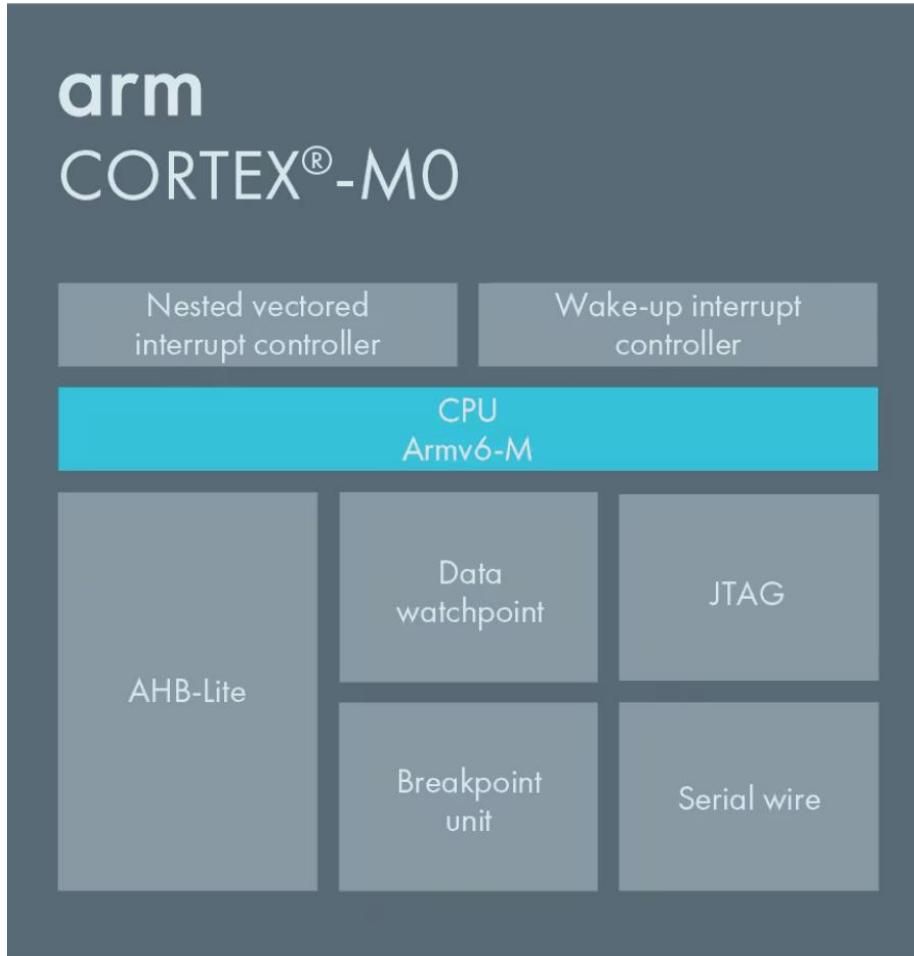




# Cortex-M0 기반의 SoC 설계 프로젝트

# System On Chip

## CORTEX-M0



## CORTEX-M0

AHB-Lite 버스는 단일 마스터 Pipe-Line 구조로 명령을 수행

Data Watch Point는 JTAG 통신의 디버깅을 위한 유닛

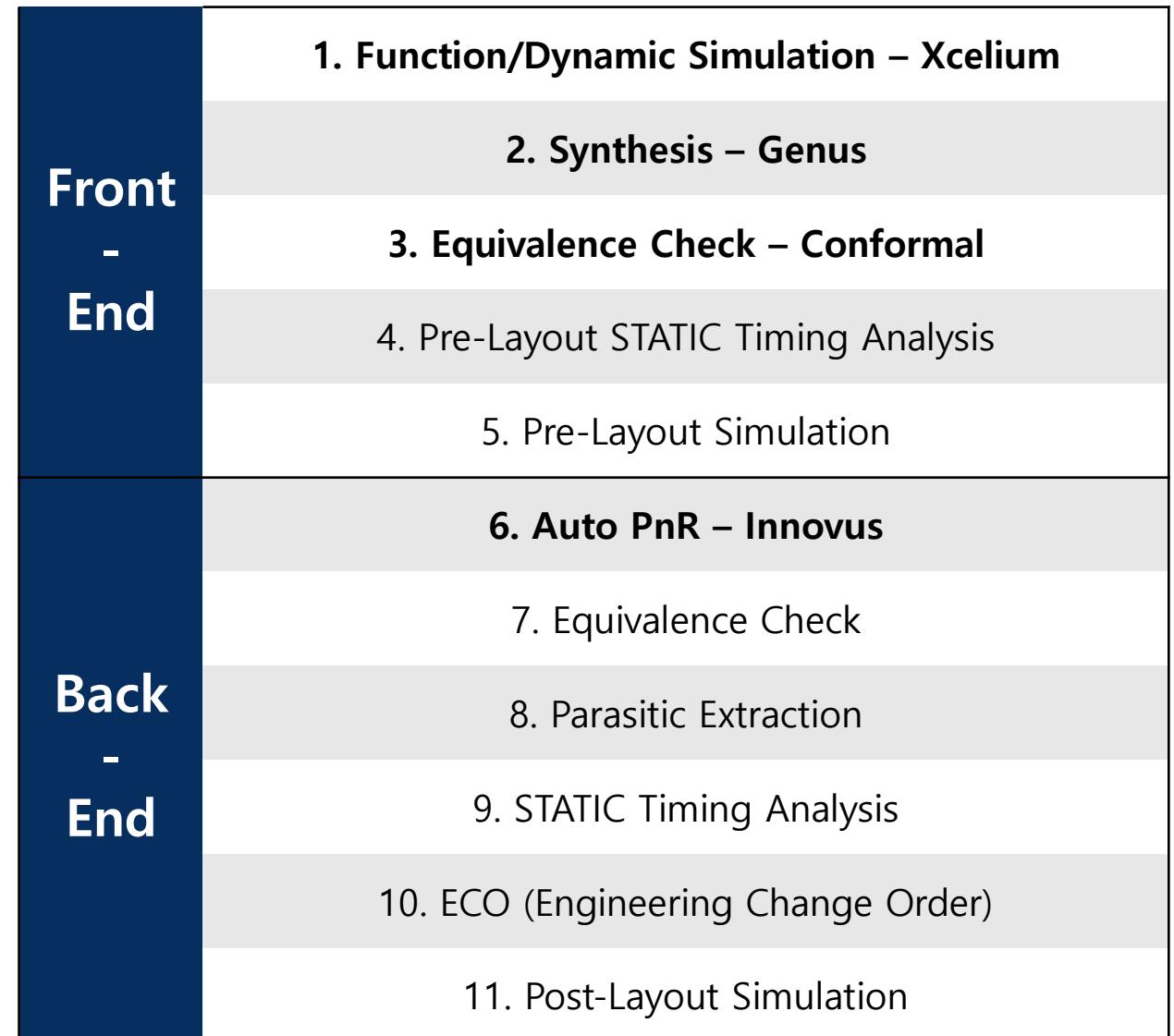
Breakpoint Unit은 플래시 메모리의 디버깅을 위한 유닛

NVIC/WIC 는 ISR을 컨트롤하기 위한 유닛

# System On Chip

## CORTEX-M0

- System On Chip 프로젝트 참여
- 해당 프로젝트는 SoC의 일부인 Cortex M0를 Front-End 단계에서 Back-End 단계까지 설계하는 내용의 교육용 자료를 만드는 프로젝트 이다.
- 프로젝트 기간: 2024.10 - 2025.03



# 디지털 라이브러리 테스트

# 실습 준비

# 디지털 라이브러리 테스트

## 실습준비

1. \$> cd SoC1 ↵

2. \$> mkdir GPDK045 ↵

```
[ex_poly1@npit ~]$ cd SoC1  
[ex_poly1@npit ~/SoC1]$ mkdir GPDK045
```

디렉토리  
생성

```
[ex_poly1@npit ~/SoC2]$ ls  
GPDK045
```

# 디지털 라이브러리 테스트

## 실습준비

1. \$> cp -a ~/digital.tar.gz . ↵

복사

```
[ex_poly1@npit GPDK045]$ cp -a ~/digital.tar.gz .
```

2. \$> tar xvzf digital.tar.gz ↵

압축해제

```
[ex_poly1@npit GPDK045]$ tar xvzf digital.tar.gz
```

```
[ex_poly1@npit GPDK045]$ ls  
digital digital.tar.gz
```

# 디지털 라이브러리 테스트

## 실습준비

1. \$> cd ↵

2. \$> source digital.cshrc ↵

```
[ex_poly1@npit GPDK045]$ cd  
[ex_poly1@npit ~]$ source digital.cshrc
```

\* \$> source ~/digital.cshrc

# 디지털 라이브러리 테스트

## 실습준비

1. cd SoC1 ↵
2. cd GPDK045 ↵
3. cd digital ↵
4. cd TEST ↵
5. cd std\_cell ↵

```
[ex_poly1@npit GPDK045]$ ls  
digital
```

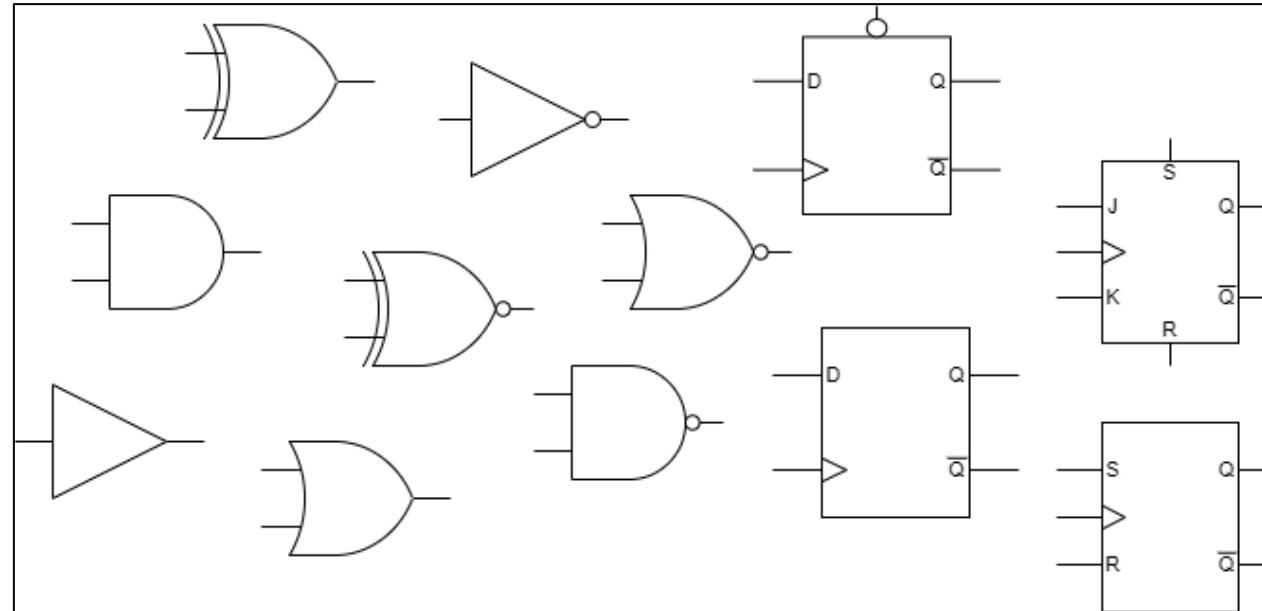
```
[ex_poly1@npit digital]$ cd TEST/  
[ex_poly1@npit TEST]$ ls  
io std_cell
```

# STD셀 라이브러리 시뮬레이션

# 디지털 라이브러리 테스트

## STD셀 이란?

1. 우리가 흔히 알고 있는 AND, OR, XOR, 플립플롭 등 기본 논리 게이트가 들어있는 셀



# LAB1

# 디지털 라이브러리 테스트

## LAB1의 목적

1. standard sell에서 몇가지 cell을 가져와 동작을 시뮬레이션
2. 게이트의 동작을 시뮬레이션 -gui옵션으로 확인
3. Dump 폴더에 파형이 자동 입력
4. 시뮬레이션 start하면 정해진 시간에 stop 확인

# 디지털 라이브러리 테스트

## LAB1 파일 확인

1. \$>cd LAB1 ↵

2. \$> ll ↵

```
[ex_poly1@npit LAB1]$ ll
total 108
-rwxr-xr-x 1 ex_poly1 rnd    34 Oct 14 00:19 clean.tcl
drwxr-xr-x 3 ex_poly1 rnd    33 Oct 14 00:44 dump
-rw-r--r-- 1 ex_poly1 rnd   319 Oct 27 15:48 func_sim.history
-rw-r--r-- 1 ex_poly1 rnd 42418 Oct 27 15:49 func_sim.log
drwxr-xr-x 2 ex_poly1 rnd    37 Oct 14 00:45 log
-rw-r--r-- 1 ex_poly1 rnd   115 Oct 15 23:40 README
-rwxr-xr-x 1 ex_poly1 rnd   662 Oct 15 23:44 run_function.tcl
-rw-r--r-- 1 ex_poly1 rnd  7026 Oct 14 21:53 shm.prof
-rw-r--r-- 1 ex_poly1 rnd   859 Oct 15 23:44 TB_std_cell.v
drwxr-xr-x 6 ex_poly1 rnd   214 Oct 27 15:48 xcelium.d
-rw-r--r-- 1 ex_poly1 rnd 23170 Oct 27 15:49 xmprof.out
-rw-r--r-- 1 ex_poly1 rnd   577 Oct 23 18:45 xrun.history
-rw-r--r-- 1 ex_poly1 rnd      5 Oct 27 15:49 xrun.key
-rw-r--r-- 1 ex_poly1 rnd  331 Oct 23 18:45 xrun.log
```

# 디지털 라이브러리 테스트

## README 파일 확인

### 1. \$> vi README ↵

- 시뮬레이션이 이런 식으로 진행되는 것을 알 수 있음

```
with option -gui  
simvision auto started  
dump folder is auto opened  
press start button and stop at $stop function
```

# 디지털 라이브러리 테스트

## clean.tcl 파일 확인

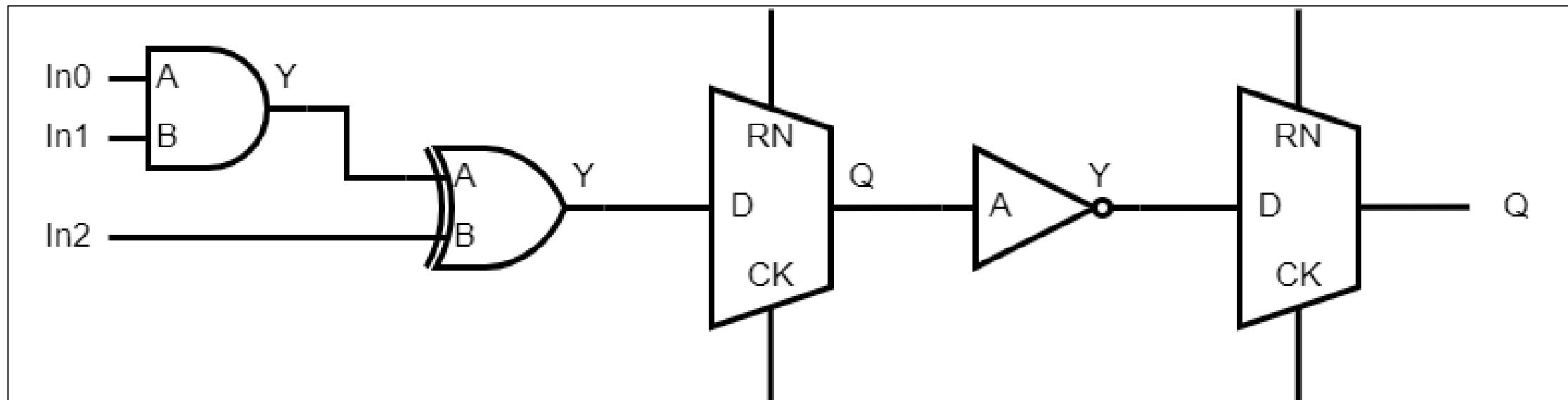
1. \$> vi clean.tcl ↵

- clean.tcl 을 실행할 시 일어나는 일들을 정리해둔 파일
- clean.tcl 실행 시 rm -rf이 자동으로 실행 됨

```
rm -rf work xcelium.d/ waves.shm
```

# 디지털 라이브러리 테스트

## TB\_std\_cell.v 구조



<그림1>

# 디지털 라이브러리 테스트

## run\_function.tcl 파일 확인

1. \$> vi run\_function.tcl ↵

- xrun 명령에 대한 여러 옵션 확인
- -gui 옵션을 통해 simvision이 자동으로 실행됨

```
#debug option is essential to view all design hierarchy
#vcs -full64 -j16 -fgp -debug_access+all -kdb -lca
#
#vcs -full64 -debug_access+all -kdb -lca \
#-debug \
#+define+function_sim \
#+v2k \
#+libext+.v+.vlib \
#+incdir+../../SAED32_EDK/lib/stdcell_rvt/verilog\
#./TB_std_cell.v \
#../../SAED32_EDK/lib/stdcell_rvt/verilog/saed32nm.v\
#-l func_sim.log

#      -libext+.v+.s
xrun -64bit \
+max_err_count:50 \
+defn \
-a \
-pri \
-prf \
-prfth \
-gui \
TB_std_cell.v \
../../../../gsclib045_all_v4.4/gsclib045_svt_v4.4/gsclib045/verilog/slow_v \
dd1v0_basicCells.v \
-l ./func_sim.log
```

g+f(go to file)

# 디지털 라이브러리 테스트

## TB\_std\_cell.v 구조

1. \$> vi TB\_std\_cell.v

:set nu

- 13행 부터 48행에서 TB의 구조 확인

```
13 AND2X1 UAND2X1 (
14     .A(IN[0]),
15     .B(IN[1]),
16     .Y(AND_RESULT)
17 );
18
19
20 XOR2X1 UXOR2X1 (
21     .A(AND_RESULT),
22     .B(IN[2]),
23     .Y(XOR_RESULT)
24 );
25
26
27
28 DFFRHQX1 UDFFRHQX1 (
29     .D(XOR_RESULT),
30     .CK(CK),
31     .RN(RN),
32     .Q(Q1)
33 );
34
35
36
37 INVX1 UINVX1 (
38     .A(Q1),
39     .Y(Q1_BAR)
40 );
41
42
43 DFFRHQX1 UDFFRHQX2 (
44     .D(Q1_BAR),
45     .CK(CK),
46     .RN(RN),
47     .Q(Q2)
48 );
```

