

ICLAB 2019 Spring

Paper Based Exam

Lec01(10%)

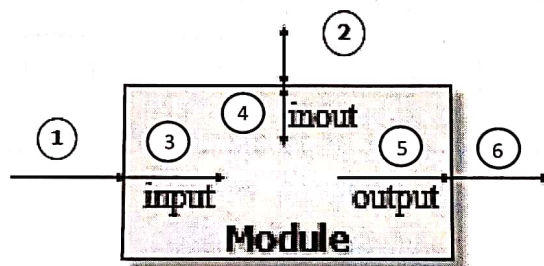
1.

a. Please clearly explain what is cell-based and full-custom design flow? (2%)

b. What is the advantage and disadvantage of them? (2%)

2.

Please write down all the possible data type below this picture. (6%)



Lec02(20%)

1.

Please write down the advantage and disadvantage for the synchronous and asynchronous (at least two things in each case). (5%)

According to the provided code, finish the below waveform. (before the dotted line the c value is 2) (2.5% + 2.5%)

c is a five-bits signal

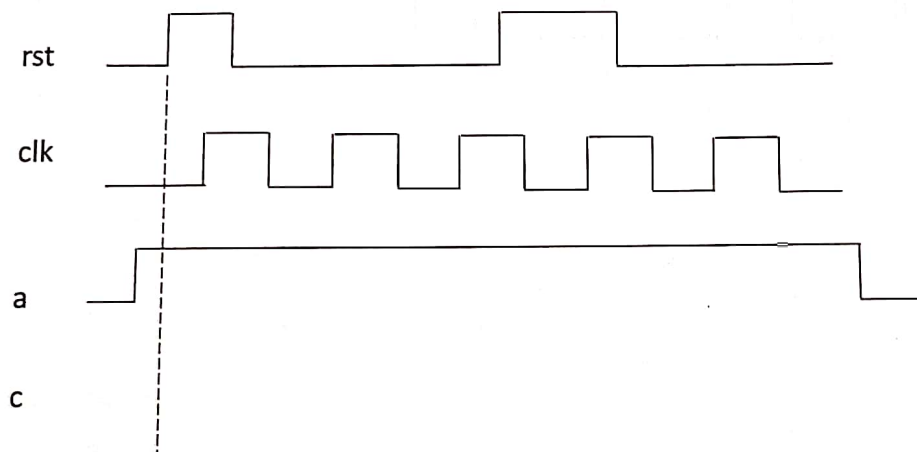
always @(posedge clk)

begin

if(rst) c <= 0;

else c <= a + 1;

end



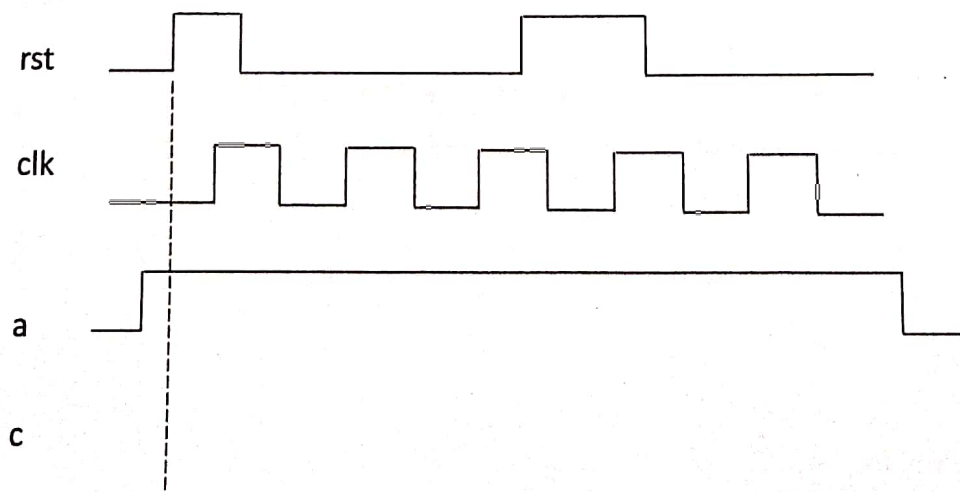
always @(posedge clk or posedge rst)

begin

if(rst) c <= 0;

else c <= a + 1;

