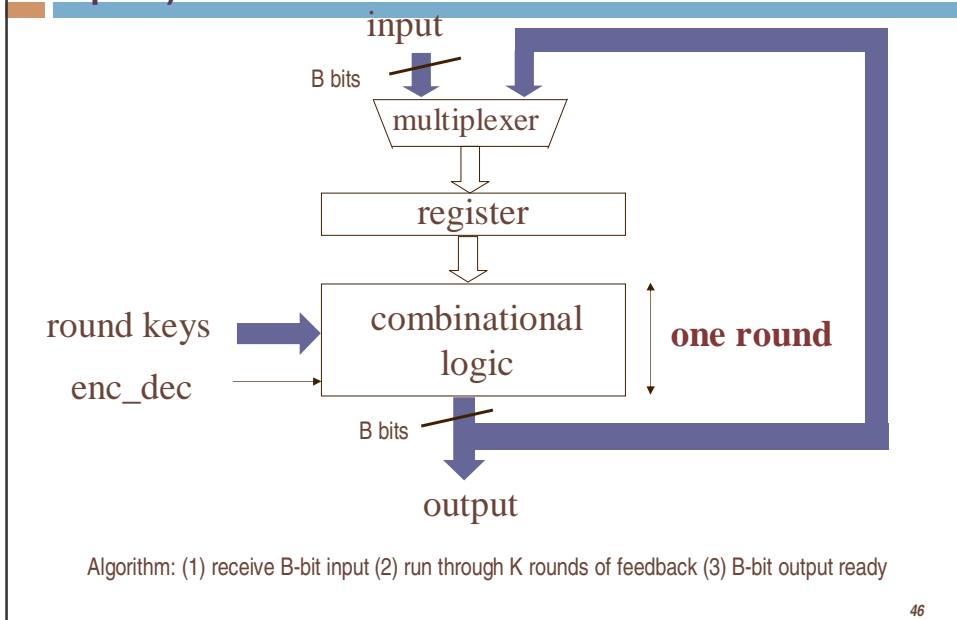
- Latency is the time between input(n) and output(n)
 - i.e. time it takes from first input to first output, second input to second output, etc.
 - Latency is usually constant for a system (but not always)
 - Also called input-to-output latency
- **Count the number of rising edges of the clock!**
 - In this example, 3 rising edges from input to output → latency is 3 cycles
- Latency is measured in clock cycles (then translated to seconds)
 - In this example, say clock period is 10 ns, then latency is 30 ns

44

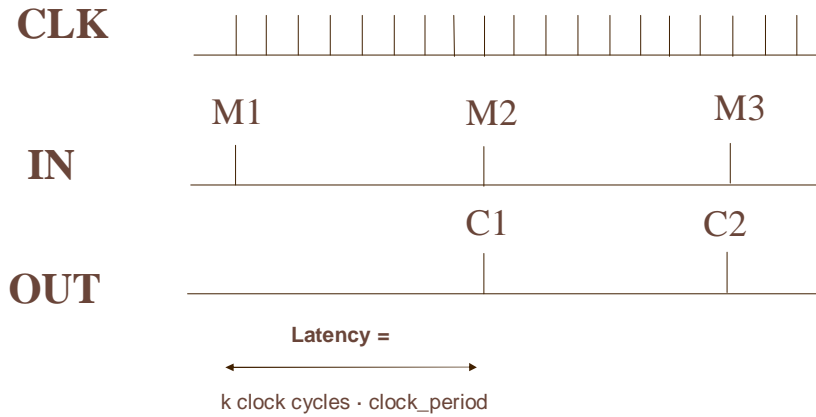
Throughput



Iterative Architecture of Block Cipher (not Stream Cipher)