# 디지털 라이브러리 테스트

검색하고 싶은 단어 Shift+3

## TB\_std\_cell.v 구조

- **Initial begin** : reg 타입으로 선언한 모든 것들에 대한 초기값 설정
- **always begin** : 반복적으로 일어나는 것에 대한 기술

```
50 initial begin
51     CK      <= 1'b0;
52     IN      <= 3'b000;
53     RN      <= 1'b1;
54
55 #199    RN      <= 1'b0;
56 #100    RN      <= 1'b1;
57
58 #500
59 $stop;
60 end
61
62 always begin
63 #2      CK      <= ~CK;
64 end
65
66 always begin
67 #2      IN      <= IN+1'b1;
68 end
```

# 디지털 라이브러리 테스트

## TB\_std\_cell.v 구조

- 70행 initial begin의 경우 특별히 ifdef기능을 이용하여 외부파형을 저장할 수 있게 하는 구문 사용함

```
70 initial begin
71     `ifdef function_sim
72         // $sdf_annotate (".../.../xx.sdf", TB_std_cell.core,,, "MAXIMUM");
73         $shm_open("./dump/TB_std_cell");
74         $shm_probe(TB_std_cell, "AC");
75         TEST <= $fopen("./log/tb_std_cell.rpt");
76     `endif
77 end
```

# LAB1

# 시뮬레이션

# 디지털 라이브러리 테스트

## LAB1 시뮬레이션

1. ./clean.tcl ↵

2. ./run\_function.tcl ↵

- xrun 실행

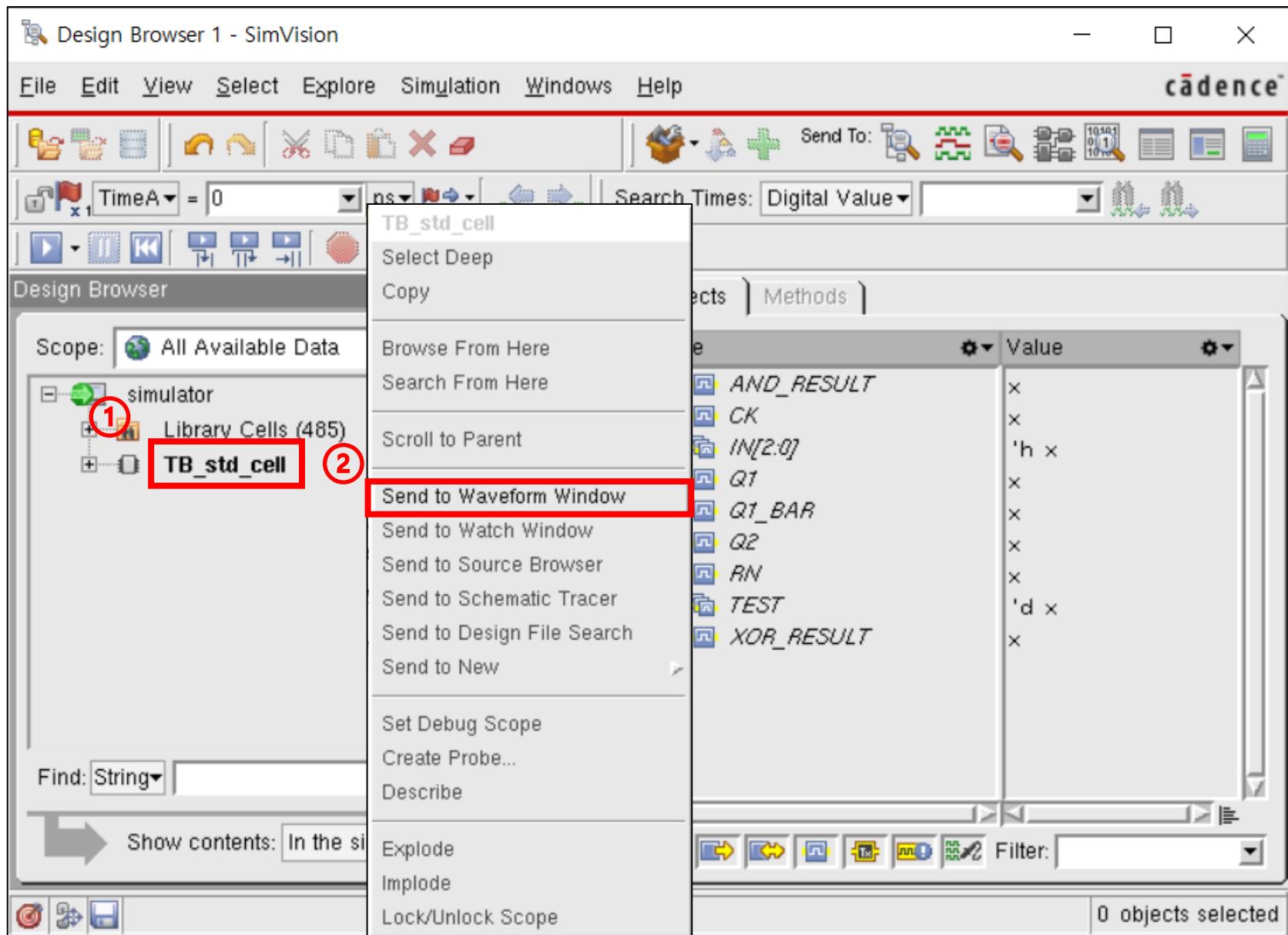
```
[ex_poly1@npit LAB1]$ ./clean.tcl
[ex_poly1@npit LAB1]$ ./run_function.tcl
```

# 디지털 라이브러리 테스트

## LAB1 시뮬레이션

### 1. xrun 실행 결과

- ① 마우스 우 클릭
- ② Send to... 클릭



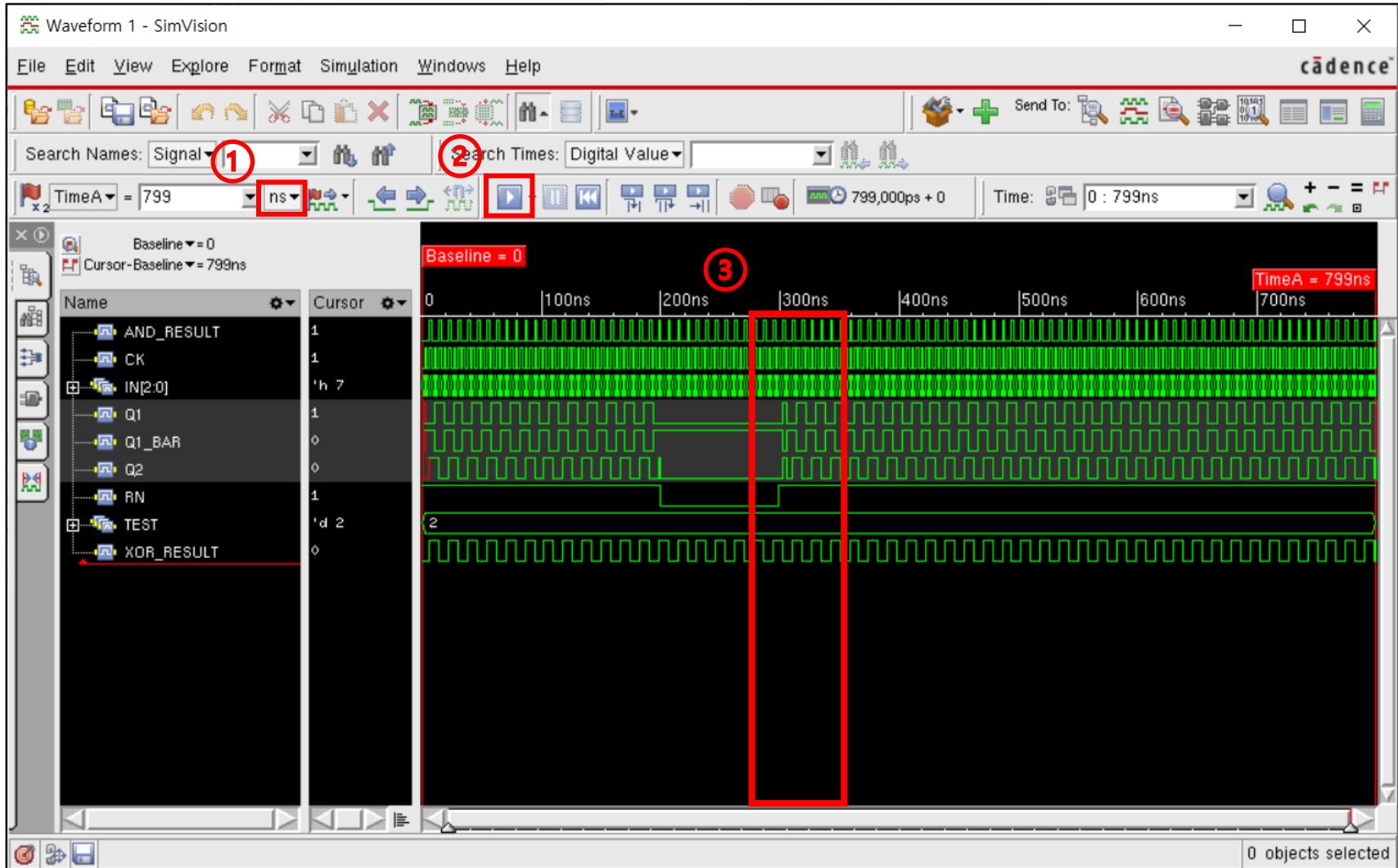
# 디지털 라이브러리 테스트

## LAB1 시뮬레이션

### 1. xrun 실행 결과

- ① ps → ns
- ② 실행 버튼 클릭
- ③ 드래그시 확대 가능

- Ctrl + 휼, 드래그로 범위 설정

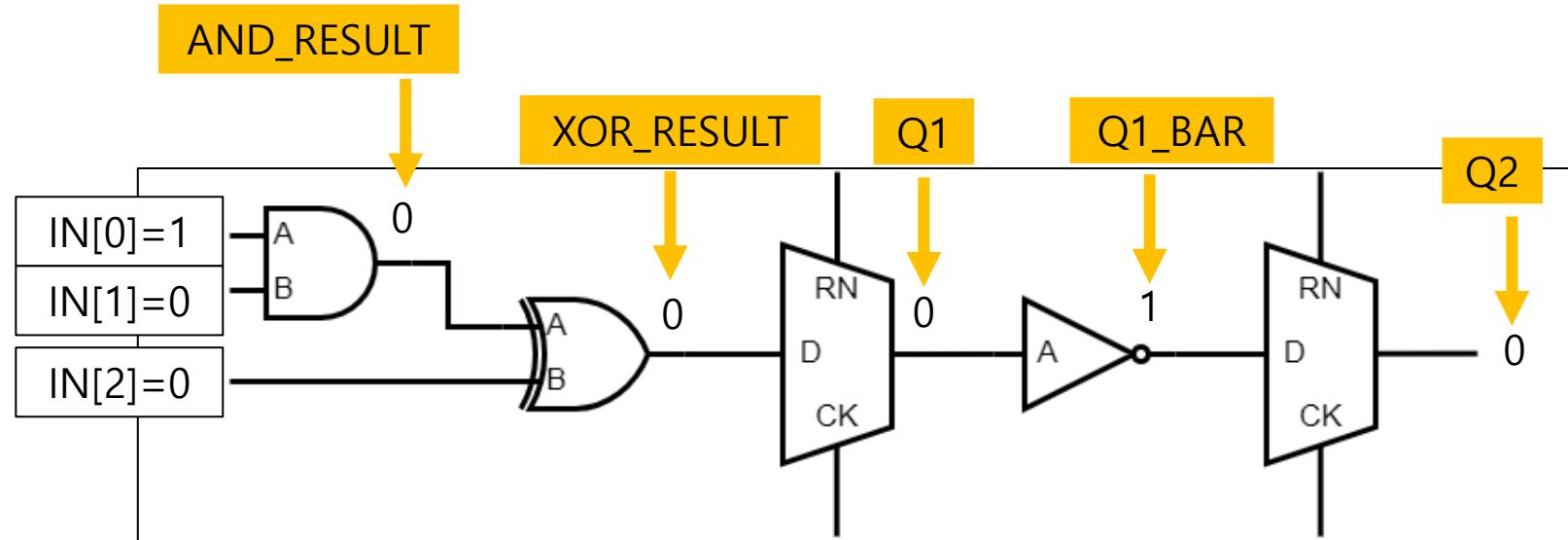


# 디지털 라이브러리 테스트

## LAB1 시뮬레이션

### 1. 파형 결과 확인

- 리셋 직후 결과 확인



# LAB2

# 디지털 라이브러리 테스트

## LAB2의 목적

1. -gui 옵션 없으므로 simvision옵션을 사용하여 시뮬레이션 확인
2. 시뮬레이션 재시작 할 필요 없이 Dump 폴더 database로 파형 확인
3. TB\_std\_se1l.v내에 \$finish 명령 확인

# 디지털 라이브러리 테스트

## LAB2 파일 확인

1. \$>cd LAB2 ↵

2. \$> ll ↵

```
[ex_poly1@npit LAB2]$ ll
total 96
-rw-r--r-- 1 ex_poly1 rnd    135 Oct 15 23:46 README
-rw-r--r-- 1 ex_poly1 rnd   860 Oct 15 23:44 TB_std_cell.v
-rwxr-xr-x 1 ex_poly1 rnd     34 Oct 14 00:19 clean.tcl
drwxr-xr-x 3 ex_poly1 rnd    33 Oct 14 00:44 dump
-rw-r--r-- 1 ex_poly1 rnd   314 Oct 27 15:49 func_sim.history
-rw-r--r-- 1 ex_poly1 rnd 42346 Oct 27 15:49 func_sim.log
drwxr-xr-x 2 ex_poly1 rnd    37 Oct 15 22:30 log
-rwxr-xr-x 1 ex_poly1 rnd   650 Oct 15 23:44 run_function.tcl
-rw-r--r-- 1 ex_poly1 rnd  7026 Oct 27 15:49 shm.prof
-rw-r--r-- 1 ex_poly1 rnd 11598 Oct 27 15:49 xmprof.out
-rw-r--r-- 1 ex_poly1 rnd   177 Oct 14 00:04 xrun.history
-rw-r--r-- 1 ex_poly1 rnd      5 Oct 15 23:29 xrun.key
-rw-r--r-- 1 ex_poly1 rnd   432 Oct 14 00:04 xrun.log
```

# 디지털 라이브러리 테스트

## README 파일 확인

### 1. \$> vi README ↵

- 시뮬레이션이 이런 식으로 진행되는 것을 알 수 있음

```
$finish in testbench file  
1) just ./run_function.tcl  
2) $> simvision  
3) open database in dump folder  
4) no need to restart simulation
```

# 디지털 라이브러리 테스트

## TB\_std\_cell.v 구조

1. \$> vi TB\_std\_cell.v

- 59행 \$finish 명령어 확인
- 나머지 LAB1과 동일

```
50 initial begin
51     CK      <= 1'b0;
52     IN      <= 3'b000;
53     RN      <= 1'b1;
54
55 #199    RN      <= 1'b0;
56 #100    RN      <= 1'b1;
57
58 #500
59 $finish;
60 end
61
62 always begin
63 #2      CK      <= ~CK;
64 end
65
66 always begin
67 #2      IN      <= IN+1'b1;
68 end
```



시뮬레이션  
799ns 까지  
진행

# LAB2

# 시뮬레이션

# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

1. \$> ./clean.tcl ↵

2. \$> ./run\_function.tcl ↵

- xrun 실행

-gui 옵션을 넣지 않아 799ns 시뮬레이션  
진행 후 아무런 창이 뜨지 않음

```
[ex_poly1@npit LAB2]$ ./clean.tcl
[ex_poly1@npit LAB2]$ ./run_function.tcl
```

# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

1. \$> //

시뮬레이션 입력 시간 확인 가능						
[ex_poly1@npit LAB2]\$ ll						
total 96						
-rw-r--r-- 1 ex_poly1 rnd 135 Oct 27 23:46 README						
-rw-r--r-- 1 ex_poly1 rnd 860 Oct 27 23:44 TB_std_cell.v						
-rwxr-xr-x 1 ex_poly1 rnd 34 Oct 27 00:19 clean.tcl						
drwxr-xr-x 3 ex_poly1 rnd 33 Oct 27 00:44 dump						
-rw-r--r-- 1 ex_poly1 rnd 314 Oct 27 15:49 func_sim.history						
-rw-r--r-- 1 ex_poly1 rnd 42346 Oct 27 15:49 func_sim.log						
drwxr-xr-x 2 ex_poly1 rnd 37 Oct 15 22:30 log						
-rwxr-xr-x 1 ex_poly1 rnd 650 Oct 15 23:44 run_function.tcl						
-rw-r--r-- 1 ex_poly1 rnd 7026 Oct 27 15:49 shm.prof						
drwxr-xr-x 6 ex_poly1 rnd 214 Oct 27 15:49 xcelium.d						
-rw-r--r-- 1 ex_poly1 rnd 11598 Oct 27 15:49 xmprof.out						
-rw-r--r-- 1 ex_poly1 rnd 177 Oct 14 00:04 xrun.history						
-rw-r--r-- 1 ex_poly1 rnd 5 Oct 15 23:29 xrun.key						
-rw-r--r-- 1 ex_poly1 rnd 432 Oct 14 00:04 xrun.log						

# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

### 1. \$> simvision

- simvision 명령으로 시뮬레이션 파형 결과 확인

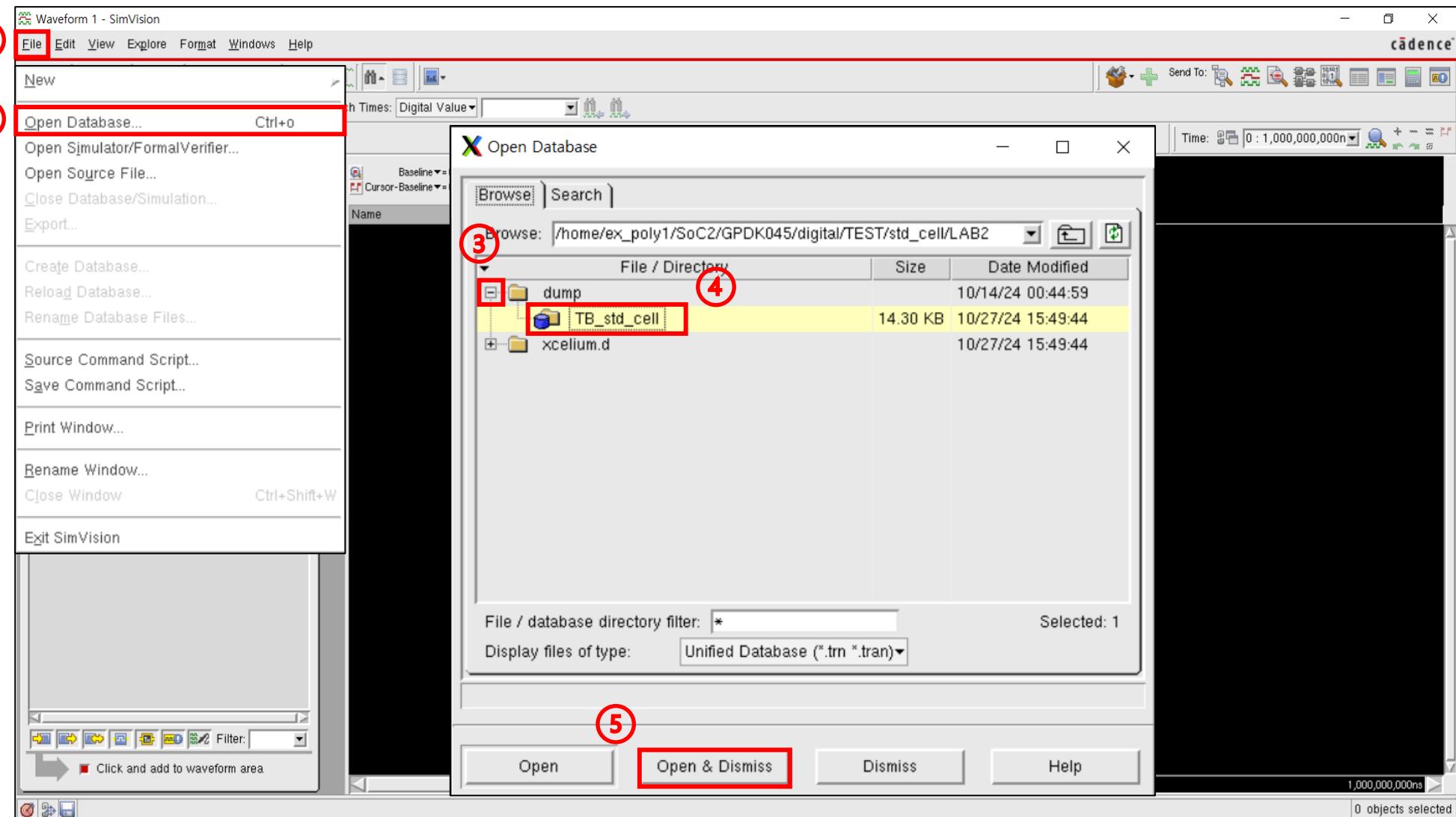
```
[ex_poly1@npit LAB2]$ simvision
simvision(64): 24.03-s005: (c) Copyright 1995-2024 Cadence Design Systems, Inc.
txe(64): 24.03-s005: (c) Copyright 1995-2024 Cadence Design Systems, Inc.
```

# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

### 1. \$> simvision

- ① File 클릭
- ② Open Database 클릭
- ③ dump 폴더 오픈
- ④ TB\_std\_cell 클릭
- ⑤ Open&Dismiss 클릭

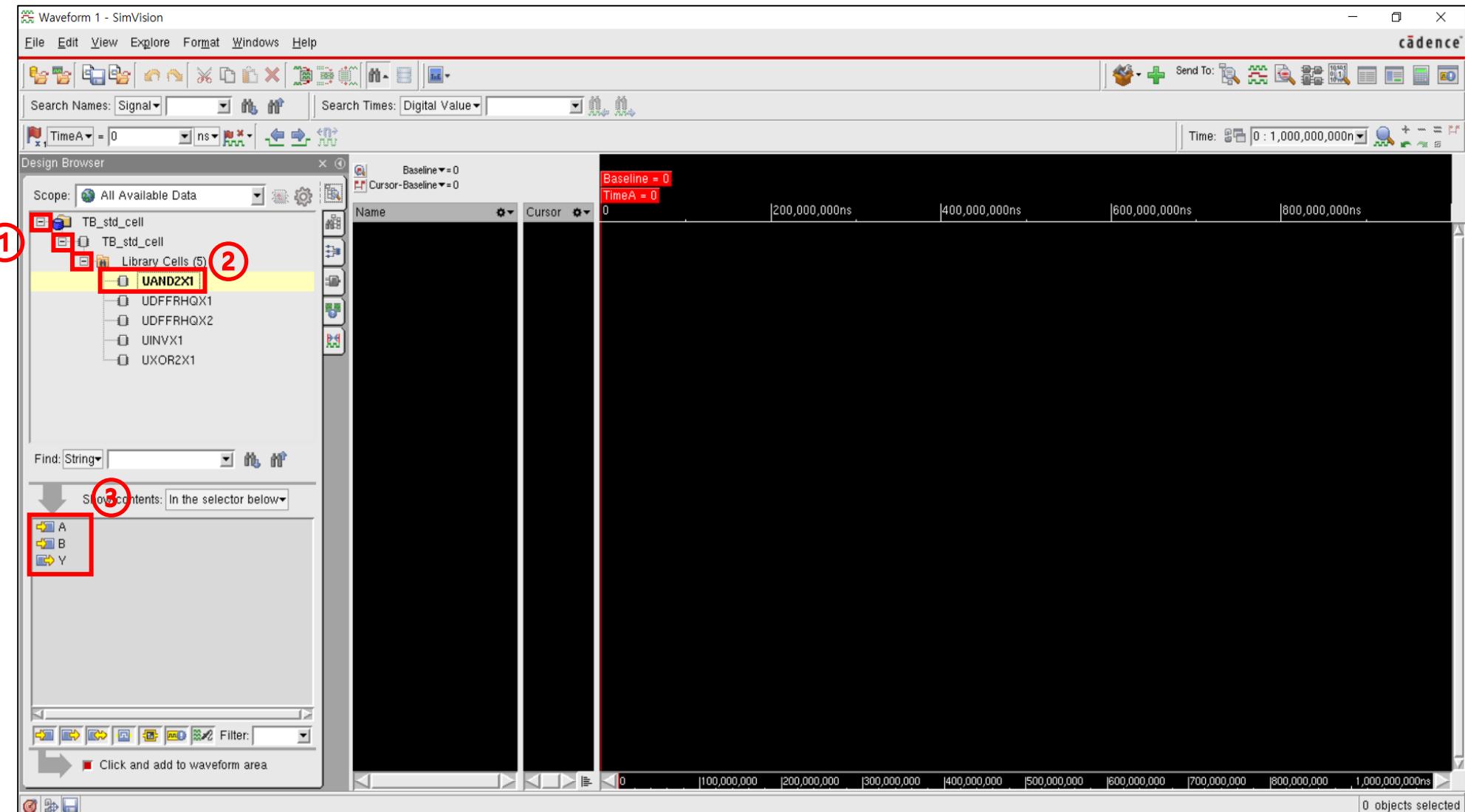


# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

### 1. \$> simvision

- ① Library Cell 오픈
- ② UAND2X1 클릭
- ③ A, B, Y 순서로 클릭

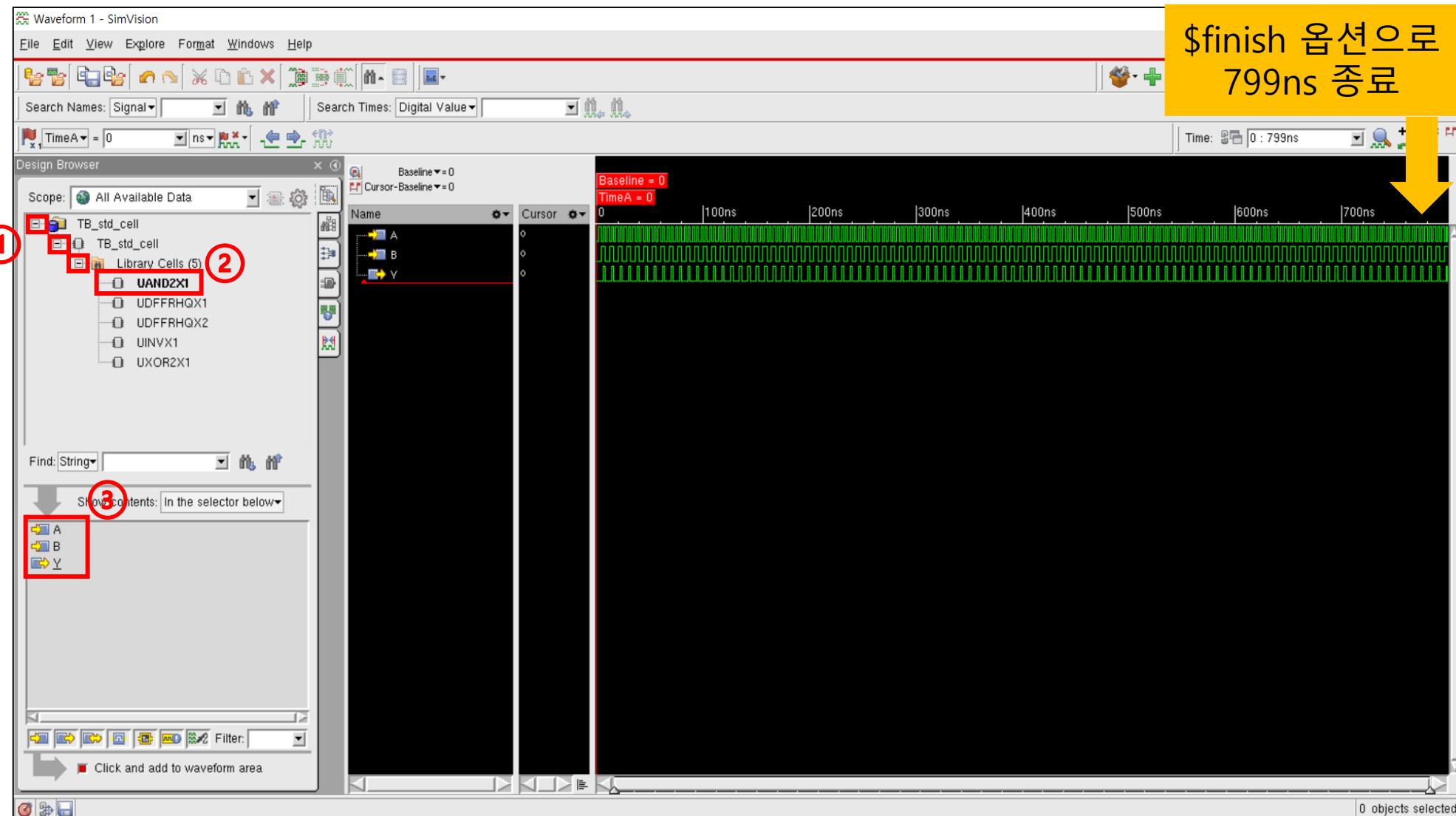


# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

### 1. \$> simvision

- ① Library Cell 오픈
- ② UAND2X1 클릭
- ③ A, B, Y 순서로 클릭



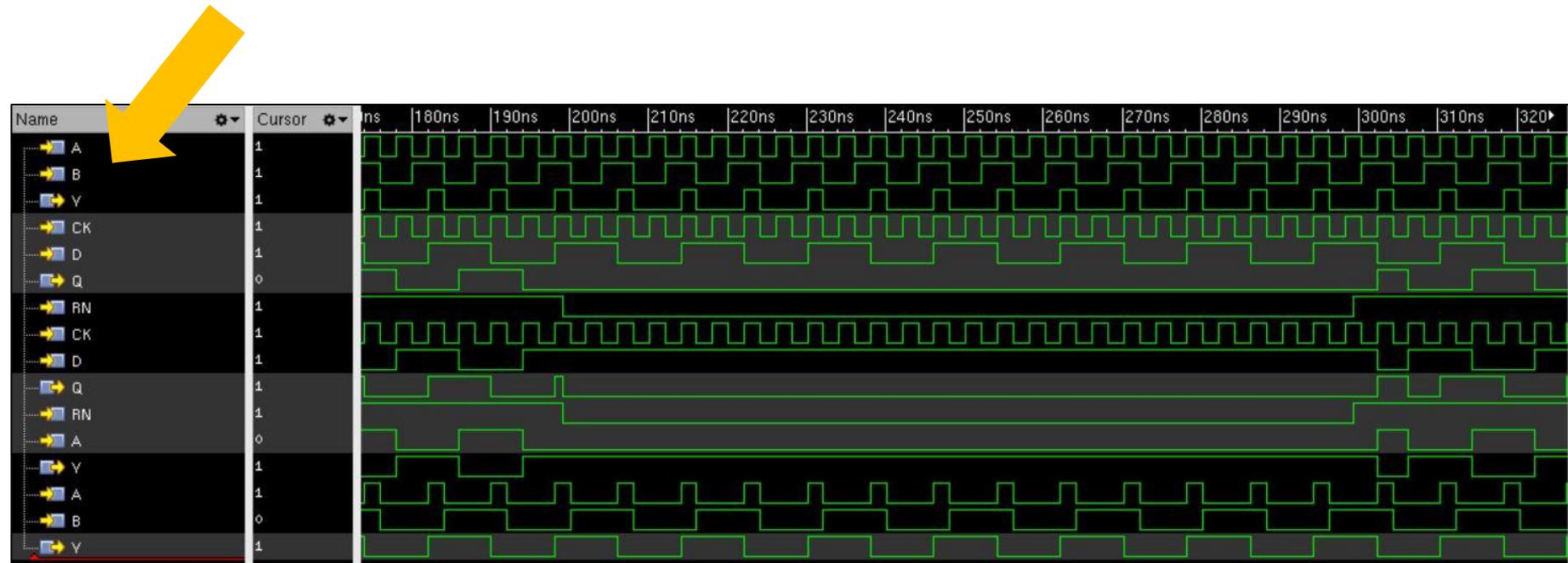
# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

### 1. \$> simvision

- LAB1과 결과가 같은 것을 알 수 있음

내가 원하는 파형만  
불러올 수 있음



# LAB3

# 디지털 라이브러리 테스트

## LAB3의 목적

1. -gui 옵션 사용 가능 simvision 확인
2. clock의 반전이 0.5ns 주기는 1ns
3. clock의 반전을 빠르게 했을 때 동작을 제대로 할 수 있나 확인
4. 최대 동작 주파수와 최대 동작 스피드 확인

# 디지털 라이브러리 테스트

## LAB3 파일 확인

1. \$>cd LAB3 ↵

2. \$> ll ↵

```
[ex_poly1@npit LAB3]$ ll
total 108
-rw-r--r-- 1 ex_poly1 rnd    178 Oct 20 22:31 README
-rw-r--r-- 1 ex_poly1 rnd    889 Oct 20 22:27 TB_std_cell.v
-rwxr-xr-x 1 ex_poly1 rnd     34 Oct 14 00:19 clean.tcl
drwxr-xr-x 3 ex_poly1 rnd    25 Oct 14 00:44 dump
-rw-r--r-- 1 ex_poly1 rnd   319 Nov  2 00:46 func_sim.history
-rw-r--r-- 1 ex_poly1 rnd 42728 Nov  2 00:48 func_sim.log
drwxr-xr-x 2 ex_poly1 rnd    29 Oct 14 00:45 log
-rwxr-xr-x 1 ex_poly1 rnd   662 Oct 15 23:44 run_function.tcl
-rw-r--r-- 1 ex_poly1 rnd  7026 Oct 14 21:53 shm.prof
-rw-r--r-- 1 ex_poly1 rnd 23608 Nov  2 00:48 xmprof.out
-rw-r--r-- 1 ex_poly1 rnd    448 Oct 21 23:31 xrun.history
-rw-r--r-- 1 ex_poly1 rnd    231 Nov  2 00:48 xrun.key
-rw-r--r-- 1 ex_poly1 rnd   331 Oct 21 23:31 xrun.log
```

# 디지털 라이브러리 테스트

## README 파일 확인

1. \$> vi README ↵

- 시뮬레이션이 이런 식으로 진행되는 것을 알 수 있음

```
with option -gui  
simvision auto started  
dump folder is auto opened  
press start button and stop at $stop function  
  
Lab3 is for high speed test  
1ns test, so toggle at each 0.5ns
```

# 디지털 라이브러리 테스트

## TB\_std\_cell.v 구조

1. \$> vi TB\_std\_cell.v



- **always begin**: CK신호 0.5ns마다 반전 주기는 1ns
- 나머진 LAB1과 동일

반복적으로 일어나는  
것에 대한 기술

```
62 // check the maximum speed
63 always begin
64 #0.5      CK      <= ~CK;
65 end
66
67 always begin
68 #2        IN      <= IN+1'b1;
69 end
```

