Lab 4. Timing Simulation

Digital System Design and Experiment

Graduate School of Convergence Science and Technology

Seoul National University

Mobile Multimedia Systems Group

Goal of Lab4

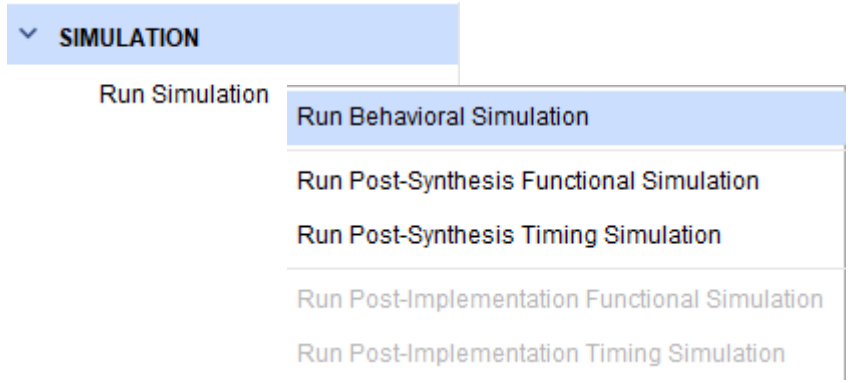
- 지난 시간에 배운 Synthesis/Implementation 방법을 복습한다.
- 4-bit Barrel Shifter를 설계해 본다.
- 설계한 Shifter를 사용한 Post-Synthesis/Implement timing simulation을 진행한다.
  - Synthesis 이후 Simulation 결과가 어떻게 변경되는지를 확인한다.
  - Implementation 이후 Simulation 결과가 어떻게 변경되는지를 확인한다.
- Barrel Shifter 의 최대 동작 Clock Speed를 찾아본다.
  - 추후 진행할 프로젝트 혹은 설계에 있어서 각각의 Standard Cell의 동작 Spec을 알아보는 법을 익힌다.

1.

Post Simulation

Various Simulation

- Simulation의 종류
  - Behavioral Simulation
  - Post-Synthesis Functional Simulation
  - Post-Synthesis Timing Simulation
  - Post-Implement Functional Simulation
  - Post-Implement Timing Simulation

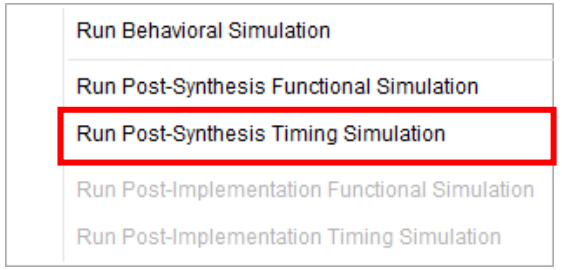


- 각각의 Simulation 역할
  - Behavioral Simulation
    - RTL 코드의 동작을 보는 Simulation. 처음 Verilog Code를 짤 때 많이 사용
  - Functional Simulation
    - Synthesis 혹은 Implementation 이후 변환된 Code의 동작을 보는 Simulation.
    - 일반적으로 Behavioral과 유사하게 나온다. 정밀한 Timing을 반영하지 않았다.
  - Timing Simulation
    - Synthesis 혹은 Implementation 이후 **Timing 정보(.sdf 파일)를 반영**한 Simulation
    - **FPGA에 올렸을 때의 동작 여부와 가장 관련 있다.**

Post Timing Simulation

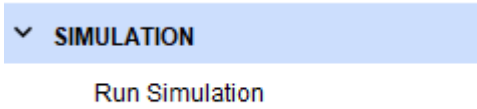
- Post Synthesis Timing Simulation
  - Synthesis 후에 활성화 된다.
  - 각각의 소자들의 Hold/Setup Time, Delay 등이 반영

Synthesis 전



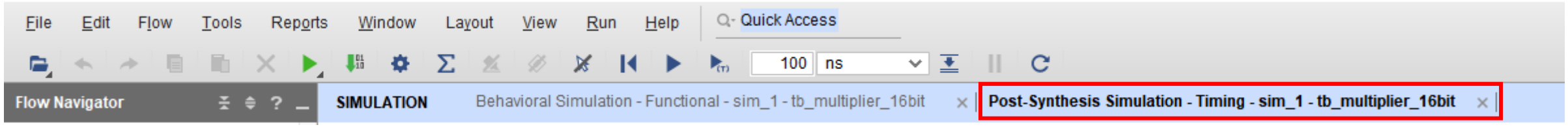
Synthesis 후

- Post Implementation Timing Simulation
  - Implementation 후에 활성화 된다.
  - 소자들의 Delay 정보들과 배치에 따른 Wire Delay 등이 반영



