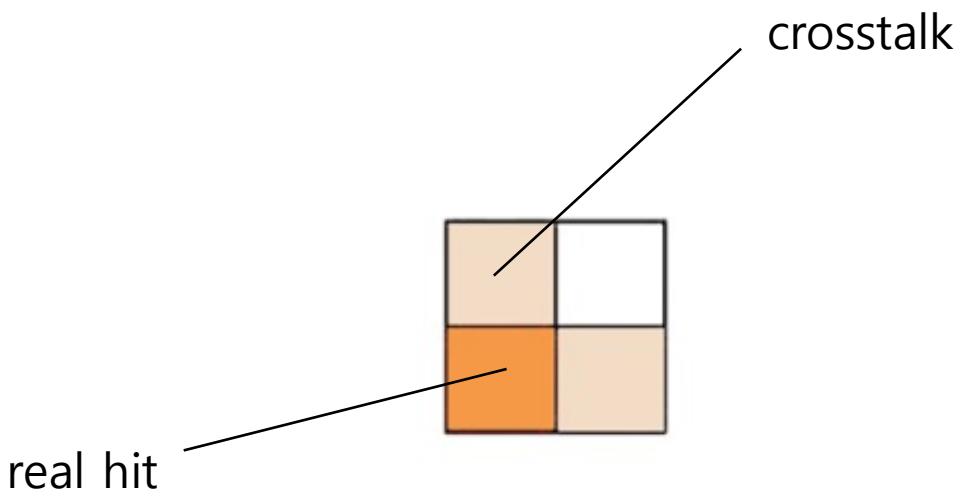
# DAQ of UBAT

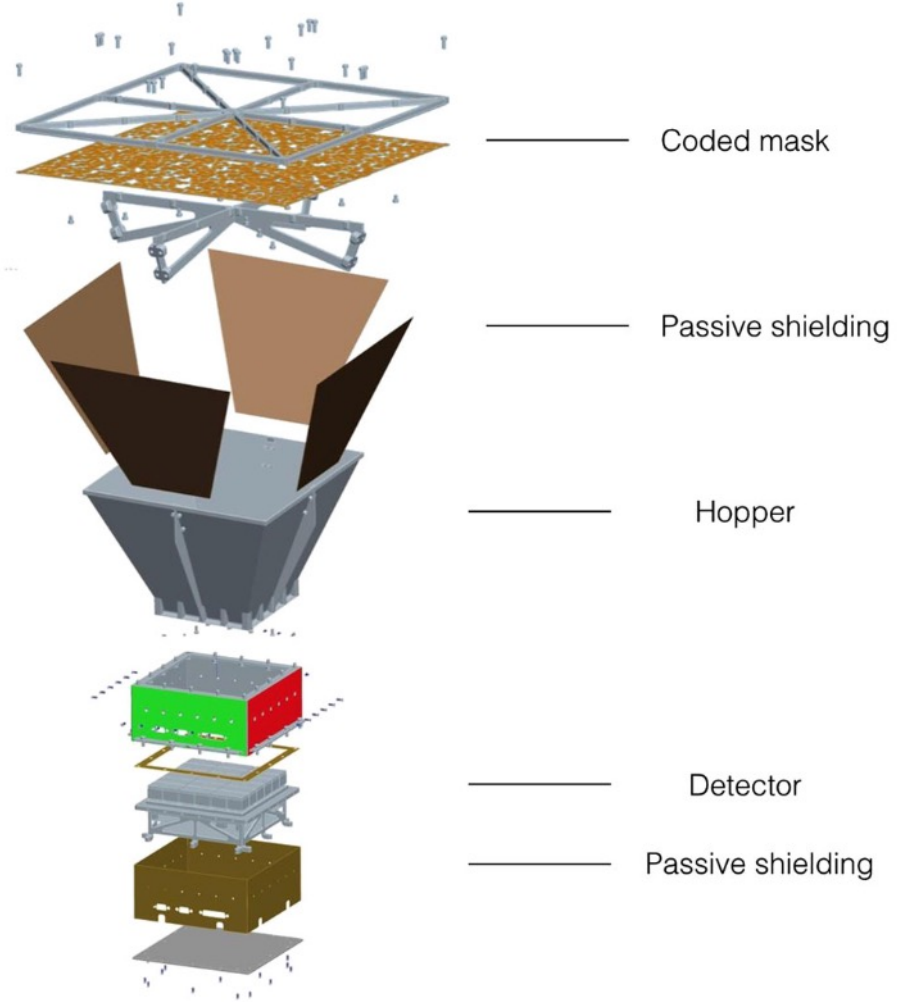
December 09, 2023

Minwoo Wang, Undergraduate, Korea University

# Ultimate Goal

- Determine whether the signal is a real hit or a crosstalk
- Find out the address of a real hit



# Study

- What is a crosstalk
- What are the criteria for distinguishing a real hit from a crosstalk
- FPGA and Verilog
- How the signal observed in the detector is transfered to the FPGA

# The Steps

1. Understanding the process of ‘hit finding algorithm’
2. Designing a hardware process
3. Coding the hardware process without delay in Verilog
4. SPACIROC : ‘spaciroc_datasheet_20110510’
5. Calibration for Photon Detector
6. Demonstrating on FPGA
7. Study BRAM
8. Input clock automatically
9. Optical sensor

10. DAQ process of MAPMT (Winter Vacation Plan)

➔ Find out the address of the *real hit*

# Paper Abstract

## 1. Block diagram of UBAT top level flow

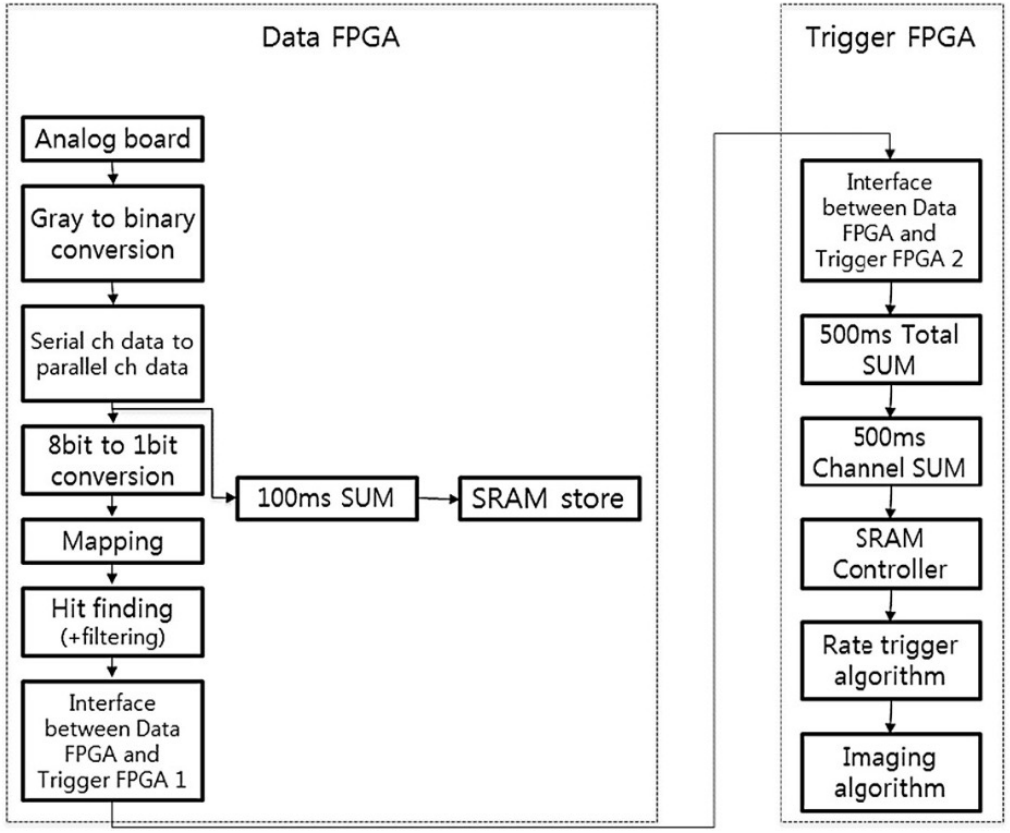



**Fig. 6** Block diagram of UBAT top level data flow. The data flow is divided between two directions: raw data is stored in SRAM as are the GRB trigger calculations

# Paper Abstract

## 2. Hit finding algorithm

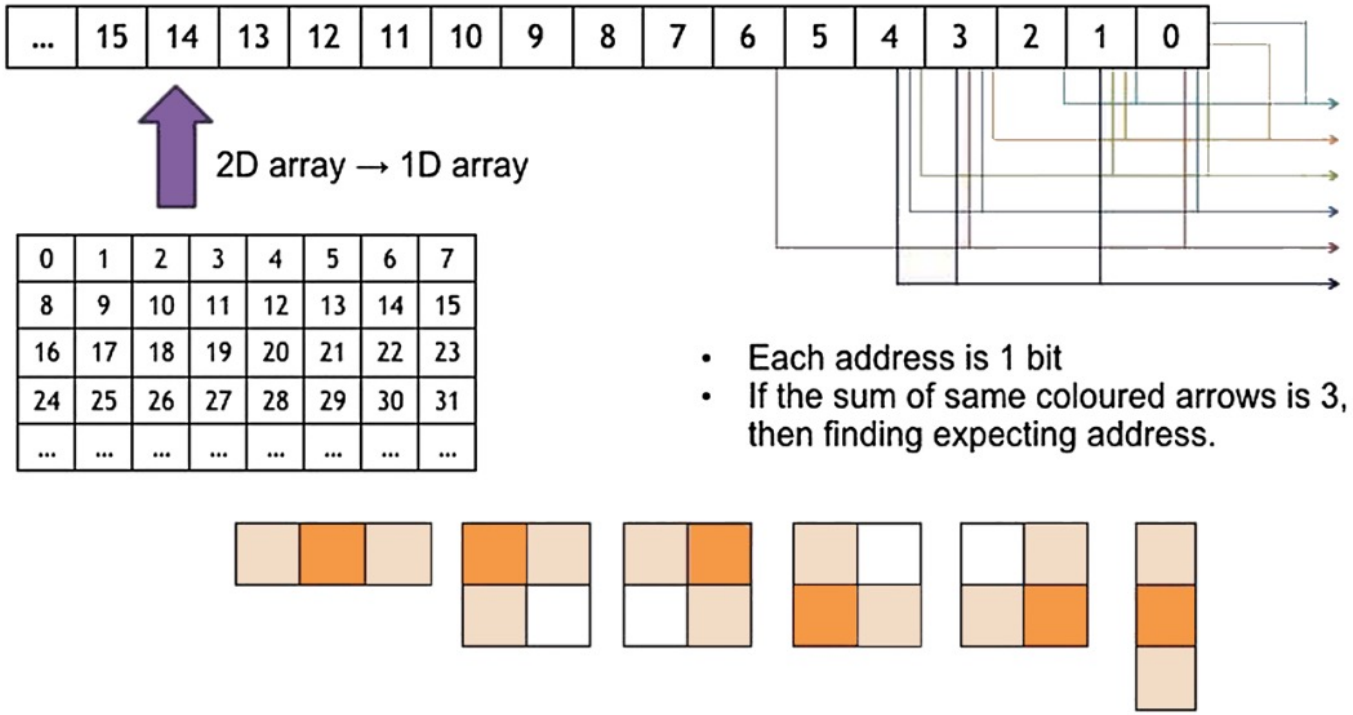



**Fig. 7** Block diagram of "hit finding" algorithm for 3-hit patterns. The *dark orange squares* represent real hits and *faint orange squares* represent hits resulting from crosstalk. Serial raw data are fed into hit finding blocks, and then the finding rulers, which are represented with different *colored arrows*, determine the center of the hit pattern

# What's the Problem?

*faint orange* (crosstalk)

➔it is difficult to find out the address of *real hit*

Since detector recognize the *faint orange* as *the real hit*

# Goal

Find out the address of the *real hit*

# The Steps

1. Understanding the process of ‘hit finding algorithm’
2. Designing a hardware process

➔ Find out the address of the *real hit*

# The Detailed Progress

1. Understanding the process of ‘hit finding algorithm’

- How the lights be detected
- What the Crosstalk is
- Mapping in ADC
- 8 bits per each pixel (= channel)
- What does 8 bits mean : intensity of the detected light
- Convert serial 8 bits to parallel 8 bits for quick calculation
- Classify according to the threshold

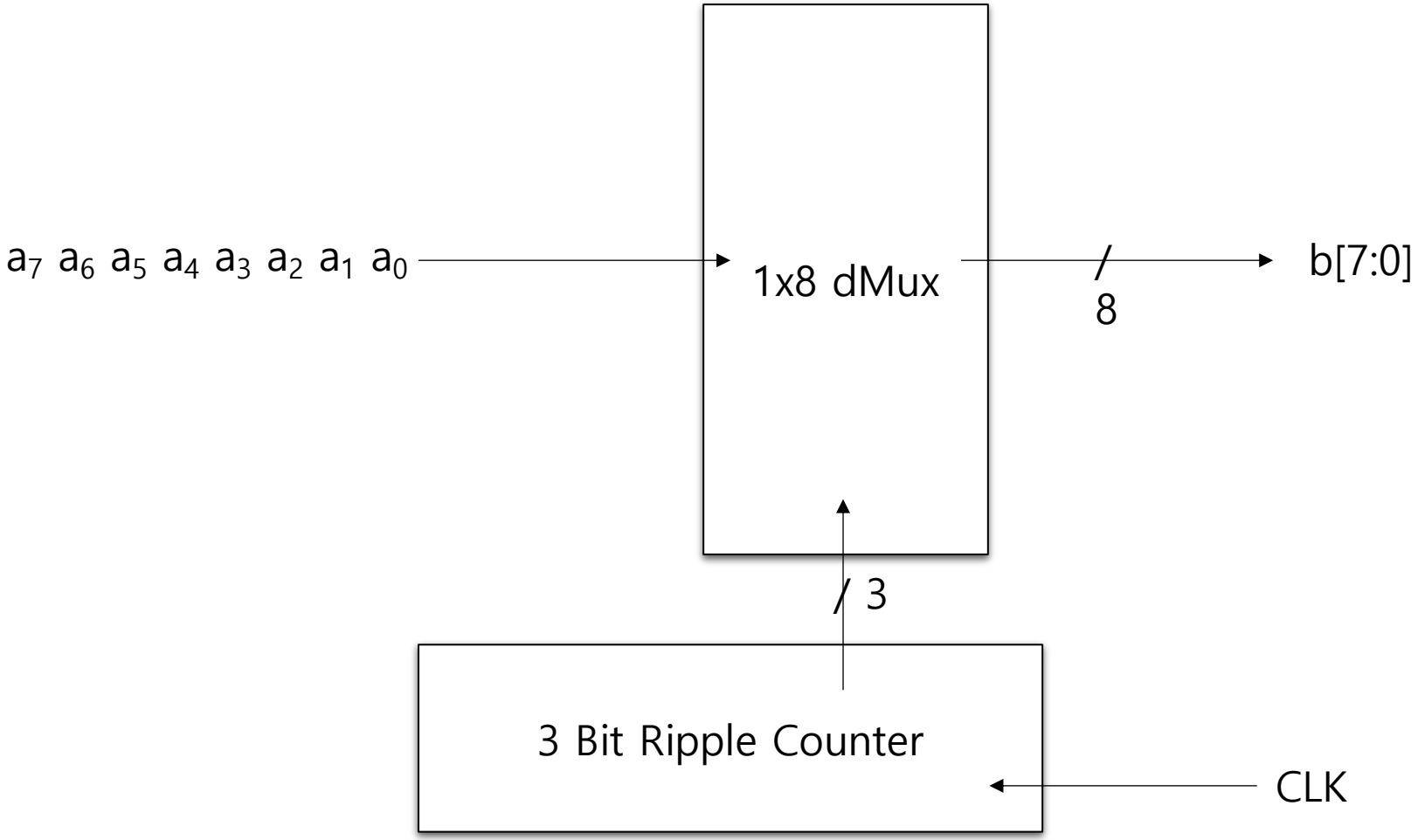
# The Detailed Progress

2. Designing a hardware process

2-1) convert serial 8 bits to parallel 8 bits
2-2) classify according the threshold

# The Detailed Progress

## 2-1) convert serial 8 bits to parallel 8 bits (SPC)



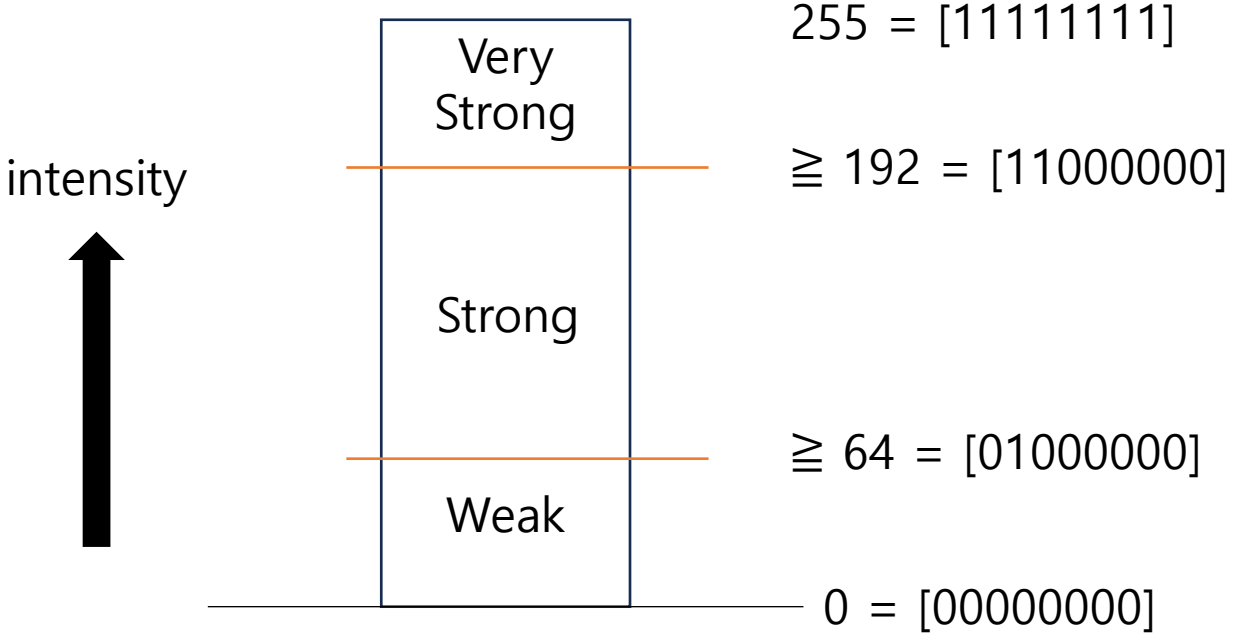
# The Detailed Progress

## 2-2) classify according the threshold

Input : the output of SPC (Serial Parallel Converter ) = 8 bits

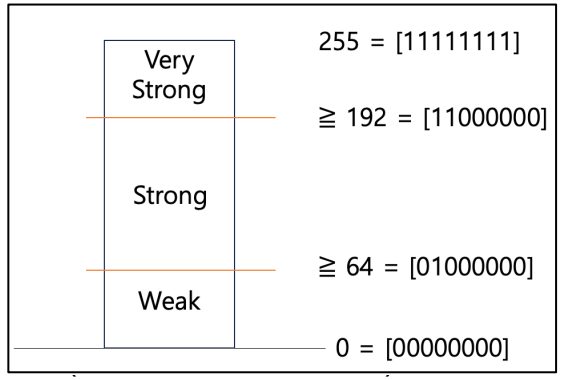
Output : 2 bits = very strong[11] / strong[10] / weak[01] / none[00]

The Threshold :

