Lab 5. Pipelined Multiplier

Digital System Design and Experiment

Graduate School of Convergence Science and Technology

Seoul National University

Mobile Multimedia Systems Group

목표

- Verilog를 이용하여 multiplier의 pipeline 구조를 설계하고 구현해본다.
- Synthesis/Implementation을 수행하고 Post Timing Simulation을 돌리면서 정상적으로 합성이 되었는지 확인한다.
- Pipelined multiplier의 최대 동작 Clock Frequency를 확인한다.

1.

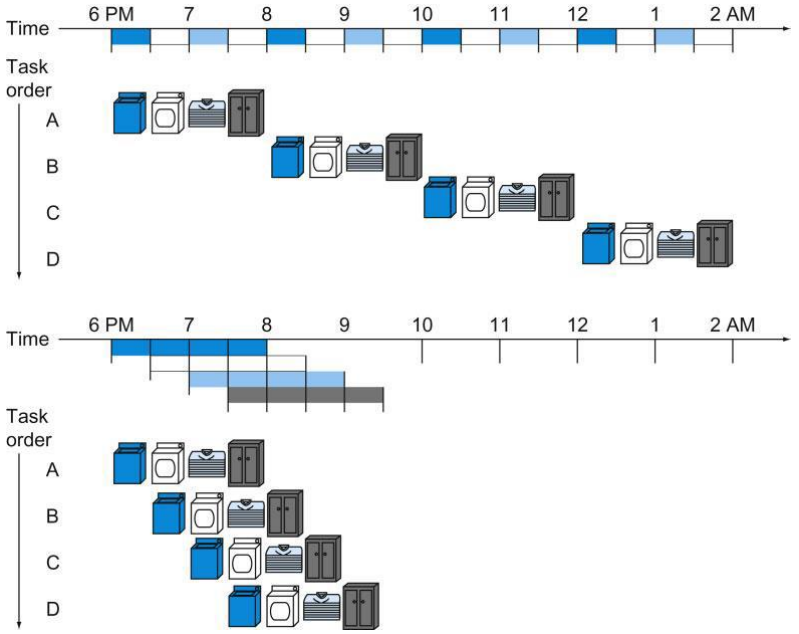
Background

Pipelining

- Improve performance
  - **overlapping execution**
  - **Parallelism (병렬화)**

- When do pipelining?
  - 매우 Heavy한 연산에서 성능 향상을 이루고 싶을 때
  - Operation이 1-Cycle 안에 끝나지 않을 때

- Ex) Sequential laundry vs Pipelined laundry
  - Sequential : 8 hours / 4 loads
    Pipelined : 3.5 hours / 4 loads
  - Speed up ➔ 8h / 3.5h = **2.3 times!**
  - More pipelining stage? ➔ 4*0.5n / (0.5n + 1.5) ≈ **4 times!**



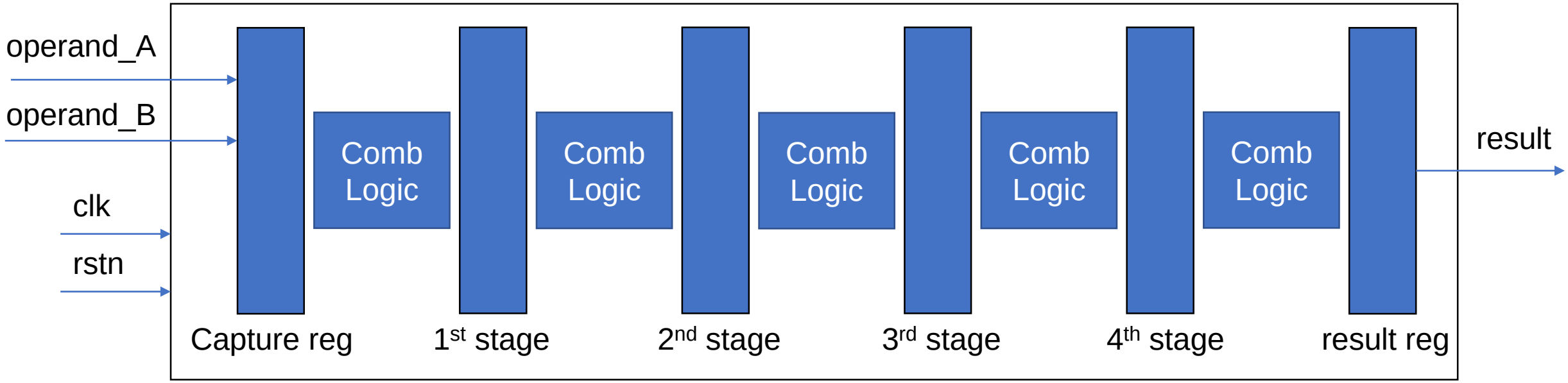
2.

