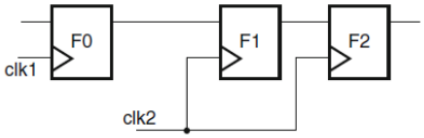
2021 Spring

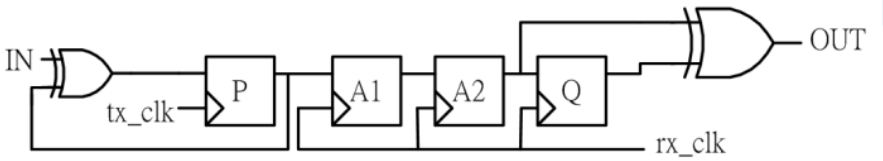
Name: **Student ID:** **Account: iclab**

Total score: 100%

1. [10%]:
   Please clearly explain how double flop synchronizer and XOR double flop synchronizer work in our design if we use them in a CDC (clock domain crossing) design and why do we need to use synchronizer in a CDC design.

   Double flop synchronizer: 

   F1 samples the asynchronous input signal into the destination clock domain, and then the signal is sampled by the same clock into F2, which is expected to be a stable and valid signal. Notice the signal must be high long enough to ensure not being lost due to metastability issue.

   XOR double flop synchronizer: 

   Similar to double flop synchronizer but with a feedback XOR circuit at the end, which makes the signal can be high for only one cycle and the output signal will also be high for only one cycle (no metastability issue need to be concerned).
   why? To avoid metastability (the unstable status due to non-ideal data transition), we have to ensure no data transition during setup/hold timing check. CDC will inevitably face this problem because the delay between registers of two different clock domain will alter through time.
2. [4%]:
   We inserted synchronizers between multi clock domains. However, without further settings, the slack will be negative in synthesis and timing violation will occur in gate level simulation. Express what action must be done to remove the negative slack issue in synthesis (2%) and timing violation issue in gate level simulation (2%).
   negative slack issue in synthesis:
   (1) Specify a multicycle path using set_multicycle_path in .sdc (design constraint file).
   (2) 01_run_dc will read .sdc file and do the synthesis and generate .sdf (standard delay format).
   (3) Specify paths that cross two different time domains using set_annotated_check in _pt.sdc file
   (4) 02_run_pt will read .sdf file generated from 01_run_dc and generate _pt.sdf (standard delay format).
   (5) 03_GATE gate simulation will read _pt.sdf file with timing information and do the simulation.
   timing violation in gate level simulation:
3. [13%]:
   There are three types of simulation flow supported by the Primetime tool for power simulation. Please

explain all of them. You need to describe the name, file of information required for simulation, and rank of precision (write “1” for the most precise simulation, write “3” for the least precise simulation). Also, there are two types of signal activity files for power simulation. Please describe them respectively.

Name of the flow (3%)	Required information (3%)	Rank of precision (3%)
VCD Flow	VCD file	1
SAIF Flow	SAIF file	2
Vector-free Flow	No file needed	3

Describe the signal activity files (4%):

VCD file: Value Change Dump, .vcd. Included in Verilog HDL IEEE Standard. Contains value changes of a signal. i.e. at what times signals change their values.

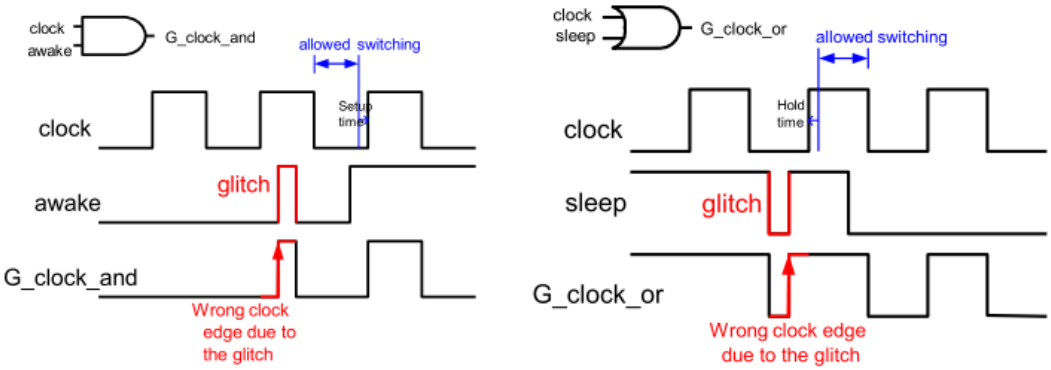
SAIF file: Switching Activity Interchange Format, .saif. Contains toggle counts and time information like how much time a signal was in 1-state, 0-state, X-state. It contains cumulative information of .vcd file.