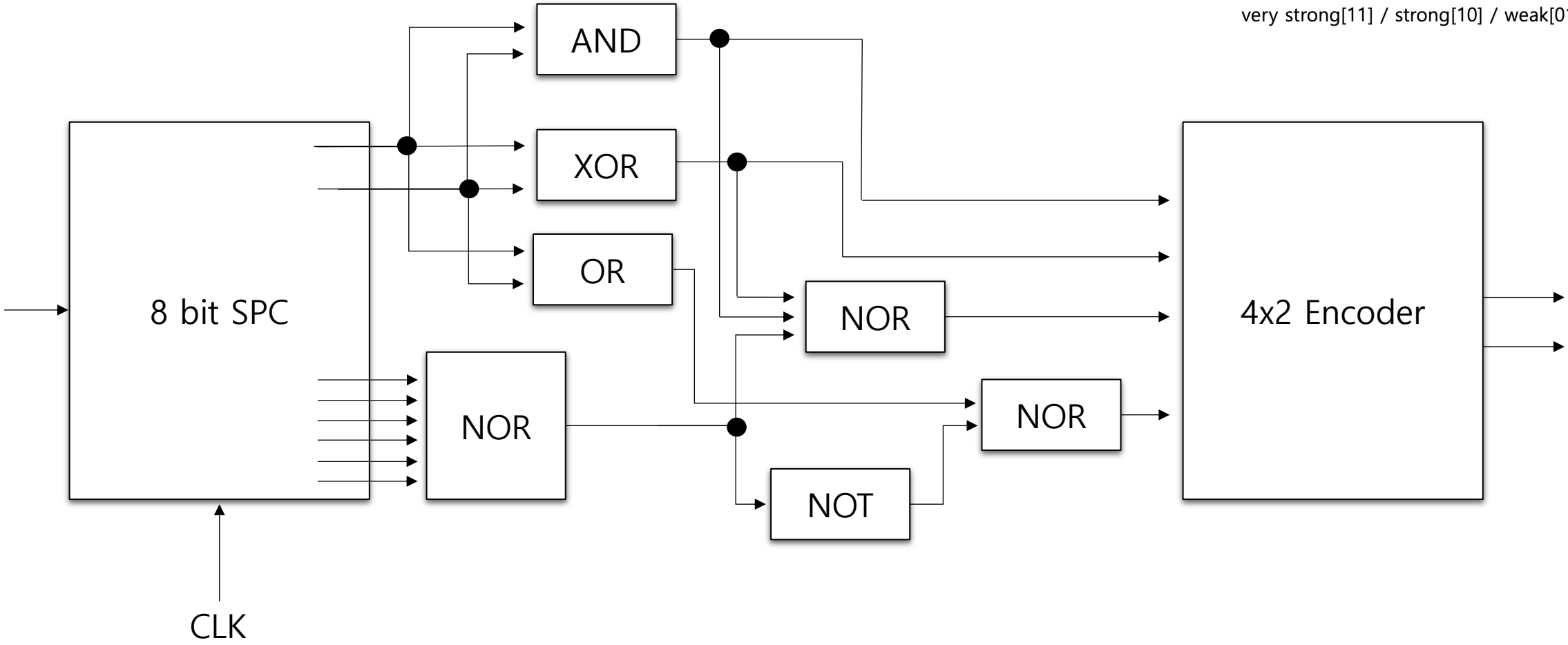


# The Detailed Progress

## 2-2) classify according the threshold




very strong[11] / strong[10] / weak[01] / none[00]



# Feedback

- Transistor의 개수 → 회로의 delay와 용량
- **Photon의 개수 → threshold 분류**
- **외부와의 통신 → 외부에서 serial로 넣어주고 output보여주기**

* Verilog simulation을 통해 delay 계산하기

# The Steps

1. ~~Understanding the process of 'hit finding algorithm'~~
2. ~~Designing a hardware process~~
3. Coding the hardware process without delay in Verilog

➔ Find out the address of the *real hit*

# The Detailed Progress

3. Coding the hardware process with Verilog

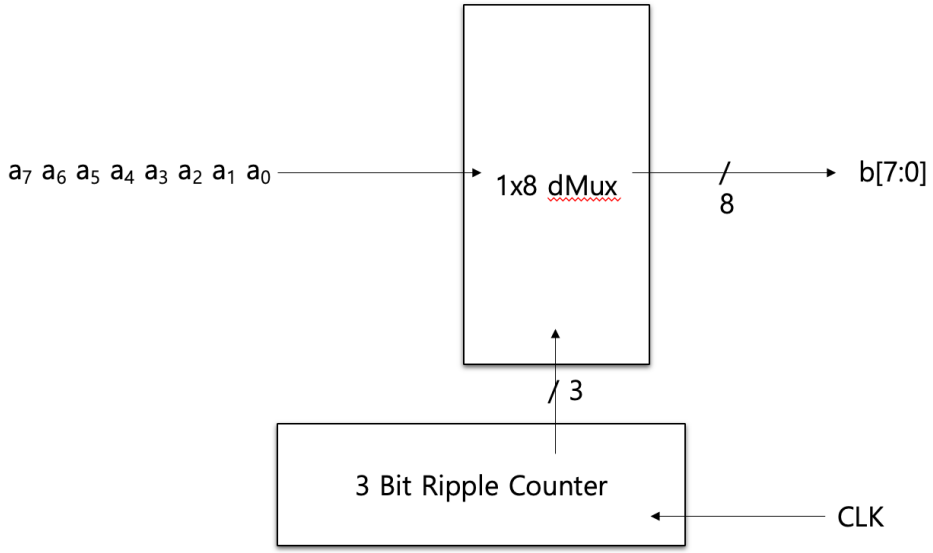
3-0) basic modules

3-1) SPC : 8bit serial parallel convertor

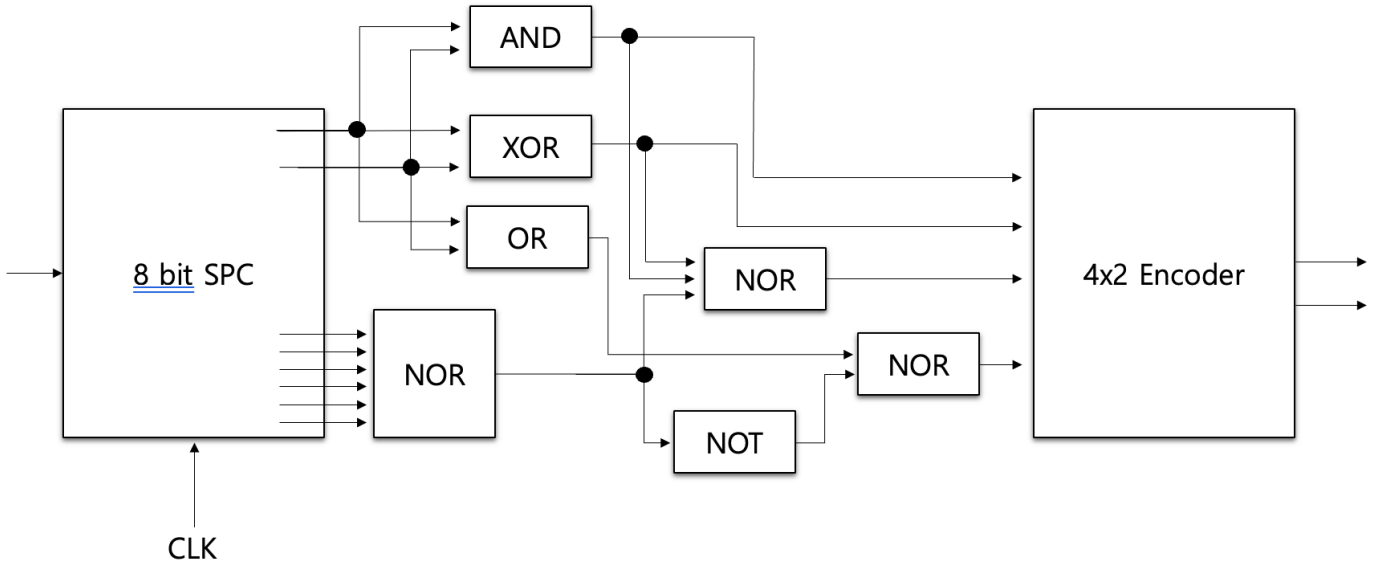
3-2) classify according the threshold

# The Hardware Process (in 2nd)

## 2-1) convert serial 8 bits to parallel 8 bits (SPC)




## 2-2) classify according the threshold



# 3-0) basic modules

Combinational Circuit : 8x1 dMux, 4x2 Encoder

Sequential Circuit : JK FF, 3bit ripple counter, ***T FF***

# 3-0) 8x1 dMux : dMux8

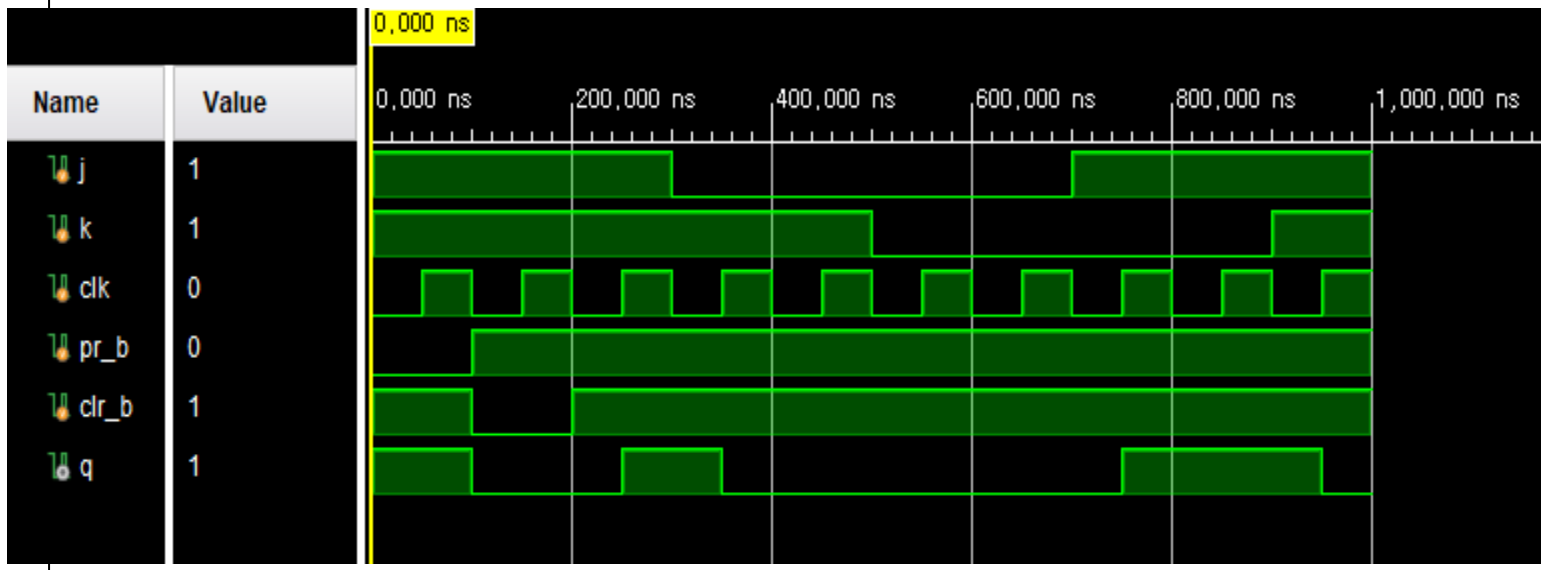
```
module dMux8(
input in,
output reg [7:0] out,
input [2:0]s
);

always @(in, s)
    begin
        case(s)
            3'b000 : begin out[7] = 0 ; out[6] = 0 ; out[5] = 0 ; out[4] = 0 ; out[3] = 0 ; out[2] = 0 ; out[1] = 0 ; out[0] = in ; end
            3'b001 : begin out[7] = 0 ; out[6] = 0 ; out[5] = 0 ; out[4] = 0 ; out[3] = 0 ; out[2] = 0 ; out[1] = in ; out[0] = 0 ; end
            3'b010 : begin out[7] = 0 ; out[6] = 0 ; out[5] = 0 ; out[4] = 0 ; out[3] = 0 ; out[2] = in ; out[1] = 0 ; out[0] = 0 ; end
            3'b011 : begin out[7] = 0 ; out[6] = 0 ; out[5] = 0 ; out[4] = 0 ; out[3] = in ; out[2] = 0 ; out[1] = 0 ; out[0] = 0 ; end
            3'b100 : begin out[7] = 0 ; out[6] = 0 ; out[5] = 0 ; out[4] = in ; out[3] = 0 ; out[2] = 0 ; out[1] = 0 ; out[0] = 0 ; end
            3'b101 : begin out[7] = 0 ; out[6] = 0 ; out[5] = in ; out[4] = 0 ; out[3] = 0 ; out[2] = 0 ; out[1] = 0 ; out[0] = 0 ; end
            3'b110 : begin out[7] = 0 ; out[6] = in ; out[5] = 0 ; out[4] = 0 ; out[3] = 0 ; out[2] = 0 ; out[1] = 0 ; out[0] = 0 ; end
            3'b111 : begin out[7] = in ; out[6] = 0 ; out[5] = 0 ; out[4] = 0 ; out[3] = 0 ; out[2] = 0 ; out[1] = 0 ; out[0] = 0 ; end
        endcase
    end

endmodule
```

# 3-0) 4x2 Encoder : Encoder4x2

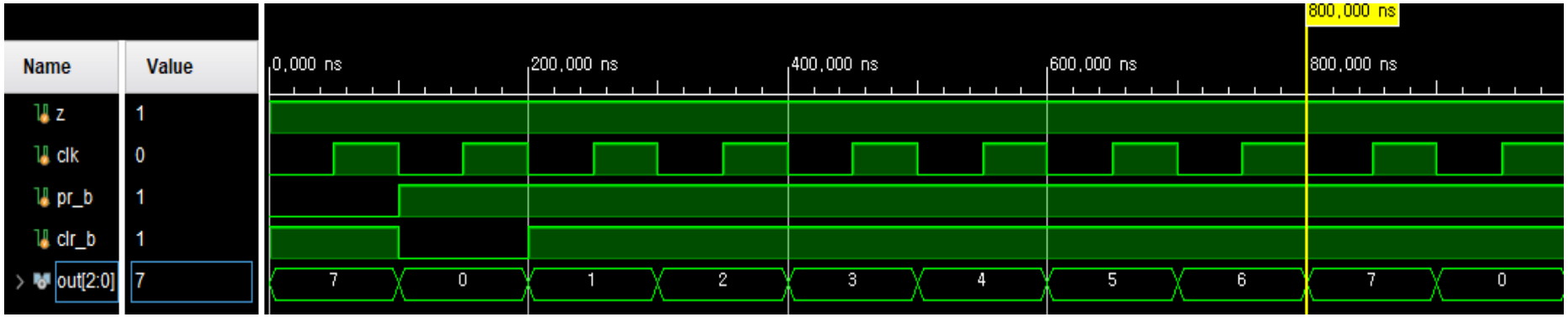
```
module Encoder4x2(in, out);

input [3:0]in;
output reg [1:0]out;

always @(in)
    begin
        case(in)
            4'b1000 : begin out[1] = 1; out[0] =1; end
            4'b0100 : begin out[1] = 1; out[0] =0; end
            4'b0010 : begin out[1] = 0; out[0] =1; end
            4'b0001 : begin out[1] = 0; out[0] =0; end
        endcase
    end

endmodule
```

# 3-0) JK FF : JKFF

```
module JKFF(
    input j,
    input k,
    input clk,
    input pr_b,
    input clr_b,
    output reg q
    );

always @ (negedge clk or negedge pr_b or negedge clr_b)
begin
    if (pr_b ==0)   q <= 1;
    else if (clr_b ==0)      q <=0;
    else
        begin
            if (j==0 && k==0)       q <=q;
            else if (j==0 && k==1)       q <=0;
            else if (j==1 && k==0)       q <=1;
            else if (j==1 && k==1)       q <=~q;
        end
end
```

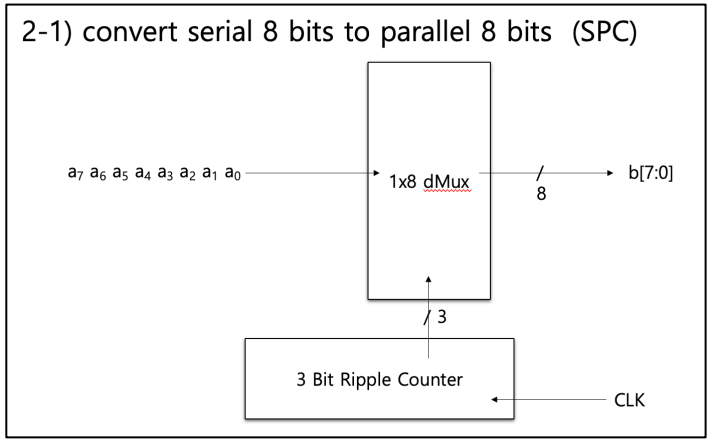


# 3-0) 3bit ripple counter : ripple_counter8




```
module ripple_counter3(
input clk,
input z, // FF의 J&K에 1 넣어줌
input pr_b,
input clr_b,
output [2:0]out
);

JKFF jk0(
.j(z),
.k(z),
.clk(clk),
.pr_b(pr_b),
.clr_b(clr_b),
.q(out[0])
);

JKFF jk1(
.j(z),
.k(z),
.clk(out[0]),
.pr_b(pr_b),
.clr_b(clr_b),
.q(out[1])
);

JKFF jk2(
.j(z),
.k(z),
.clk(out[1]),
.pr_b(pr_b),
.clr_b(clr_b),
.q(out[2])
);

endmodule
```

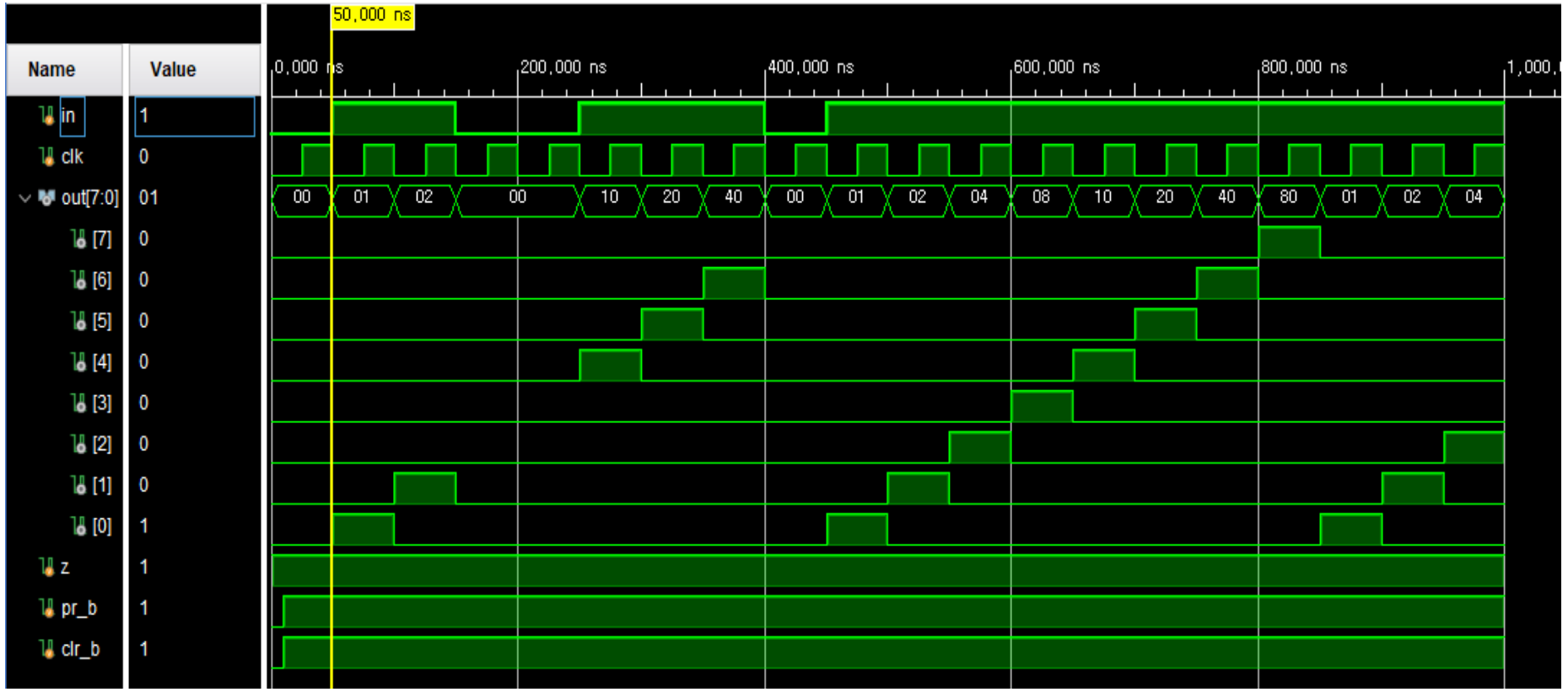
# 3-1) SPC : 8bit serial parallel convertor




```
module SPC(

input in,
input clk,
output [7:0]out,

input z,
input pr_b,
input clr_b
);

wire [2:0]s;

ripple_counter3 RC3(
.clk(clk),
.z(z),
.pr_b(pr_b),
.clr_b(clr_b),
.out(s)
);

dMux8 DMux(
.in(in),
.out(out),
.s(s)
);

endmodule
```

**Serial input 8bit :**
**11001110 (line 44~51) ~400ns**

400ns ~ : 8bit dMux 작동확인



# 3-1) Feature of SPC

8bit serial input should be equal to 8 times period of the clk

Since the output of the dmux changes every period of the clk

→ Memory is required to store the output (TFF)