Pipelined Multiplier

Overview of Multiplier

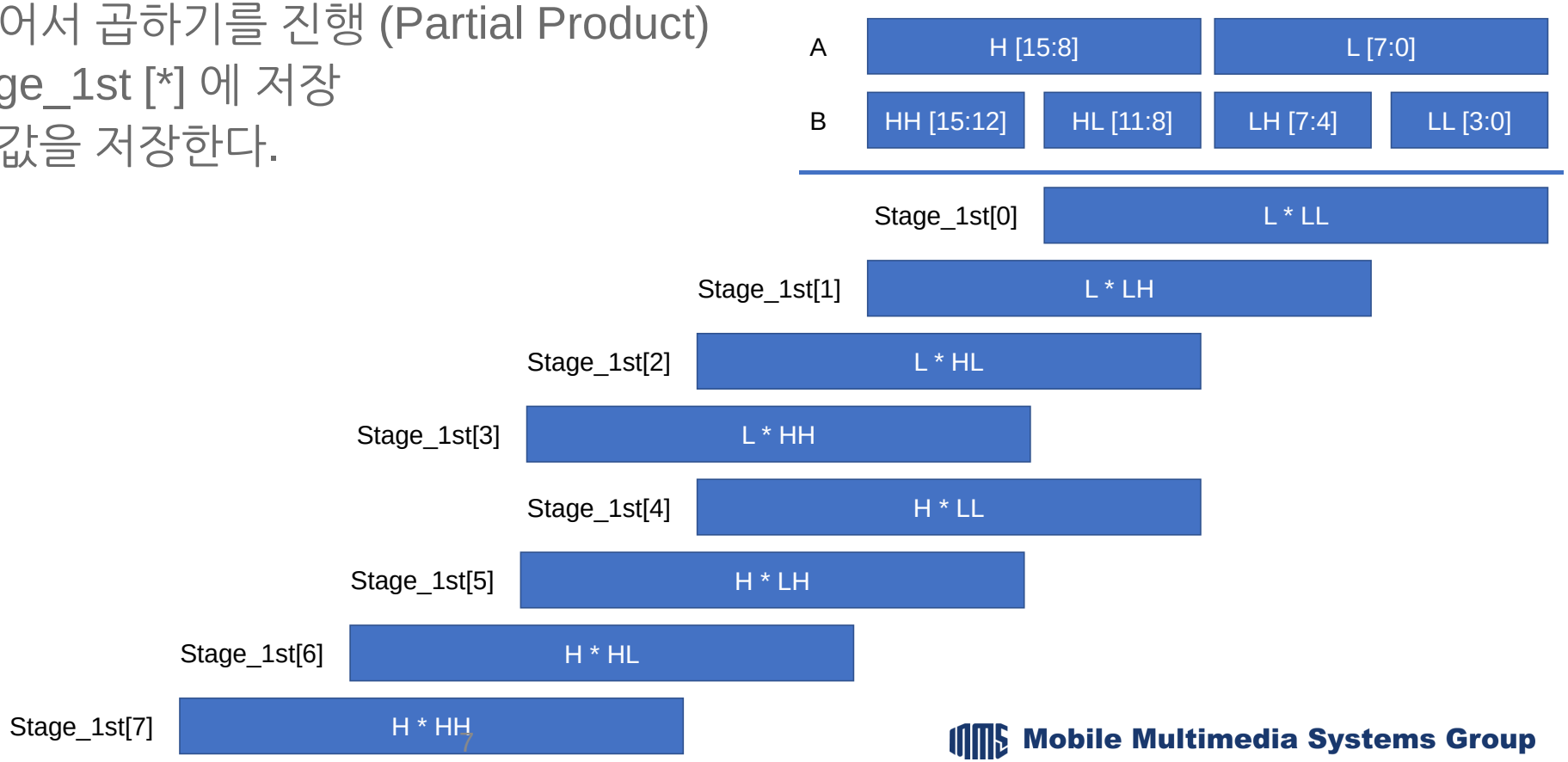
- 16-bit unsigned 두 개(A, B)를 받아서 곱셈 결과를 출력
  - Pipeline을 적용한 형태의 Multiplier를 제작
  - Input이 들어가서 Capture 되는 Clock부터 **6-Cycle**이 걸려서 result를 출력