end



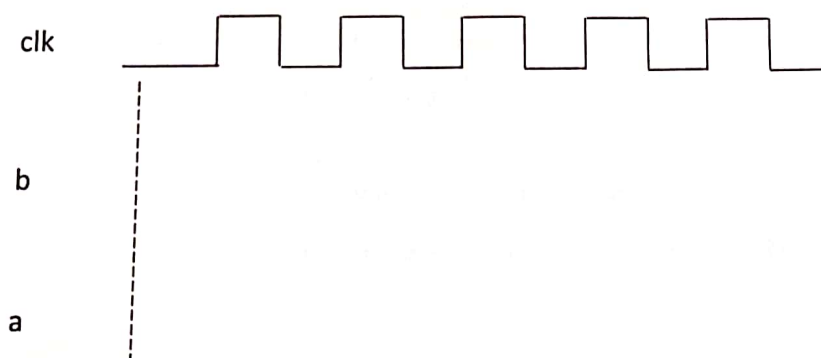
2.

According to the provided code, finish the below waveform.

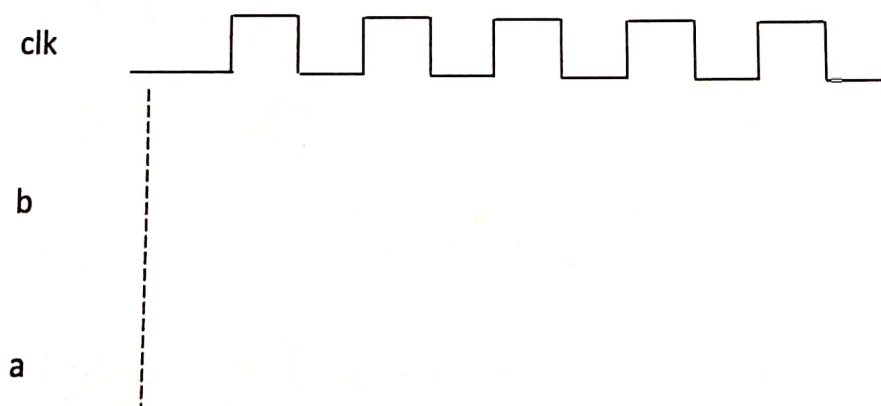
(the initial block start at the dotted line and the #1 means the very little delay)

Clock period is 10ns.(5% + 5%)

```
initial begin
    a=0; b=1;
end
always @(posedge clk)
begin
    b = #1 a;
    a = #1 b;
end
```



```
initial begin
    a=0; b=1;
end
always@(posedge clk)
begin
    b <= #1 a;
    a <= #1 b;
end
```



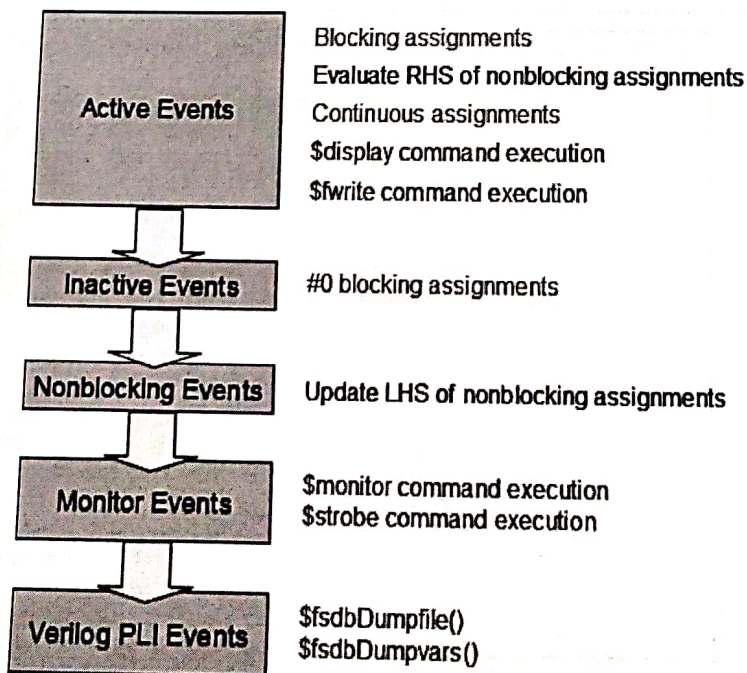
Lec03(20%)