T FF : 0 = no change / 1 = toggle

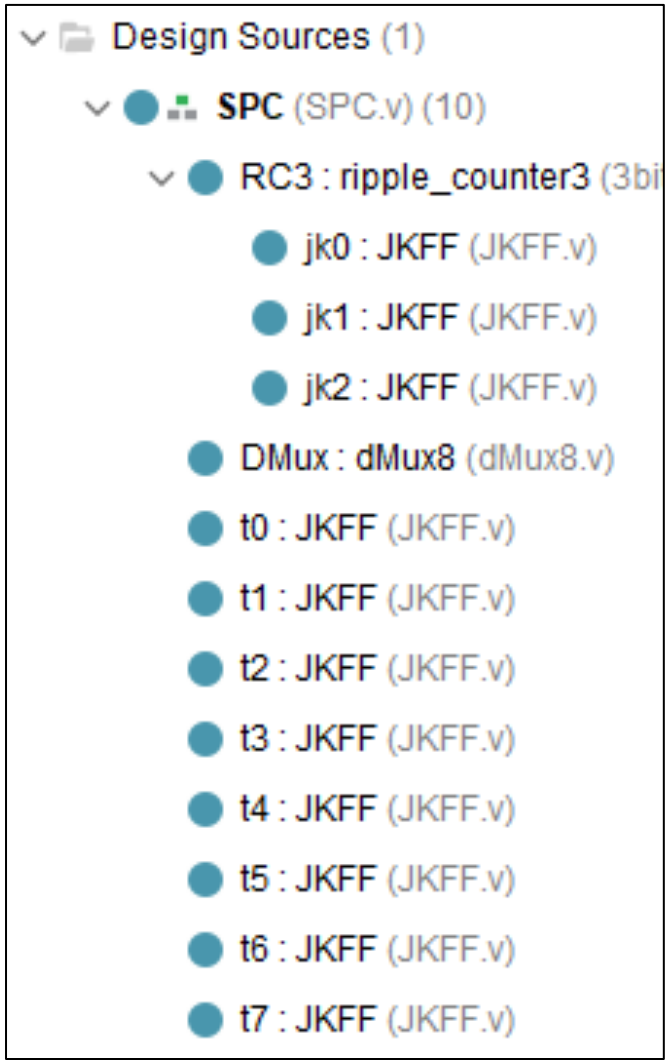
> Since the initial output is zero &

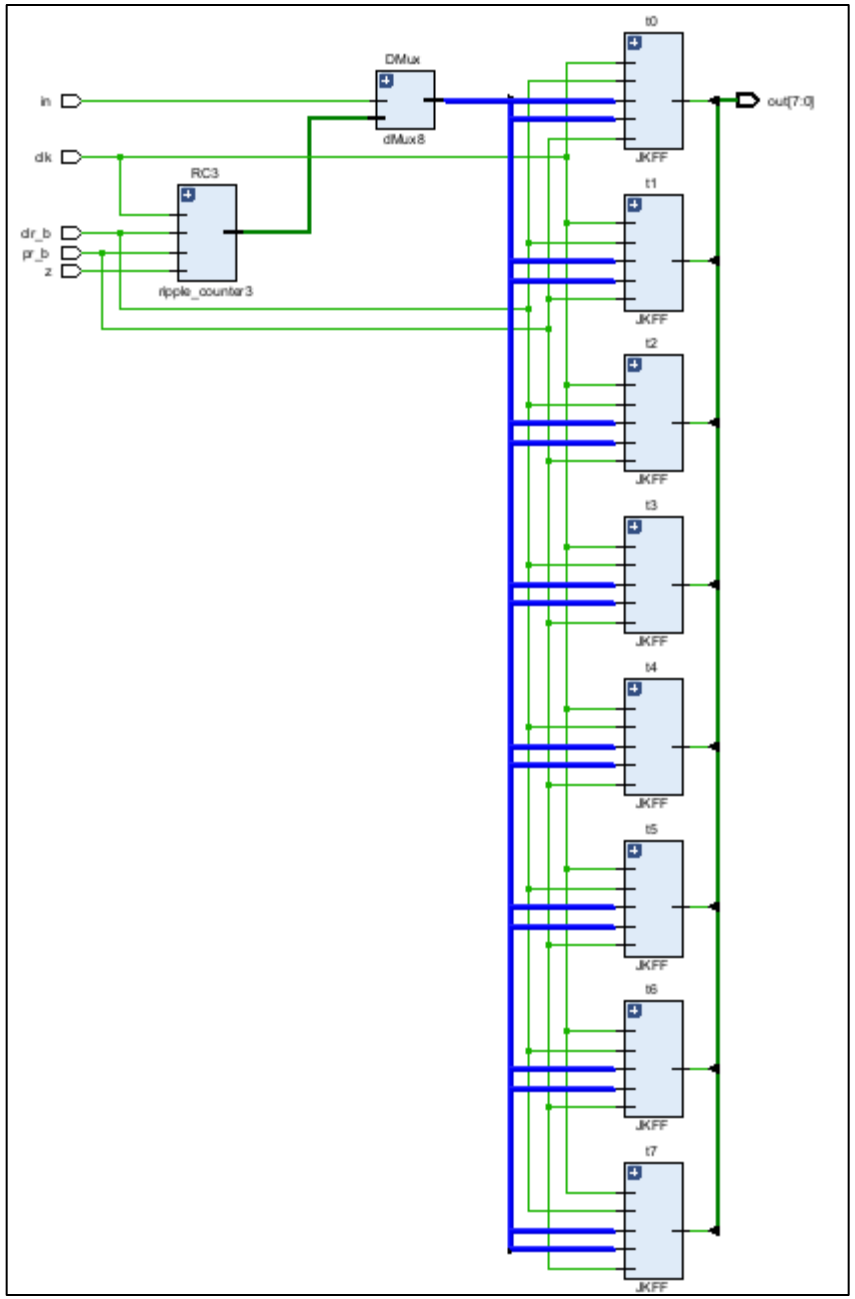
**the input 1 information must be stored**

# 3-1) SPC with T FF

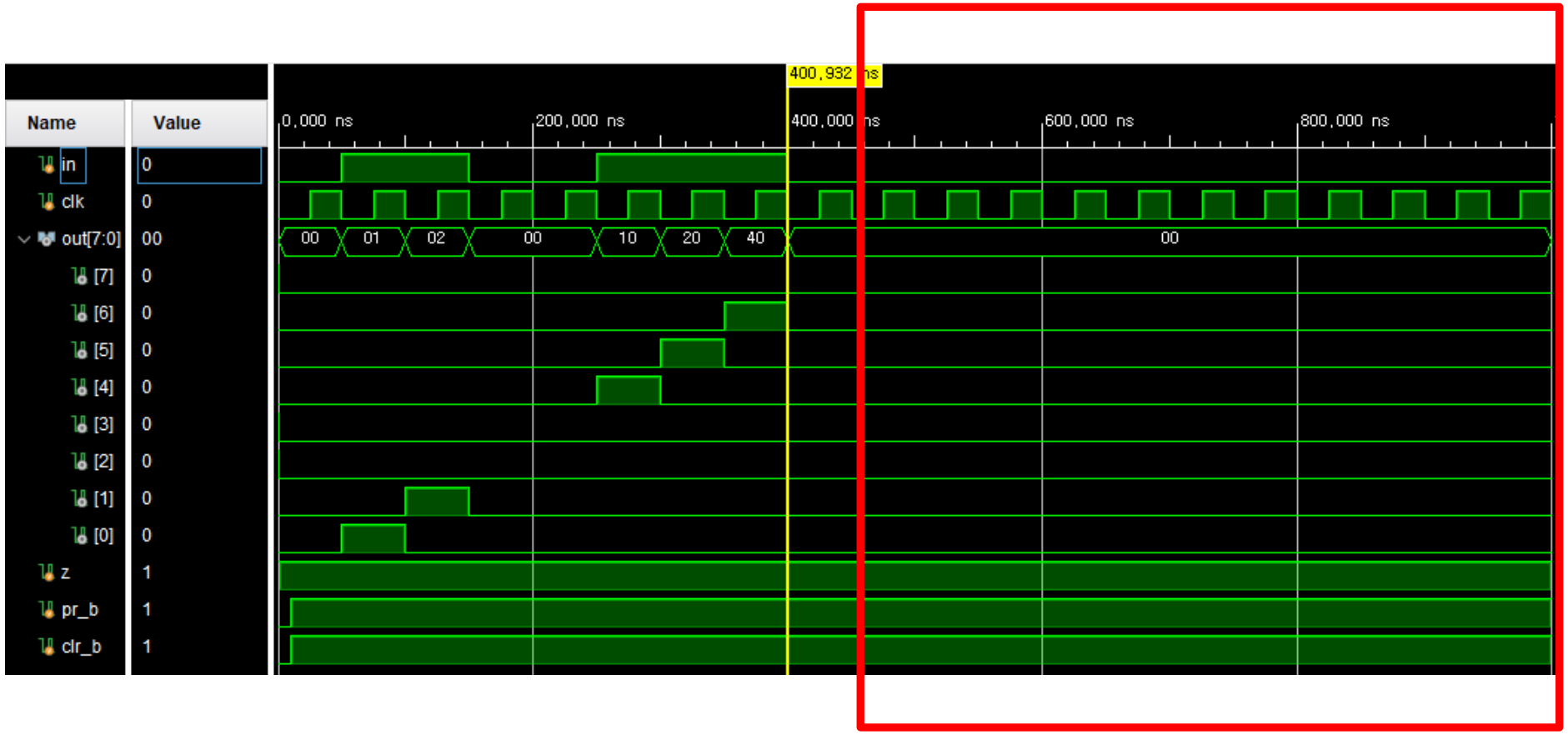
```
module SPC(

input in,
input clk,
output [7:0]out,

input z,
input pr_b,
input clr_b
);

wire [2:0]s; // select bit : output of ripple counter
wire [7:0]o; // output of dmux
```

```
JKFF t0(
.j(o[0]),
.k(o[0]),
.clk(clk),
.pr_b(pr_b),
.clr_b(clr_b),
.q(out[0])
);
```



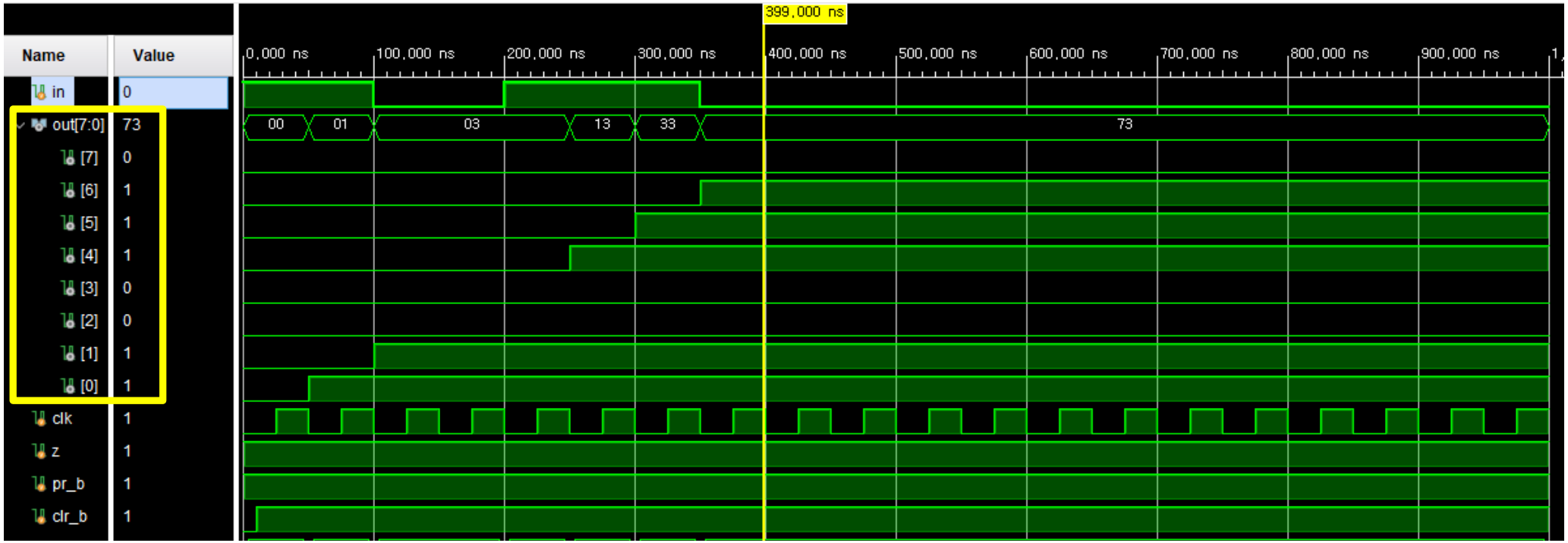
# 3-1) Simulation without T FF




Input : $11001110_{(2)}$ => intensity : 01110011

$a_0$ ~$a_7$ $a_7$ ~$a_0$

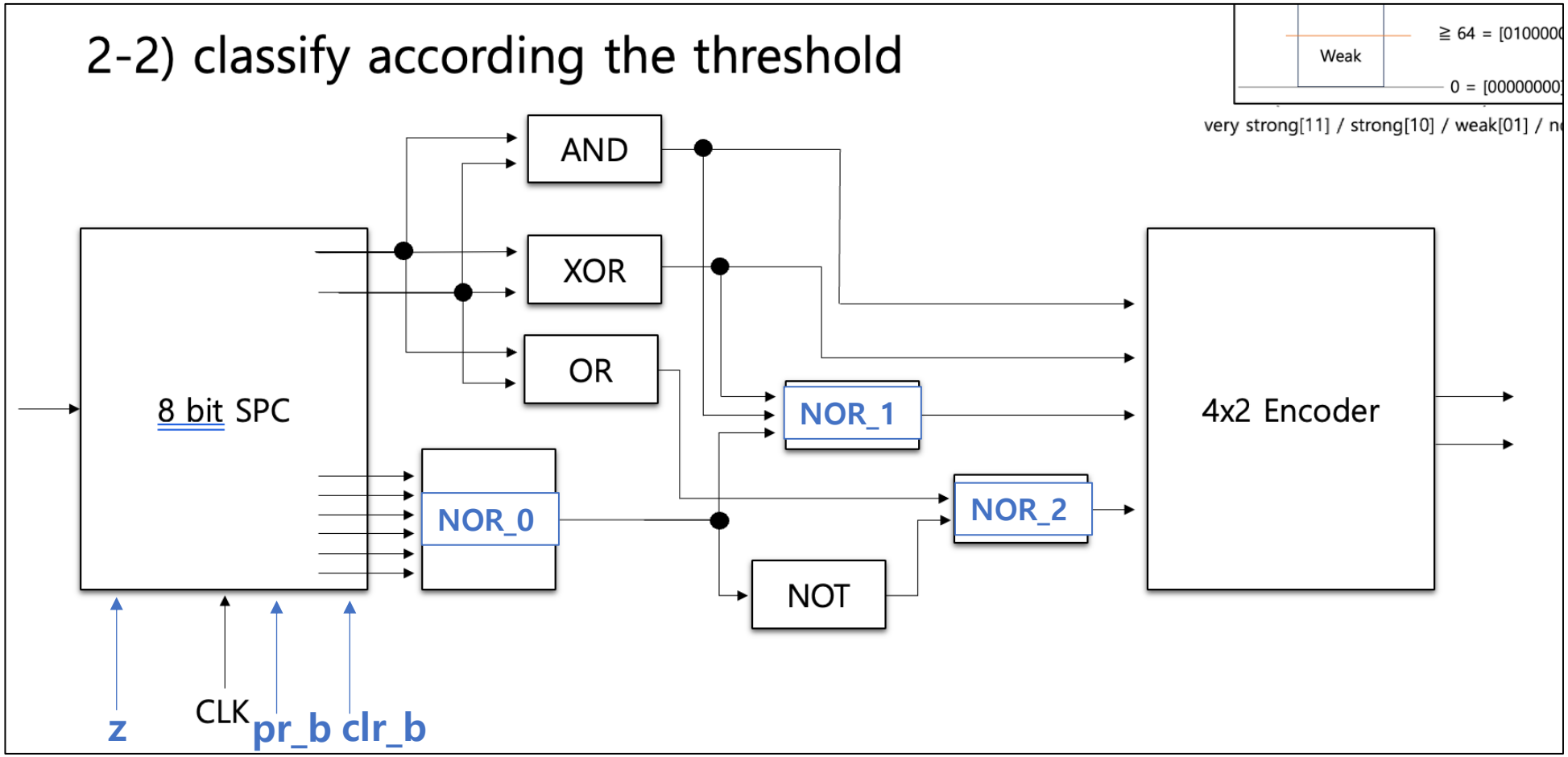
# 3-1) Simulation with T FF



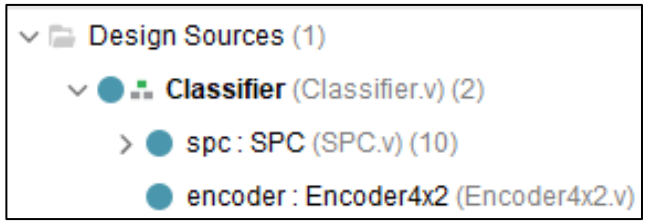

Input : $11001110_{(2)}$ => intensity : 01110011

$a_0$ ~$a_7$ $a_7$ ~$a_0$

# 3-2) Classifier



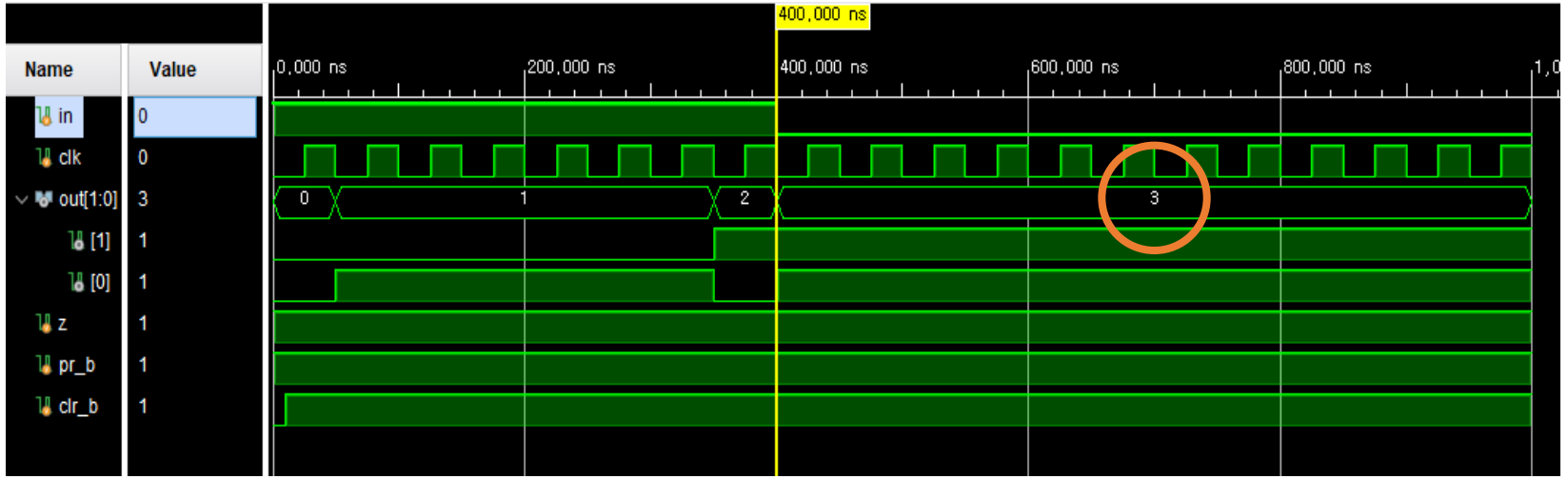
# 3-2) Classifier




```
module Classifier(in, clk, out, z, pr_b, clr_b);

input in;
input clk;
output [1:0]out;

input z;
input pr_b;
input clr_b;

wire [7:0]w;
wire AND, XOR, OR, NOR_0, NOR_1, NOR_2, NOT;
wire [3:0]i;

SPC spc(
.in(in),
.clk(clk),
.out(w),
.z(z),
.pr_b(pr_b),
.clr_b(clr_b)
);

assign AND = w[7] & w[6];
assign XOR = w[7] ^ w[6];
assign OR = w[7] | w[6];
assign NOR_0 = ~( w[5] | w[4] | w[3] | w[2] | w[1] | w[0] );

assign NOR_1 = ~( AND | XOR | NOR_0 );
assign NOT = ~ NOR_0;
assign NOR_2 = ~( OR | NOT );

assign i[3] = AND;
assign i[2] = XOR;
assign i[1] = NOR_1;
assign i[0] = NOR_2;

Encoder4x2 encoder(i,out);

endmodule
```

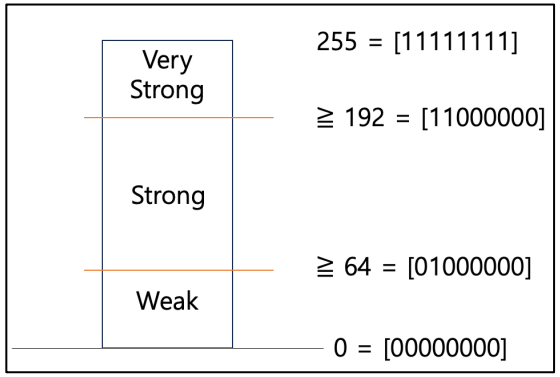
# 3-2) Classifier : Very Strong

```
module Classifier_tb;

reg in;
reg clk;
wire [1:0]out;

reg z, pr_b, clr_b;

Classifier inst(in, clk, out, z, pr_b, clr_b);

initial  // input : 11111111  <-> intensity : 11111111
    begin
    pr_b = 1; clr_b = 0; z=1; clk =0; in=1;
    #10 clr_b = 1;
    #40 in = 1;
    #50 in = 1;
    #50 in = 1;
    #50 in = 1;
    #50 in = 1;
    #50 in = 1;
    #50 in = 1;
    #50 in = 0; // check for dmux8
    end

always #25 clk = ~clk;


endmodule
```