# LAB3

## 시뮬레이션

# 디지털 라이브러리 테스트

## LAB3 시뮬레이션

### 1. \$> simvision

- simvision 명령 후 시뮬레이션 결과 확인 함

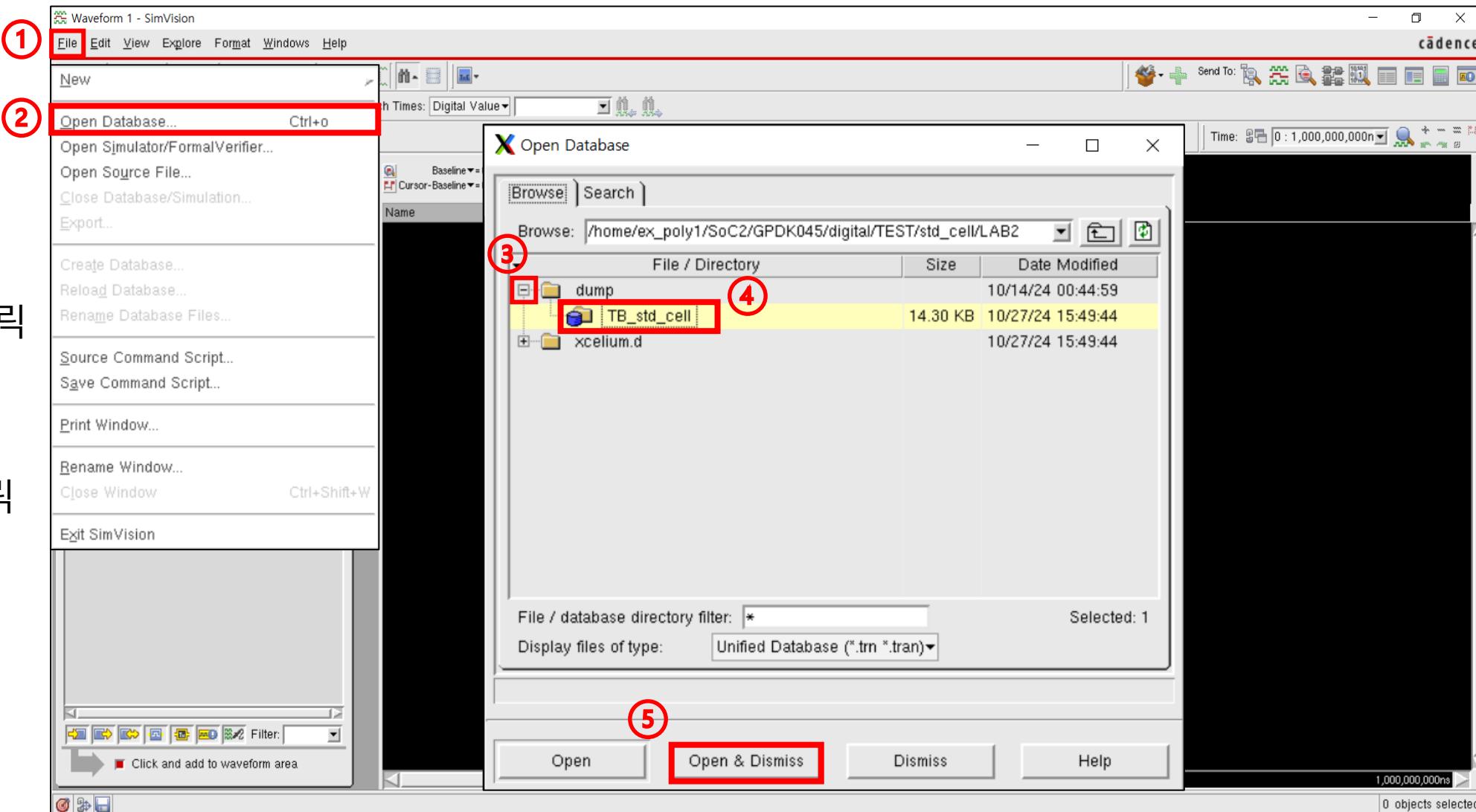
```
[ex_poly1@npit LAB2]$ simvision
simvision(64): 24.03-s005: (c) Copyright 1995-2024 Cadence Design Systems, Inc.
txe(64): 24.03-s005: (c) Copyright 1995-2024 Cadence Design Systems, Inc.
```

# 디지털 라이브러리 테스트

## LAB3 시뮬레이션

### 1. \$> simvision

- ① File 클릭
- ② Open Database 클릭
- ③ dump 폴더 오픈
- ④ TB\_std\_cell 클릭
- ⑤ Open&Dismiss 클릭