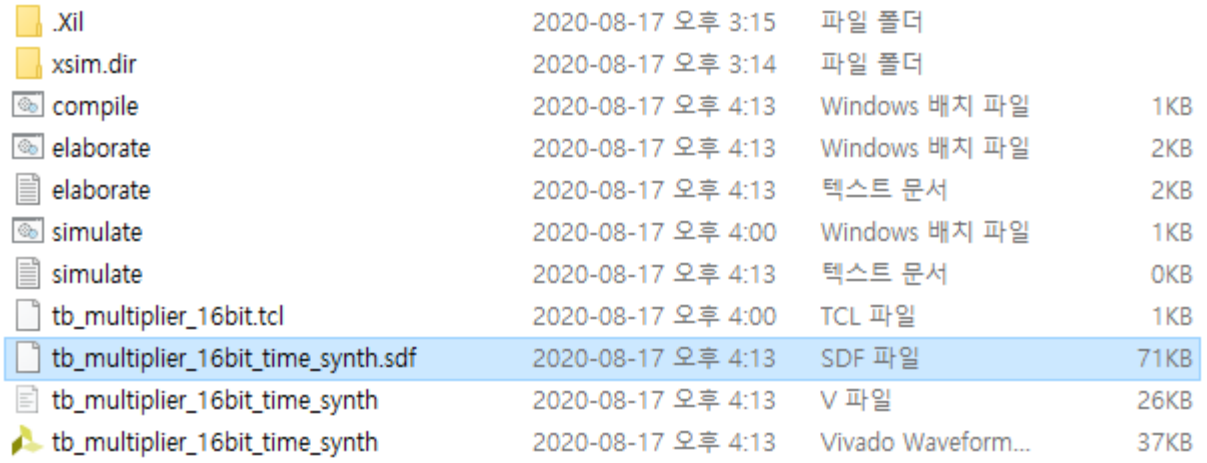
Post Timing Simulation

- Run Timing Simulation을 누르면?
  - 아래 그림의 빨간 박스와 같이 Simulation 창이 하나 더 나타나게 된다.
  - 각각의 이름을 눌러서 전환이 가능하다.
  - 각각의 이름 옆에 있는 x 표시를 통해서 끌 수 있다.

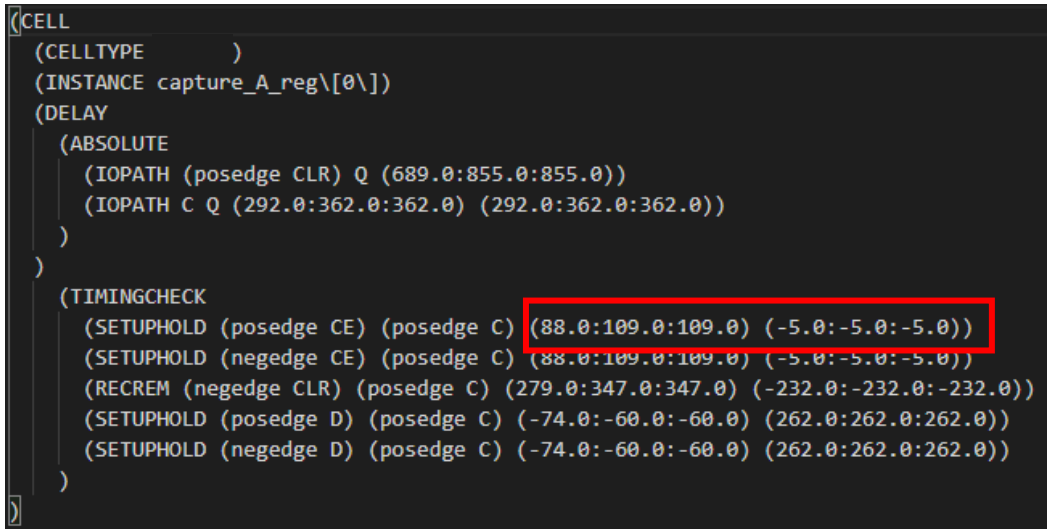


SDF 파일

- SDF 파일 생성
  - Vivado에서는 오로지 Run **** Timing Simulation을 눌러야 생성
  - 프로젝트이름.sim/sim_1/synth/timing/xsim/ 경로에 존재
- SDF 파일 구조
  - 수업시간에 배운 [min:typ:max] 와 같은 구문이 존재
  - 보다 정확한 Simulation을 위해 Tool이 자동으로 만들어 줌



생성된 SDF 파일



SDF 파일 내부

2.

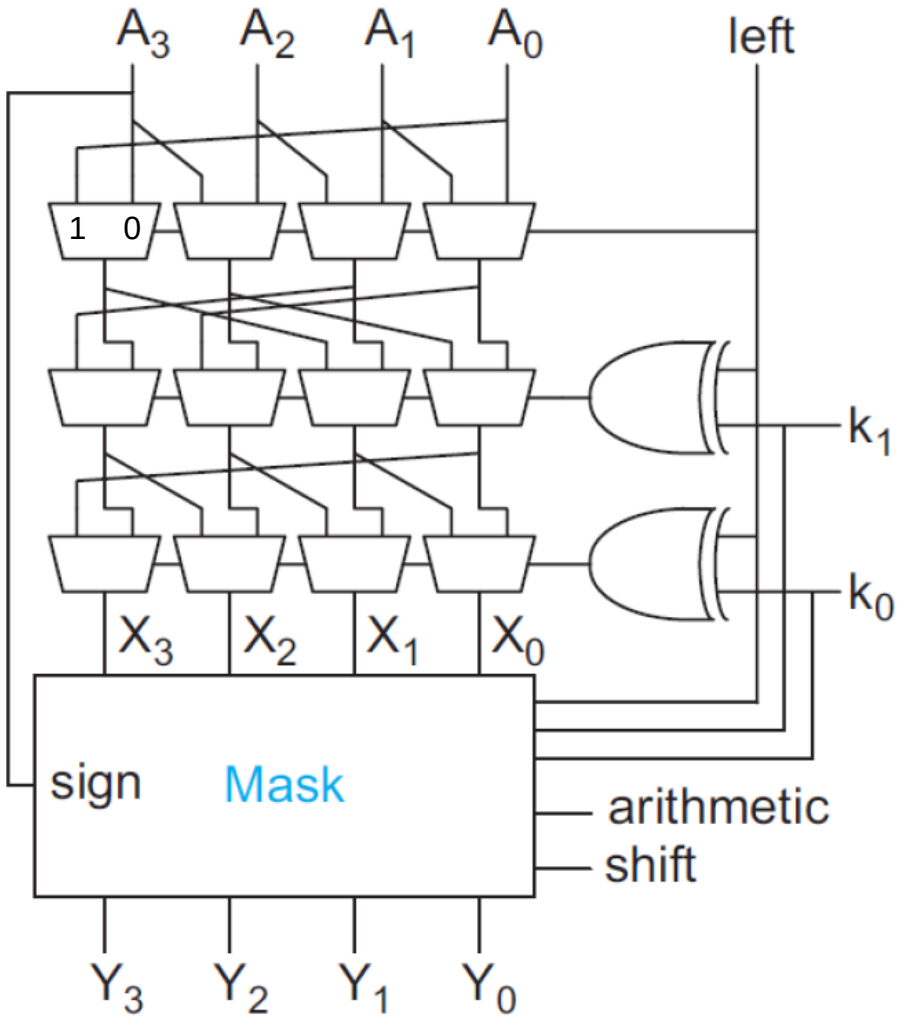
4-bit barrel shifter design with synth./impl.