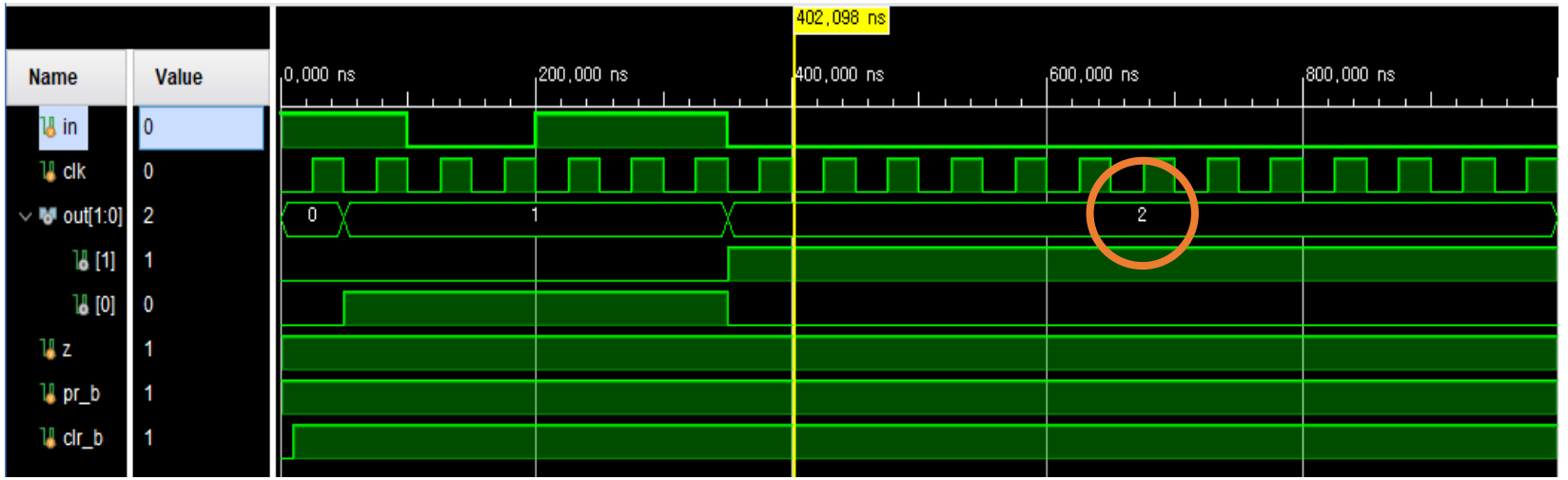

Input : $11111111_{(2)}$ => intensity : 11111111

$a_0 \sim a_7$ $a_7 \sim a_0$



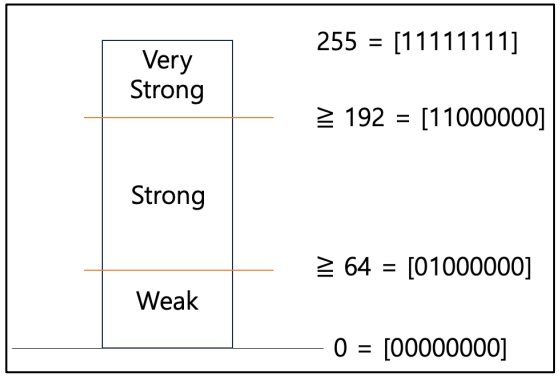
# 3-2) Classifier : Strong

```
module Classifier_tb;

reg in;
reg clk;
wire [1:0]out;

reg z, pr_b, clr_b;

Classifier inst(in, clk, out, z, pr_b, clr_b);

initial  // input : 11001110 <-> intensity : 01110011
    begin
    pr_b = 1; clr_b = 0; z=1; clk =0; in=1;
    #10 clr_b = 1;
    #40 in = 1;
    #50 in = 0;
    #50 in = 0;
    #50 in = 1;
    #50 in = 1;
    #50 in = 1;
    #50 in = 0;
    #50 in = 0; // check for dmux8
    end

always #25 clk = ~clk;


endmodule
```



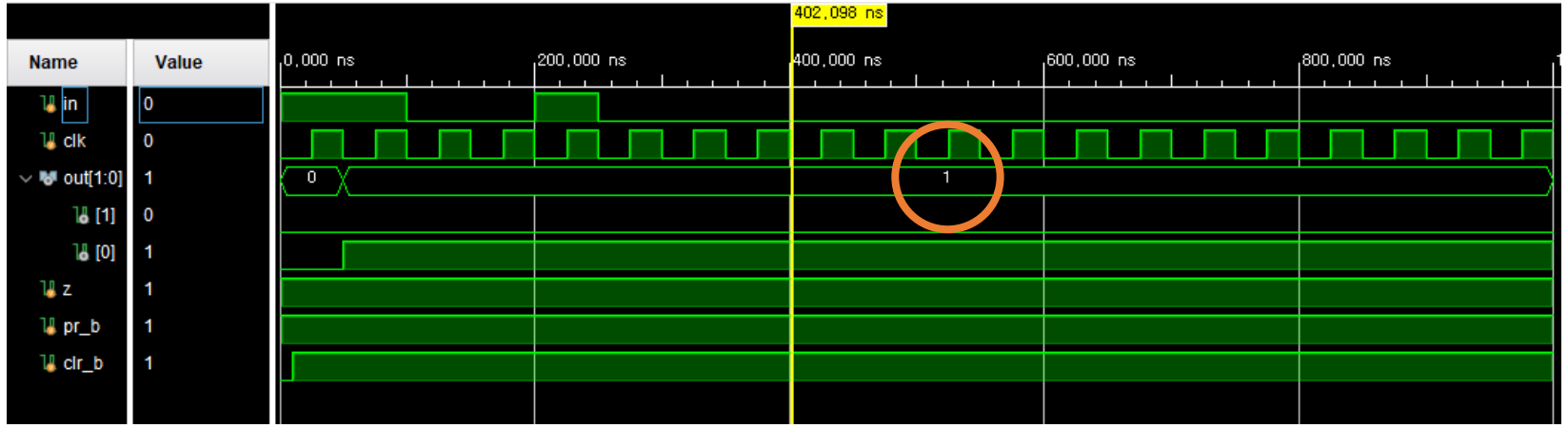

Input : $11001110_{(2)}$ => intensity : 01110011

$a_0 \sim a_7$ $a_7 \sim a_0$

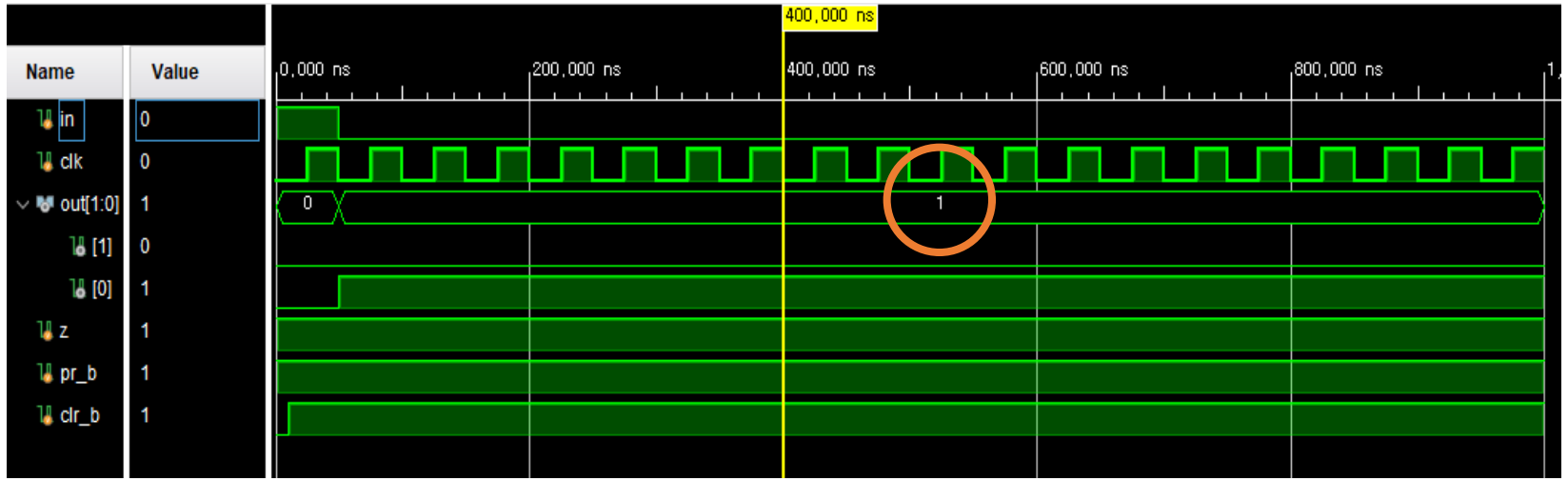


# 3-2) Classifier : Weak

```
module Classifier_tb;

reg in;
reg clk;
wire [1:0]out;

reg z, pr_b, clr_b;

Classifier inst(in, clk, out, z, pr_b, clr_b);

initial  // input : 1000000 <-> intensity : 0000001
    begin
    pr_b = 1; clr_b = 0; z=1; clk =0; in=1;
    #10 clr_b = 1;
    #40 in = 0;
    #50 in = 0;
    #50 in = 0;
    #50 in = 0;
    #50 in = 0;
    #50 in = 0;
    #50 in = 0;
    #50 in = 0; // check for dmux8
    end

always #25 clk = ~clk;


endmodule
```




Input : $11001000_{(2)}$ => intensity : 00010011

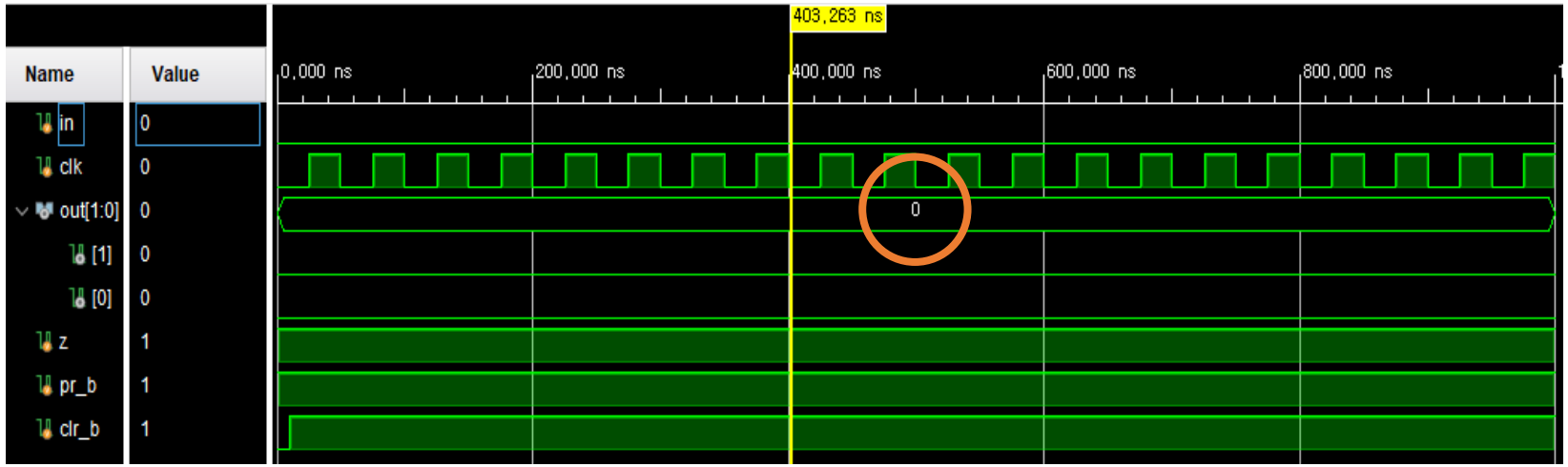



Input : $10000000_{(2)}$ => intensity : 00000001

$a_0$ ~$a_7$ $a_7$ ~$a_0$

# 3-2) Classifier : None

```
module Classifier_tb;

reg in;
reg clk;
wire [1:0]out;

reg z, pr_b, clr_b;

Classifier inst(in, clk, out, z, pr_b, clr_b);

initial  // input : 0000000  <-> intensity : 0000000
    begin
    pr_b = 1; clr_b = 0; z=1; clk =0; in=0;
    #10 clr_b = 1;
    #40 in = 0;
    #50 in = 0;
    #50 in = 0;
    #50 in = 0;
    #50 in = 0;
    #50 in = 0;
    #50 in = 0;
    #50 in = 0; // check for dmux8
    end

always #25 clk = ~clk;


endmodule
```



# Plan

4. Quantitative Criteria for the Thresholds > Thesis
5. Coding the hardware process *with delay* in Verilog
   (proportional to # of transistors)

# Feedback

- Thesis 참조할 때 UBAT 파일 내부에 있는 Mux 칩의 스펙을 확인하기

> SPACIROC_datasheet // 64ch readout 처리하는 칩의 규격

(UBAT -> Electronics -> Analog -> Datasheet)

# The Steps

1. ~~Understanding the process of ‘hit finding algorithm’~~
2. ~~Designing a hardware process~~
3. ~~Coding the hardware process without delay in Verilog~~
4. SPACIROC : ‘spaciroc_datasheet_20110510’
5. Calibration for Photon Detector

➔ Find out the address of the *real hit*

# The Detailed Progress

4. SPACIROC : ‘spaciroc_datasheet_20110510’

4-0) ASIC
4-1) Specification for the chip
4-2) Process of SPACIROC
4-3) MAROC chip

# 4-1) Specification for the chip

SPACIROC : Spatial Photomultiplier Array Counting and Integrating ReadOut Chip

SPACIROC is designed to readout 64 channels Multi-anode photomultipliers (MAPMT). The main features of this ASIC are to count detected photons and to perform charge to time (Q-to-T) conversion. SPACIROC offers 64 inputs dedicated to the anodes of one MAPMT and 1 input for the dynode. The specifications for the chip are the following, assuming the MAPMT gain is $10^6$:

- Photon Counting : 64 channels
- Q-to-T converter : 1 channel for last dynode + 8 internal channels (summed signal)
- 100% trigger efficiency for charge greater than 1/3 photoelectron (p.e.) or 50fC with 30 ns double pulse resolution
- Q-to-T converter input range: 2pC - 400pC (13 p.e. - 2500 p.e )
- Power consumption : 1 mW/channel
- 9 data serial outputs

# 4-2) Process of SPACIROC

SPACIROC : Spatial Photomultiplier Array Counting and Integrating ReadOut Chip

For the photon counting, the preamplifier, the shapers and the discriminators are largely inherited from the MAROC3[3] chip. The input signal passes through a low-noise and low input impedance preamplifier with an individual variable gain of 8-bit. Afterwards, the amplified signal could be fed through shapers or into the Q-to-T converter. Currently, the chip offers 3 different discriminator outputs for discriminating the detected photons. Depending on the selected shaper and discriminator, this chip could reach a double pulse resolution up to 30ns. To count the detected photons, a digital module was built for each channel around an 8-bit counter which could operate up to 100 MHz. At the end of each acquisition window (GTU=2.5μs), the counter values are transmitted through 8 serial links.

# 4-3) MAROC chip

## MAROC: Multi-Anode ReadOut Chip for MaPMTs

P. Barrillon, S. Blin, M. Bouchel, T. Caceres, C. de La Taille, *member IEEE*, G. Martin, P. Puzo, N. Seguin-Moreau

*Laboratoire de l'Accélérateur Linéaire, IN2P3-CNRS, Université Paris Sud 11*

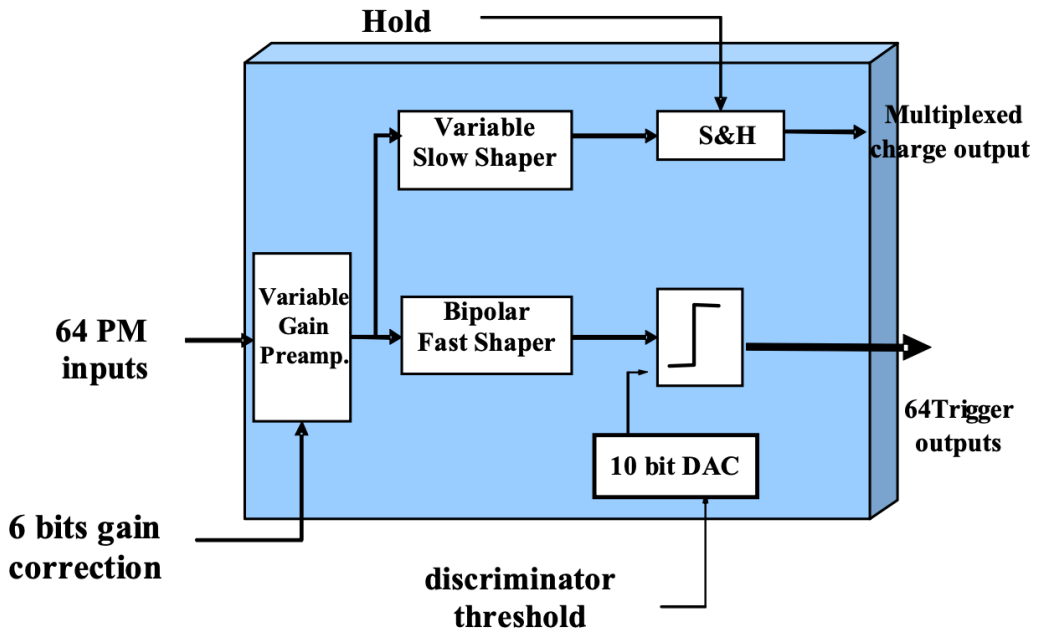



Fig. 3. Bloc diagram of the MAROC ASIC.