4. [7%]:

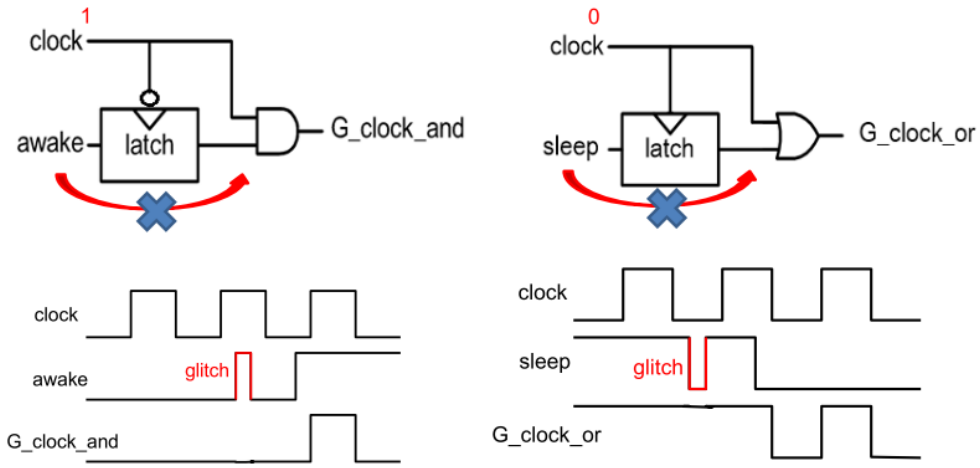
Clock gating can be implemented by either AND-gating or OR-gating. Which one is better? (1%) Why?(3%) Explain why a glitch may happen when we are using clock gating and how to avoid it?(3%)

OR-gating is better because it consumes less power than AND-gating.

Glitch may happen when the signal changed in forbidden half cycle (for AND-gating, the allowed switching cycle is clock at low period; for OR-gating, clock at high period). As the waveform below show:



To avoid gltich, and latch to AND-gating or OR-gating.



5. [4%]:

(1) Please describe the major functionality of “typedef” syntax.

(2) Please describe the major functionality of “enum” syntax.

(1) “typedef”: User defined type (UDT). Allow users to create their own names for type definitions that they will use frequently.

(2) “enum”: Like enumeration in c++, which defines a set of named values, it is a distinct type whose value is restricted to a range of values. Also, it provides built-in assertion.

6. [10%]:

(1) Please explain what are pre_randomize() and post_randomize() function in every class? How to use these functions?

(2) Please write the corresponding code if we want to create a class called “random_val” which contains a random member variable called “rand_val”, which is a 32-bits variable. The possible values of this variable are 32’h10000000, 32’h01000000, and 32’h00000001 and these values need to be cyclic randomized without repeating.

(1) pre_randomize() and post_randomize() are two callback functions that are automatically called by randomize() function before and after computing random values. We can use pre_randomize() function to set the constraints for randomization, and use post_randomize() function to do some process to the randomized values. To use these functions, we need to override exiting empty pre_randomize() and post_randomize(). For example:

```
class Apple{
    int color;
    constraint c{ /* … */ }
    function pre_randomize();
        /* what you want to do before randomization */
    endfunction
    function pre_randomize();
        /* what you want to do after randomization */
    endfunction
endclass
```

(2)

```
class random_val;
    randc int flag;
    bit [31:0] rand_var;
    constraint limit { flag inside { 1, 2, 3}; }
    function post_function();
        if (flag==1)    rand_var = 32’h10000000 ;
        else if (flag==2)    rand_var = 32’h01000000 ;
        else if (flag==3)    rand_var = 32’h00000001 ;
    endfunction
endclass
```

7. [12%]:

If we want to check whether our design hits the specification, we can write the assertion in the checker.sv file. There are three signals, A, B, C, in the design, and three signals will only be high consecutive unfixed cycles. Please finish the three assertions below. Notice that you cannot use always block here.

(1) If A happens, 1 or more cycles later B should be 0 when C happens.
(2) If A changes from 1 to 0 between one rising clock, then B must happen 10 times in a row after 2-10 cycles.
(3) If A is 0, B was high before 2 clock cycles.