4bit Barrel Shifter 설계

- Barrel shifter
  - Rotation, Logical shift, Arithmetic shift 등의 연산을 수행
  - 순수 Combinational Logic만을 사용
  - 단일 Clock 내에 k-bit shift / rotate 하는데 많이 사용

- 구성
  - 크게 2개의 부분으로 구성
  - Mux로 Rotate결과를 만드는 부분
  - Masking을 통해서 원하는 결과를 만드는 부분

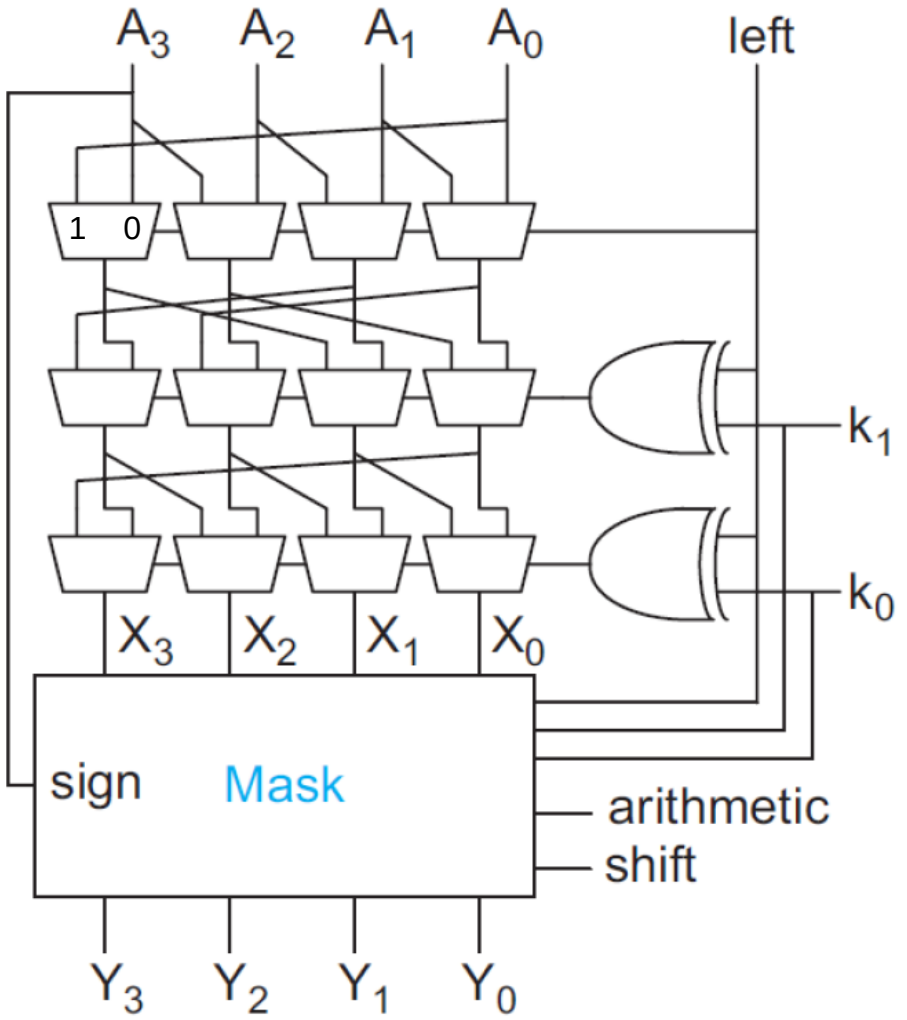
- 참고
  - Masking 개념은 구글에 <bit masking>을 검색해보자
    - Ex. A[3:0] & (~B[3:0])
  - Right Arithmetic Shift가 매우 어려울 수 있다
    - 우선 손으로 Making Logic을 적어보는 것을 추천