Fig. 3. represents the bloc diagram of MAROC with its main features. The chip has 64 "super common base" inputs, 64 trigger outputs and a multiplexed charge output. Each channel is made of a variable gain preamplifier with low input tunable impedance (50-100 Ω), a low offset and a low bias current (20µA) in order to minimize the cross talk. This variable gain allows compensating for the PM gain dispersion up to a factor 4 to an accuracy of 6% with 6 bits. The amplified current feeds then two paths:

- A slow shaper path which consists in a CRRC$^2$ shaper and a Sample and Hold Widlar differential buffer. This S&H block stores the charge in a 2 pF capacitor and deliver a multiplexed charge measurement with a 5 MHz readout speed.
- A fast (15 ns) shaper path made of a CRRC bipolar shaper followed by a discriminator. The threshold is set by an internal 10 bit DAC composed by a 4 bits thermometer DAC allowing coarse tuning (200 mV per step) and a 7 bits mirror DAC used for fine tuning (3 mV per step). A trigger output is produced.

https://indico.fnal.gov/event/49220/attachments/143287/181205/maroc_atlas.pdf

# The Detailed Progress

## 5. Calibration for Photon Detector

Space Sci Rev (2018) 214:16
DOI 10.1007/s11214-017-0454-5

CrossMark

**UBAT of UFFO/*Lomonosov*: The X-Ray Space Telescope to Observe Early Photons from Gamma-Ray Bursts**

**S. Jeong[1,2,3] · M.I. Panasyuk[4,5] · V. Reglero[6] · P. Connell[6] · M.B. Kim[1] · J. Lee[1] · J.M. Rodrigo[6] · J. Ripa[7] · C. Eyles[6] · H. Lim[1] · G. Gaikov[1] · H. Jeong[1] · V. Leonov[1] · P. Chen[7,8] · A.J. Castro-Tirado[3,9] · J.W. Nam[7,8] · S. Svertilov[4,5] · I. Yashin[4] · G. Garipov[4] · M.-H.A. Huang[10] · J.-J. Huang[7] · J.E. Kim[1] · T.-C. Liu[7] · V. Petrov[4] · V. Bogomolov[4,5] · C. Budtz-Jørgensen[11] · S. Brandt[11] · I.H. Park[1,2]**

Received: 31 January 2017 / Accepted: 27 November 2017

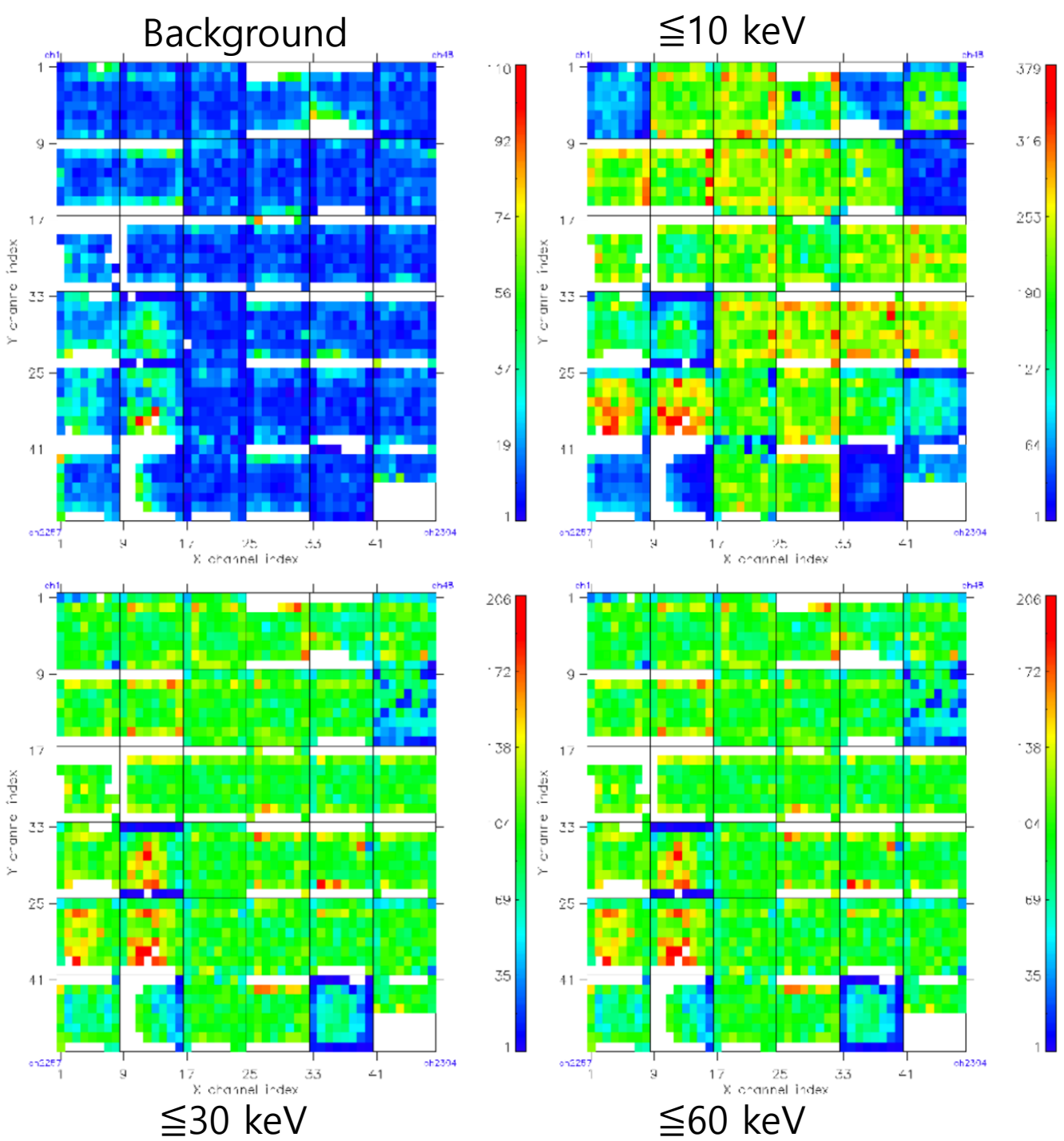
## 4 Pre-flight Calibration of the UBAT Detector

### 4.1 Detector Uniformity

Response variations among the detector pixels were investigated through uniform X-ray illumination at energies of ≤10 keV, ≤30 keV and ≤50 keV. The detector response of the UBAT is adjustable in three different ways. First, we adjusted the parameters of the ASICs' preamplifiers to change the output signal level fed into the MAPMT. Next, high voltages in the range between 942 V and 1044 V were adjusted for groups of 4 MAPMTs placed on analog boards. Finally, preamplifier output thresholds for the photon counting mode were set differently on an analog ASIC by ana X-ray energies,

**Table 3** Detected count rates with a uniform illumination of the UBAT detector from different X-ray energies. The exposure time was 186 s, frame time 10 ms, and the area of active detector pixels 138.7 $cm^2$

| Set-up | Total counts (cnts) | Total counts—background (cnts) | Count rate due to the source ($cnts/cm^2/s$) |
|---|---|---|---|
| Background | 30 460 | 0 | 0 |
| 10 keV | 320 049 | 289 589 | 9.72 |
| 30 keV | 182 545 | 152 085 | 5.10 |
| 50 keV | 167 716 | 137 256 | 4.61 |



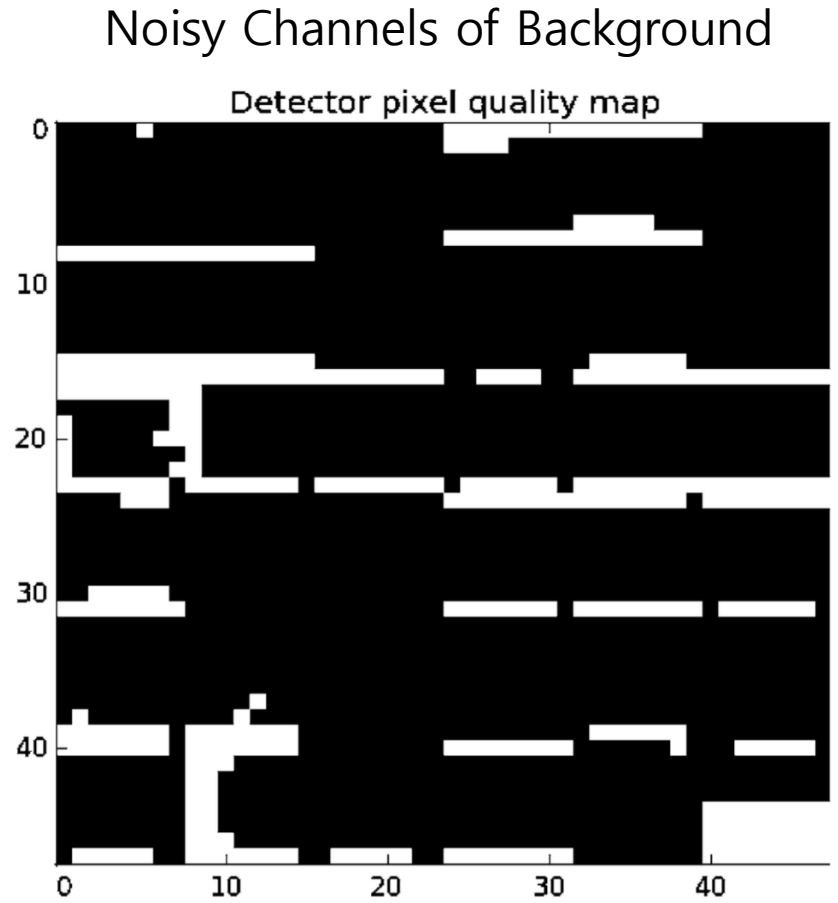
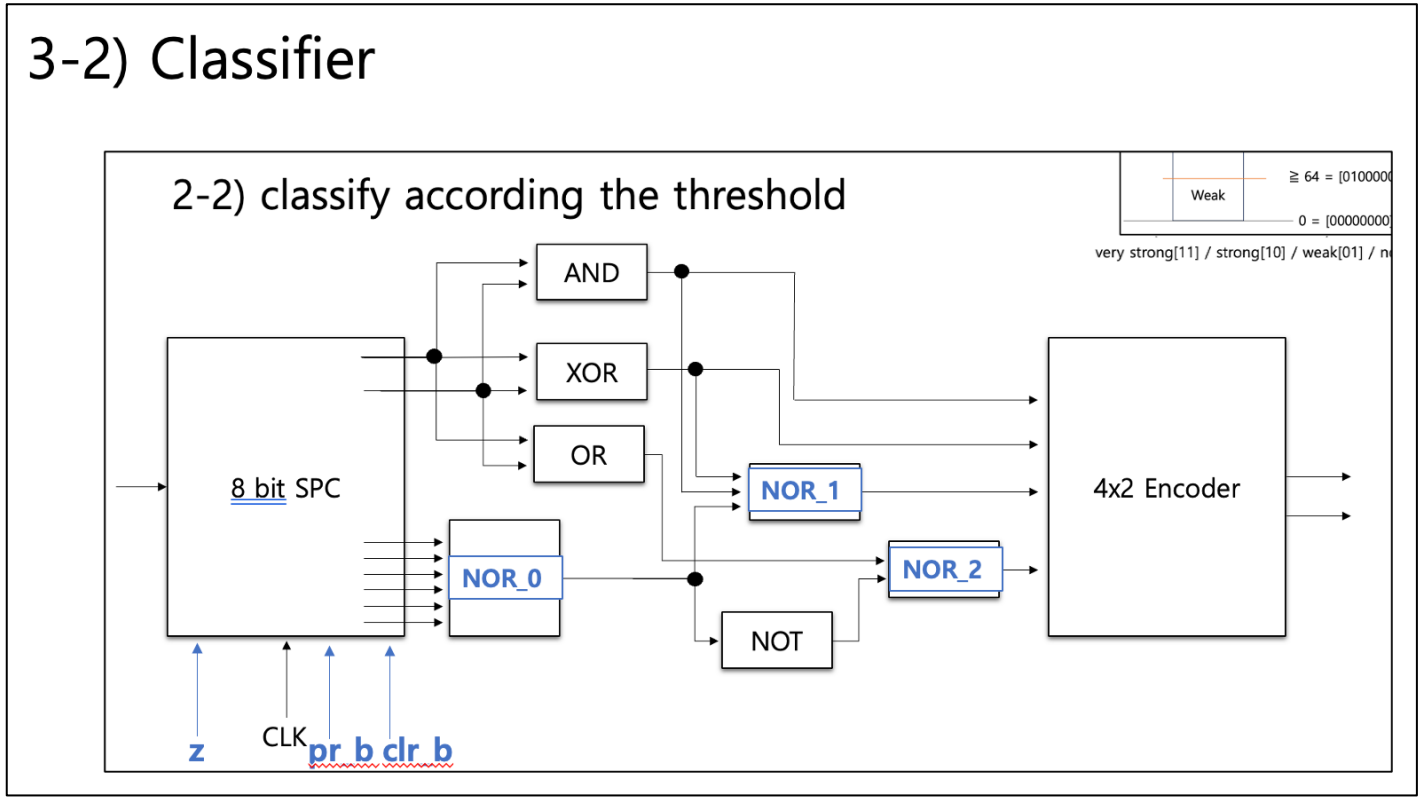




**Fig. 12** Detector response of the UBAT flight model integrated over ~186 s in a dark room. Hot, that is, noisy, channels are switched off as shown in Fig. 13. The *color scale* indicates the total number of counts per pixel. *White* indicates zero counts (resulting mainly from switched off pixels). *Top left*: No X-ray source, *top right*: X-ray energies ≤10 keV, *bottom left*: X-ray energies ≤30 keV, *bottom right*: X-ray energies ≤60 keV

# Idea

Add a Subtractor to the Classifier in Step3
to **remove the Background noise**



# Idea

- Subtractor :
 for removing the Background noise

- Register :
 for saving the Background noise

> Insert both directly after the SPC (serial parallel convertor)

# Basic Process of the Idea

> Input(t) from the output of SPC
> Input(t-1) from the Register

In Subtractor,

Input(t) – Input(t-1) = without noise

Weakness) Checked only once directly after the burst

# Advanced Process of the Idea

Input(t) – Input(t-1)  [from the basic idea]

➔ Checking for the burst occurrence w/ a threshold

➔ Generate Trigger signal

Background noise

➔ mean of the noise before the burst

➔ Burst signal – Background noise = without noise

# Feedback

광도 곡선 Dirac-Delta Function의 형태로 나오지 않을 수 있음

➔ burst의 발생을 체크하는 방법을 다시 구상해보기.

# The Steps

1. ~~Understanding the process of 'hit finding algorithm'~~
2. ~~Designing a hardware process~~
3. ~~Coding the hardware process without delay in Verilog~~
4. ~~SPACIROC : 'spaciroc_datasheet_20110510'~~
5. ~~Calibration for Photon Detector~~
6. Demonstrating on FPGA

➔ Find out the address of the *real hit*

# The Detailed Progress

6. Demonstrating on FPGA

6-0) .xdc file
6-1) result
6-2) disadvantage & improvement

# 6-0) .xdc file

Classifier.v | Classifier.xdc * | Untitled 1

C:/Users/admin/.Xilinx/XlinxProjects/project/Classifier/Classifier.srcs/constrs_1/new/Classifier.xdc

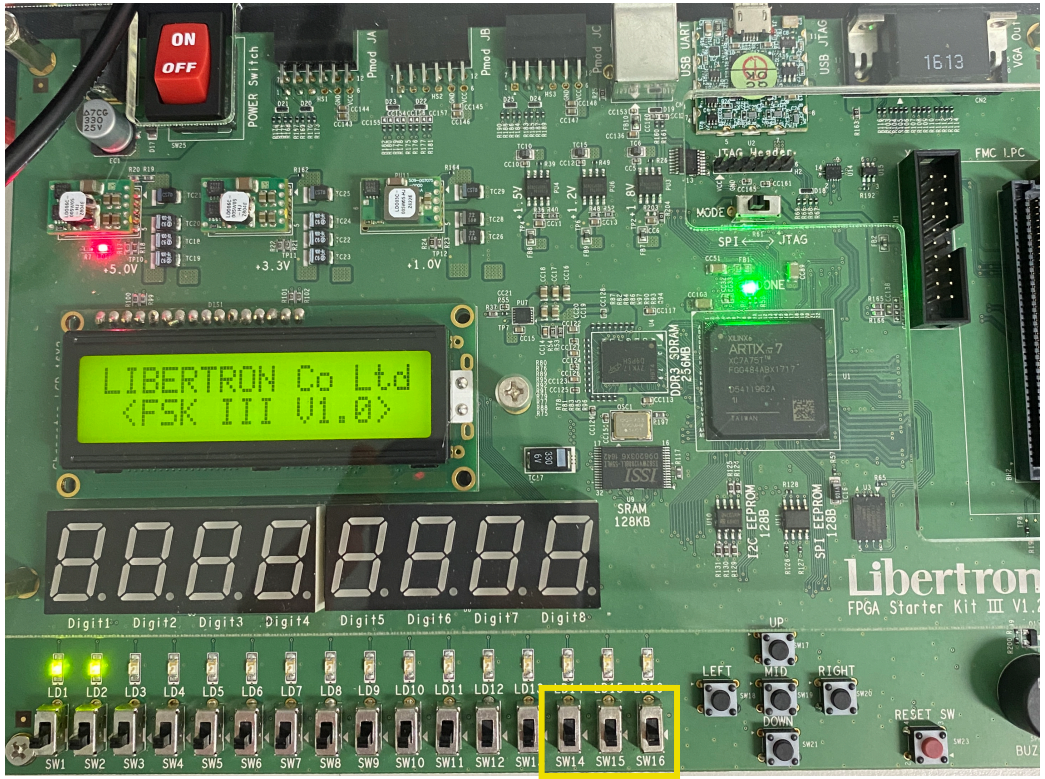
```
1  set_property -dict {PACKAGE_PIN G21 IOSTANDARD LVCMOS33} [get_ports in]; # PUSH_SW_MID
2  set_property -dict {PACKAGE_PIN J4 IOSTANDARD LVCMOS33} [get_ports clk]; # DIP_SW<1>
3
4  set_property -dict {PACKAGE_PIN F15 IOSTANDARD LVCMOS33} [get_ports out[1]]; # LED<0>
5  set_property -dict {PACKAGE_PIN F13 IOSTANDARD LVCMOS33} [get_ports out[0]]; # LED<1>
6
7  set_property -dict {PACKAGE_PIN AB15 IOSTANDARD LVCMOS33} [get_ports z]; # DIP_SW<15>
8  set_property -dict {PACKAGE_PIN AA15 IOSTANDARD LVCMOS33} [get_ports pr_b]; # DIP_SW<14>
9  set_property -dict {PACKAGE_PIN AB13 IOSTANDARD LVCMOS33} [get_ports clr_b]; # DIP_SW<13>
10
11 set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {pr_b_IBUF}]
12 set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {clk_IBUF}]
```

Classifier.v | Classifier.xdc * | Untitled 1

C:/Users/admin/.Xilinx/XlinxProjects/project/Classifier/Classifier.srcs/s

```
21
22
23  module Classifier(in, clk, out, z, pr_b, clr_b);
24
25  input in;
26  input clk;
27  output [1:0]out;
28
29  input z;
30  input pr_b;
31  input clr_b;
32
```

# 6-1) result

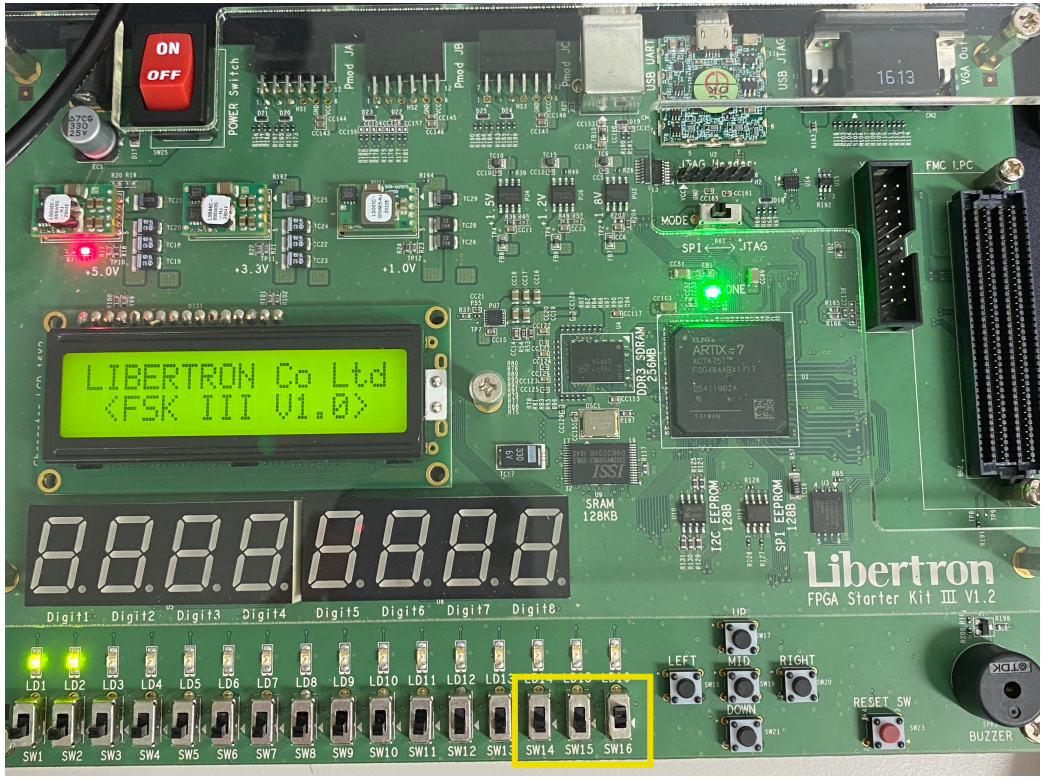



[ initial state ]