- Pipeline을 직접 Coding하는 것은 여러가지 방법이 존재
  - 간단하지만 성능은 보통인 방법을 실습해볼 예정



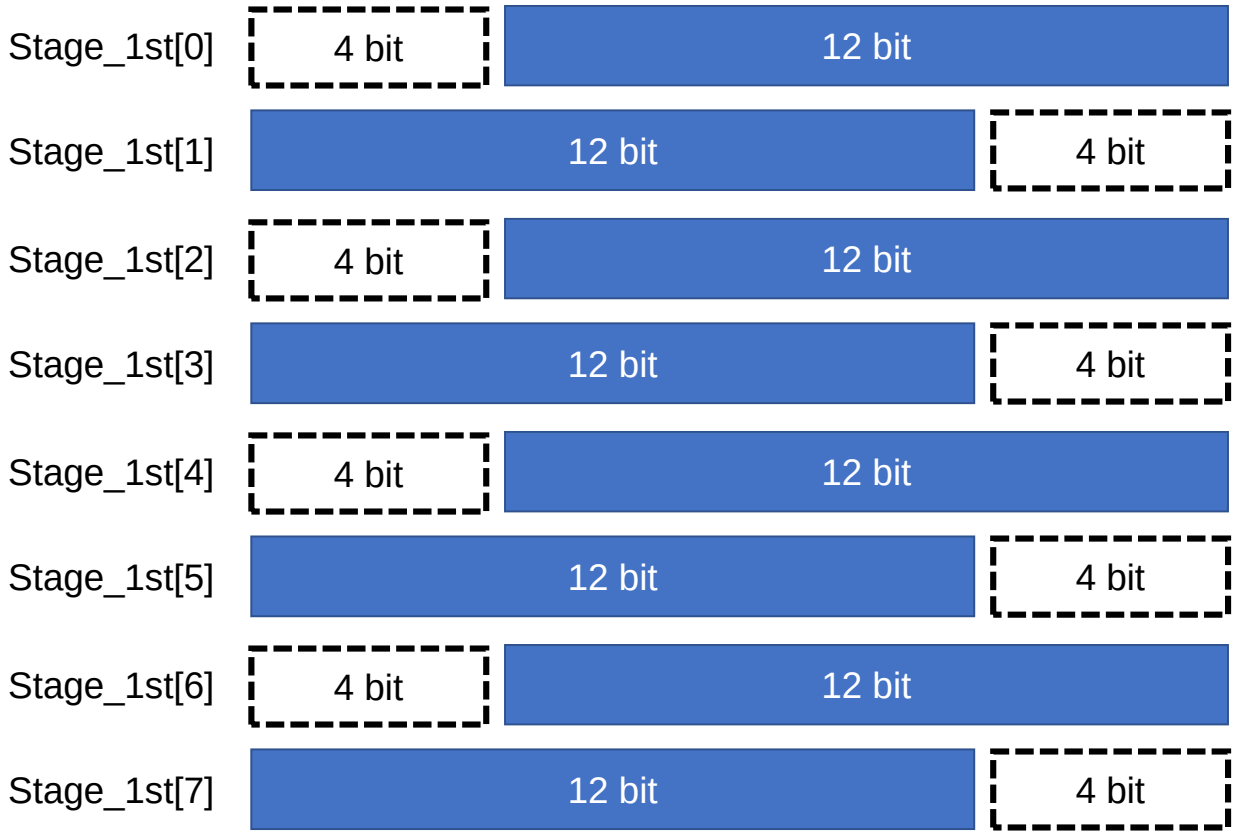
Pipelining Multiplier

- Capture reg
  - Input이 안정적으로 되기 위해 Capture를 수행하는 stage
  - Input을 Register에 값을 바로 Update 한다.
- 1st stage
  - Capture된 input을 나누어서 곱하기를 진행 (Partial Product)
  - 각각의 결과를 12bit stage_1st [*] 에 저장
  - 다음 Stage에서 각각의 값을 저장한다.