4bit Barrel Shifter 설계

- Mux로 Rotate결과를 만드는 부분 (assign문 사용)
  - 0 : Mux의 오른쪽 입력을 출력함
  - 1 : Mux의 왼쪽 입력을 출력함
- Masking 부분 (case문 사용)
  - Shift, Left, Arithmetic signal에 따라 다르게 동작해야 함
  - Masking은 쉽게 4'hF 비트와 And / Or / Not / >> / << 등을 통해서 구현가능 (Sign bit도 사용하는 경우 존재)

{shift, left, arithmetic}	Masking	동작
000	None	R Rotation
001	None	R Rotation
010	None	L Rotation
011	None	L Rotation
100	0 masking	R Logic Shift
101	Sign masking	R Arithmetic Shift
110	0 masking	L Logic Shift
111	0 masking	L Arithmetic Shift



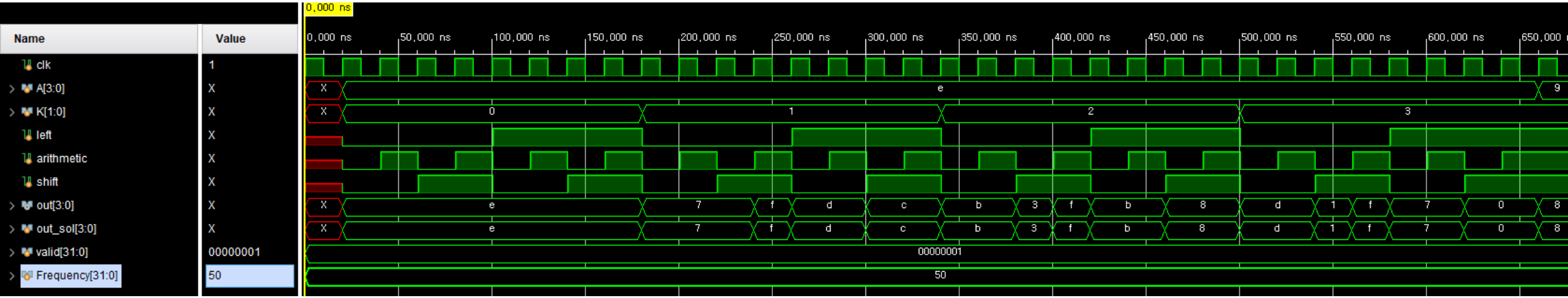
Target Behavior

- Testbench
  - Testbench는 주어지며, Behavior Model은 아래와 사진과 같이 나타난다.

- Testbench 주의사항
  - 주어진 Testbench에서는 Input이 0.6ns 지연되어서 들어간다.
    - Flip-Flop(FF)에서 C-Q Delay를 0.6ns로 가정하고서 Testbench를 구현함
    - 다른 FPGA 혹은 SoC를 만들 경우 FF의 CQ Delay를 파악할 필요 존재
  - Testbench의 **define 부분을 바꾸면 Clock Frequency를 변경할 수 있다.** →

```
`define HALF_CLK_PERIOD 10
`define FULL_CLK_PERIOD (2*`HALF_CLK_PERIOD)
```

    - Frequency Term으로 간단하게 확인할 수 있도록 함 (Radix decimal로 변경해서 보기)



Synthesis 주의 사항

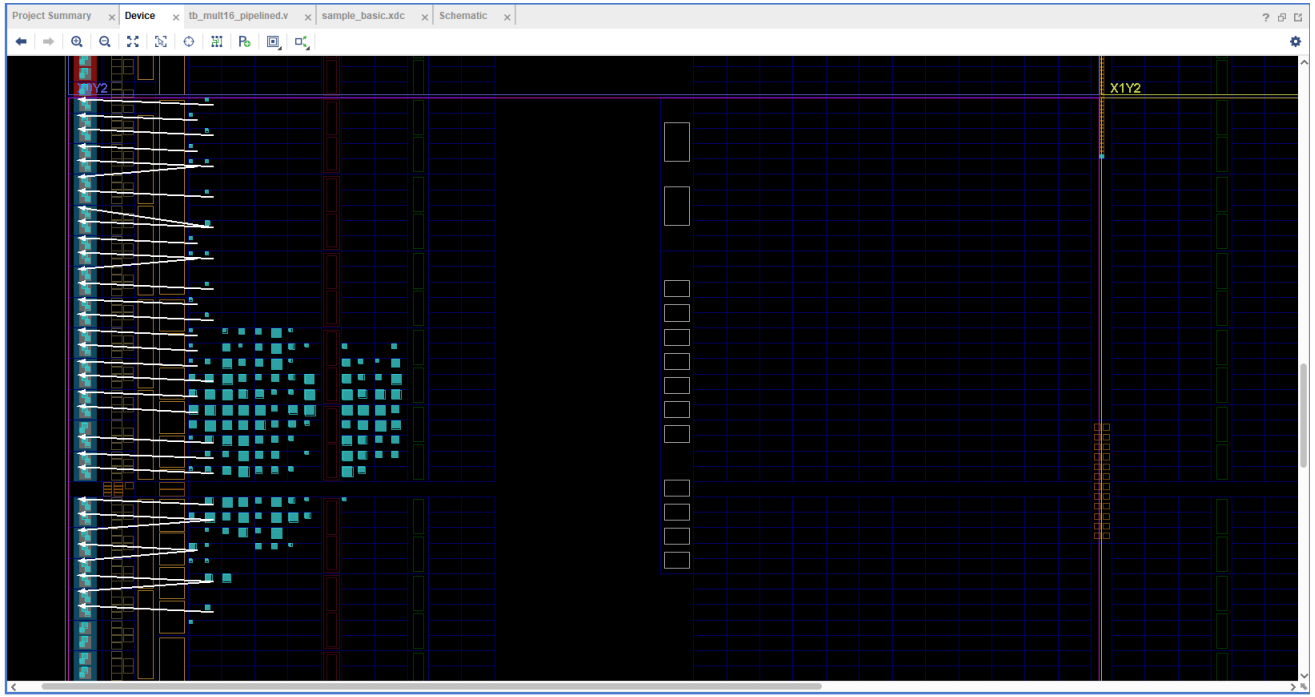
- Top 모듈과 내부 모듈
  - Top 모듈의 경우, IO Pad와 연결을 하는 모듈이다. → IO Buffer가 필수!!!
  - 내부 모듈의 경우, 모듈끼리 연결을 해서 계산하는 모듈이다. → IO Buffer가 필요 X