1. According to the figure of priority below, if we wrote the program below, what will the simulation output show? Please write the result row by row separately (not only the data but also the words that show in the command.).(10%)

```

1 module priority;
2
3 reg a, b;
4
5 initial begin
6     a = 0;
7     b = 0;
8     #1;
9     a = 0;
10    b = 1;
11    a <= b;
12    b <= a;
13
14    $display("%0dns : \display: a=%b b=%b" , $stime, a, b);
15    $monitor("%0dns : \monitor: a=%b b=%b" , $stime, a, b);
16    $strobe ("%0dns : \strobe : a=%b b=%b\n", $stime, a, b);
17    #0 $display("%0dns : #0      : a=%b b=%b" , $stime, a, b);
18
19    #1 $monitor("%0dns : \monitor: a=%b b=%b" , $stime, a, b);
20    $strobe ("%0dns : \strobe : a=%b b=%b\n", $stime, a, b);
21    $display("%0dns : \display: a=%b b=%b" , $stime, a, b);
22    $fwrite(fp, "%0dns : \fwrite : a=%b b=%b\n", $stime, a, b);
23    #0 $display("%0dns : #0      : a=%b b=%b" , $stime, a, b);
24
25 end
26
27 endmodule

```



2. There are 6 syntax errors in the following code, please find 5 errors and correct them.(10%)

```
`timescale 1ns/10ps
module TESTBED;

wire in_a, in_b, in_sel;
wire rst, clk;
wire out;

initial begin
    `ifdef RTL
        $fsdbDumpfile("DESIGN.fsdb");
        $fsdbDumpvars(0,"mda");
    `endif
    `ifdef GATE
        $fsdbDumpfile("DESIGN SYN.fsdb");
        $fsdbDumpvars(0,"mda");
    `endif
end

DESIGN (.clk(clk),.rst(rst),.in_a(in_a),.in_b(in_b),.in_sel(in_sel),.out(out));

PATTERN(.clk(clk),.rst(rst),.in_a(in_a),.in_b(in_b),.in_sel(in_sel),.out(out));

endmodule
```

```
module PATTERN( clk, rst, in_a, in_b, in_sel, out);

output in_a, in_b, in_sel;
output rst, clk;
input out;

reg in_a, in_b, in_sel, rst, clk;

always #1 clk=~clk;
initial begin
    clk=0;rst=1;
    #5rst=0;
    in_a=0;in_b=1;in_sel=0;
    #5sel=1;
end

endmodule
```

```
module DESIGN( clk, rst, in_a, in_b, in_sel, out);

input in_a, in_b, in_sel;
input rst, clk;
output reg out;

reg 2_to_1;

assign out=(rst==1'b1)? 1'b0:2_to_1;
always @(posedge clk or posedge rst) begin
    mux_task;
end

task mux_task;
begin
    if(in_sel==0) 2_to_1 = in_a;
    else 2_to_1 = in_b;
end
endtask

endmodule
```

Lec04(20%)

1. Briefly describe what operator inference and instantiate IP are. What are their respective advantages and disadvantages?(5%)

2. (a) Briefly describe what clk-to-Q propagation delay and logic propagation delay are?(5%)

(b) Give the following formulas in terms of (t_{cycle} : clock cycle 、 t_{setup} : setup time 、 t_{hold} : hold time 、 t_{pcq} : clk-to-Q propagation delay 、 t_{ccq} : clk-to-Q contamination delay 、 t_{pd} : logic propagation delay 、 t_{cd} : logic contamination delay).(10%)

	Setup time	Hold time
Data required time =		
Data arrival time =		
Criterion :	() > ()	() > ()

Lec05(15%)

1. When using memory, we use memory compiler to generate two files used in our design flow, v file and lib file. Separately explain what are this two files and which part of design flow are they used in.(10%)

2. Why using sequential logic to describe control signals of memory such as WEN, A, D, ..., or else will be easier to face hold time violation in gate level simulation.(5%)


Lec06(15%)

1. Please list three specify libraries in synthesis flow, and give a brief description of these libraries.(7.5 %)
2. Please explain the difference between Top-down compile and Bottom-up compile, and list their respective advantage.(7.5%)

1.