Input 00000000

Clr_b = 0 / pr_b =0 / z = 0

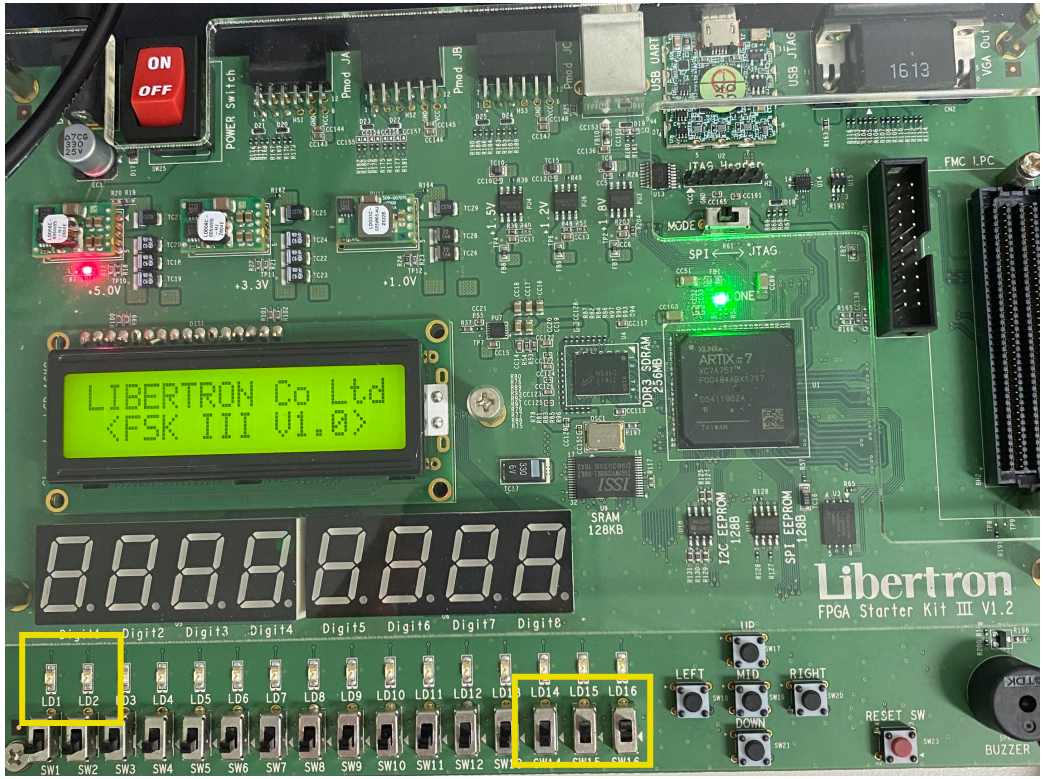
# 6-1) result




Input 00000000

Clr_b = 0 / pr_b =0 / z = 1

# 6-1) result

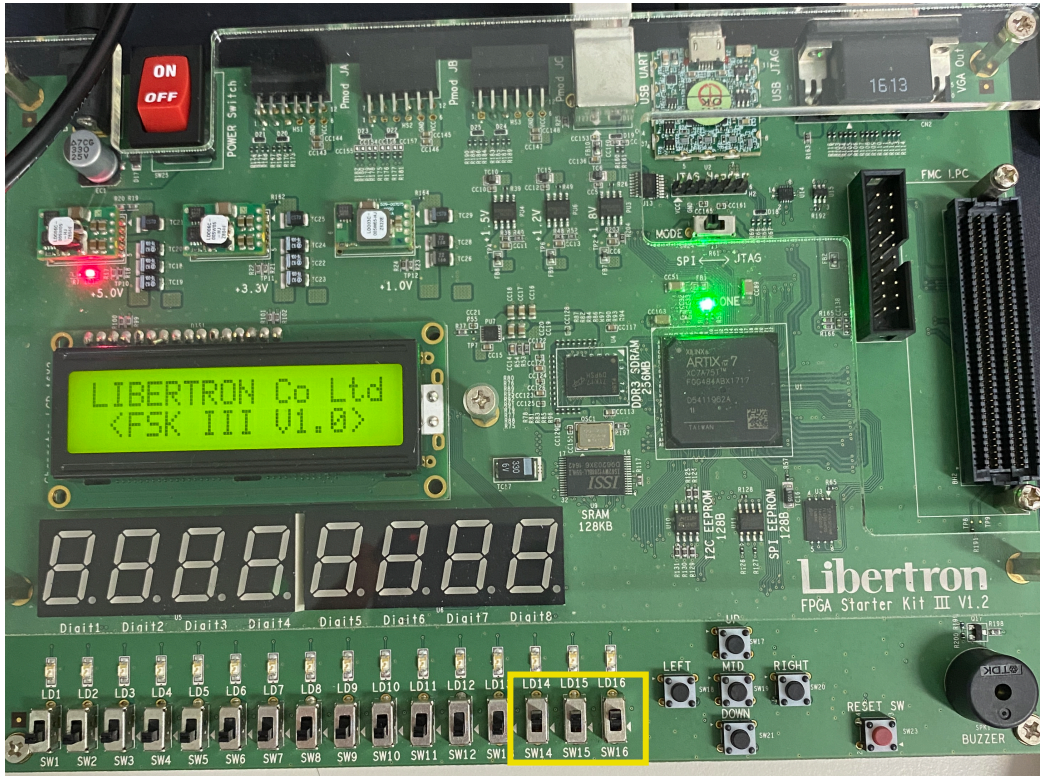



Input 00000000

Clr_b = 0 / pr_b =1 / z = 1

→ **clear**

# 6-1) result




Input 00000000

Clr_b = 1 / pr_b =1 / z = 1

# 6-1) result

Input varies according to each clk cycle

00000000 / 00

00000001 / 01 ( clk * 1 )

00000011 / 01

00000111 / 01

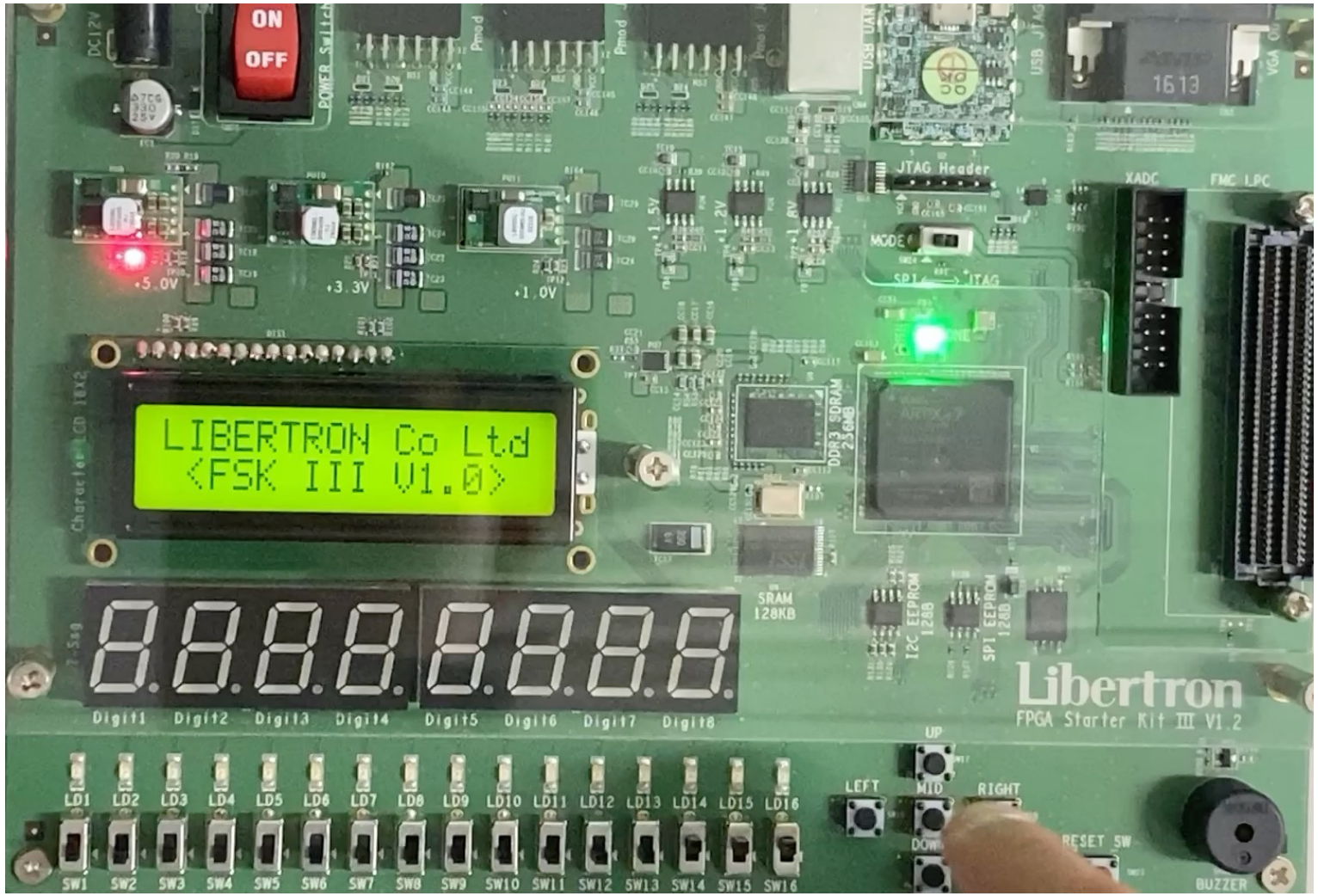
00001111 / 01

00011111 / 01

00111111 / 01

01111111 / 10 ( clk * 2 )

11111111 / 11 ( clk * 3 )



# 6-2) weakness & improvement

Disadvantage

→ Impossible to check

the results of the input with 0's btw. 1 such as $10011101_{(2)}$

Improvement

→ Add PSC (Parallel Serial Converter) before SPC

# Feedback

- 수동으로 clk, input을 주지 말고 **'Block RAM'**을 활용하여 FPGA 내에서 다 처리하기

- Header, enable을 활용하여 input signal에 대한 정보 담기.

  (En = 1이 들어왔을 때 header가 ~~이면 뒤의 몇 bit는 어떤 정보이다.)

* 보드에서 수동으로 하는 부분은 reset, enable 정도만

# The Goals

1. ~~Understanding the process of 'hit finding algorithm'~~
2. ~~Designing a hardware process~~
3. ~~Coding the hardware process without delay in Verilog~~
4. ~~SPACIROC : 'spaciroc_datasheet_20110510'~~
5. ~~Calibration for Photon Detector~~
6. ~~Demonstrating on FPGA~~
7. Study BRAM

➔ Find out the address of the *real hit*

# The Detailed Progress

## 7. Study BRAM

7-0) What is BRAM
7-1) Procedure
7-2) Simulation
7-3) Problems & Supplementations

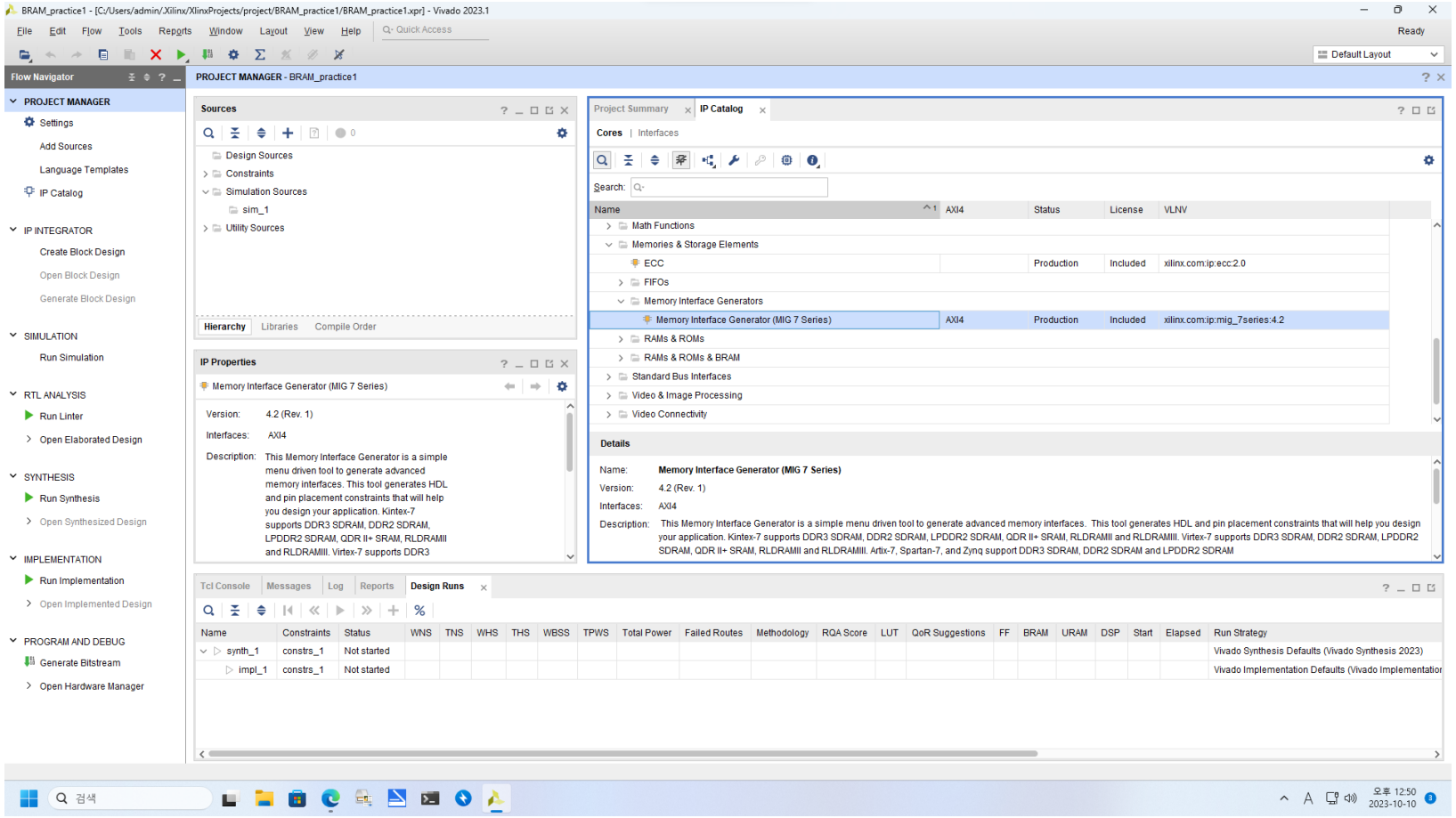
# 7-0) What is BRAM?

Block RAM :