Assertion1: assertion property (@(posedge clk)) …… ;
Assertion2: assertion property (@(posedge clk)) …… ;
Assertion3: assertion property (@(posedge clk)) …… ;

```
Assertion1: assert property ( @(posedge clk) A ##[1:$] C |-> B===0    )
else begin
      $display("Assertion1 is violated");
      $fatal;
end
Assertion2: assert property ( @(posedge clk) %fall(A) |-> ##[2:10] B [*10] )
else begin
      $display("Assertion2 is violated");
      $fatal;
end
Assertion3: assert property ( @(posedge clk) (A===0) |-> $past(B, 2)===1 )
else begin
      $display("Assertion3 is violated");
      $fatal;
end
```

8. [6%]:
Given the Verilog code shown below, calculate the total toggle coverage percentage of register 'data' after simulation. Please briefly explain your reasons. 60%

```
reg [4:0] data;
initial begin
        data = 8;       // 01000
        #50;
        data = 18;      // 10010
        #50;
        data = 10;      // 01010
        #50;
        data = 27;      // 11011
        #50;
        data = 19;      // 10011
end
```

	Data[4]	Data[3]	Data[2]	Data[1]	Data[0]
0 → 1	O	O	X	O	O
1 → 0	O	O	X	X	X

9. [10%]:

What libraries and files are used in APR. Please explain the purpose of including these files.

1. Lib Library: Includes timing information for each cell.
2. LEF Library: Process and cell information. Ex: metal layer, routing pitch, …
3. RC Extraction: For further power analysis and simulation.
4. CeltIC Library: cdB model (noise model) for each cell. Used in SI optimization phase.
5. GDSII: Planar geometric shapes and other information about the layout in hierarchical form.
6. DESIGN_SYN.v: The target design you want to do APR with.
7. CHIP_SHELL.v: To specify IO, IO power, core power of the chip.
8. CHIP.io: IO pad location file. To specify the locations of each IO pad.

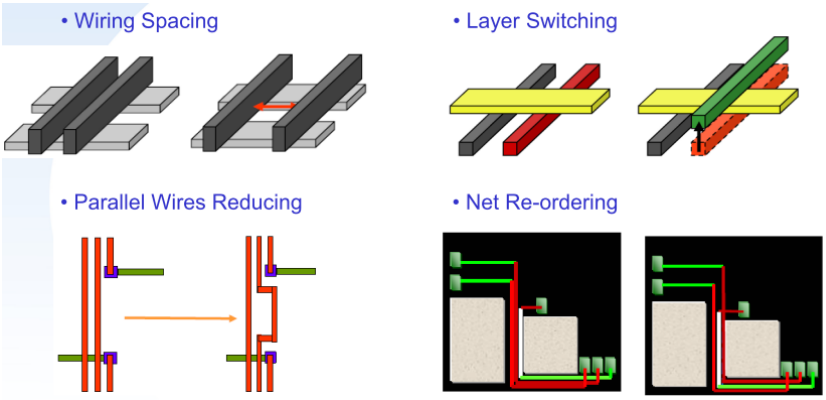
10. [6%]:

Explain what's Signal Integrity (SI) Issue, and write at least two solutions.

Signal Integrity (SI) is a set of measures of the quality of an electrical signal. All signals are subject to effects such as noise, distortion and loss. Thus, we need to make sure no errors will occur due to these effects.

The main issues are crosstalk, charge sharing, supply noise, leakage, propagated noise, overshoot, under shoot.

SI prevention:



11. [6%]:

With a more advanced process, does IR drop tend to be more serious or more trivial? List at least two reasons.

More serious.

As fabrication technologies have become smaller, the size of the wires has also been getting smaller, while physical chip dimensions have stayed roughly the same. This means that wires have become thinner but have not gotten shorter. That leads to an increase in resistance per unit length.

Another trend in semiconductor design has been a reduction in operating voltage, meaning that small changes in supply voltage may represent an increasing percentage of the digital swing and potentially lead to incorrect logic values being seen.

12. [12%]:

a. [4%] What is the difference between power/rail analysis?

b. [4%] What input files do we need when running vetor-based power analysis and rail analysis respectively? What's the content in the files?

c. [4%] If we use different input files to do the vector-based power analysis, will the result of rail

analysis be different? Why?

a. power analysis analyze how much energy is consumed by the circuit, while rail analysis analyze whether the powerplan can provide enough power for each cell.

b. (RCGen.tch, lefdef.layermap, *.cl, *.vcd)(*.cl, static_VDD.ptiavg, static_VSS.ptiavg)

c. yes. Using different input files to do vector-based power analysis will generate different output files (power.rpt, static_VDD.ptiavg, static_VSS.ptiavg). Rail analysis will take the files vector-based power analysis generated (static_VDD.ptiavg, static_VSS.ptiavg) to analyze. Therefore, different input files for power analysis will affect rail analysis as well.