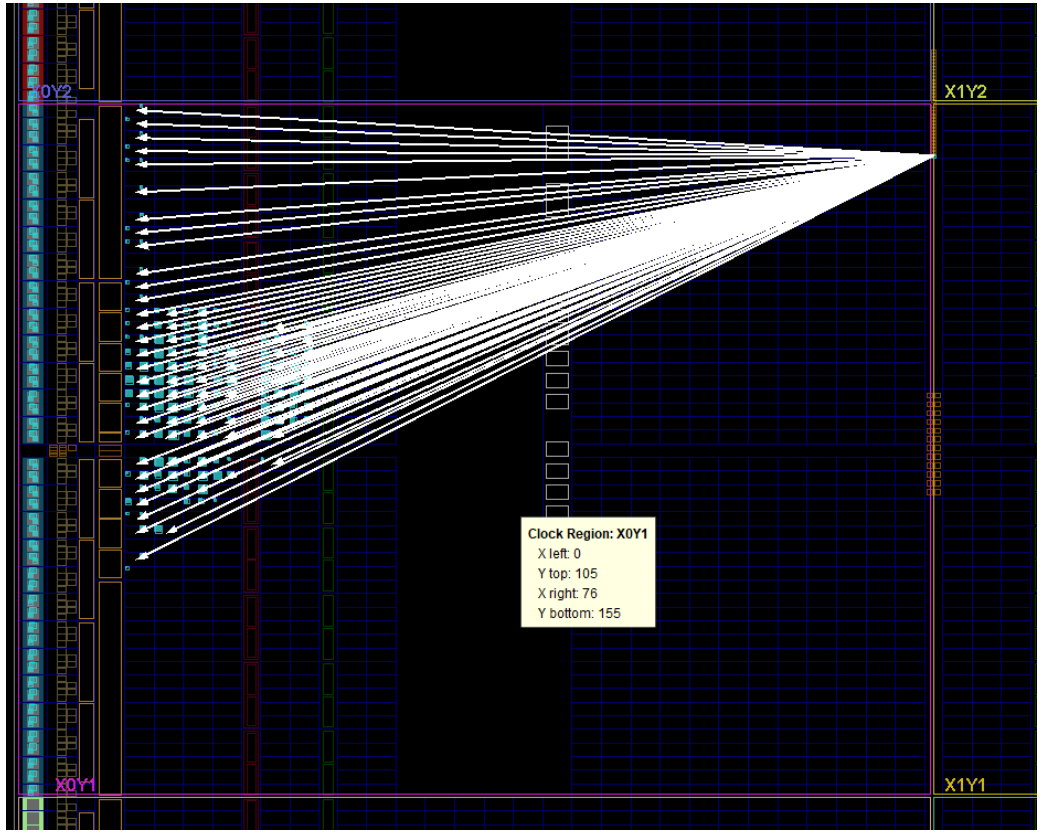
- IO Buffer를 Synthesis시 넣고 빼는 방법
  - Top 모듈로 두고서 Synthesis를 하는 경우, 무조건 IO Buffer가 생성
  - 내부 모듈을 Top으로 두고 시뮬레이션을 돌리는 경우, 각각의 Port에 다음의 문구를 삽입하여, 보다 정확한 시뮬레이션을 진행
    (* io_buffer_type = "none" *)
  - Implementation 시에는 위 문구를 빼야 함

```
module Barrel_Shifter(
        // input [3:0] A,
        // input [1:0] K,
        // input left,
        // input arithmetic,
        // input shift,
        // output reg [3:0] out
        (* io_buffer_type = "none" *) input [3:0] A,
        (* io_buffer_type = "none" *) input [1:0] K,
        (* io_buffer_type = "none" *) input left,
        (* io_buffer_type = "none" *) input arithmetic,
        (* io_buffer_type = "none" *) input shift,
        (* io_buffer_type = "none" *) output reg [3:0] out
    );
```