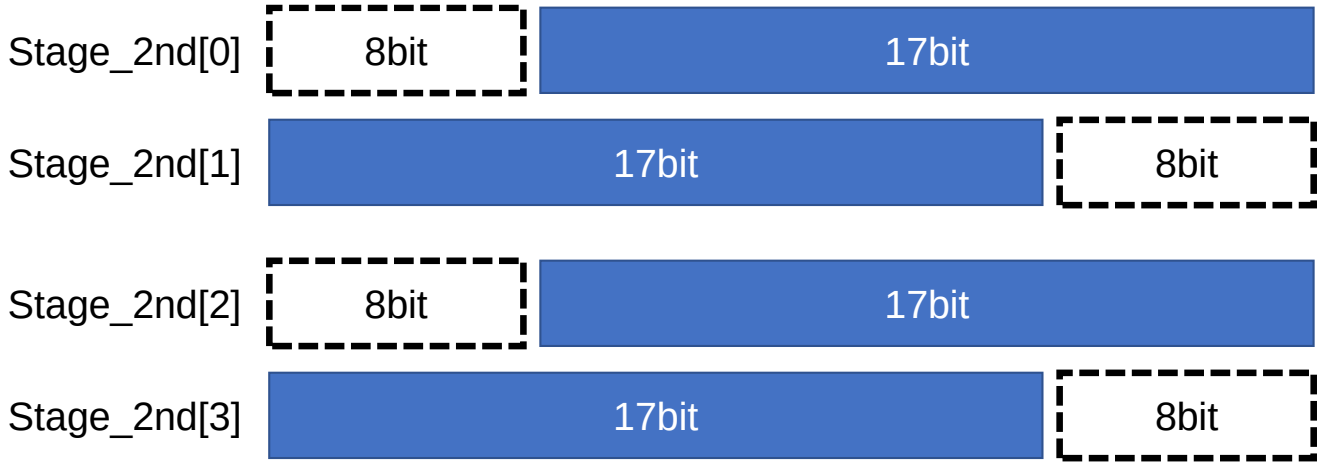
Pipelining Multiplier

- 2nd stage (17bit)
  - Stage_1st에서 다음을 더한다.
    - 0번과 1번
    - 2번과 3번
    - 4번과 5번
    - 6번과 7번
  - 더해줄 때에는 각각의 자릿수를 맞춰준다.

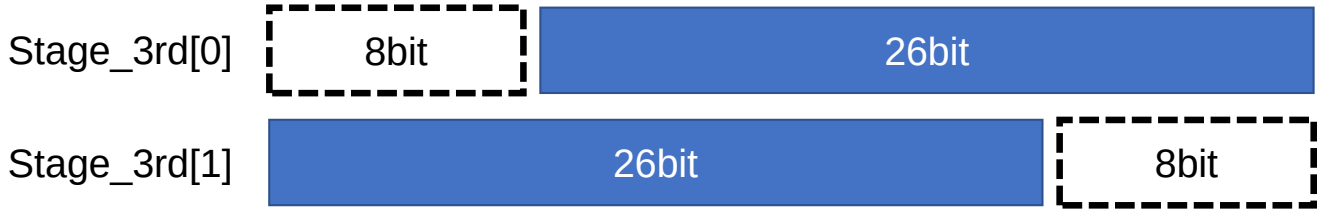


Pipelining Multiplier

- 3rd stage (26bit)
  - Stage_2nd에서 다음을 더한다.
    - 0번과 1번
    - 2번과 3번
  - 더해줄 때에는 각각의 자릿수를 맞춰준다.



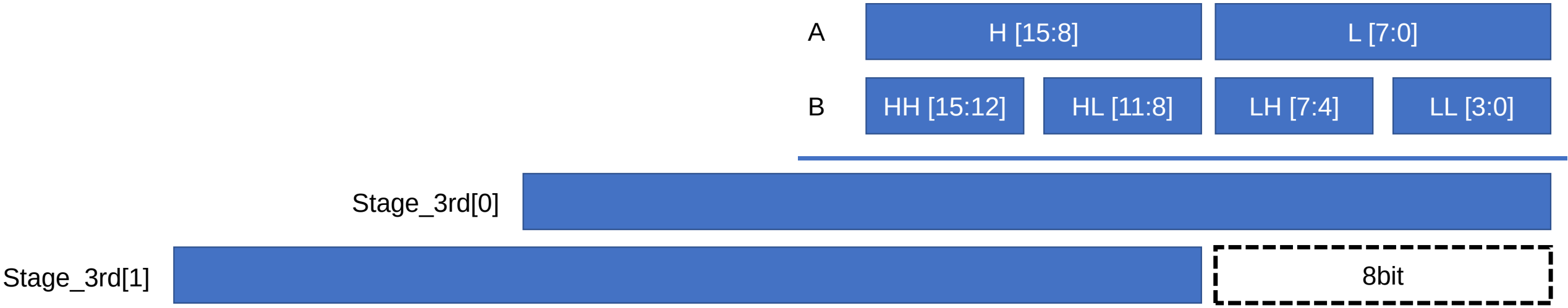
- 4th stage (35bit)
  - Stage_3rd 에서 다음을 더한다.
    - 0번과 1번
  - 더해줄 때에는 각각의 자릿수를 맞춰준다.