Iterative Architecture of Block Cipher

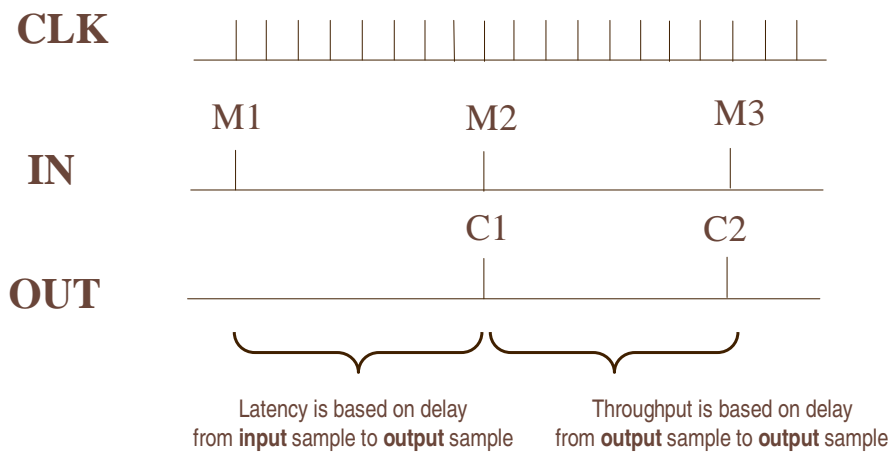


$$\text{Throughput} = (\text{bits per output sample}) / (\text{time between consecutive output samples})$$

$$= (B \text{ bits / sample}) / (k \text{ clock cycles} \cdot \text{clock_period})$$

47

KEY POINT: Latency versus Throughput!!



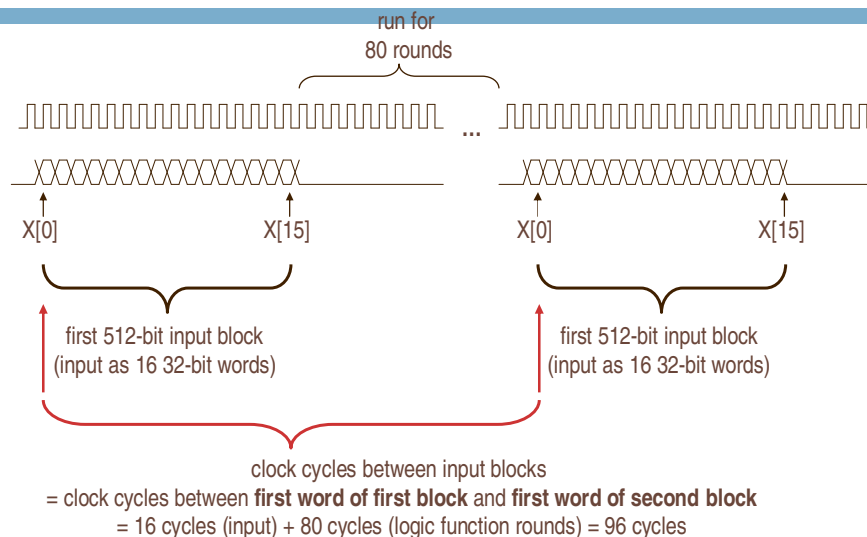
48

Throughput for Hash Function

- For normal systems, typically calculate output throughput:
 - Output throughput = (bits per output sample) / (time between consecutive output samples)
- Hash function throughput is different
 - Output occurs only once at end of operation
 - Output throughput does not make any sense because output throughput depends on input message size
 - For hash functions, we measure input throughput for blocks
 - Input throughput = (bits per input block) / (time between consecutive input blocks)
 - = (bits per input block) / (clock cycles between input blocks * clock period)
 - Example:
 - bits per input block = 512 bits
 - clock cycles between input blocks = 96 cycles
 - clock period = 20 ns (i.e. 50 MHz)
 - input throughput = 512 bits / (96 * 20 ns) = 267 Mbits/second
 - A throughput of 1000 Mbits/second is excellent for a hash function

49

Hash Function Throughput Calculation

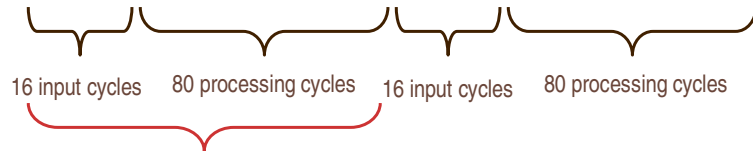


(this simple example ignores any preprocessing and postprocessing states, which would make >96)