Synthesis 주의 사항

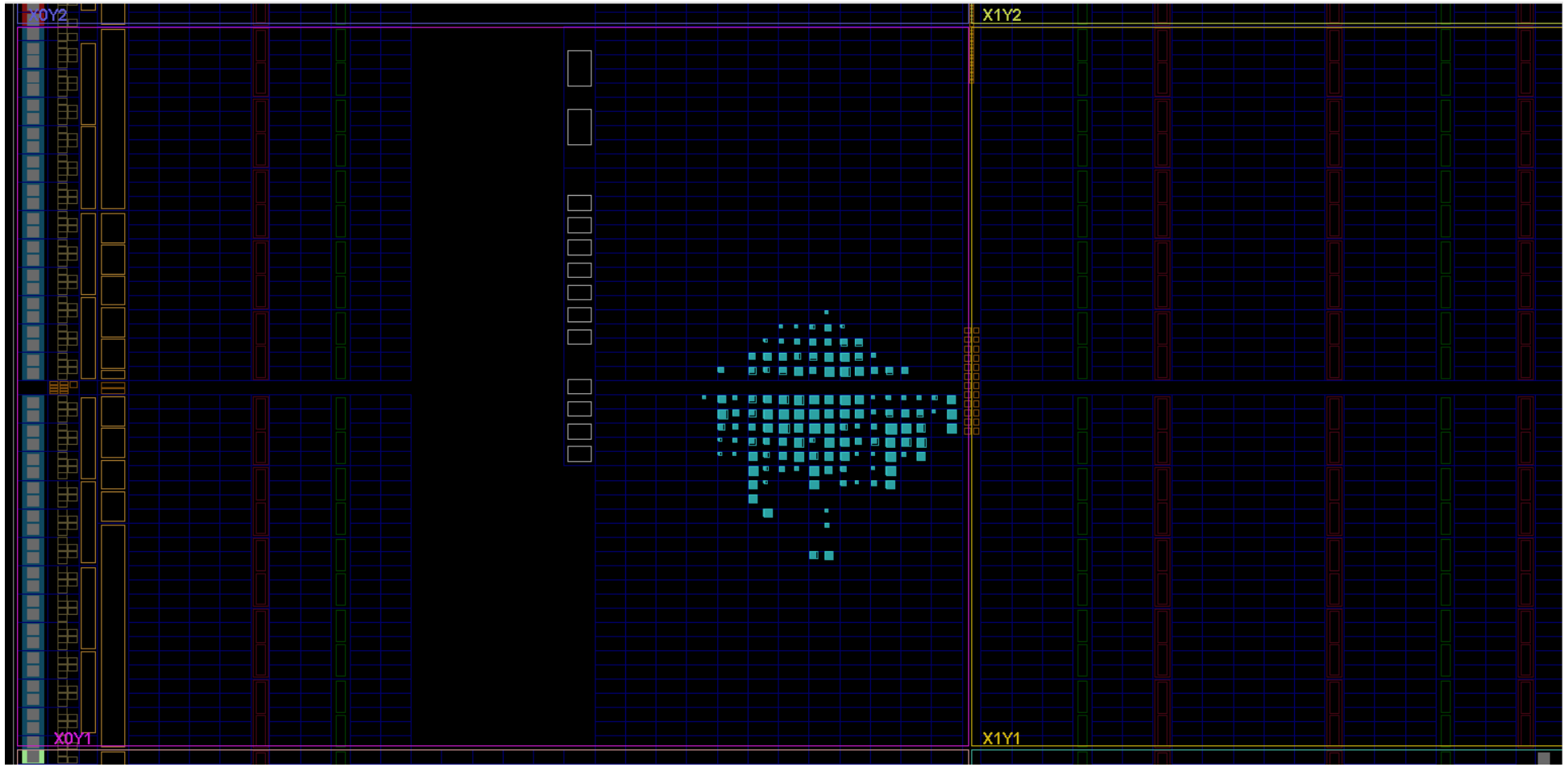
- Implementation시 top모듈과 내부 모듈이 어떻게 다른가?
  - Top Module → IO Buff와 연결되는 모습 (좌) // Clock Port & Buff와 연결되는 모습 (우)





Synthesis 주의 사항

- Implementation시 top모듈과 내부 모듈이 어떻게 다른가?
  - 내부 Module → 아무것도 없이 모여 있다.



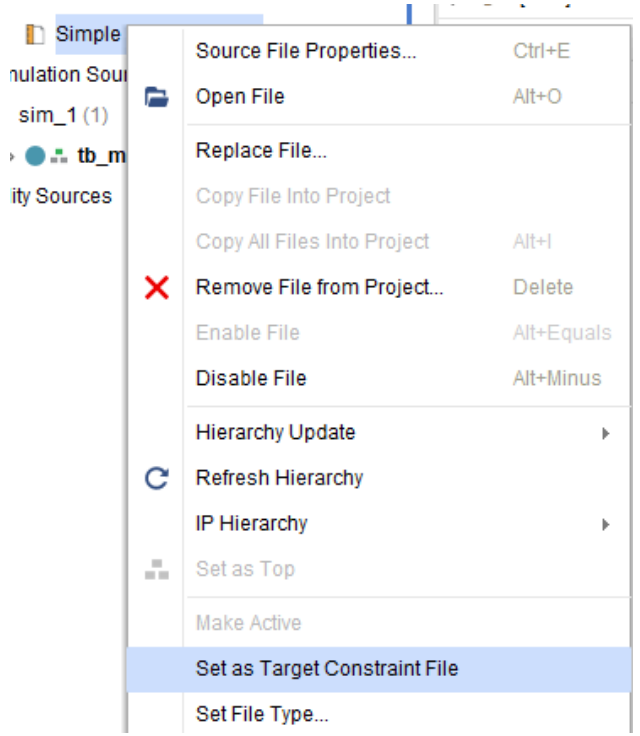
3.