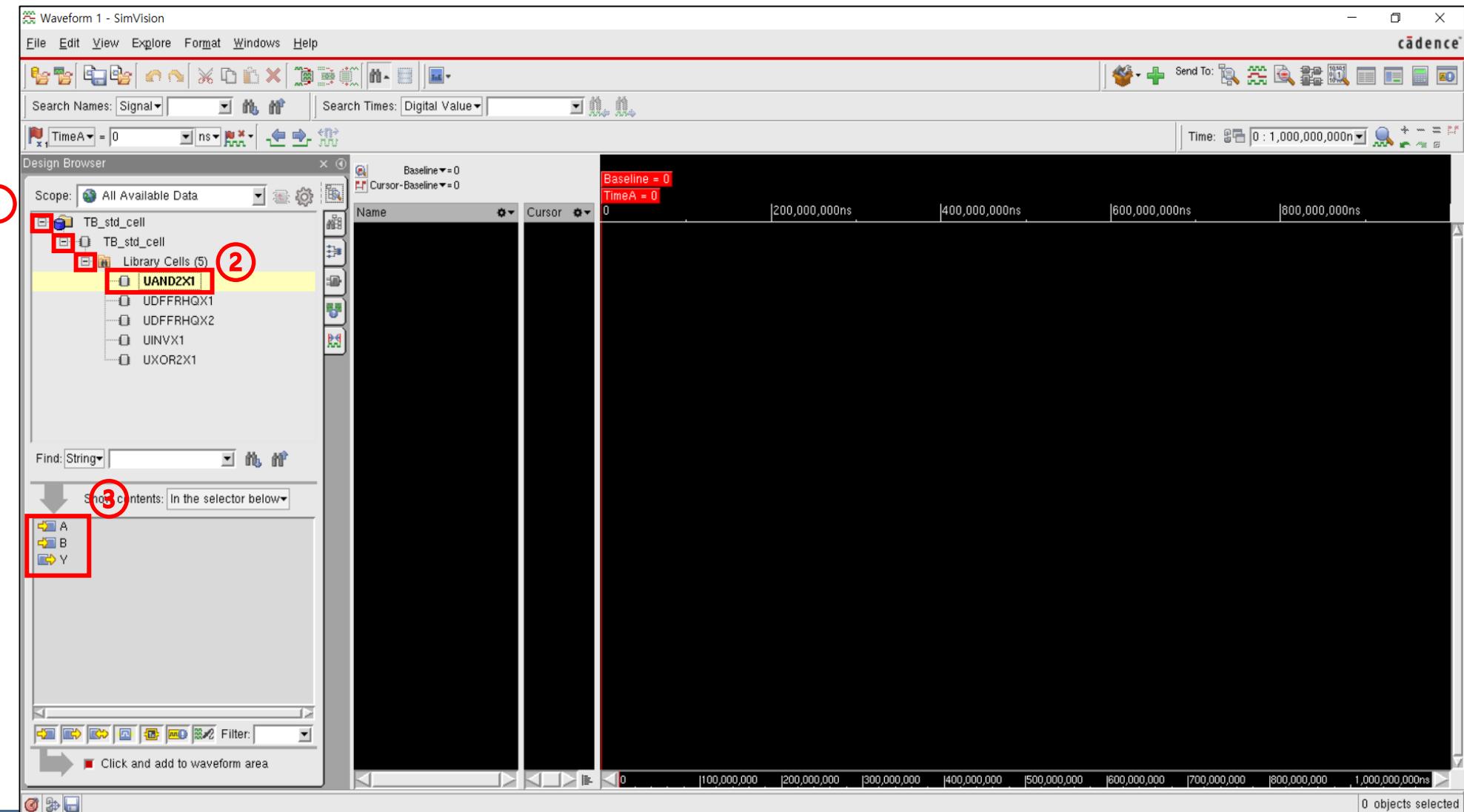


# 디지털 라이브러리 테스트

## LAB3 시뮬레이션

### 1. \$> simvision

- ① Library Cell 오픈
- ② UAND2X1 클릭
- ③ A, B, Y 순서로 클릭

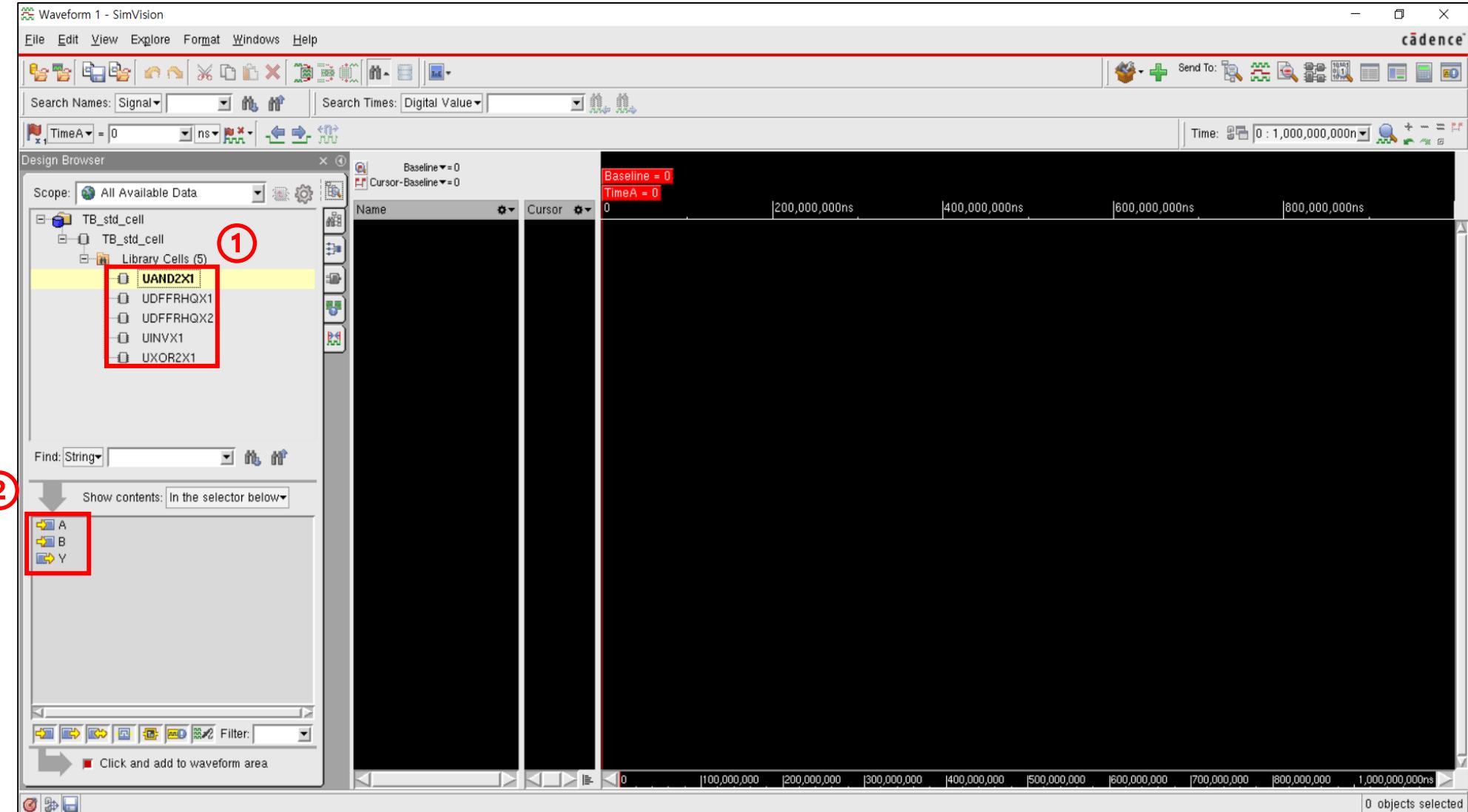


# 디지털 라이브러리 테스트

## LAB3 시뮬레이션

### 1. \$> simvision

- 모든 파형 입력

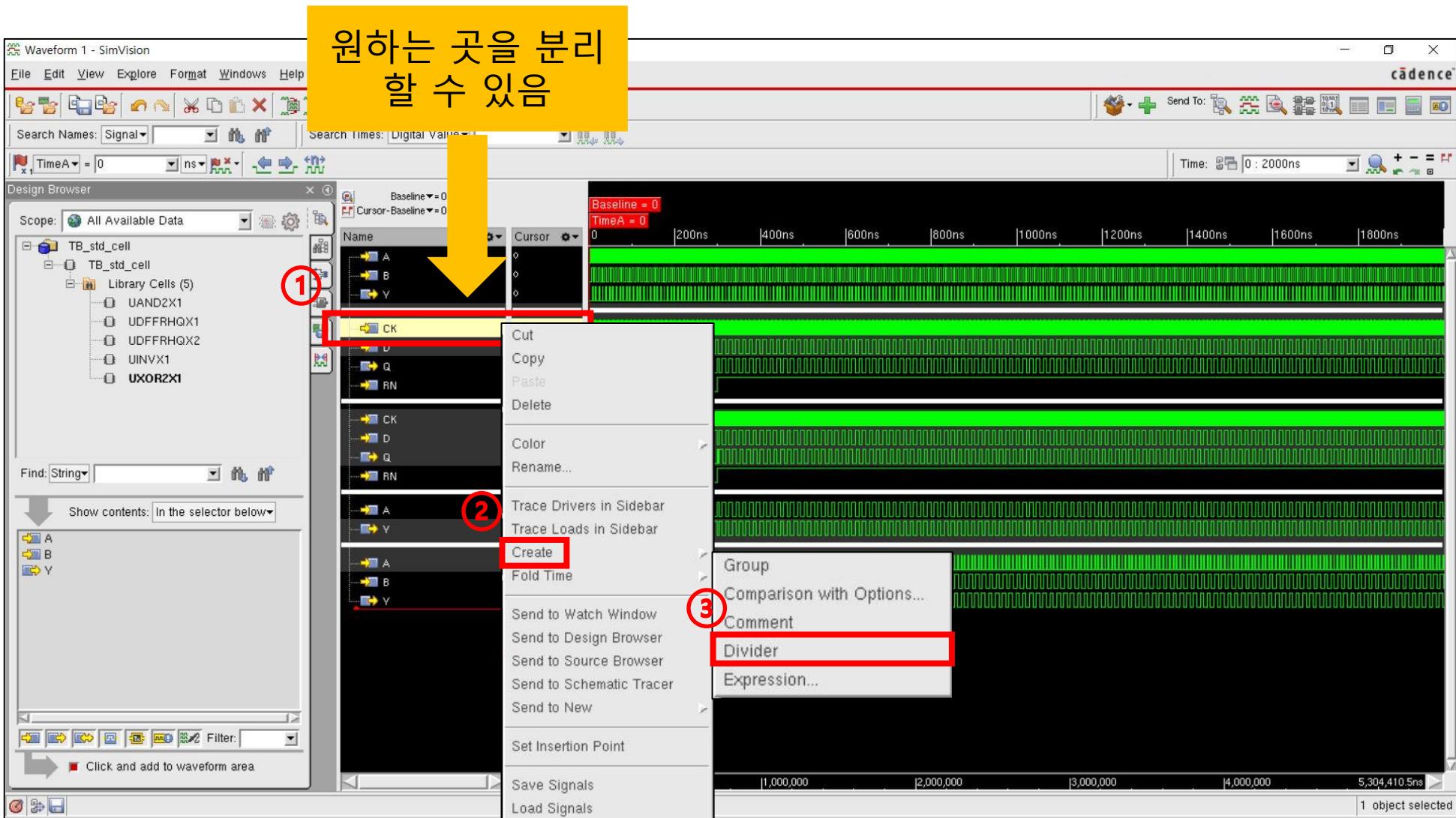


# 디지털 라이브러리 테스트

## LAB3 시뮬레이션

### 1. \$> simvision

- ① CK 클릭 후 우클릭
- ② Create 클릭
- ③ Divider 클릭

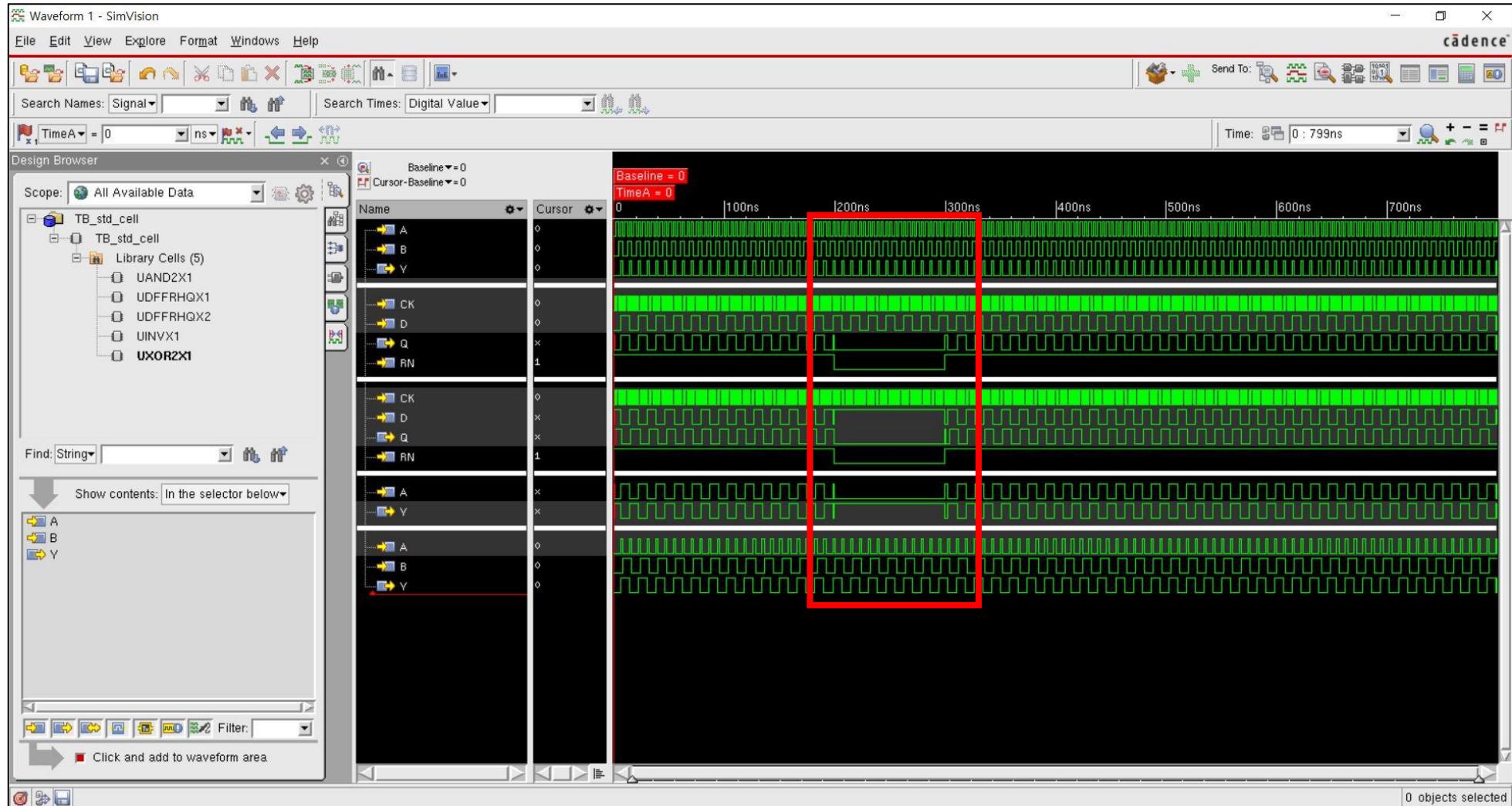


# 디지털 라이브러리 테스트

## LAB3 시뮬레이션

### 1. \$> simvision

- 파형 드래그 하여 실행 결과 확인



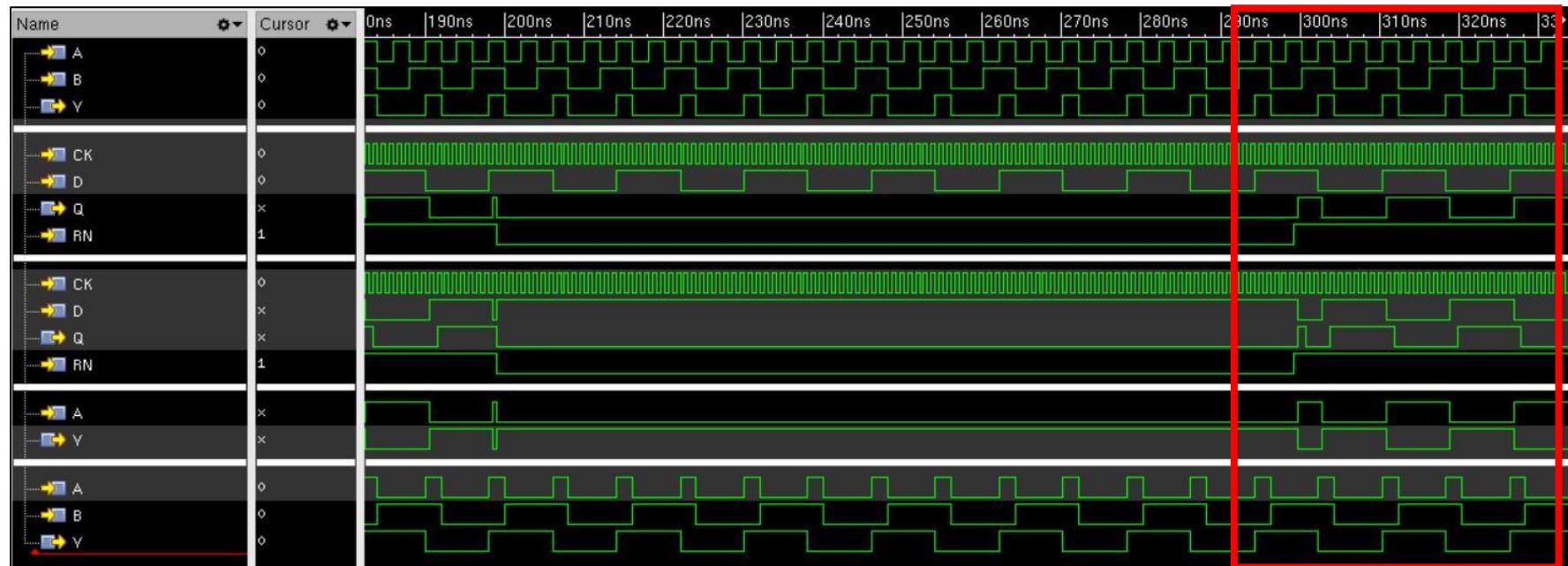
# 디지털 라이브러리 테스트

## LAB3 시뮬레이션

### 1. \$> simvision

- 리셋 신호에 맞추어 출력 신호  
가 잘 반응 함
- 셀이  $f = 1/T$  이므로 최대  
1GHz에서도 잘 동작함

확대해서 확인



# 디지털 라이브러리 테스트

## LAB3 시뮬레이션

### 1. \$> simvision

- D-FF의 리셋 직후의 동작을 살펴보아야 함
- 셀이 리셋 직후에도 CK신호와 D신호에 맞춰 출력이 잘 나옴
- 빨간 상자 확대해서 한번 더 확인

RN신호가 0 일 때 동작을 안하고  
RN에 1신호가 들어오면 다시 동작



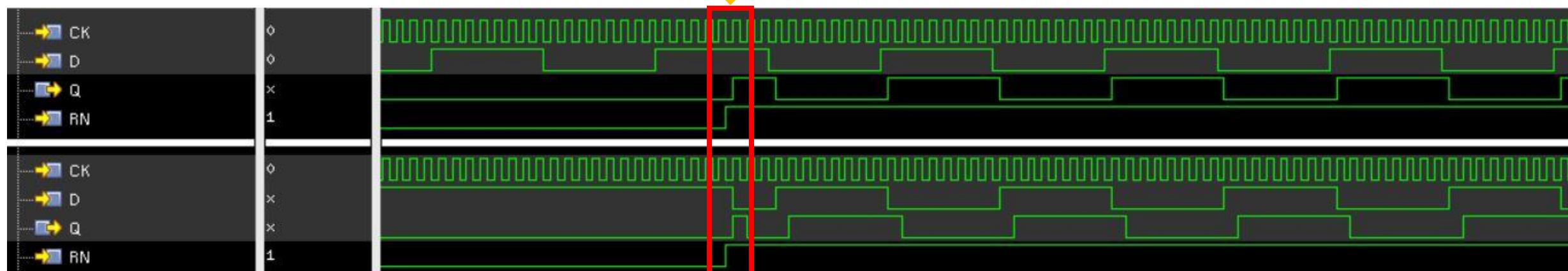
# 디지털 라이브러리 테스트

## LAB3 시뮬레이션

### 1. \$> simvision

- 동작에 문제는 없지만 사용 범위를 알아보기 위해 500MHz, 750MHz에서도 시뮬레이션 후 결과 확인 필요

리셋 직후 CK신호가 상승하는 타이밍에 맞추어 D값 출력



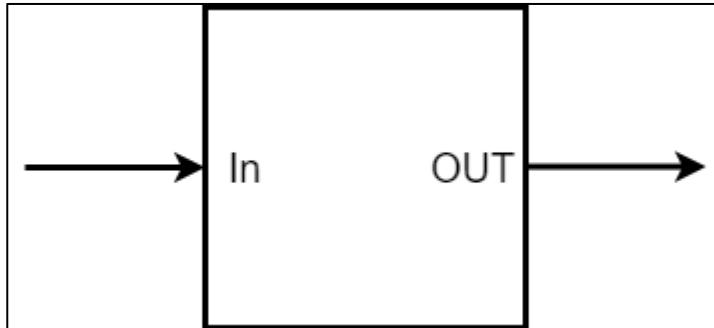
# IO셀 라이브러리 시뮬레이션

# 디지털 라이브러리 테스트

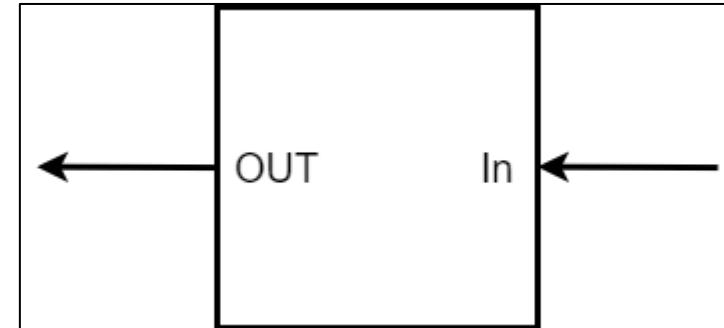
## IO셀 이란?

1. 신호가 들어가기도 나오기도 하는 양방향 셀(Input-Output)
2. ESD회로: 외부 정전기로 인한 회로 손상 방지용으로 사용
3. VDD, VSS를 연결해주는 셀(Core영역으로 Power 인가에 용이)

<하나의 셀에서 IN OUT 둘 다 가능>



<그림4>



<그림5>

# LAB1

# 디지털 라이브러리 테스트

## LAB1의 목적

1. IO셀의 기본적인 동작 확인
2. PADDB, PADDI, PADDO의 구조와 동작 이해

# 디지털 라이브러리 테스트

## LAB1 파일 확인

1. \$>cd LAB1 ↵

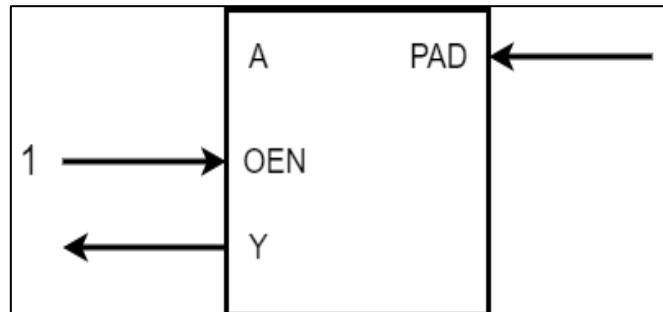
2. \$> ll ↵

```
[ex_poly1@npit LAB1]$ ll
total 32
-rw-r--r-- 1 ex_poly1 rnd 1008 Oct 19 02:53 TB_io_cell.v
-rwxr-xr-x 1 ex_poly1 rnd   34 Oct 19 02:05 clean.tcl
-rw-r--r-- 1 ex_poly1 rnd  272 Nov  2 17:08 func_sim.history
-rw-r--r-- 1 ex_poly1 rnd 3421 Nov  2 17:32 func_sim.log
-rwxr-xr-x 1 ex_poly1 rnd  615 Oct 15 23:57 run_function.tcl
-rw-r--r-- 1 ex_poly1 rnd 4634 Nov  2 17:32 xmprof.out
-rw-r--r-- 1 ex_poly1 rnd  388 Nov  2 17:32 xrun.key
```

# 디지털 라이브러리 테스트

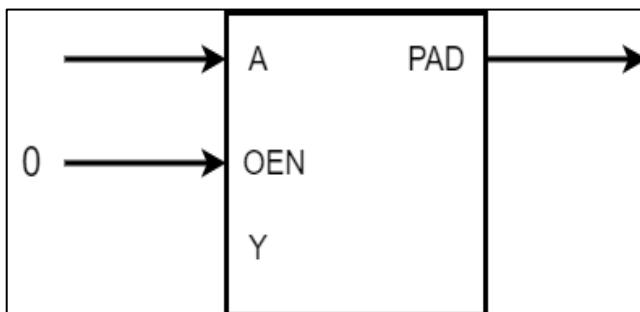
## TB\_io\_cell.v 구조

① PADDB\_IN



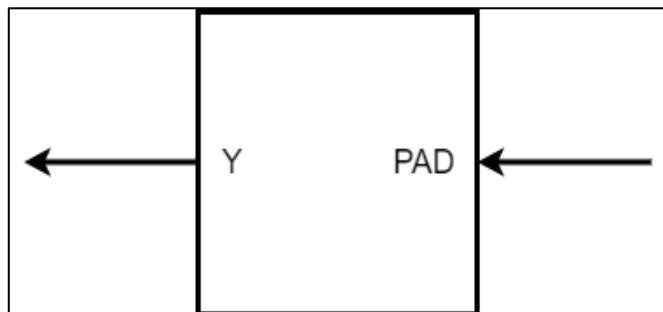
<그림3>

② PADDB\_OUT



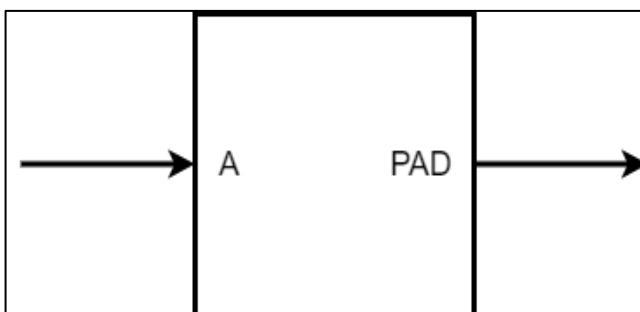
<그림6>

③ PADDI



<그림7>

④ PADDO



<그림8>

```
20 // Bi-directional Cell
21 // PAD to Y
22 PADDB UPADDB_IN(
23     .A(),
24     .OEN(OEN_IN),
25     .PAD(PAD_B_IN),
26     .Y(Y_B_IN)
27 );
28
29
30 // Bi-directional Cell
31 // A to PAD
32 PADDB UPADDB_OUT(
33     .A(A_B_OUT),
34     .OEN(OEN_OUT),
35     .PAD(PAD_B_OUT),
36     .Y()
37 );
38
39 // Input Buffer
40 PADDI UPADDI(
41     .PAD(PAD_IN),
42     .Y(Y_IN)
43 );
44
45 // Output Driver
46 PADDO UPADDO(
47     .A(A_OUT),
48     .PAD(PAD_OUT)
49 );
50
```

①

②

③

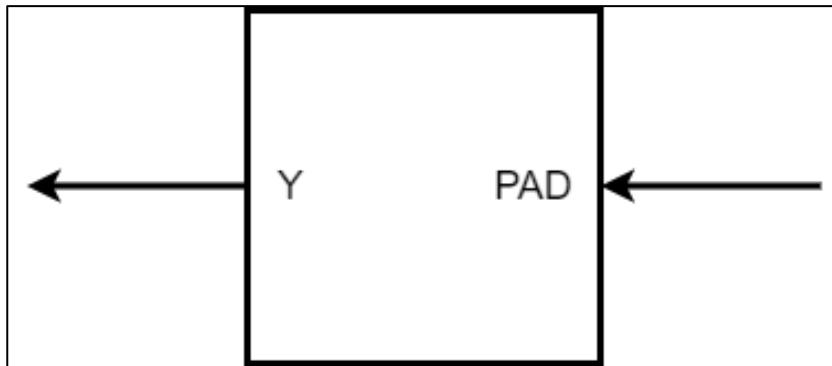
④

# 디지털 라이브러리 테스트

## TB\_io\_cell.v 구조

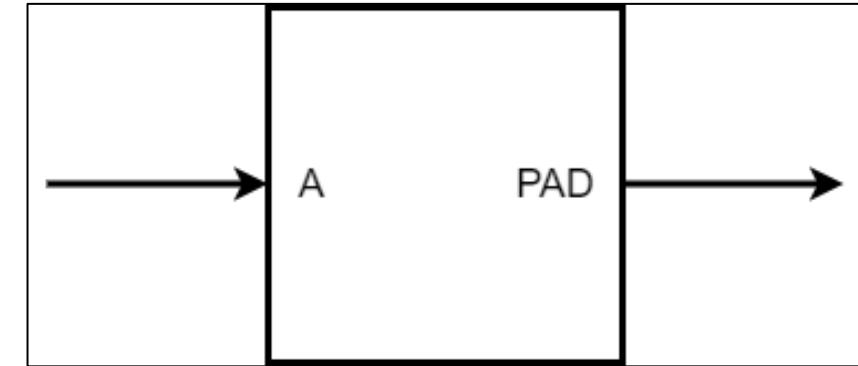
### 1. 한 방향 출력

1. PADDI



<그림7>

2. PADDO



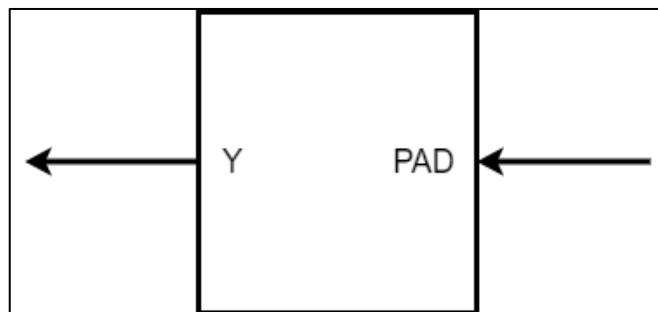
<그림8>

# 디지털 라이브러리 테스트

## TB\_io\_cell.v 구조

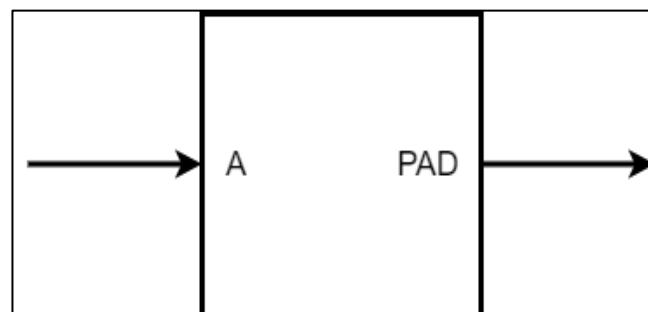
1. PADDI는 PAD가 입력하고 Y가 출력되는 방향으로만 사용 가능
2. PADDO는 A가 입력하고 PAD가 출력되는 방향으로만 사용 가능

1. PADDI



<그림7>

2. PADDO



<그림8>

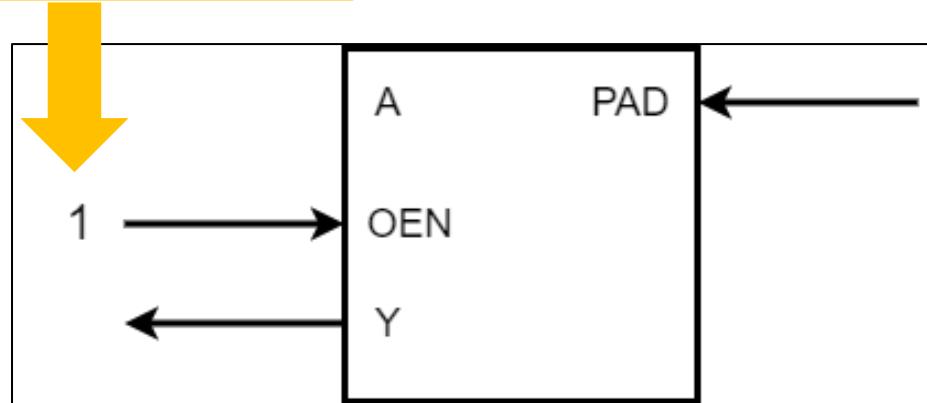
```
20 // Bi-directional Cell
21 // PAD to Y
22 PADDB UPADDB_IN(
23     .A(),
24     .OEN(OEN_IN),
25     .PAD(PAD_B_IN),
26     .Y(Y_B_IN)
27 );
28
29
30 // Bi-directional Cell
31 // A to PAD
32 PADDB UPADDB_OUT(
33     .A(A_B_OUT),
34     .OEN(OEN_OUT),
35     .PAD(PAD_B_OUT),
36     .Y()
37 );
38
39 // Input Buffer
40 PADDI UPADDI(
41     .PAD(PAD_IN),
42     .Y(Y_IN)
43 );
44
45 // Output Driver
46 PADDO UPADDO(
47     .A(A_OUT),
48     .PAD(PAD_OUT)
49 );
50
```