FPGA 내부에서 RAM, ROM, FIFO 등의 기능을 구현하기 위해 사용될 수 있는 FPGA 내부 logic

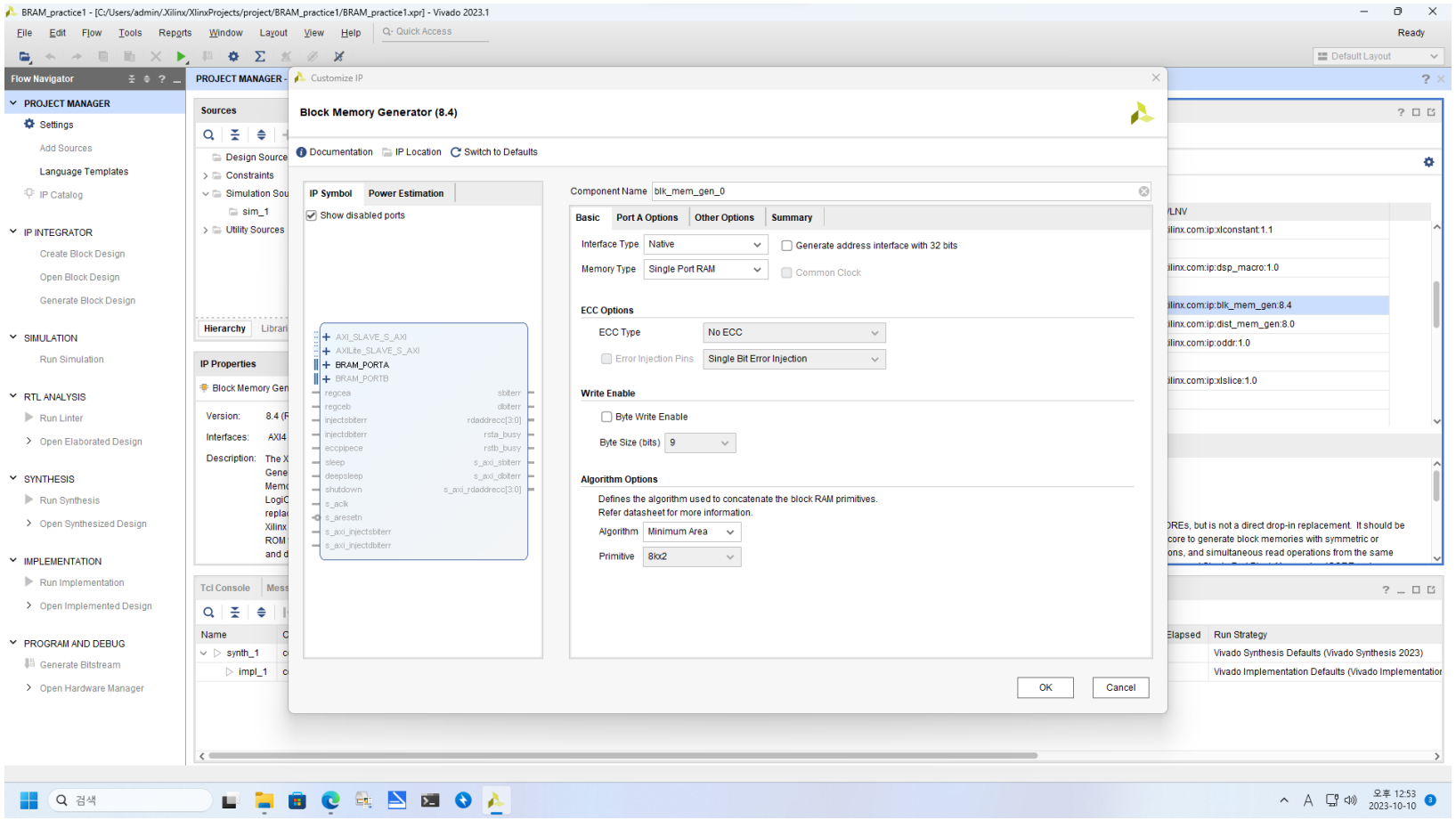
# 7-1) Procedure

IP Catalog → Basic Elements → Memory Elements → Block Memory Generator



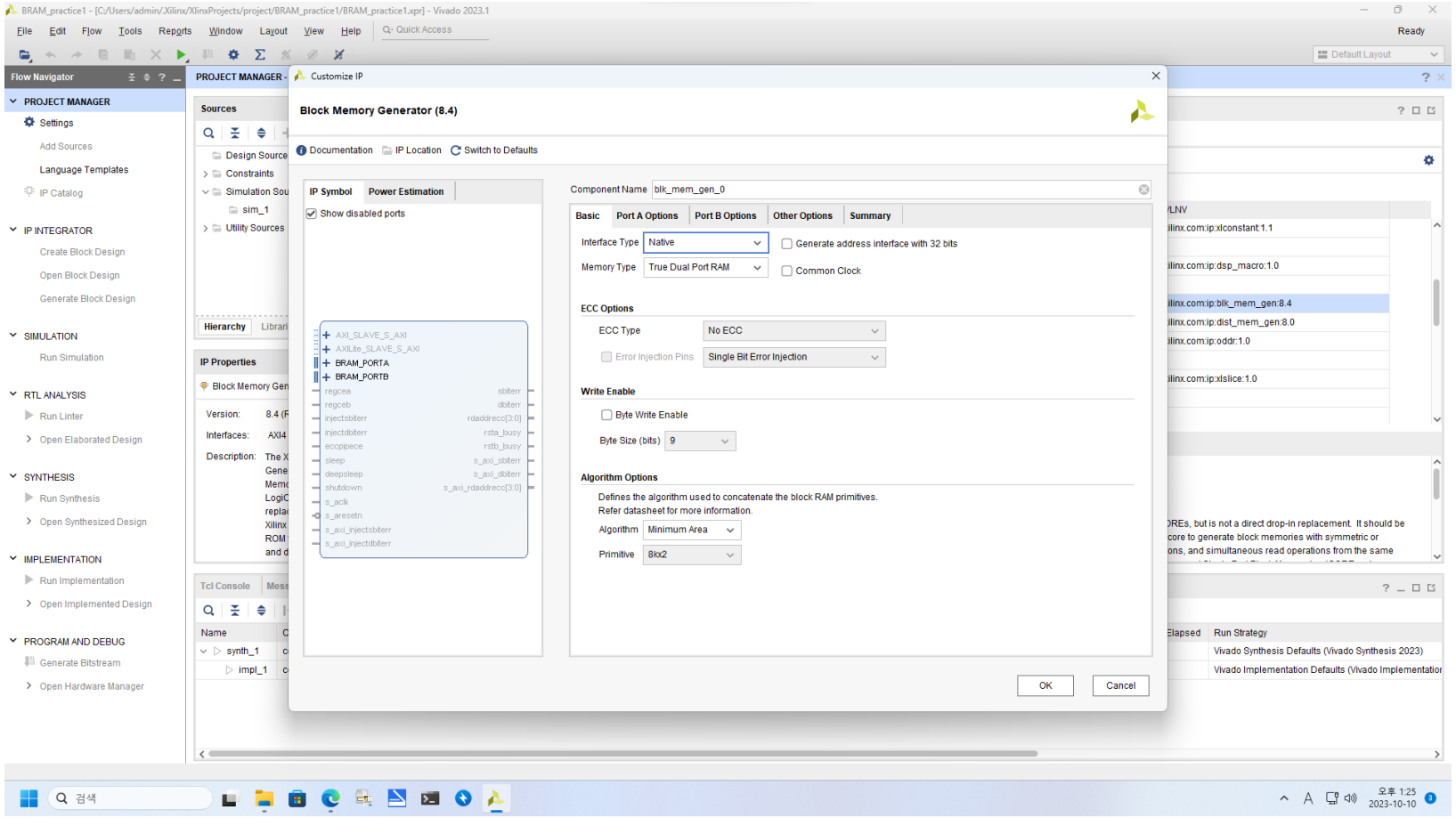
# 7-1) Procedure

## SPRAM (Single Port RAM)  [default]



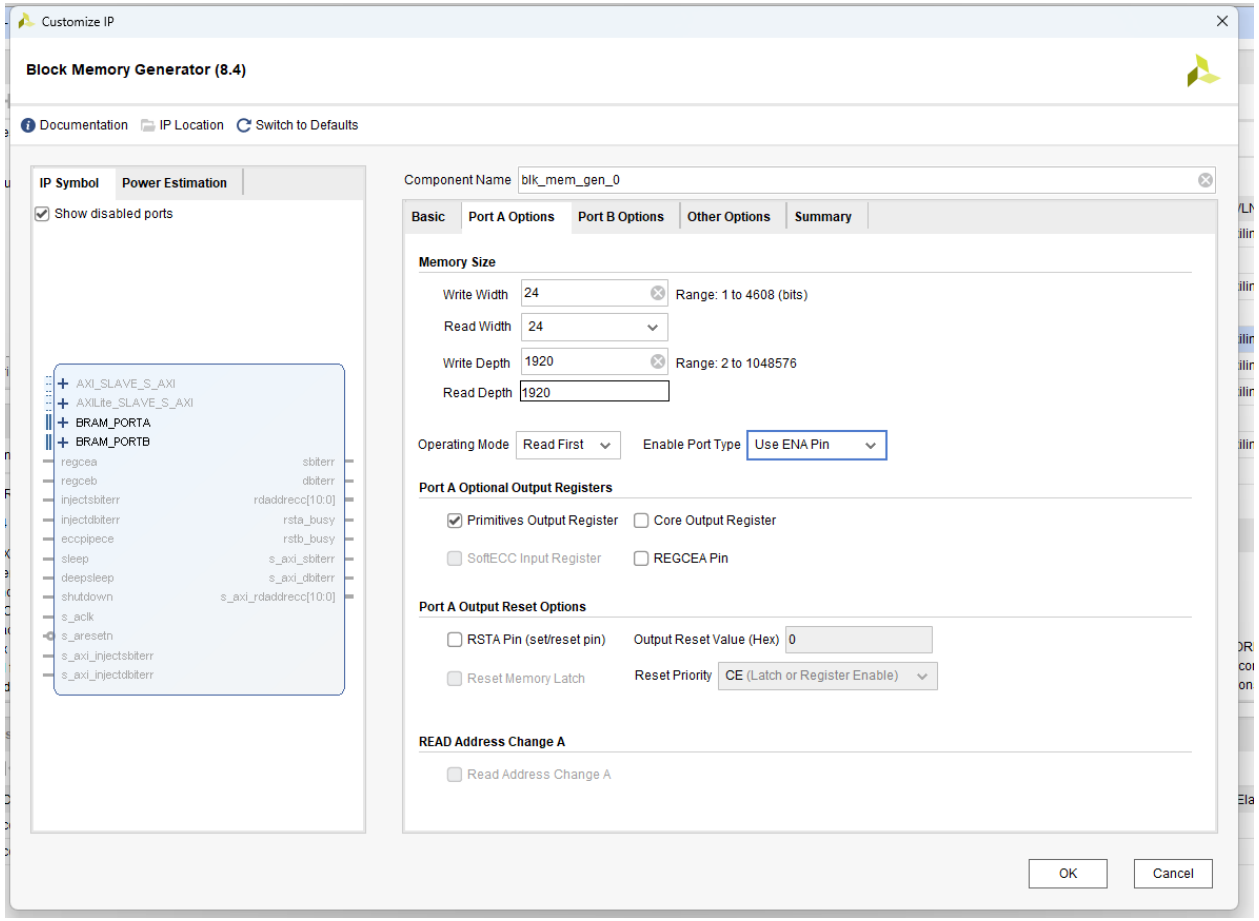
# 7-1) Procedure

## SPRAM (Single Port RAM) → TDRAM (True Dual RAM)



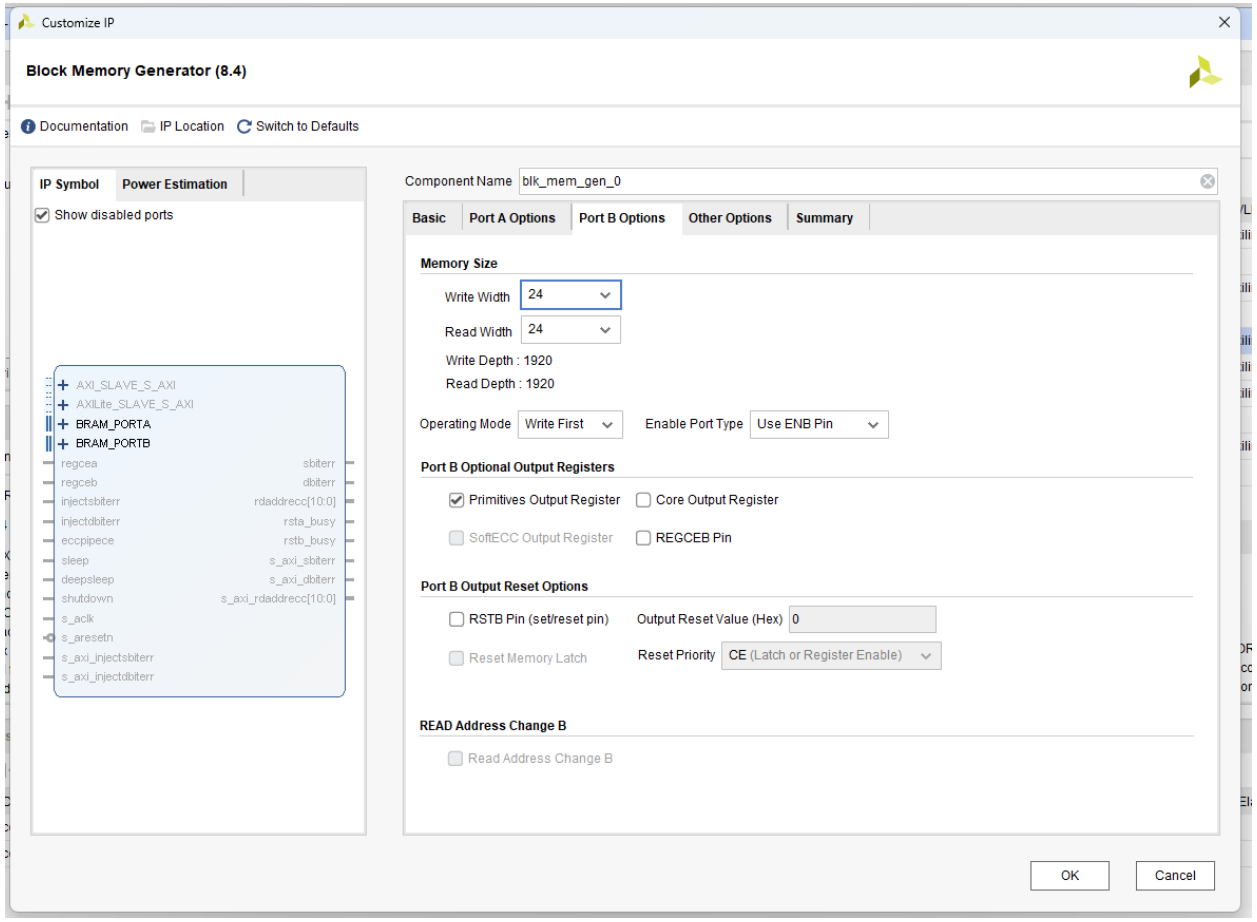
# 7-1) Procedure

## Port A Options



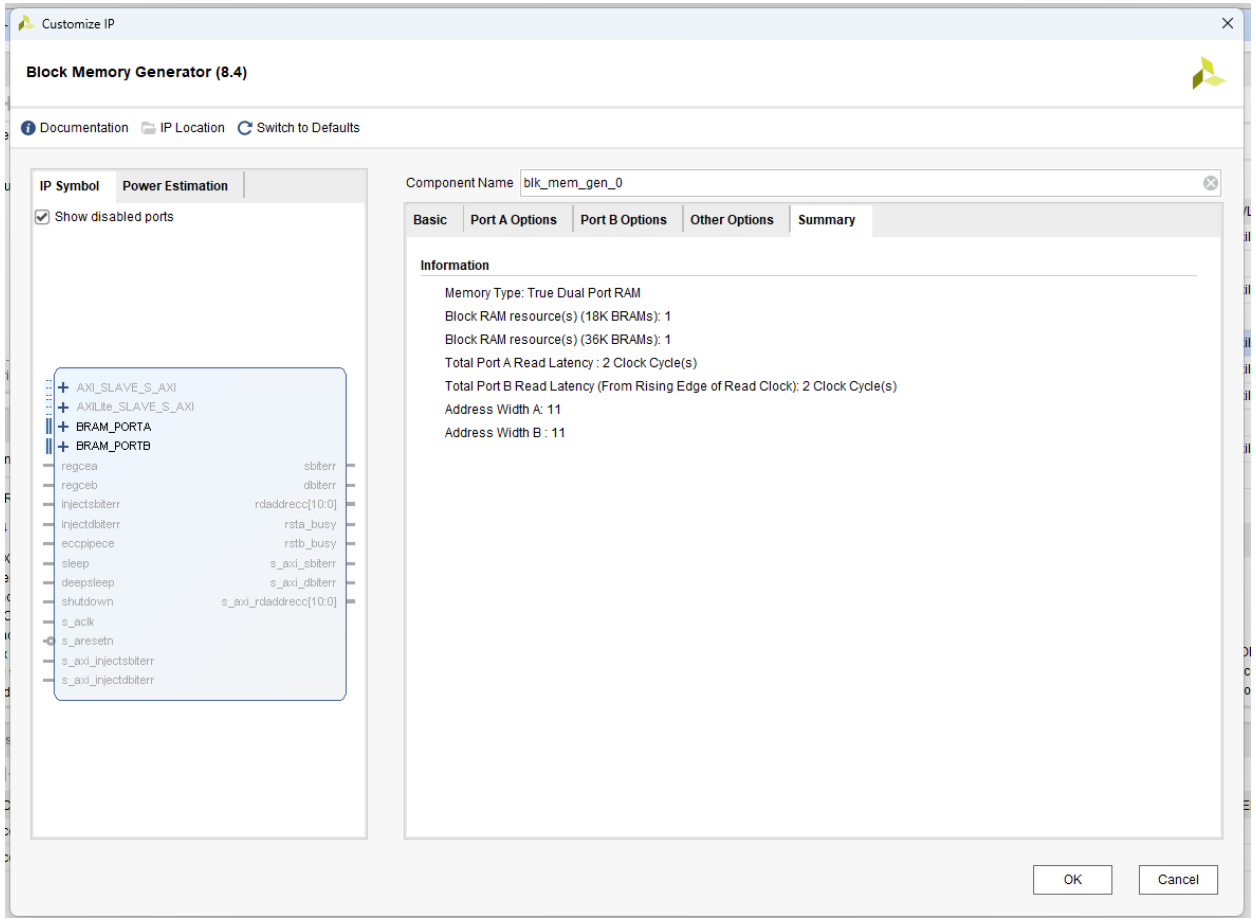
# 7-1) Procedure

## Port B Options

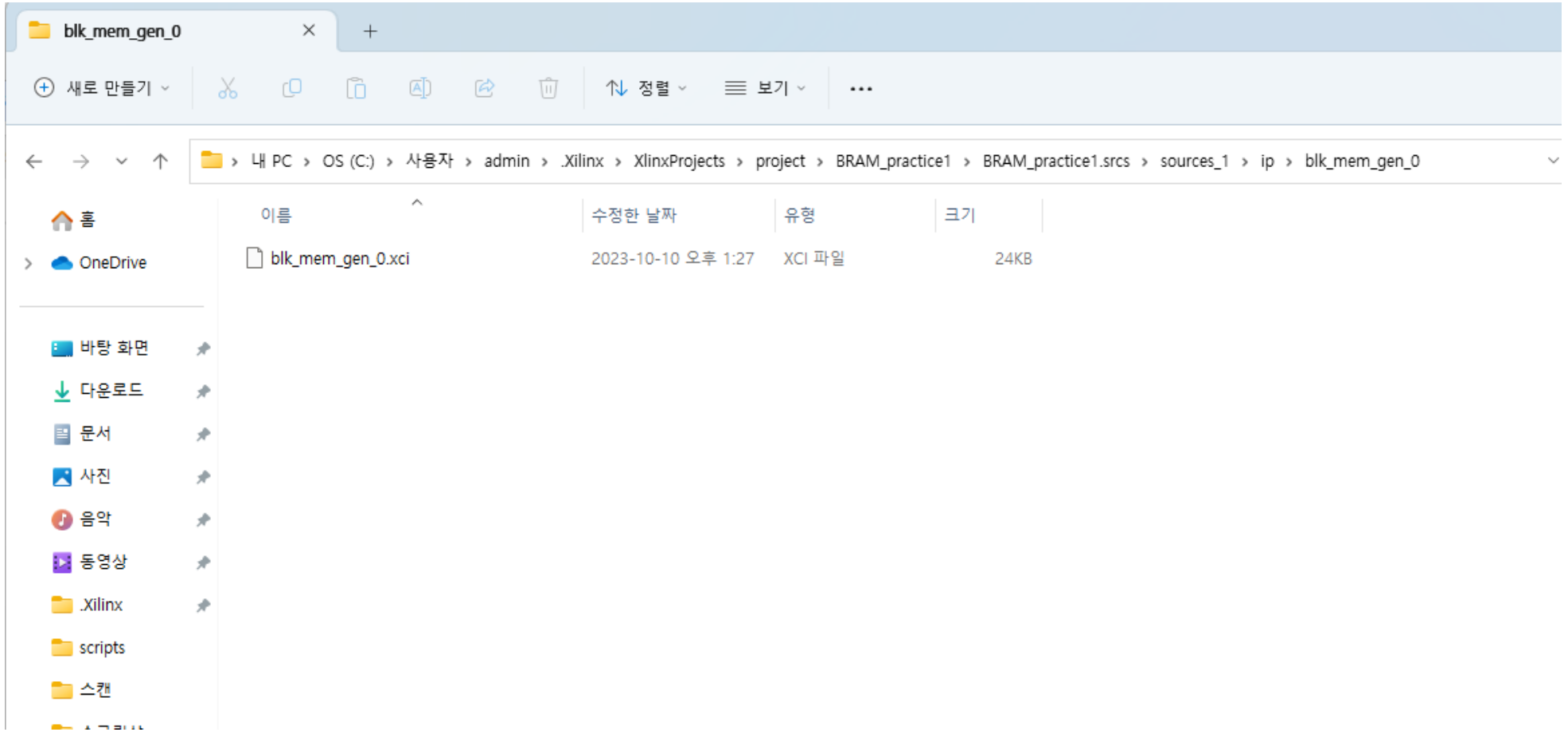


# 7-1) Procedure

## Summary

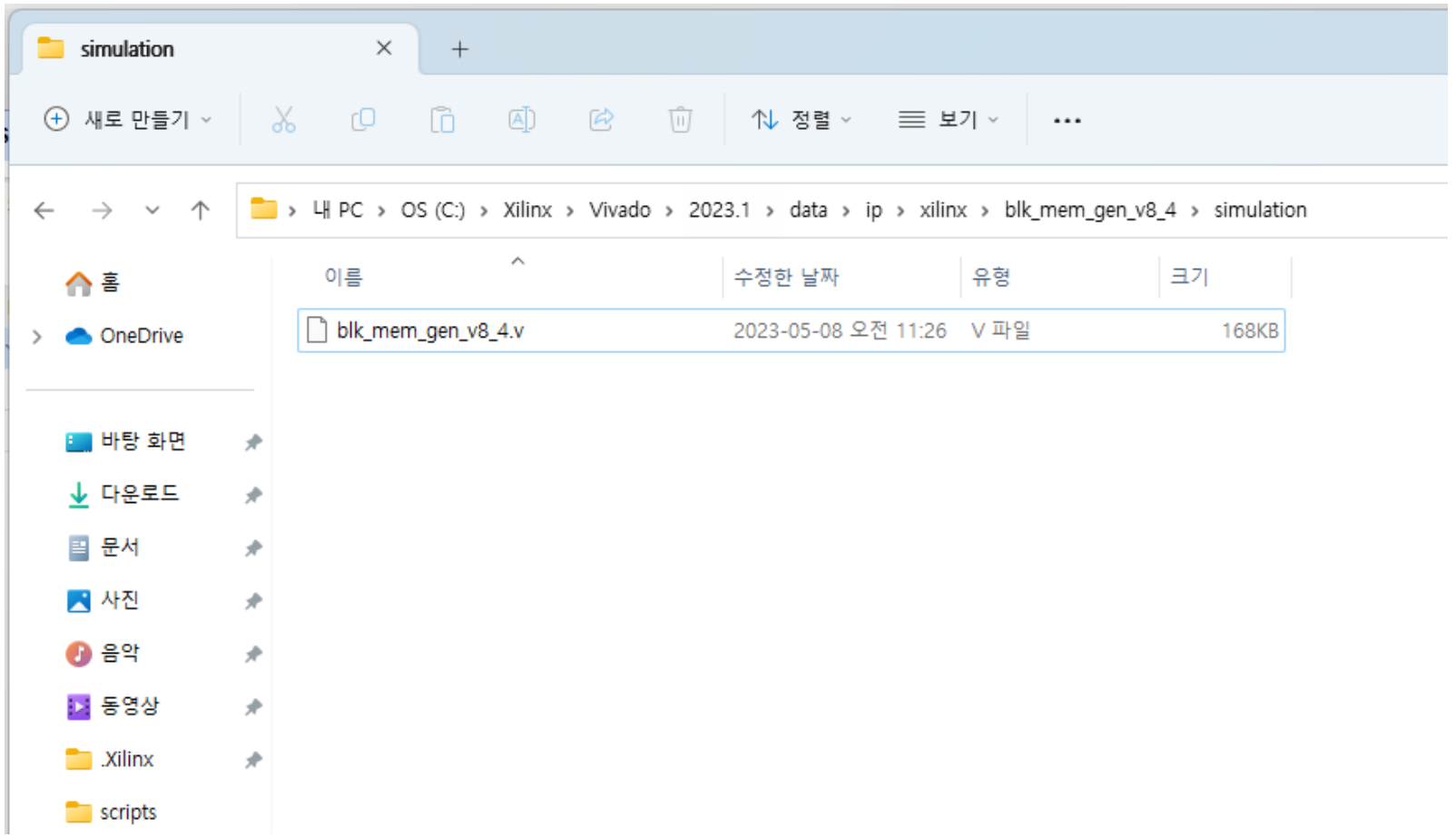


# 7-1) Procedure

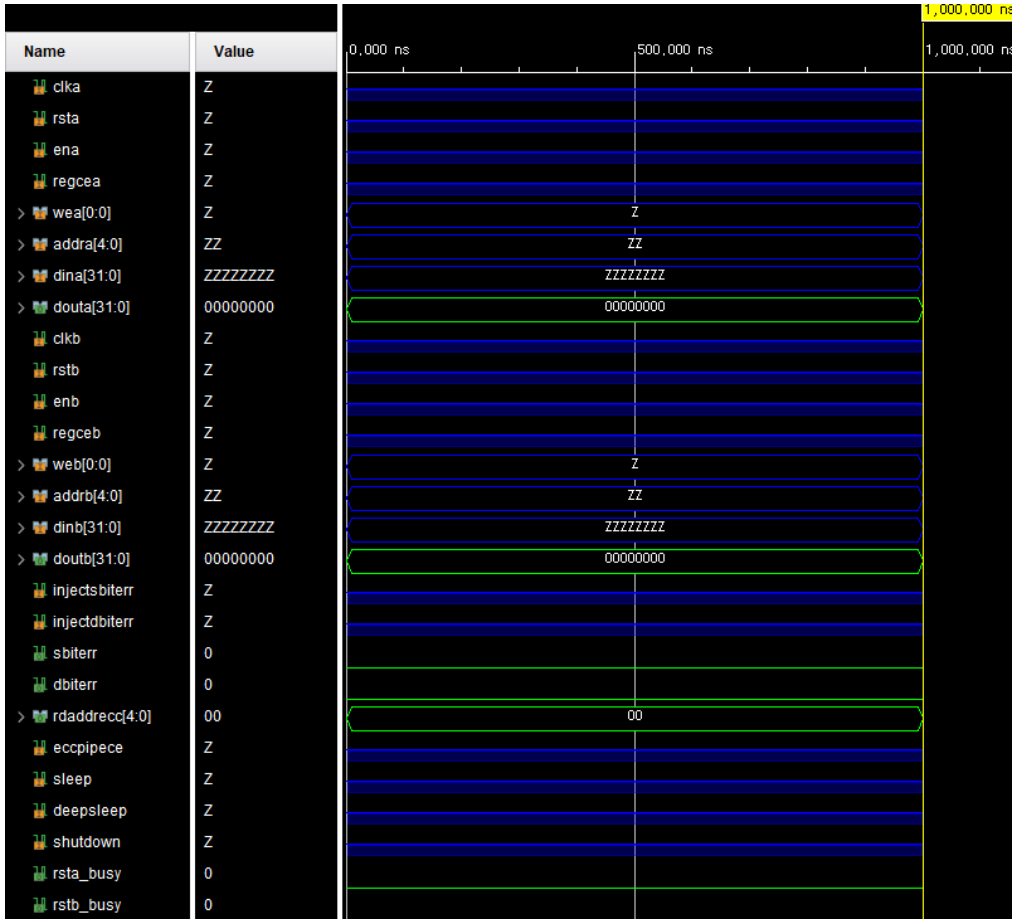


# 7-2) Simulation

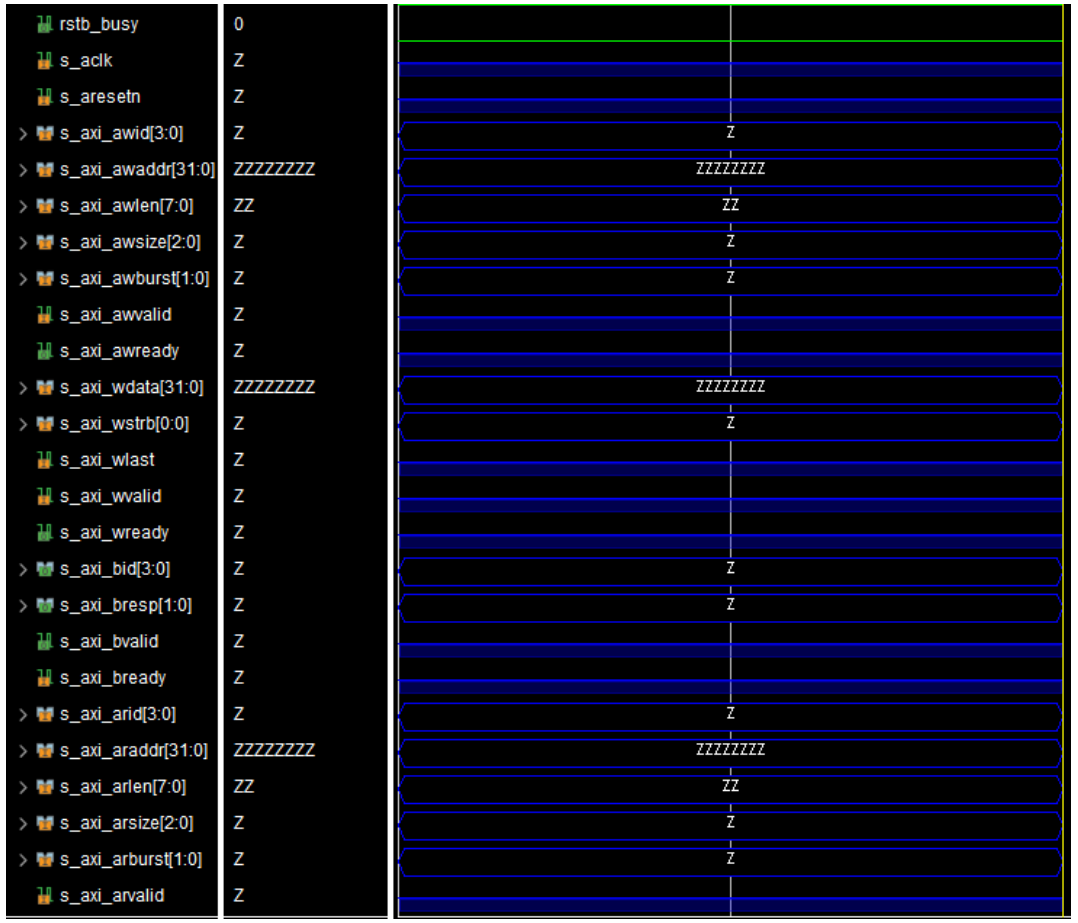
Testbench code [default]