50

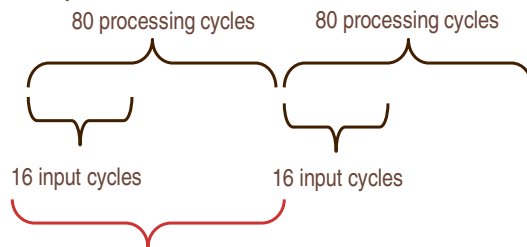
Improving Throughput by Overlapping Input Cycles and Processing Cycles (with same clock period)

Inefficient Implementation:



clock cycles between input blocks = $80 + 16 = 96 \rightarrow$ throughput = $512 \text{ bits} / (96 * 20 \text{ ns}) = 267 \text{ Mb/s}$

Efficient Implementation:



clock cycles between input blocks = $80 \rightarrow$ throughput = $512 \text{ bits} / (80 * 20 \text{ ns}) = 320 \text{ Mb/s}$

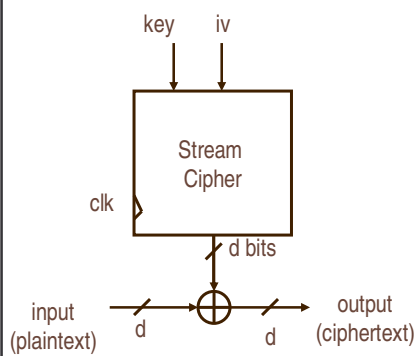
51

Increasing Hash Function Throughput

- To increase hash function throughput
- **Increase:** bits per input block
 - **INVALID:** we cannot do this since input block size is fixed.
- **Decrease:** number of cycles between input blocks
 - **VALID:** we can "overlap" reading of input data and computation of hash function, combine states when possible, etc.
- **Decrease:** clock period
 - **VALID:** design the hash function such that the critical path is as short as possible.

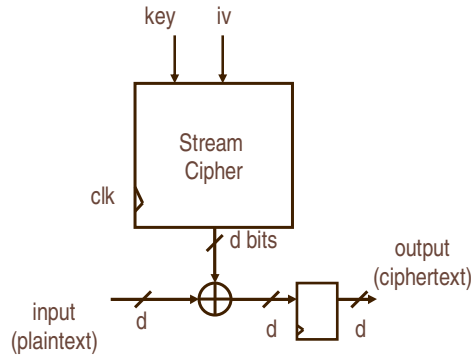
52

Simple stream cipher: timing



Latency = 0 (no rising edges between input and output)

Throughput = (d bits) / (1 · clock_period)

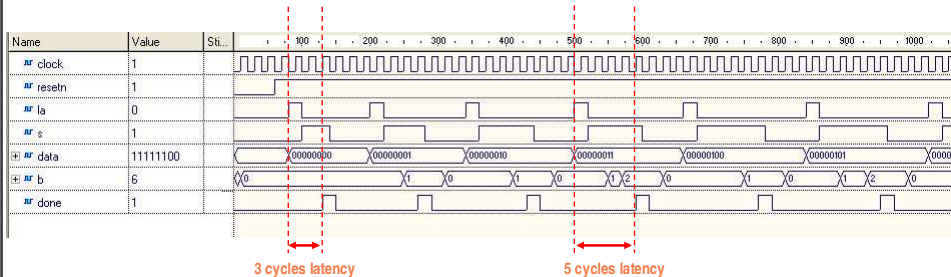


Latency = 1 · clock_period

Throughput = (d bits) / (1 · clock_period)

53

Latency is not always constant



- Bit-counting example from lecture 6
- Define input as DATA, define output as DONE (signals when B is valid)
- When DATA = 0, latency is 3 clock cycles!
- When DATA = 3, latency is 5 clock cycles!

54

Advanced Testbenches

55

Records

```
type opcodes is (add, sub, andd, orr);  
type reg_number is range 0 to 7;
```

```
type instruction is record  
  opcode: opcodes;  
  source_reg1: reg_number;  
  source_reg2: reg_number;  
  dest_reg: reg_number;  
  displacement: integer;  
end record instruction;
```

```
constant add_instr_1_3: instruction :=  
  (opcode => add,  
   source_reg1 => 1,  
   source_reg2 => 3,  
   dest_reg => 1,  
   displacement => 0);
```

56

Assert

Assert is a **non-synthesizable** statement whose purpose is to write out messages on the screen when problems are found during simulation.

Depending on the **severity of the problem**, The simulator is instructed to continue simulation or halt.