- Result stage
  - Result에 4^{th}-stage의 35bit 중 32bit만을 할당한다. (unsigned 연산이므로 고려 사항 x)
  - 앞의 3bit는 overflow bit로 활용할 수 있으나, 이번 실험에서는 제외.

Pipelining Multiplier

- Q1. 왜 4th-stage에서는 8-bit 만큼만 자리를 맞춰주었을까?
- A1. 아래 그림과 같이 나타나기 때문!

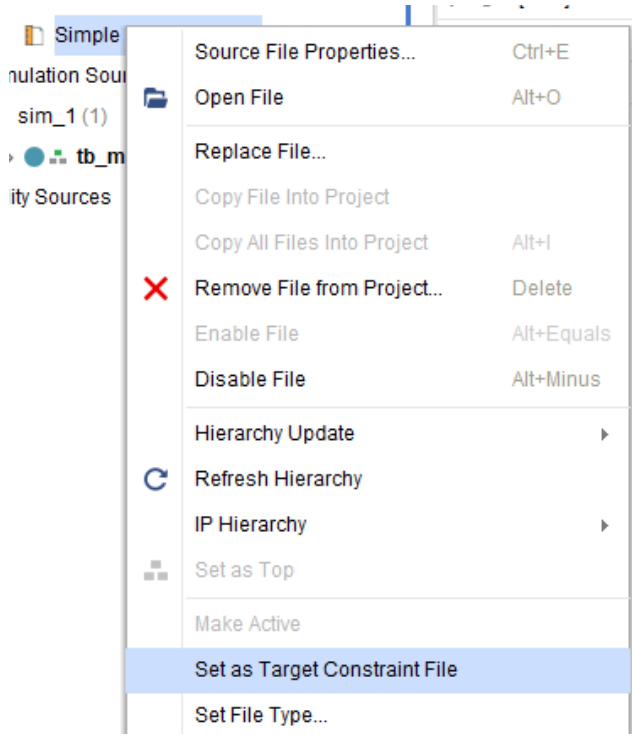


3.

Overview of Lab 5

실험 수행 내용

- 16bit-Multiplier 설계
  - 조건에 맞는 16bit Multiplier를 설계한다.
  - Behavioral Simulation을 비교해 보면서 조건에 맞는지 확인
  - 모든 시뮬레이션에는 성공했는지, 실패했는지 여부가 나옴
- 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
  - 결과가 얼마나 차이가 나는지 확인
- 16bit-Multiplier standard Cell의 최대 동작 Frequency를 확인
  - Testbench에서 define에 있는 Period를 바꿔가면서 시뮬레이션을 진행
  - 각각의 define의 숫자는 1ns 단위의 숫자이다. (예시 : 100인 경우, 100ns를 의미)
  - 최대 몇 Hz까지 정상적으로 동작을 하는지를 확인