# 디지털 라이브러리 테스트

## TB\_io\_cell.v 구조

- 양방향 출력
- OEN 뜻은 Output Enable Negative이며 OEN값을 0 또는 1로 조절하면 하나의 셀로 입출력을 모두 처리할 수 있음

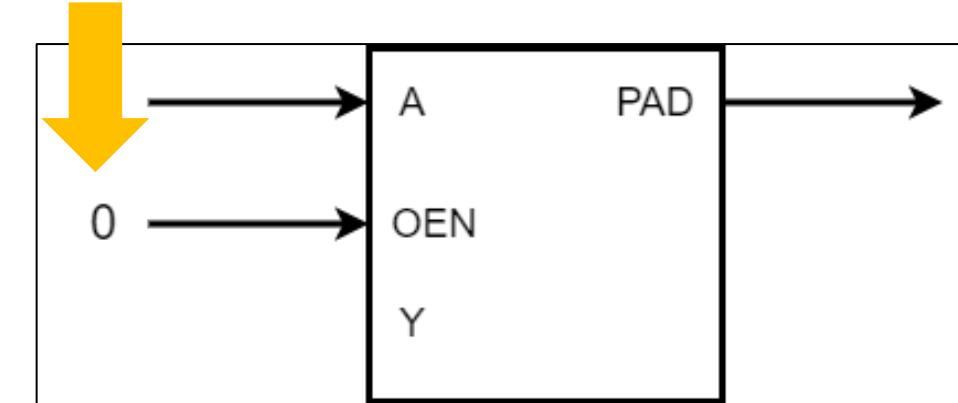
PAD에 입력을 하려면  
OEN 값이 1이어야 됨



1. PADDB\_IN

<그림3>

PAD가 출력을 하려면  
OEN 값이 0이어야 됨



2. PADDB\_OUT

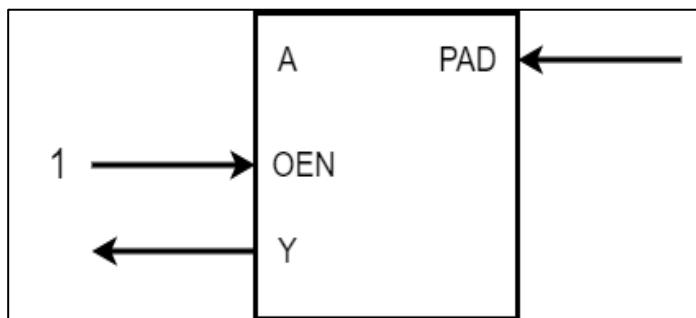
<그림6>

# 디지털 라이브러리 테스트

## TB\_io\_cell.v 구조

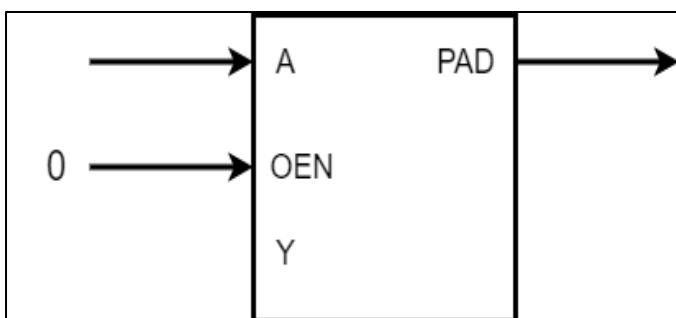
1. PADDB\_IN은 PAD로 입력 받고 Y로 출력함
2. PADDB\_OUT은 A로 입력 받고 PAD로 출력함

1. PADDB\_IN



<그림3>

2. PADDB\_OUT



<그림6>

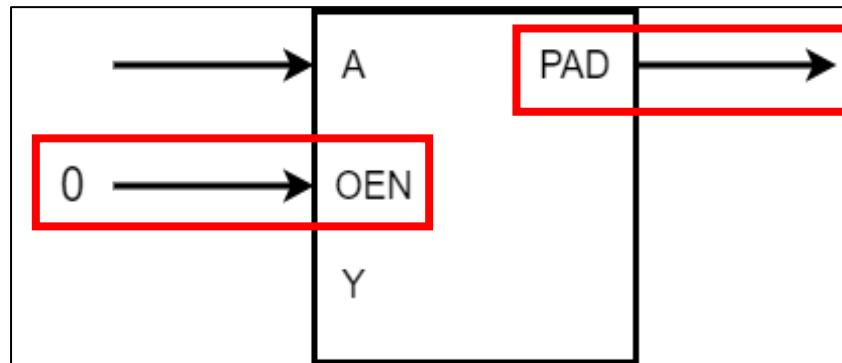
```
20 // Bi-directional Cell
21 // PAD to Y
22 PADDB_UPADDB_IN(
23     .A(),
24     .OEN(OEN_IN),
25     .PAD(PAD_B_IN),
26     .Y(Y_B_IN)
27 );
28
29
30 // Bi-directional Cell
31 // A to PAD
32 PADDB_UPADDB_OUT(
33     .A(A_B_OUT),
34     .OEN(OEN_OUT),
35     .PAD(PAD_B_OUT),
36     .Y()
37 );
38
39 // Input Buffer
40 PADDI_UPADDI(
41     .PAD(PAD_IN),
42     .Y(Y_IN)
43 );
44
45 // Output Driver
46 PADDO_UPADDO(
47     .A(A_OUT),
48     .PAD(PAD_OUT)
49 );
50
```

# 디지털 라이브러리 테스트

## TB\_io\_cell.v 구조

1. PAD가 출력을 하므로 OEN 값은 0이어야 함

### 1. PADDB\_OUT



<그림6>

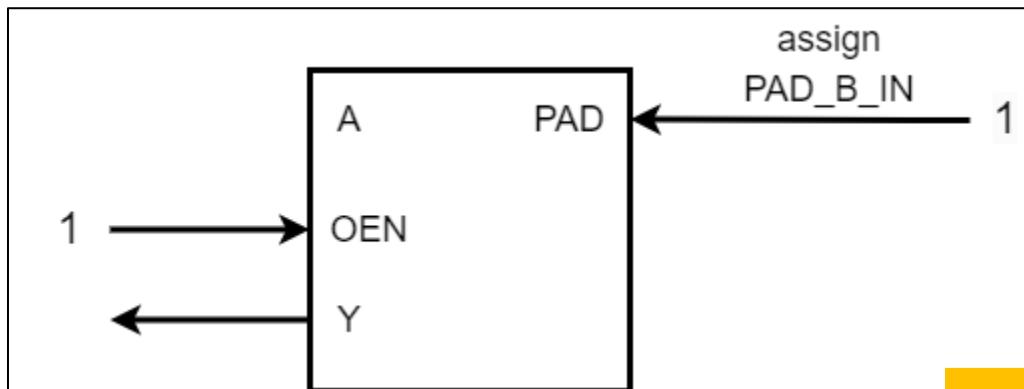
```
53 initial begin
54
55     OEN_IN <= 1'b1;
56
57     A_B_OUT<=1'b1;
58     OEN_OUT<=1'b1;
59     #20
60     OEN_OUT <=1'b0;
61     #20
62     A_B_OUT<=1'b0;
63
64     PAD_IN<= 1'b0;
65     #20
66     PAD_IN <=1'b1;
67
68     A_OUT<= 1'b0;
69     #20
70     A_OUT<= 1'b1;
71
72     #100
73     $stop;
74 end
```

# 디지털 라이브러리 테스트

## TB\_io\_cell.v 구조

1. PAD에 입력을 하므로 OEN 값이 1이어야 함

### 1. PADDB\_IN



<그림9>

assign으로 1 입력

PAD\_B\_IN을 reg로 했을 시  
오류 발생 wire로 수정

```
3
4
5 reg OEN_IN;
6 //reg PAD_B_IN;
7 wire PAD_B_IN;
8 wire Y_B_IN;
9
```

```
21 // PAD to Y
22 PADDB_UPADDB_IN(
23     .A(),
24     .OEN(OEN_IN),
25     .PAD(PAD_B_IN),
26     .Y(Y_B_IN)
27 );
```

```
51 assign PAD_B_IN = 1'b1;
```

# LAB1

# 시뮬레이션

# 디지털 라이브러리 테스트

## LAB1 시뮬레이션

1. ./clean.tcl ↵

2. ./run\_function.tcl ↵

```
[ex_poly1@npit LAB1]$ ./clean.tcl
[ex_poly1@npit LAB1]$ ./run_function.tcl
```

- xrun 실행

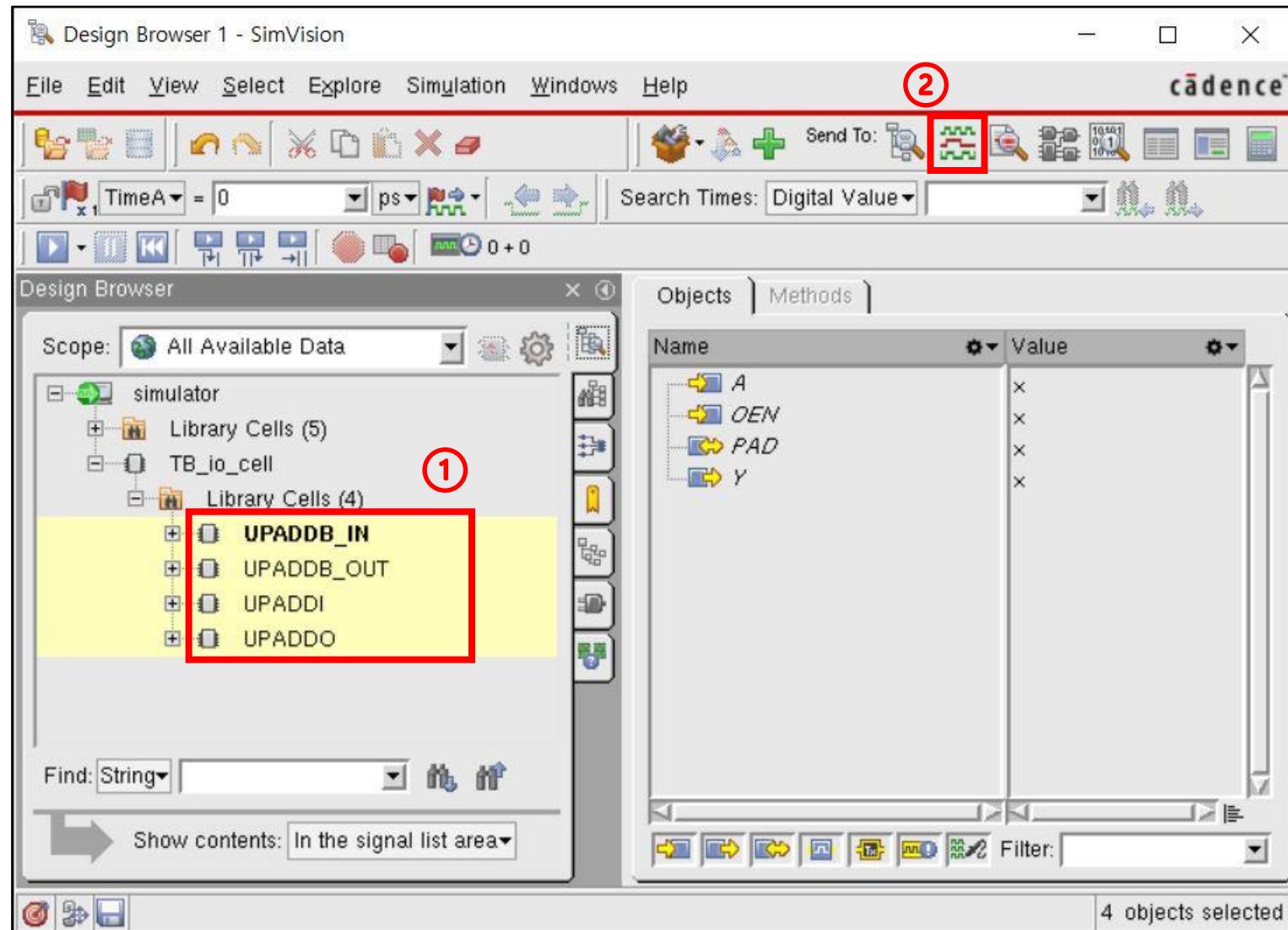
```
xrun -64bit \
+max_err_count+50 \
+define+function_sim \
-access +rwc \
-profile \
-profthread \
-gui \
TB_io_cell.v \
../../../../giolib045_v3.5/vlog/pads_FF_s1vg.v \
-l ./func_sim.log
```