57

Assert - syntax

```
ASSERT condition  
REPORT "message"  
[SEVERITY severity_level ];
```

The message is written when the condition is FALSE.

severity_level can be:

Note, Warning, Error (default), or Failure.

58

Assert - Examples

```
assert initial_value <= max_value
  report "initial value too large"
  severity error;

assert packet_length /= 0
  report "empty network packet received"
  severity warning;

assert false
  report "Initialization complete"
  severity note;
```

59

Report - syntax

```
REPORT "message"
[SEVERITY severity_level ];
```

The message is always written.

Severity_level can be:

Note (default), Warning, Error, or Failure.

60

Report - Examples

```
report "Initialization complete";  
  
report "Current time = " & time'image(now);  
  
report "Incorrect branch" severity error;
```

61

Generating reports in the message window

```
reports: process(clk_trigger) begin  
    if (clk_trigger = '0' and clk_trigger'EVENT) then  
        case segments is  
            when seg_0 => report time'image(now) & ": 0 is displayed" ;  
            when seg_1 => report time'image(now) & ": 1 is displayed" ;  
            when seg_2 => report time'image(now) & ": 2 is displayed" ;  
            when seg_3 => report time'image(now) & ": 3 is displayed" ;  
            when seg_4 => report time'image(now) & ": 4 is displayed" ;  
            when seg_5 => report time'image(now) & ": 5 is displayed" ;  
            when seg_6 => report time'image(now) & ": 6 is displayed" ;  
            when seg_7 => report time'image(now) & ": 7 is displayed" ;  
            when seg_8 => report time'image(now) & ": 8 is displayed" ;  
            when seg_9 => report time'image(now) & ": 9 is displayed" ;  
        end case;  
    end if;  
end process;
```

62

Using Arrays of Test Vectors In Testbenches

63

Testbench for Seven Segment Output

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY sevenSegmentTB IS
END sevenSegmentTB;

ARCHITECTURE testbench OF sevenSegmentTB IS

COMPONENT sevenSegment
PORT (
    bcdInputs      : IN      STD_LOGIC_VECTOR(3 DOWNTO 0);
    seven_seg_outputs : OUT   STD_LOGIC_VECTOR(6 DOWNTO 0)
);
end COMPONENT;

CONSTANT PropDelay: time := 40 ns;
CONSTANT SimLoopDelay: time := 10 ns;
```

64

Testbench for Seven Segment Output cont'd

```
TYPE vector IS RECORD
  bcdStimulus: STD_LOGIC_VECTOR(3 downto 0);
  sevSegOut:   STD_LOGIC_VECTOR(6 downto 0);
END RECORD;

CONSTANT NumVectors: INTEGER:= 10;

TYPE vectorArray is ARRAY (0 TO NumVectors - 1) OF vector;

CONSTANT vectorTable: vectorArray := (
  (bcdStimulus => "0000", sevSegOut => "0000001"),
  (bcdStimulus => "0001", sevSegOut => "1001111"),
  (bcdStimulus => "0010", sevSegOut => "0010010"),
  (bcdStimulus => "0011", sevSegOut => "0000110"),
  (bcdStimulus => "0100", sevSegOut => "1001100"),
  (bcdStimulus => "0101", sevSegOut => "0100100"),
  (bcdStimulus => "0110", sevSegOut => "0100000"),
  (bcdStimulus => "0111", sevSegOut => "0001111"),
  (bcdStimulus => "1000", sevSegOut => "0000000"),
  (bcdStimulus => "1001", sevSegOut => "0000100")
);
```

65

Testbench for Seven Segment Output cont'd

```
SIGNAL StimInputs:   STD_LOGIC_VECTOR(3 downto 0);
SIGNAL CaptureOutputs: STD_LOGIC_VECTOR(6 downto 0);

BEGIN
  u1: sevenSegment PORT MAP (
    bcdInputs => StimInputs,
    seven_seg_outputs => CaptureOutputs);
  LoopStim: PROCESS
  BEGIN
    FOR i in 0 TO NumVectors-1 LOOP
      StimInputs <= vectorTable(i).bcdStimulus;
      WAIT FOR PropDelay;

      ASSERT CaptureOutputs == vectorTable(i).sevSegOut
        REPORT "Incorrect Output"
        SEVERITY error;