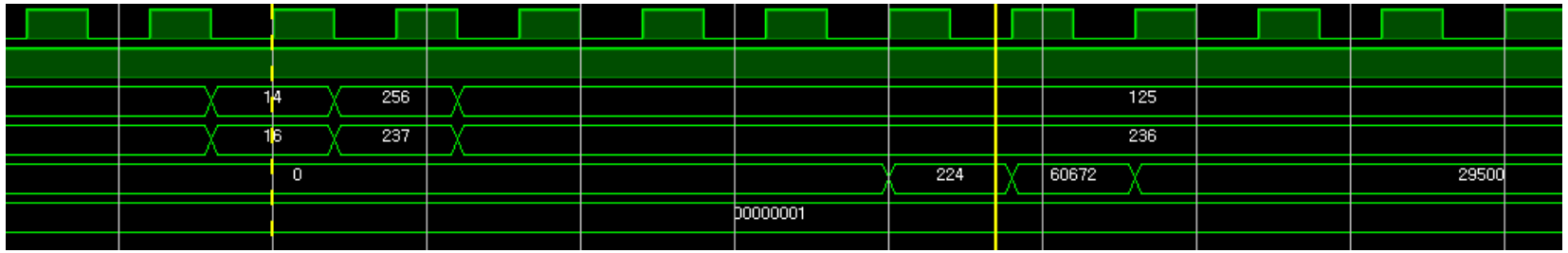


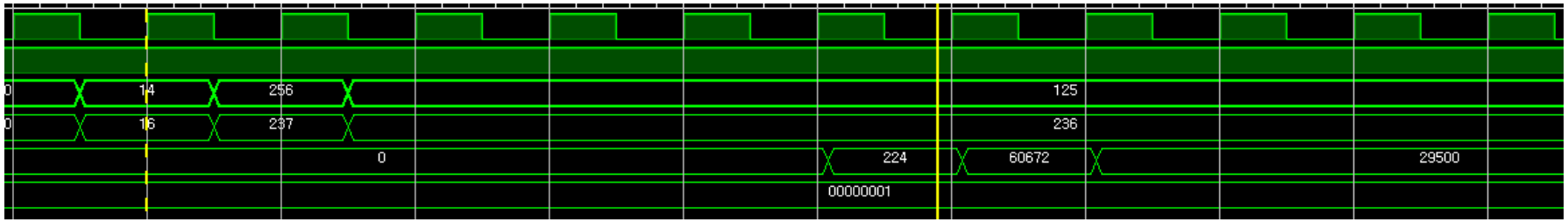
실험 제공 파일

- **mult16_pipelined.v**
  - Input / Output이 지정된 module 파일
  - 동작을 모두 설계해야 함
- tb_mult16_pipelined.v
  - Clock Period를 변경할 수 있는 TB
  - Define 부분만 변경
- sample_basic.xdc
  - 간단하게 주어진 xdc
  - 수정하지 말 것!

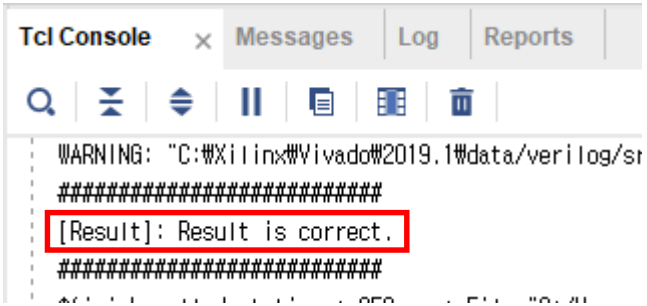
Simulation Result Check

- 결과: 설계한 multiplier에 대해, tb_multiplier를 사용하여 simulation
  - Behavior simulation (위), Timing Simulation (아래) – 아래는 실제 구현시 다를 수 있습니다.

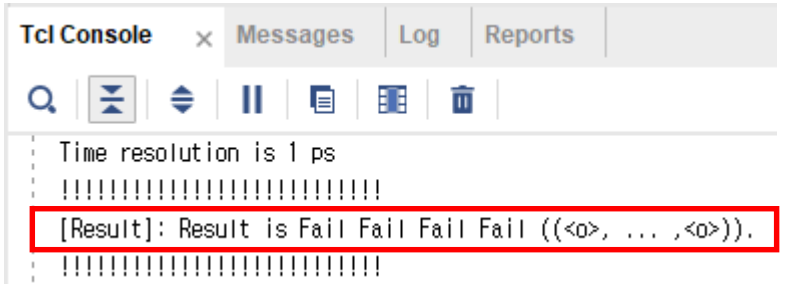




- Tcl Console 을 통해 올바른 결과인지 확인 가능.



TEST PASS



TEST FAIL

결과 제출

- 제출 파일 (압축하여 eTL 업로드)
  - 구현한 Source file
  - 스크린샷 (아래의 내용이 모두 들어가도록)
    - 정상적으로 도는 Post Synthesis Simulation 결과 (Waveform과 Tcl Console)
    - 자신의 학번 이름
    - 최대 Frequency
    - 아래 사진의 빨간 박스 부분 (Post synthesis 결과임을 보이기 위함)

- File name: “학번_이름_lab5.zip”
  - ex) 2019-12345_홍길동_lab5.zip

- Deadline: 10월 14일 (Friday) 23:59분까지