Overview of Lab4

실험 수행 내용

- 4bit Barrel Shifter 설계
  - 조건에 맞는 4bit Barrel Shifter를 설계한다.
  - Behavioral Simulation을 비교해 보면서 조건에 맞는지 확인
  - 모든 시뮬레이션에는 성공했는지, 실패했는지 여부가 TCL 창에 나옴

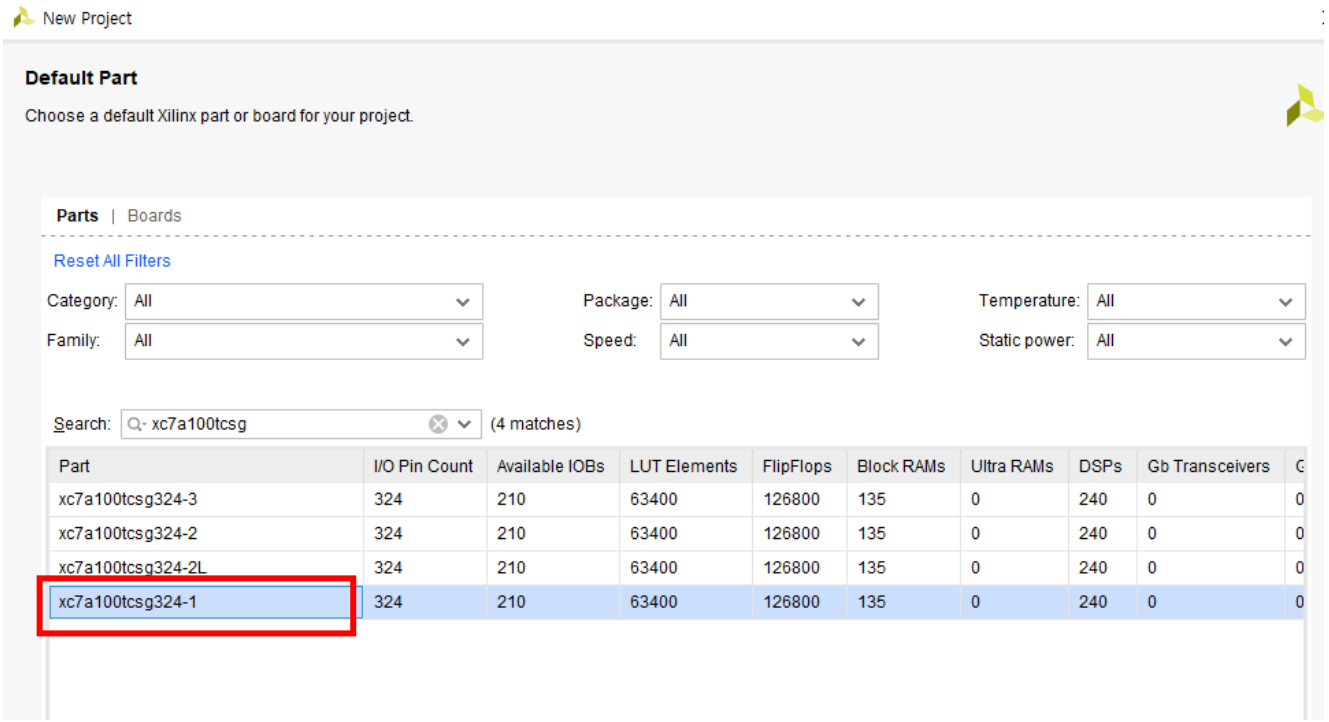
```
##########################
[Result]: Result is correct.
##########################
```



- Synthesis 수행
  - 올라온 Constraint 파일을 추가 및 Target Constraint 파일로 만든다.
  - Run Synthesis를 눌러 수행
- Post Synthesis Timing Simulation
  - 앞에서 설명한 Post Synthesis Timing Simulation을 수행
- 4bit Barrel Shifter 의 최대 동작 Frequency를 확인
  - Testbench에서 define에 있는 HALF_Period를 바꿔가면서 시뮬레이션을 진행
  - 각각의 define의 숫자는 1ns 단위의 숫자이다. (예시 : 100인 경우, 100ns를 의미)
  - Frequency 변수를 보면서 최대 몇 Hz까지 정상적으로 동작을 하는지를 확인 (제출 파일에 포함)
    (HALF_CLK_PERIOD를 소수점 1자리 까지 바꿔보면서 확인)

실험 제공 파일 & Part 설정

- **Barrel_Shifter.v**
  - Input / Output이 지정된 Module 파일
  - 동작을 모두 설계해야 함
  - // code in here // 위치에 코드 작성
  - 매우 간단하니 어렵게 생각하지 말 것
- **Barrel_Shift_TB.v**
  - Clock Period를 변경할 수 있는 TB
  - Define Clock 부분만 변경.
- Simple_Constraint.xdc
  - 간단하게 주어진 XDC
  - 수정하지 말 것!