      WAIT FOR SimLoopDelay;
    END LOOP;
  END PROCESS;
END testbench;
```

Verify outputs!

66

File I/O

67

File I/O Example

- Example of file input/output using a counter
- Text file is vectorfile.txt
 - Has both input data and EXPECTED output data
 - **Will compare VHDL output data with EXPECTED output data!**

68

Counter with load

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY loadCnt IS
PORT (
    data:          IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    load:          IN STD_LOGIC;
    clk: IN STD_LOGIC;
    rst: IN STD_LOGIC;
    q:  OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
);
END loadCnt;
```

69

Counter with load cont'd

```
ARCHITECTURE dataflow OF loadCnt IS
SIGNAL cnt: STD_LOGIC_VECTOR (7 DOWNTO 0);

BEGIN
    counter: PROCESS (clk, rst)
    BEGIN
        IF (rst = '1') THEN
            cnt <= (OTHERS => '0');
        ELSIF (clk'event AND clk = '1') THEN
            IF (load = '1') THEN
                cnt <= data;
            ELSE
                cnt <= cnt + 1;
            END IF;
        END IF;
    END PROCESS;
    q <= cnt;
END dataflow ;
```

70

vectorfile.txt

```
#Format is Rst, Load, Data, Q
#load the counter to all 1s
0 1 11111111 11111111
#reset the counter
1 0 10101010 00000000
#now perform load/increment for each bit
0 1 11111110 11111110
0 0 11111110 11111111
#
0 1 11111101 11111101
0 0 11111101 11111110
#
0 1 11111011 11111011
0 0 11111011 11111100
#
0 1 11110111 11110111
0 0 11110111 11111000
```

input

EXPECTED output (i.e. VHDL should produce this output if it implements the circuit correctly) corresponding to the input

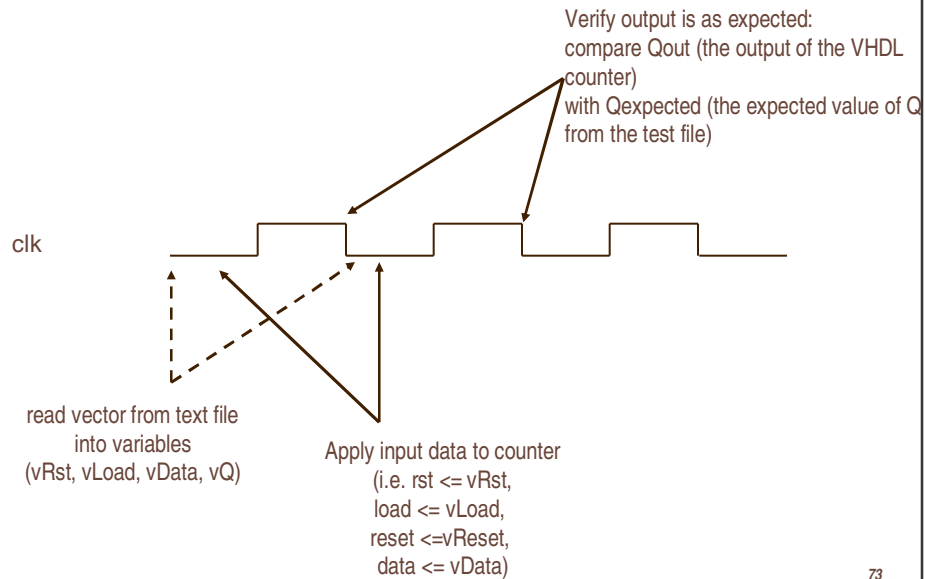
71

vectorfile.txt cont'd

```
#
0 1 11101111 11101111
0 0 11101111 11110000
#
0 1 11011111 11011111
0 0 11011111 11100000
#
0 1 10111111 10111111
0 0 10111111 11000000
#
0 1 01111111 01111111
0 0 01111111 10000000
#
#check roll-over case
0 1 11111111 11111111
0 0 11111111 00000000
#
# End vectors
```

72

Methodology to test vectors from file



Testbench (1)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_textio.all;

LIBRARY std;
USE std.textio.all;