Cell-based ASIC

- ✓ **Cell-based IC (CBIC)**
 - use **pre-designed** logic cells (known as standard cells) and micro cells (e.g. microcontroller)
 - designers save time, money, and reduce risk
 - each standard cell can be optimized individually
 - all mask layers are customized
 - custom blocks can be embedded




ICLAB NCTU Institute of Electronics

7

Full-Custom Design

- ✓ **An engineer designs some or all of the logic cells, circuits, layout specifically for one ASIC**
 - required cells/IPs are not available
 - existing cell libraries are not fast enough
 - logic cells are not small enough or consume too much power
 - technology migration (mixed-mode design)
 - **demand long design cycle**
- ✓ **Not our focus**



ICLAB NCTU Institute of Electronics

10

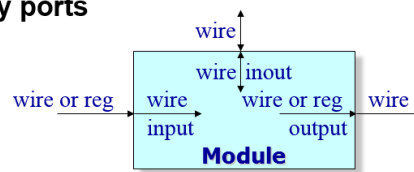
2.

Port

✓ Interface is defined by ports

✓ Port declaration

- input : input port
- output : output port
- inout : bidirectional port



✓ Port connection

- input : only wire can be assigned to represent this port **in** the module
- output : only wire can be assigned to represent this port **out** of module
- inout : **register assignment is forbidden** neither in module nor out of module **[Tri-state]**



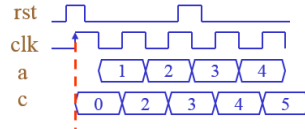
1.

Flip-Flop Designs

✓ Register with synchronous reset

- Syntax: @(posedge clk) : for synchronous reset

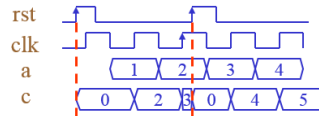
```
EX.
always @(posedge clk)
begin
    if(rst)    c <= 0;
    else      c <= a + 1;
end
```



✓ Register with asynchronous reset

- Syntax: @(posedge clk or posedge rst)

```
EX.
always @(posedge clk or posedge rst)
begin
    if(rst)    c <= 0;
    else      c <= a + 1;
end
```



Flip-Flop Designs – Synchronous Resets (cont.)

✓ Advantages

- Easier work with cycle based simulator
- Typical recommended for DFT design
 - DFT (Design for test)
- Easier for ECO (Engineering Change Order)
- Glitch filtering from reset combinational logic
 - Filter the small glitch of reset between clock
- Glitch filtering if reset is in a mission-critical application
 - When reset is generated by a set of internal condition, it can filter the glitch of logic equations between clock.

✓ Disadvantages

- May not be able come out unknown X during simulation
- Adds delay to data path
- Can't be reset without clock signal



Flip-Flop Designs – Asynchronous Resets (cont.)

✓ Advantages

- Reset is immediate
- No problem related to unknown X propagation in simulation
- Does not interfere or add extra delay to data path
- Very easy to implement
- Reset is independent of clock signal

✓ Disadvantages

- Noisy reset line could cause unwanted reset
 - This can be filtered
- Difficult for ECO
- DFT prefer synchronous designs



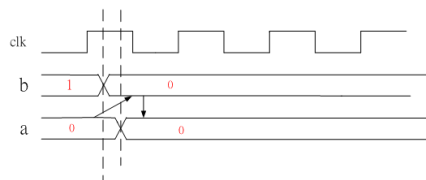
2.

Assignments – Procedural Assignment (cont.)

✓ Blocking procedural assignment

- Syntax: **<value> = <timing_control><expression>**
- Must be executed before executing the statement that follow it in a sequential block.
 - Example

```
initial begin
  a=0;b=1;clk=0;
end
always @(posedge clk)
begin
  b = #1 a;
  a = #1 b;
end
```



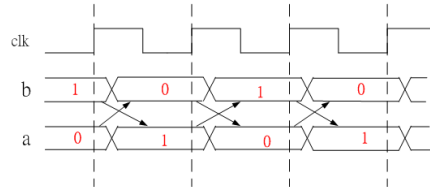
Assignments – Procedural Assignment (cont.)