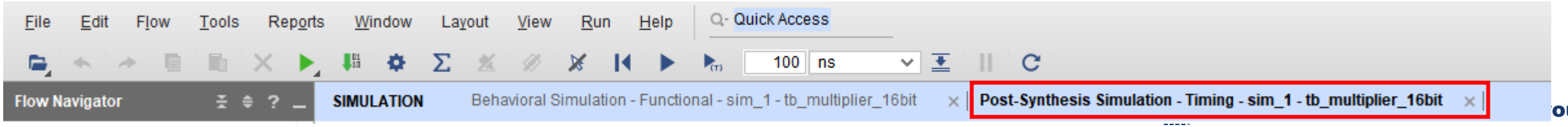


- Project 생성 시, Part는 xc7a100tcsg324-1 을 사용할 것

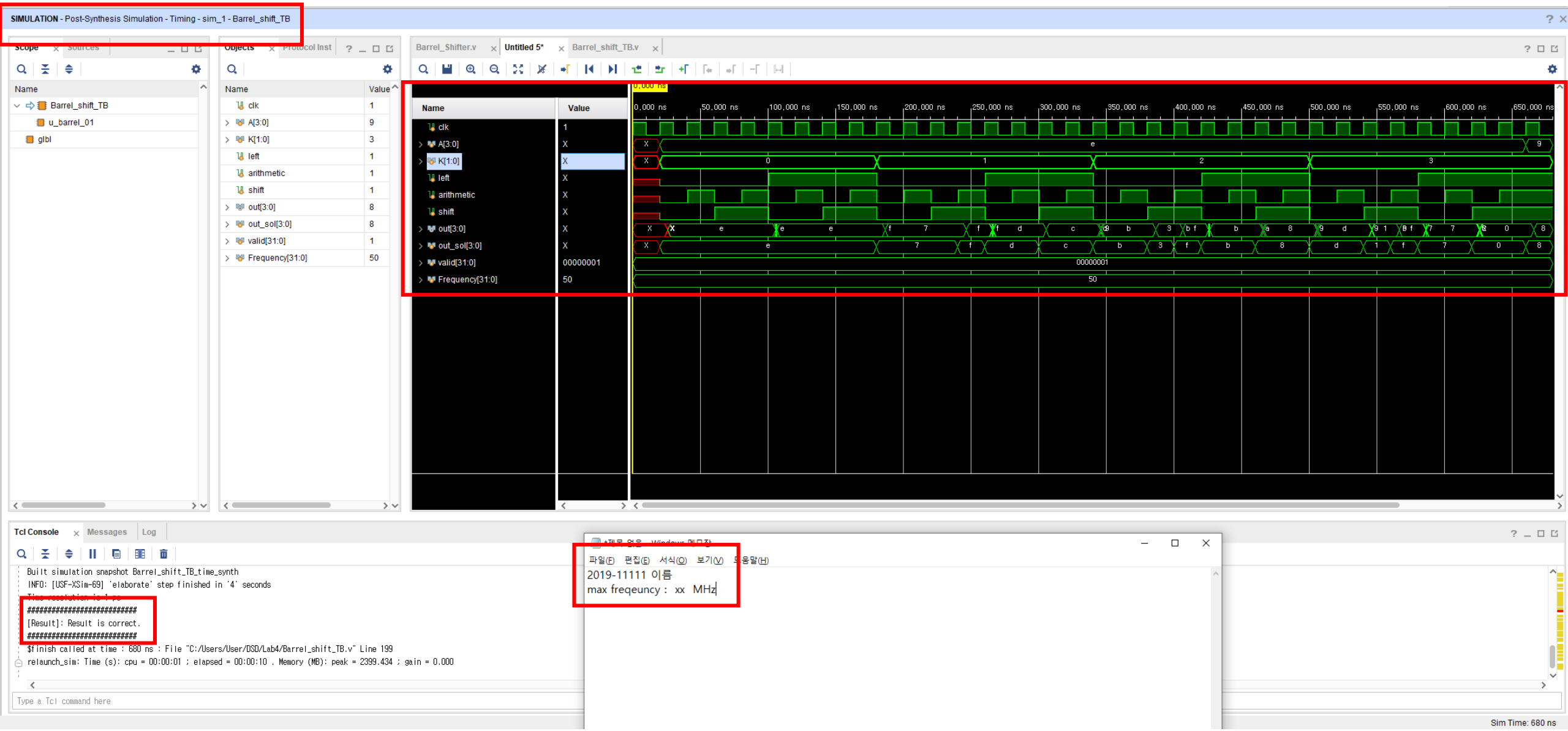
결과 제출

- 제출 파일
  - 구현한 Source file
  - 스크린샷 (한 장에 아래의 내용이 모두 들어가야 함 , **다음 슬라이드의 캡쳐 예시 참고**)
    - Post Synthesis Simulation - timing 결과 임을 알 수 있도록 (아래 빨간색 박스가 보이게)
    - “Result is correct” in Tcl Console
    - 자신의 학번 이름
    - 최대 Frequency (Post Synthesis timing simulation 기준)

- File name: “학번_이름_lab4.zip”
  - ex) 2020-12345_홍길동_lab4.zip
- Deadline: 10월 7일 (금요일) 23:59분까지



결과 캡쳐 예시


SIMULATION - Post-Synthesis Simulation - Timing - sim_1 - Barrel_shift_TB
Barrel_shift_TB
u_barrel_01
glbl
clk
A[3:0]
K[1:0]
left
arithmetic
shift
out[3:0]
out_sol[3:0]
valid[31:0]
Frequency[31:0]
00000001
50
Built simulation snapshot Barrel_shift_TB_time_synth
INFO: [USF-XSim-69] 'elaborate' step finished in '4' seconds
[Result]: Result is correct.
$finish called at time : 680 ns : File "C:/Users/User/DSD/Lab4/Barrel_shift_TB.v" Line 199
relaunch_sim: Time (s): cpu = 00:00:01 ; elapsed = 00:00:10 . Memory (MB): peak = 2399.434 ; gain = 0.000
2019-11111 이름
max freqeuncy : xx MHz
Sim Time: 680 ns