ENTITY loadCntTB IS
END loadCntTB;
ARCHITECTURE testbench OF loadCntTB IS

COMPONENT loadCnt
  PORT (
    data:          IN   STD_LOGIC_VECTOR (7 DOWNTO 0);
    load:          IN   STD_LOGIC;
    clk:           IN   STD_LOGIC;
    rst:           IN   STD_LOGIC;
    q:             OUT  STD_LOGIC_VECTOR (7 DOWNTO 0)
  );
END COMPONENT;
```

Testbench (2)

```
FILE vectorFile: TEXT OPEN READ_MODE is "vectorfile.txt";

SIGNAL Data: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL Load: STD_LOGIC;
SIGNAL Rst: STD_LOGIC;
SIGNAL Qout: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL Qexpected: STD_LOGIC_VECTOR(7 DOWNTO 0);

SIGNAL TestClk: STD_LOGIC := '0';
CONSTANT ClkPeriod: TIME := 100 ns;
BEGIN

-- Free running test clock
TestClk <= NOT TestClk AFTER ClkPeriod/2;

-- Instance of design being tested
u1: loadCnt PORT MAP (Data => Data,
                      load => Load,
                      clk => TestClk,
                      rst => Rst,
                      q => Qout
                      );
```

75

Testbench (4)

```
-- File reading and stimulus application

readVec: PROCESS
  VARIABLE VectorLine: LINE;
  VARIABLE VectorValid: BOOLEAN;
  VARIABLE vRst: STD_LOGIC;
  VARIABLE vLoad: STD_LOGIC;
  VARIABLE vData: STD_LOGIC_VECTOR(7 DOWNTO 0);
  VARIABLE vQ: STD_LOGIC_VECTOR(7 DOWNTO 0);
  VARIABLE space: CHARACTER;
```

76

Testbench (5)

```
BEGIN

    WHILE NOT ENDFILE (vectorFile) LOOP
        readline(vectorFile, VectorLine); -- put file data into line

        read(VectorLine, vRst, good => VectorValid);
        NEXT WHEN NOT VectorValid;
        read(VectorLine, space);
        read(VectorLine, vLoad);
        read(VectorLine, space);
        read(VectorLine, vData);
        read(VectorLine, space);
        read(VectorLine, vQ);

        WAIT FOR ClkPeriod/4;
        Rst <= vRst;
        Load <= vLoad;
        Data <= vData;
        Qexpected <= vQ;

        WAIT FOR (ClkPeriod/4) * 3;
    END LOOP;
```

77

Testbench (6)

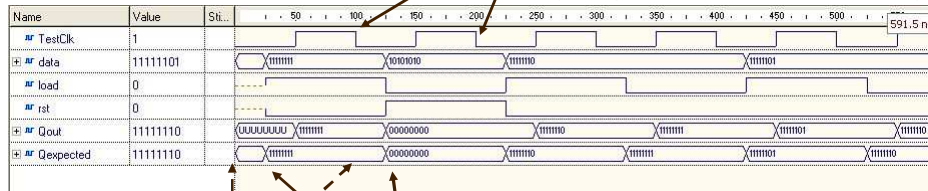
```
    ASSERT FALSE
        REPORT "Simulation complete"
        SEVERITY NOTE;
    WAIT;
END PROCESS;

-- Process to verify outputs
verify: PROCESS (TestClk)
variable ErrorMsg: LINE;
BEGIN
    IF (TestClk'event AND TestClk = '0') THEN
        IF Qout /= Qexpected THEN
            write(ErrorMsg, STRING("Vector failed "));
            write(ErrorMsg, now);
            writeline(output, ErrorMsg);
        END IF;
    END IF;
END PROCESS;
END testbench;
```

78

Simulation Waveform

Verify output is as expected:
compare Q (the output of the VHDL
counter)
with Qexpected
(the expected value of Q from the test file)



read vector from text file
into variables
(vRst, vLoad, vData, vQ)

Apply input data to counter
(i.e. `rst <= vRst`,
`load <= vLoad`,
`reset <= vReset`,
`data <= vData`)

79

Hex format

In order to read/write data in the hexadecimal
notation, replace

read with `hread`, and
write with `hwrite`

80

Note on test file

- This example showed a test file that had both the control commands (i.e. load, reset), and the actual data itself
- Often the test file just has the input and output vectors (and no load, reset, etc.)