✓ Non-Blocking procedural assignment

- Syntax: `<value> <= <timing_control><expression>`
- The statements are executed **within the same time step** without regard to order or dependence upon each other.
- Evaluating Non-blocking divided into two steps:

- Example

```
initial begin
  a=0;b=1;clk=0;
end
always@(posedge clk)
begin
  b <= #1 a;
  a <= #1 b;
end
```



- ◆ Step1 : at posedge clk the simulator evaluates the RHS's new value, and scheduled when to update LHS.
- ◆ Step2 : at 1ns after posedge clk the simulator updates LHS. Because the given delay (#1) is expired.



Lecture03

1.

```
# 1ns :$display: a=0 b=1
# 1ns :#0       : a=0 b=1
# 1ns :$monitor: a=1 b=0
# 1ns :$strobe  : a=1 b=0
#
# 2ns :$display: a=1 b=0
# 2ns :#0       : a=1 b=0
# 2ns :$monitor: a=1 b=0
# 2ns :$strobe  : a=1 b=0
```

2.

1. In module DESIGN(...), the variable "out" should be declared as "wire " instead of "reg".
 2. In module DESIGN(...), the variable "2_1" violates the naming rule, it should begin with _ or alphabets, such as "w2_1".
 3. In module DESIGN(...), task can't be synthesizable. So can not be use in design.
 4. In module PATTERN (...), there's no finish in pattern. The simulation will keep running.
 5. In dumping the waveform in TESTBED, it need to add the cmd "\$sdf_annotate("DESIGN _SYN.sdf", My_ DESIGN);" in gate simulation.
 6. In TESTBED, when instantiating DESIGN and PATTERN modules, a name should be given, such as DESIGN mux_1(.clk(clk),.....);
-(more than 6)

Lecture04

1.

Ans:

Operator inference:

- ✓ Use the HDL operator in description
- ✓ Convenient
- ✓ Sometimes it is inefficient when synthesizing
- ✓ Supply default function only, cannot use special function.

Instantiate IP:

- ✓ To instantiate a synthetic module manually and explicitly
- ✓ Need to include a reference to the synthetic module in HDL code.
- ✓ Supply different architecture for implementation
- ✓ Apply pre-compiling sub-blocks speeds up the synthesis for large design.

2.

(a) Propagation delay is the maximum amount of time from an input changes until all outputs reach steady state. Clk-to-Q is the propagation delay in flip-flop, and logic propagation delay is the propagation delay in combinational logic.

(b)

	Setup time	Hold time
Data required time =	$t_{cycle} - t_{setup}$	t_{hold}
Data arrival time =	$t_{pcq} + t_{pd}$	$t_{ccq} + t_{cd}$
Criterion :	$(t_{cycle} - t_{setup}) > (t_{pcq} + t_{pd})$	$(t_{ccq} + t_{cd}) > (t_{hold})$

Lecture 05

1. When using memory, we use memory compiler to generate two files used in our design flow, v file and lib file. Separately explain what are these two files and which part of design flow are they used in.

v file : **Behavior model** which cannot be synthesized. Use in **rtl simulation** and **gate level simulation**.

lib file : **Library for synthesis** & APR. It gives information such as delay, timing constraint, area, capacitance of ports of the memories, that enable the synthesizer to perform static timing analysis (STA) based on these data. However unlike designware, the memory will not be synthesized out. Since memory is a hard macro, a compacted layout architecture which is not built from the standard cell gate. Only the timing / area and other information are used for **synthesis**.

2. Why using sequential logic to describe control signals of memory such as WEN, A, D, ..., or else will be easier to face hold time violation in gate level simulation.

With sequential logic, Flip-Flop output is connected to the Memory input. If the $\text{clk} \rightarrow \text{q}$ delay is smaller than the hold time check of memory, a hold time violation will occur.

Lecture06

Synthetic_library:

Specifies additional **DesignWare libraries** for optimization purposes.

Efficient implementations for adders, comparators, multipliers

Link_library:

Specifies a list of libraries that Design Compiler can use to resolve design references.

Target_library:

During synthesis, compiler selects gates from target library(usually put **technology library** here).

It also calculates the timing of the circuit, using the vendor-supplied (UMC, TSMC...) timing data of the .lib or .db file.

Top-down compile, in which the top-level design and all its subdesigns are compiled together

Bottom-up compile, in which the individual subdesigns are compiled separately, starting from the bottom of the hierarchy and proceeding up through the levels of the hierarchy until the top-level design is compiled.

Top-down compile achieve less synthesis time but **Bottom-up compile** achieve better performance