# 디지털 라이브러리 테스트

## LAB1 시뮬레이션

### 1. xrun 실행 결과

- ① 모두 클릭
- ② 웨이브 아이콘 클릭

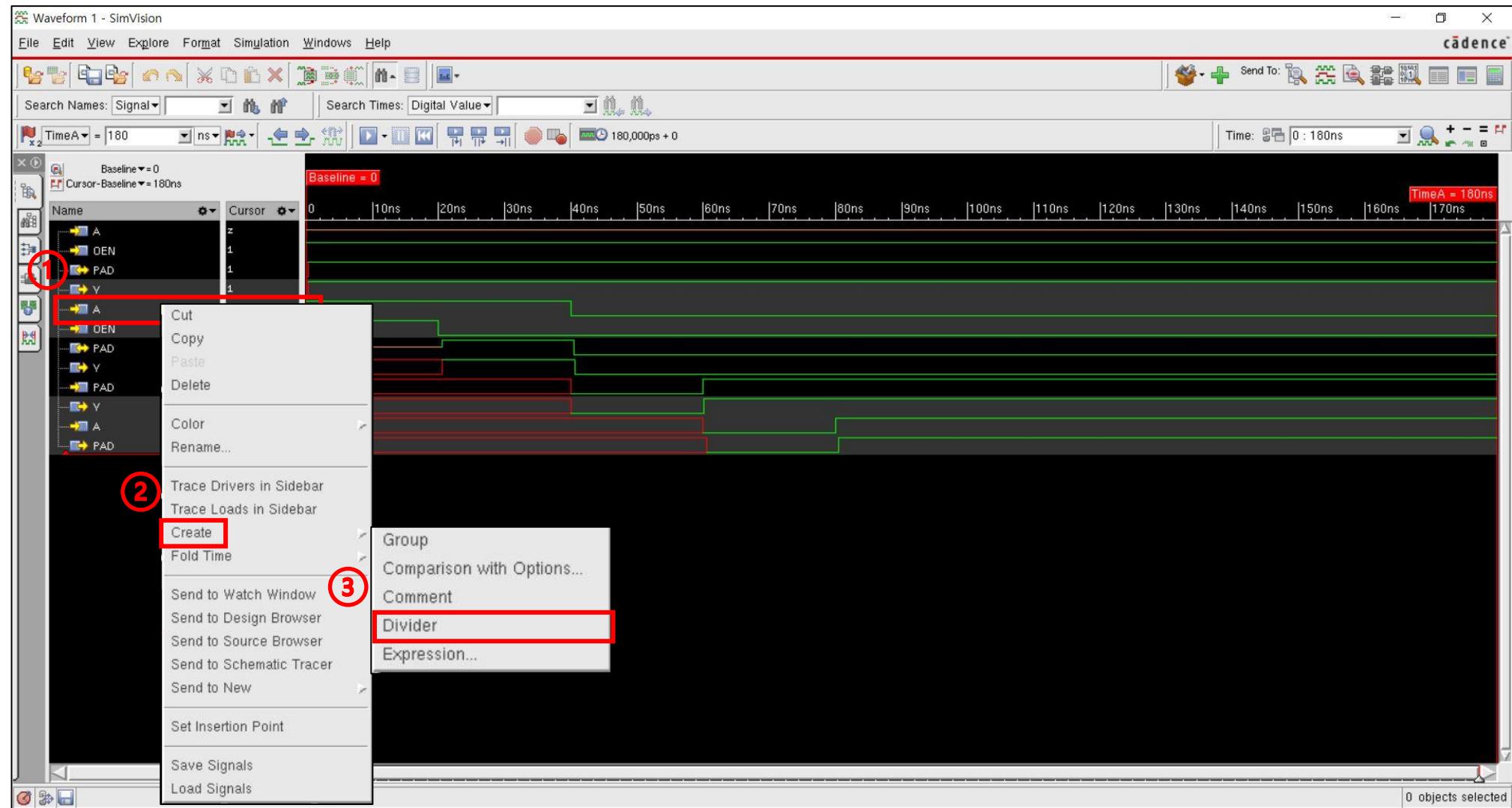


# 디지털 라이브러리 테스트

## LAB1 시뮬레이션

### 1. xrun 실행 결과

- ① A 클릭 후 우클릭
- ② Create 클릭
- ③ Divider 클릭

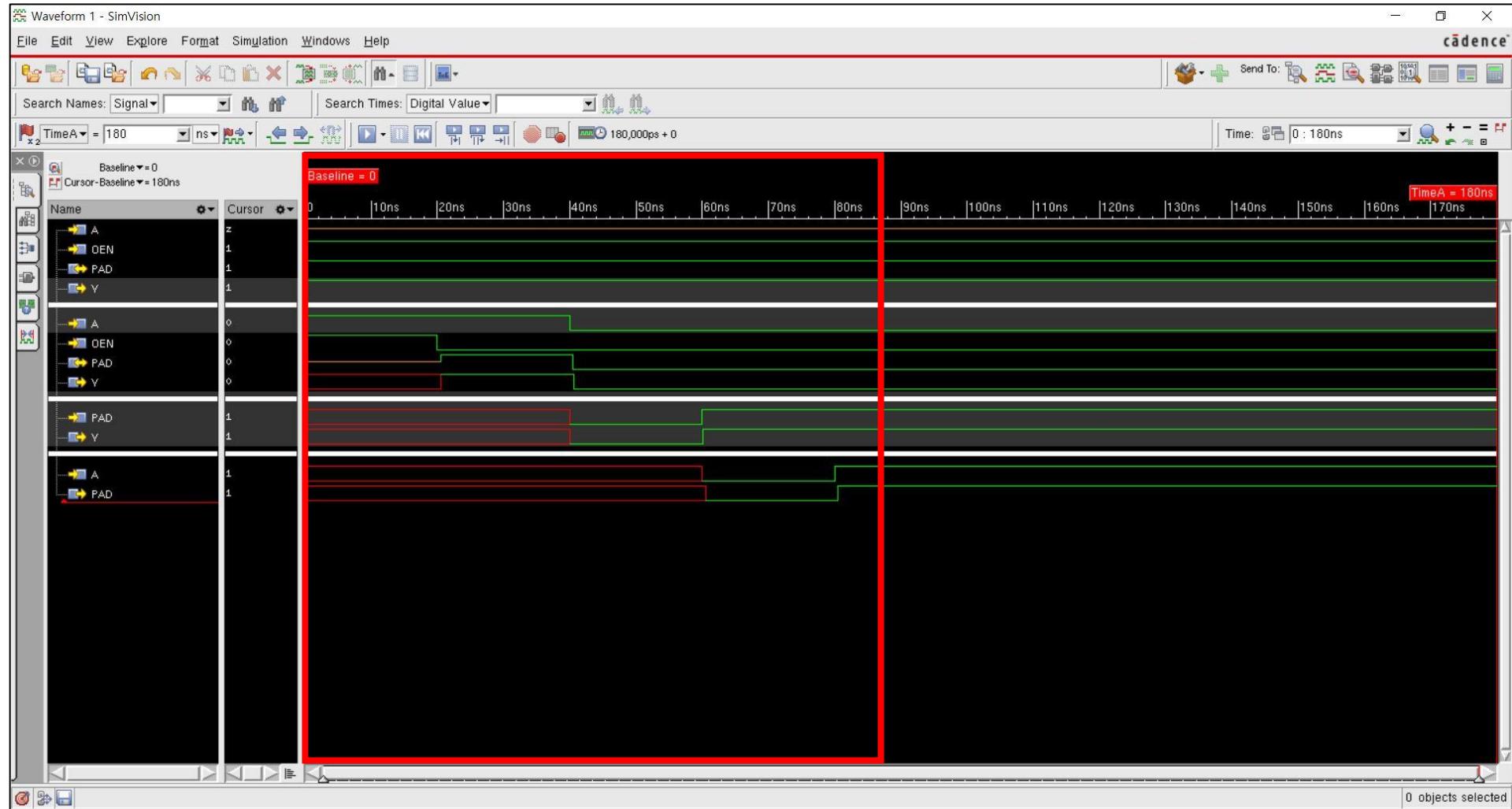


# 디지털 라이브러리 테스트

## LAB1 시뮬레이션

### 1. xrun 실행 결과

- 드래그 결과 확인



# 디지털 라이브러리 테스트

## LAB1 시뮬레이션

### 1. xrun 실행 결과

- 시뮬레이션 동작확인



# 디지털 라이브러리 테스트

## LAB1 시뮬레이션

### 1. xrun 실행 결과

- 시뮬레이션 동작확인
- 버퍼동작 문제없음

```
32 PADDB UPADDB_OUT(  
33     .A(A_B_OUT),  
34     .OEN(OEN_OUT),  
35     .PAD(PAD_B_OUT),  
36     .Y());  
37 );
```

```
53 initial begin  
54     OEN_IN <= 1'b1;  
55  
56     A_B_OUT<=1'b1;  
57     OEN_OUT<=1'b1;  
58     #20  
59     OEN_OUT <=1'b0;  
60     #20  
61     A_B_OUT<=1'b0;  
62  
63     PAD_IN<= 1'b0;  
64     #20  
65     PAD_IN <=1'b1;  
66  
67     A_OUT<= 1'b0;  
68     #20  
69     A_OUT<= 1'b1;  
70  
71     #100  
72     $stop;  
73  
74 end
```

OEN의 초기값을 1로 시작 후  
0으로 반전시켜 정상적으로  
동작하는지 확인



# LAB2

# 디지털 라이브러리 테스트

## LAB2의 목적

### 1. IO셀의 성능확인

- 최대 동작 스피드와 허용 가능한 신호의 가용범위 알아보기

# 디지털 라이브러리 테스트

## LAB2 파일 확인

1. \$>cd LAB2 ↵

2. \$> ll ↵

```
[ex_poly1@npit LAB2]$ ll
total 32
-rw-r--r-- 1 ex_poly1 rnd 1135 Oct 20 23:34 TB_io_cell.v
-rwxr-xr-x 1 ex_poly1 rnd    34 Oct 19 02:05 clean.tcl
-rw-r--r-- 1 ex_poly1 rnd   272 Nov  2 22:32 func_sim.history
-rw-r--r-- 1 ex_poly1 rnd 3237 Nov  3 00:03 func_sim.log
-rwxr-xr-x 1 ex_poly1 rnd   615 Oct 15 23:57 run_function.tcl
-rw-r--r-- 1 ex_poly1 rnd 4812 Nov  3 00:03 xmprof.out
-rw-r--r-- 1 ex_poly1 rnd   354 Nov  3 00:03 xrun.key
```

# 디지털 라이브러리 테스트

## TB\_io\_cell.v 구조

1. 8행에서 PAD\_B\_IN\_reg를 reg로 선언
2. 83행에서 0.5ns마다 반전을 해주기 위해 wire인 PAD\_B\_IN에 assign 명령으로 PAD\_B\_IN\_reg값을 입력해 줌

0.5ns 반전, 주기는 1ns

```
1 `timescale 1ns/1ps
2 module TB_io_cell(
3 );
4
5 reg OEN_IN;
6 //reg PAD_B_IN;
7 wire PAD_B_IN;
8 reg PAD_B_IN_reg; 초기값은 0으로 지정
9 wire Y_B_IN;
```

```
55 initial begin
56     PAD_B_IN_reg <= 1'b0;
57     OEN_IN <= 1'b1;
```

```
79 always begin
80     #0.5 PAD_B_IN_reg <= ~PAD_B_IN_reg;
81 end
82
83 assign PAD_B_IN = PAD_B_IN_reg;
```

# 디지털 라이브러리 테스트

## TB\_io\_cell.v 구조

1. \$> vi TB\_io\_cell.v -d ..../LAB1/TB\_io\_cell.v

- Tip: LAB1 TB파일과 LAB2 TB파일의 차이를 한눈에 보여줌

The screenshot shows a terminal window at the bottom with the command:

```
[ex_poly1@npit LAB2]$ vi TB_io_cell.v -d ..../LAB1/TB_io_cell.v
```

Two code editors are displayed above the terminal. The left editor is labeled "TB\_io\_cell.v" and the right is "B1/TB\_io\_cell.v". Both files contain Verilog code for a test bench. The right file has several lines highlighted in pink, while the left file has them in blue. A yellow callout box with the text "-d: Differential의 약자로 어디가 다른 지 보여줌" is pointing to the "-d" option in the terminal command.

```
File Edit View Search Terminal Help
50      .A(A_OUT),
51      .PAD(PAD_OUT)
52 );
53
54 initial begin
55     PAD_B_IN_reg <= 1'b0;
56     OEN_IN <= 1'b1;
57
58     A_B_OUT<=1'b1;
59     OEN_OUT<=1'b1;
60
61     #20
62     OEN_OUT <=1'b0;
+ 63 +-- 9 lines: #20-
64     A_OUT<= 1'b1;
65
66     #100
67     $stop;
68 end
69
70 always begin
71     #0.5 PAD_B_IN_reg <= ~PAD_B_IN_reg;
72 end
73
74 assign PAD_B_IN = PAD_B_IN_reg;
```

```
47      .A(A_OUT),
48      .PAD(PAD_OUT)
49 );
50
51 assign PAD_B_IN = 1'b1;
52 initial begin
53     OEN_IN <= 1'b1;
54
55     A_B_OUT<=1'b1;
56     OEN_OUT<=1'b1;
57
58     #20
59     OEN_OUT <=1'b0;
```

```
[ex_poly1@npit LAB2]$ vi TB_io_cell.v -d ..../LAB1/TB_io_cell.v
```

# LAB2

# 시뮬레이션

# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

1. ./clean.tcl ↵

```
[ex_poly1@npit LAB1]$ ./clean.tcl  
[ex_poly1@npit LAB1]$ ./run_function.tcl
```

2. ./run\_function.tcl ↵

- xrun 실행

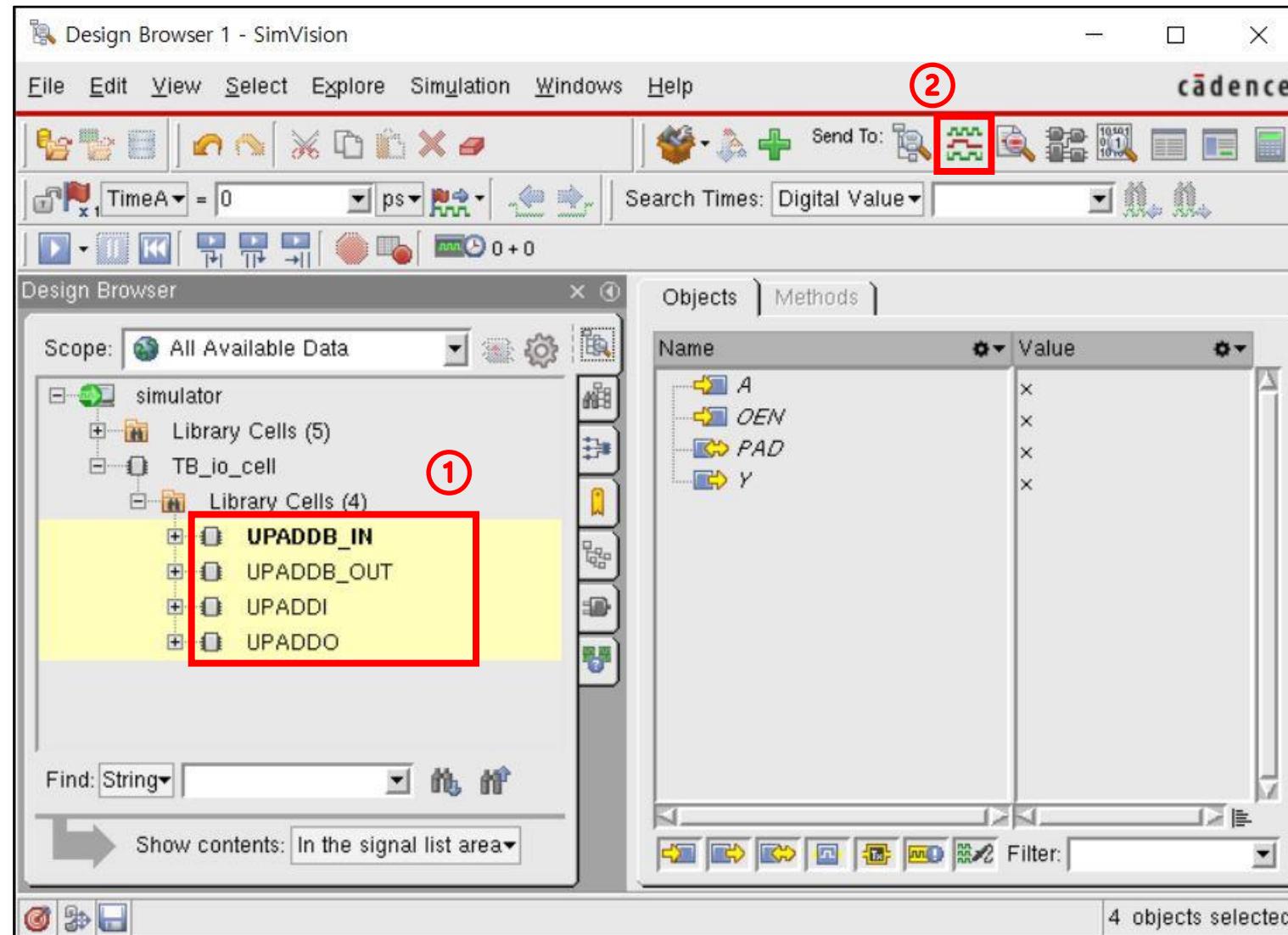
```
xrun -64bit \  
+max_err_count+50 \  
+define+function_sim \  
-access +rwc \  
-profile \  
-profthread \  
-gui \  
TB_io_cell.v \  
../../../../giolib045_v3.5/vlog/pads_FF_s1vg.v \  
-l ./func_sim.log
```

# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

### 1. xrun 실행 결과

- ① 모두 클릭
- ② 웨이브 아이콘 클릭

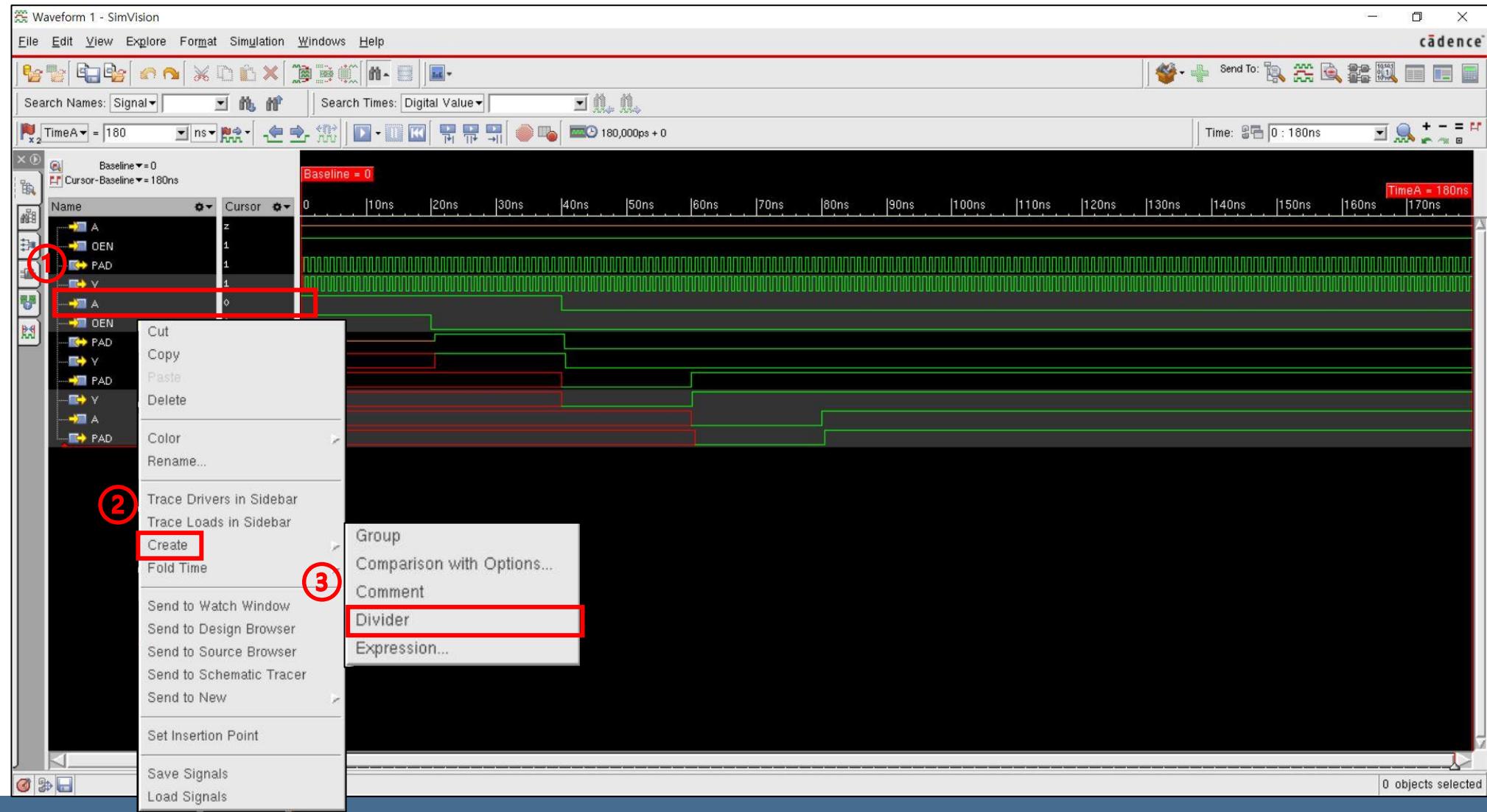


# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

### 1. xrun 실행 결과

- ①: A 클릭 후 우클릭
- ②: Create 클릭
- ③: Divider 클릭

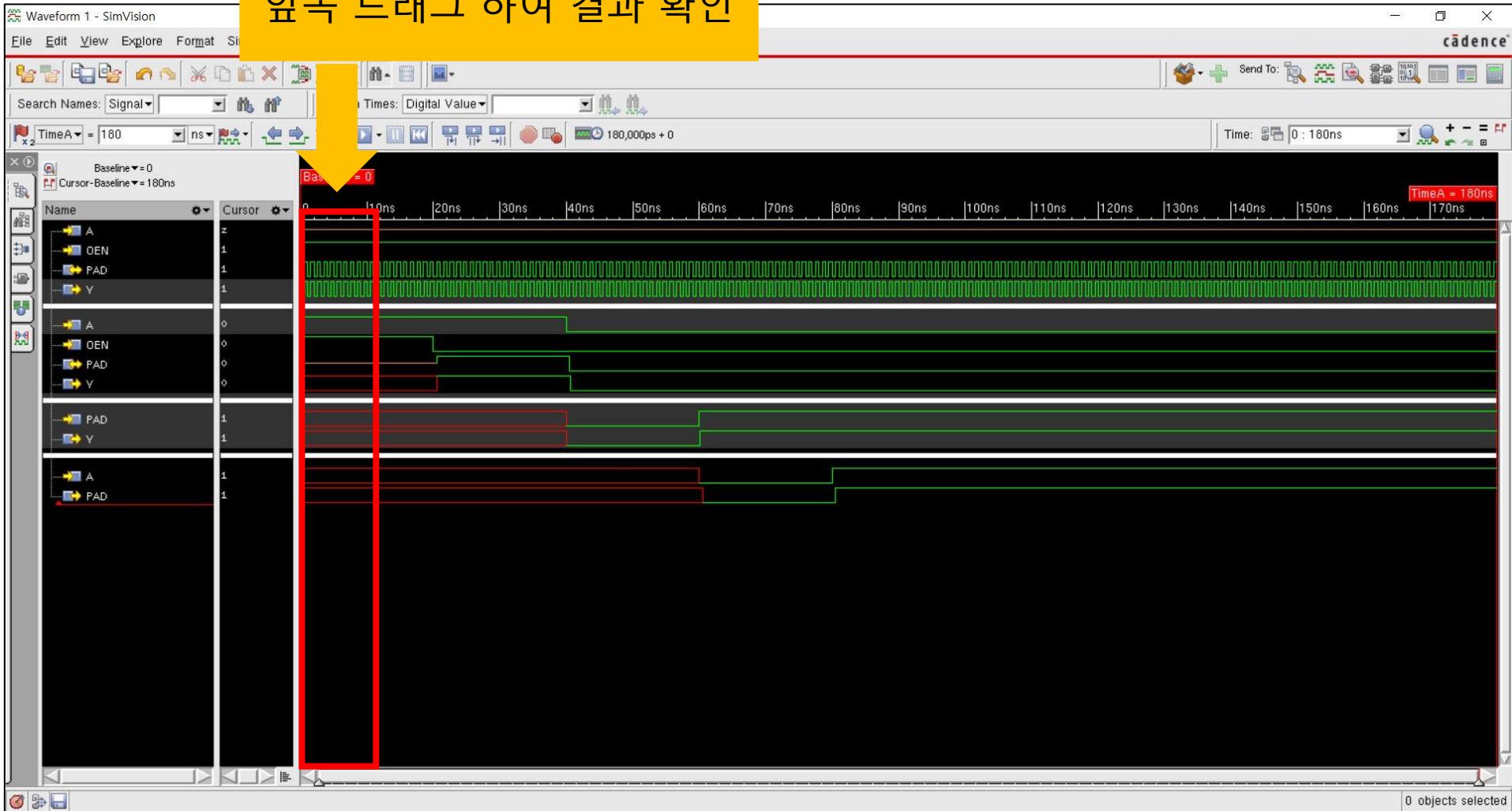


# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

앞쪽 드래그 하여 결과 확인

### 1. xrun 실행 결과

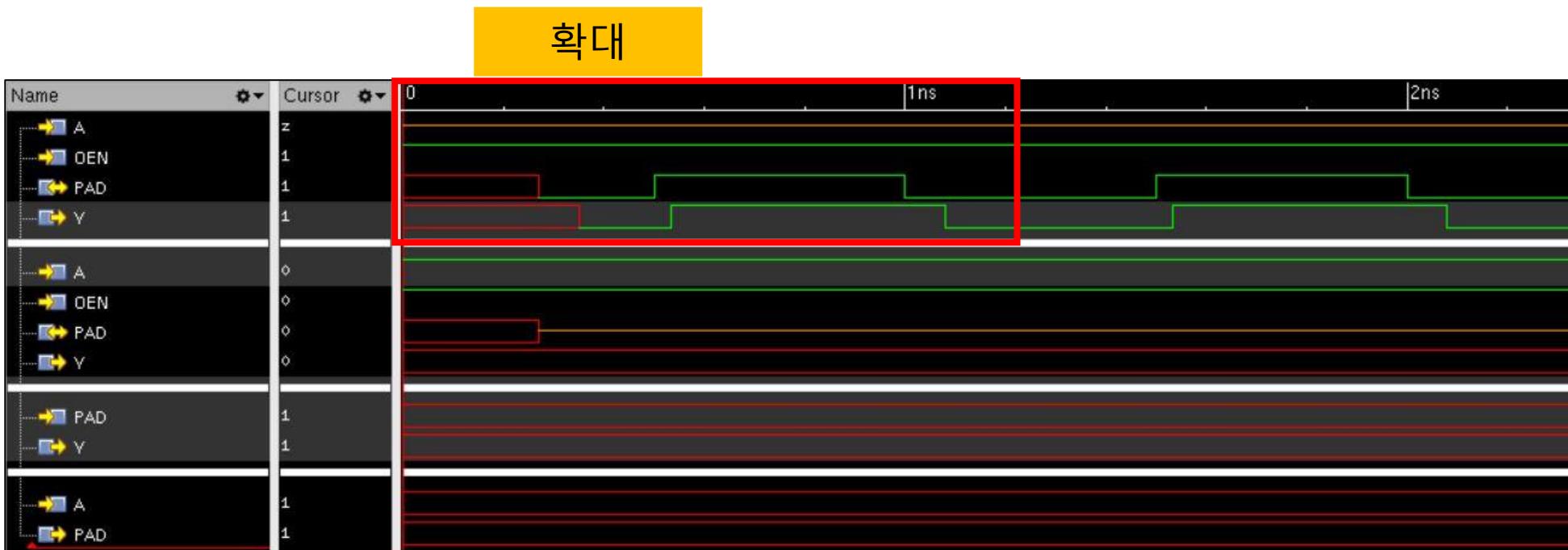


# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

### 1. xrun 실행 결과

- PADDB\_IN의 동작 확인

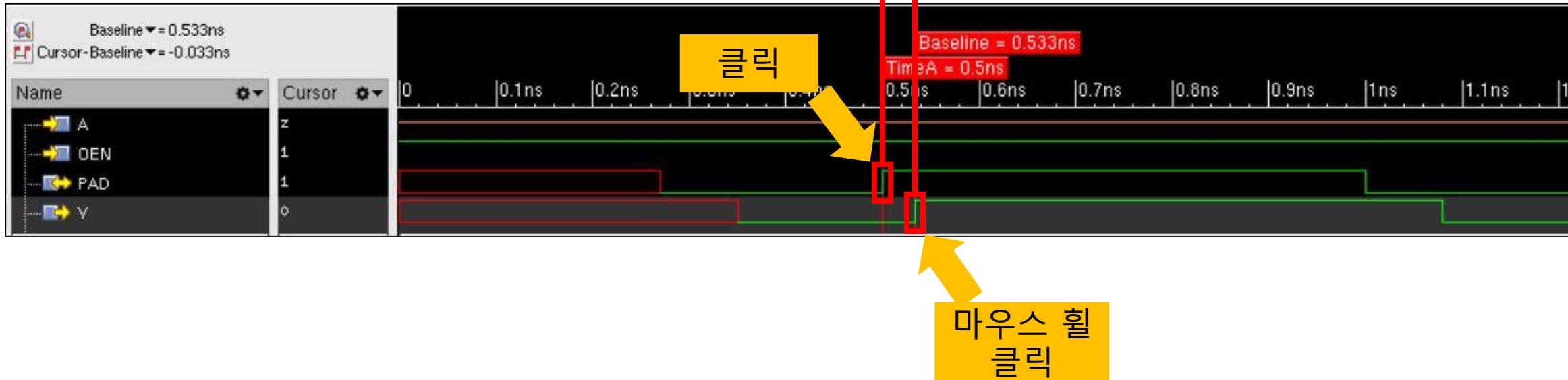


# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

### 1. PADDB\_IN 시뮬레이션

- Rising 신호

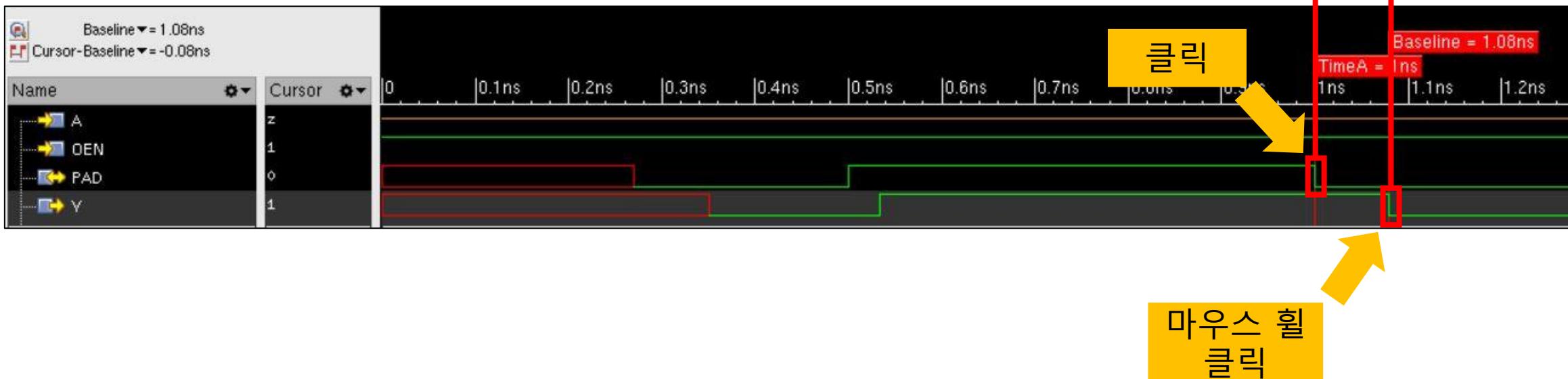


# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

### 1. PADDB\_IN 시뮬레이션

- Falling 신호



# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

### 1. PADDB\_IN 시뮬레이션 결과

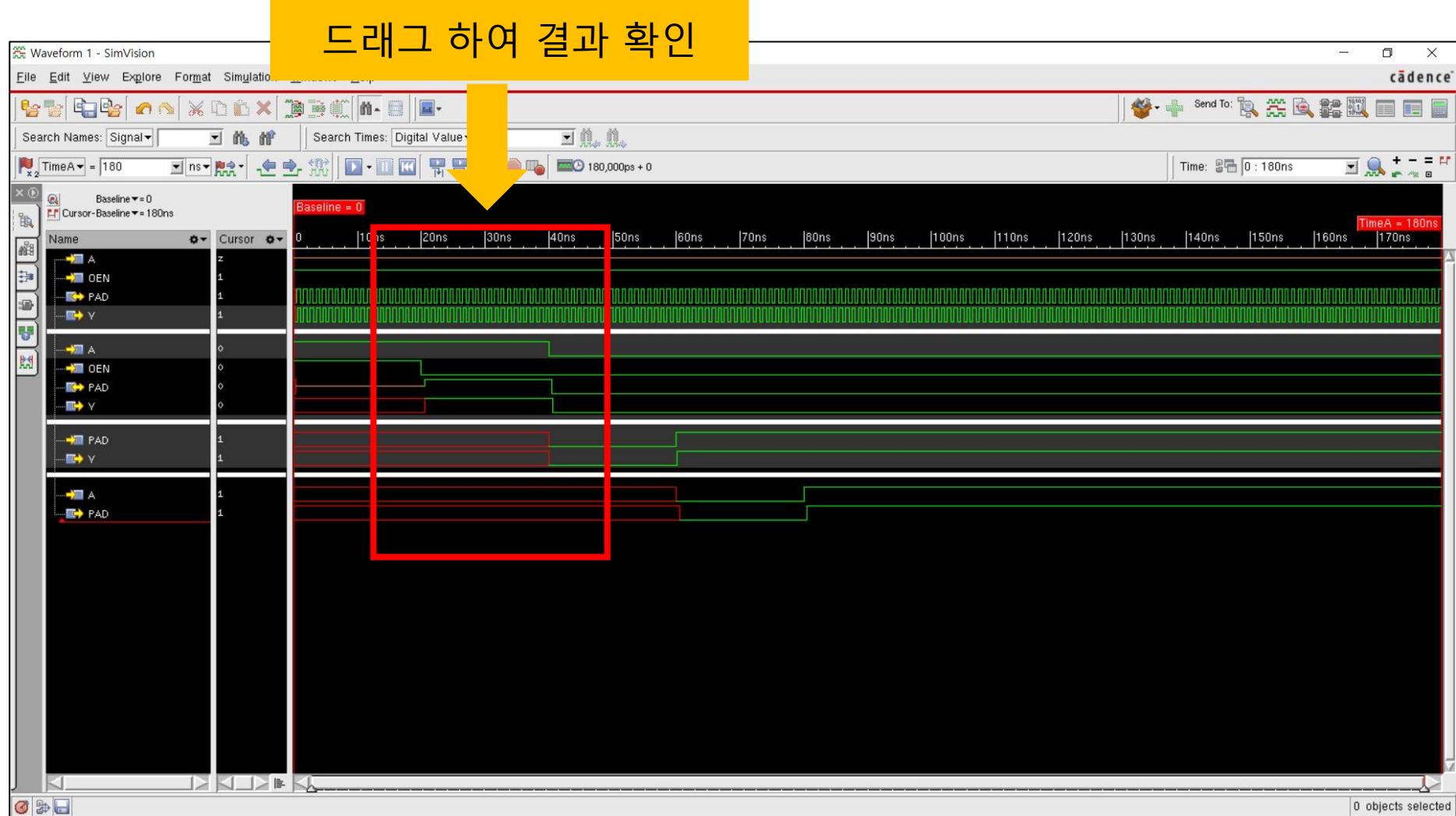
- IO셀의 Cell delay가 크지 않기 때문에 고속 신호(1GHz) 처리에 적합

# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

### 1. PADDB\_OUT 실행 결과

- OEN의 반응 시간



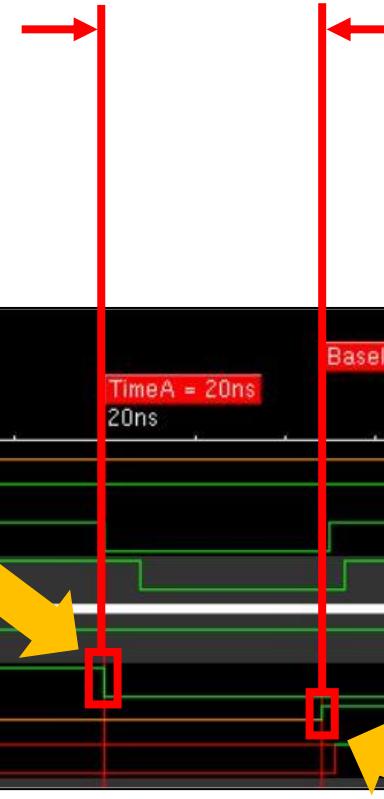
# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

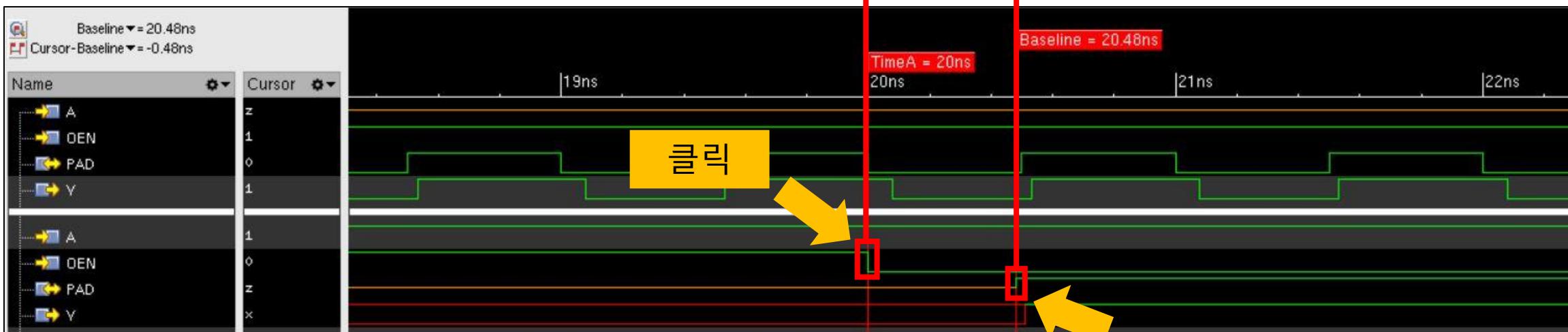
### 1. PADDB\_OUT 시뮬레이션

- Rising 신호

Cell delay  
0.48ns



마우스 훨  
클릭



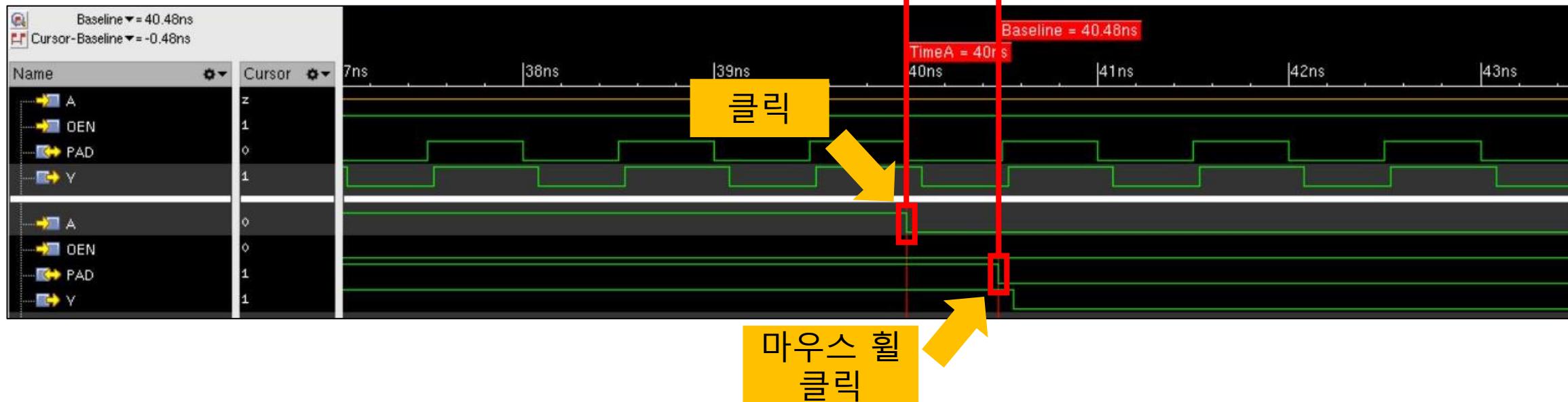
# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

### 1. PADDB\_OUT 시뮬레이션

- Falling 신호

Cell delay  
0.48ns



# 디지털 라이브러리 테스트

## LAB2 시뮬레이션

### 1. PADDB\_OUT 시뮬레이션

- IO셀의 Cell delay가 어느 정도 생기기 때문에 고속 신호(1GHz) 처리에 부적합
- 안정적으로 동작되는 가용 범위를 알아보기 위해 500MHz, 750MHz에서도 시뮬레이션 후 결과 확인 필요