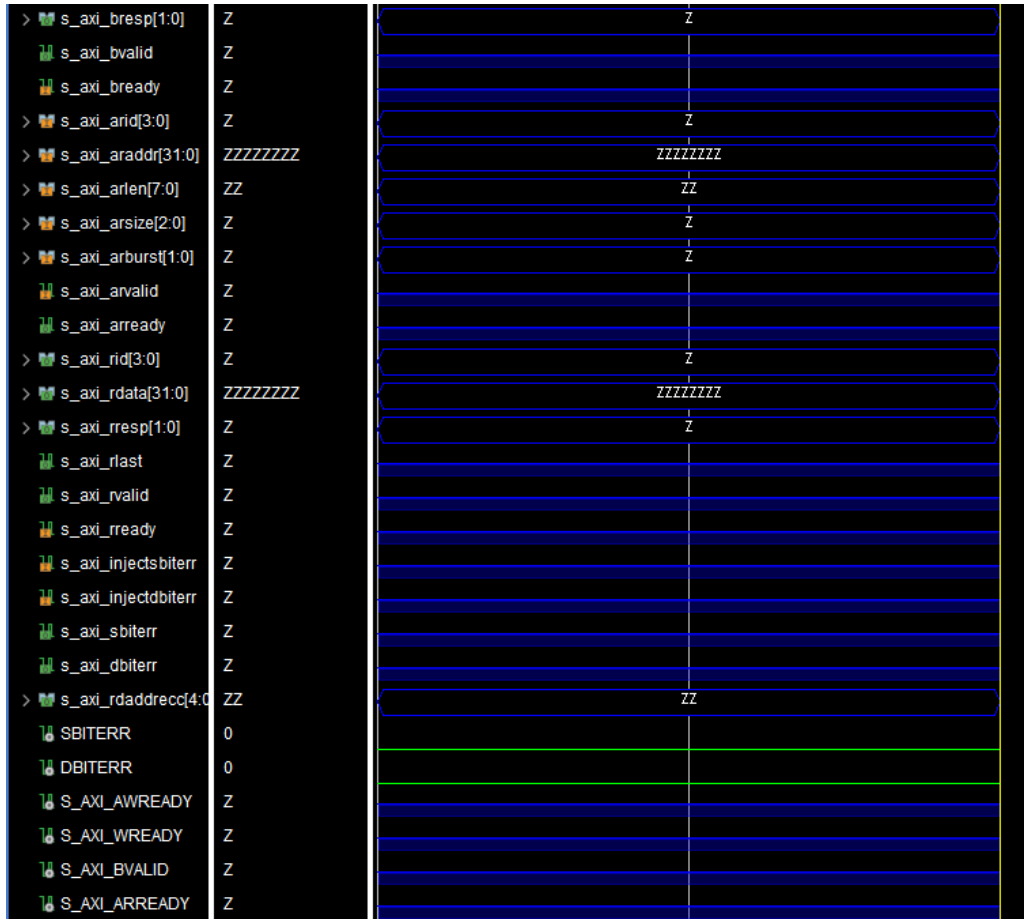
# 7-2) Simulation



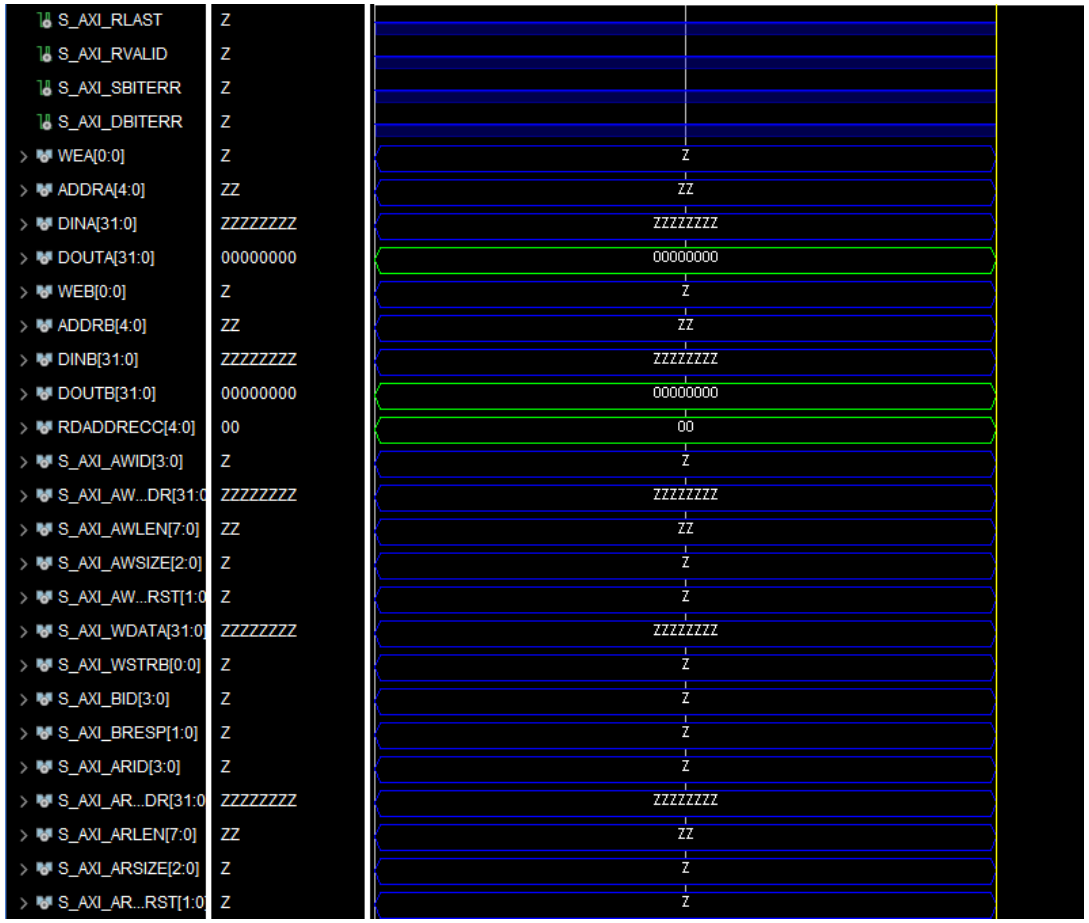
# 7-2) Simulation



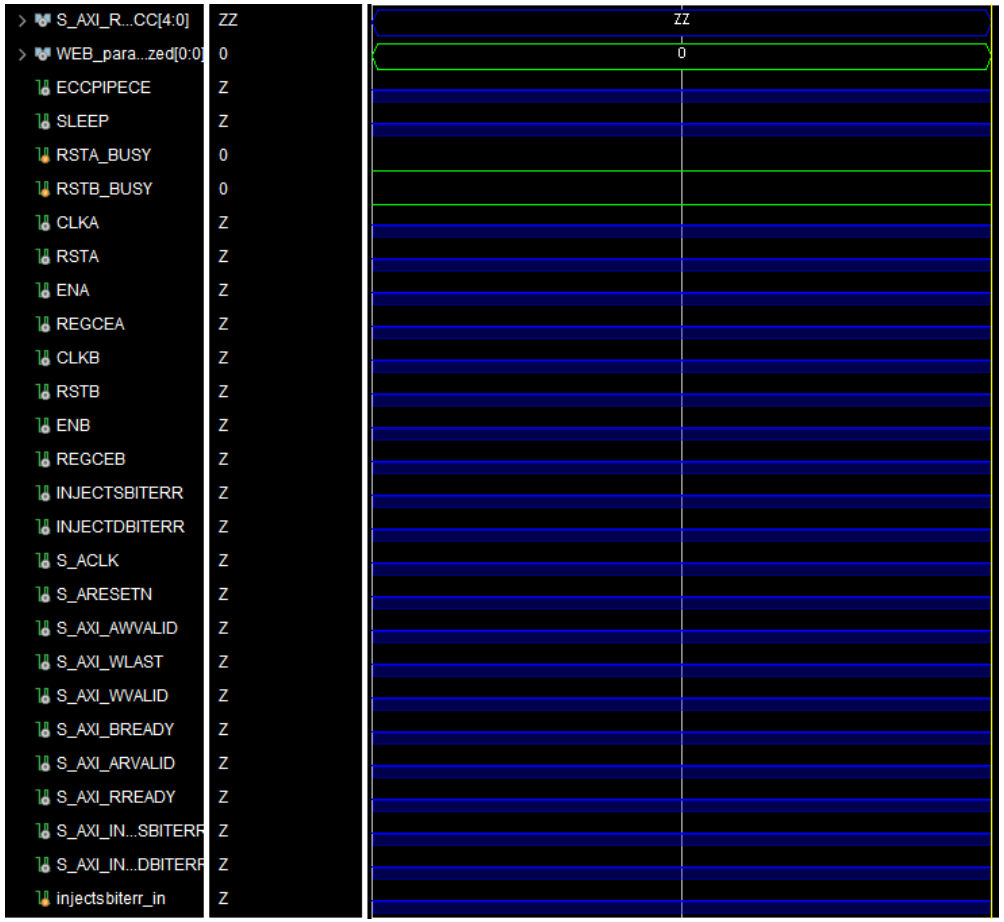
# 7-2) Simulation



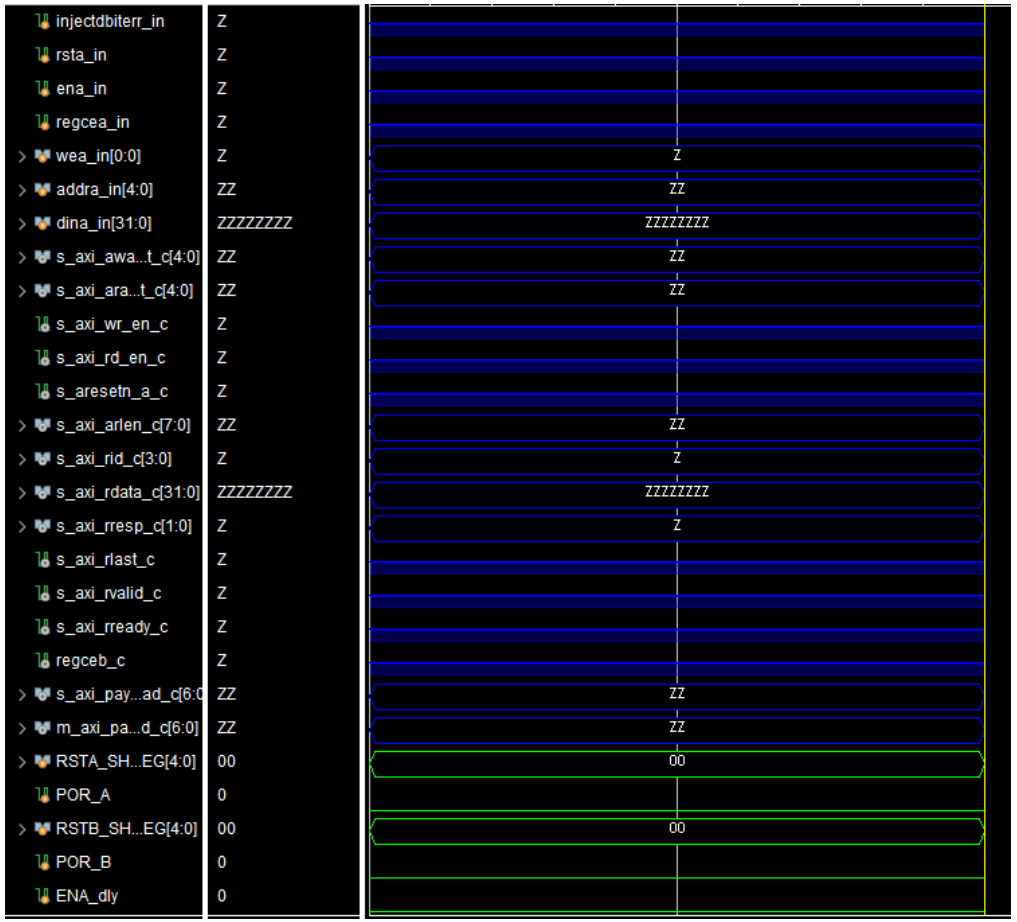
# 7-2) Simulation



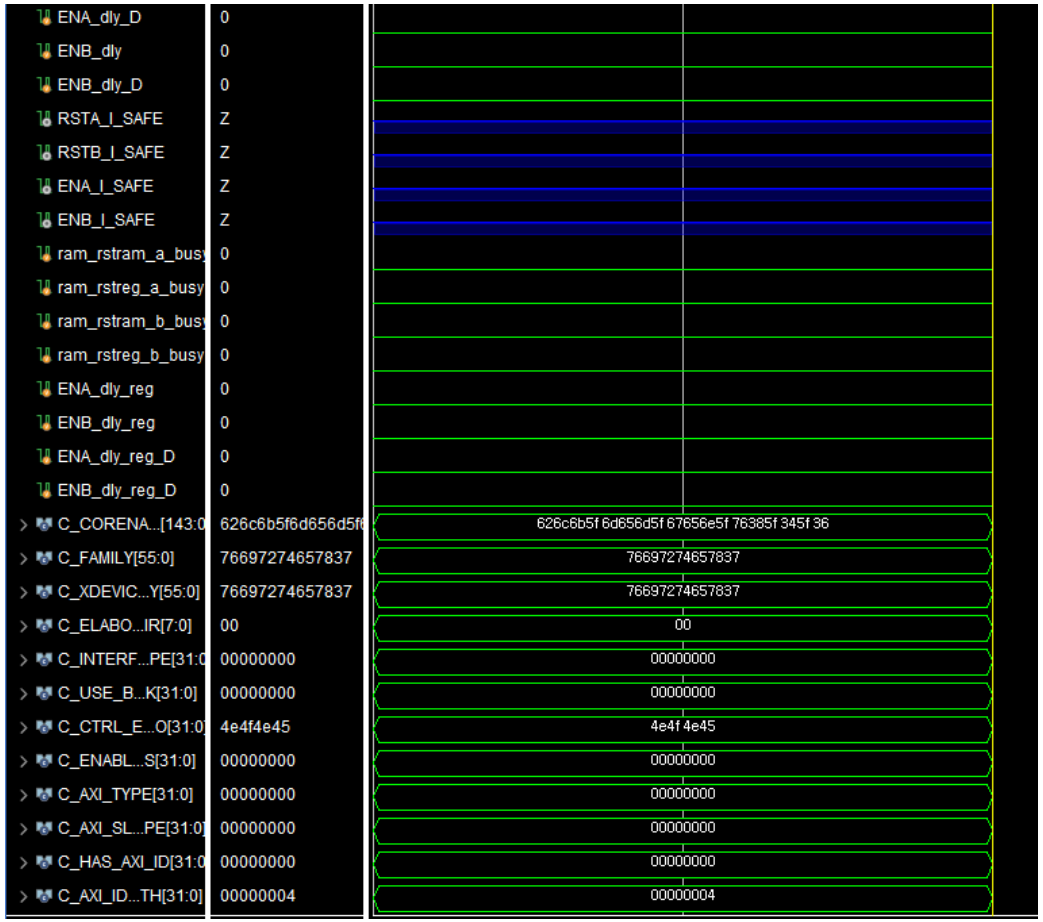
# 7-2) Simulation



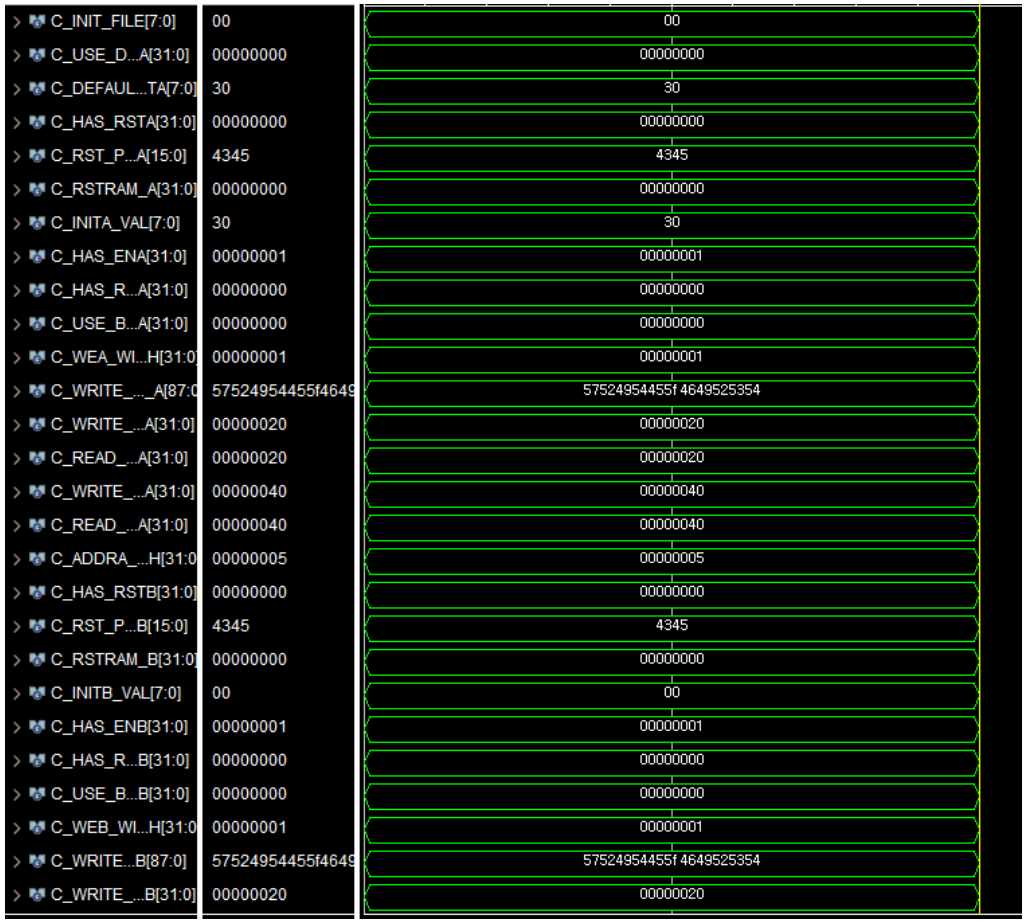
# 7-2) Simulation



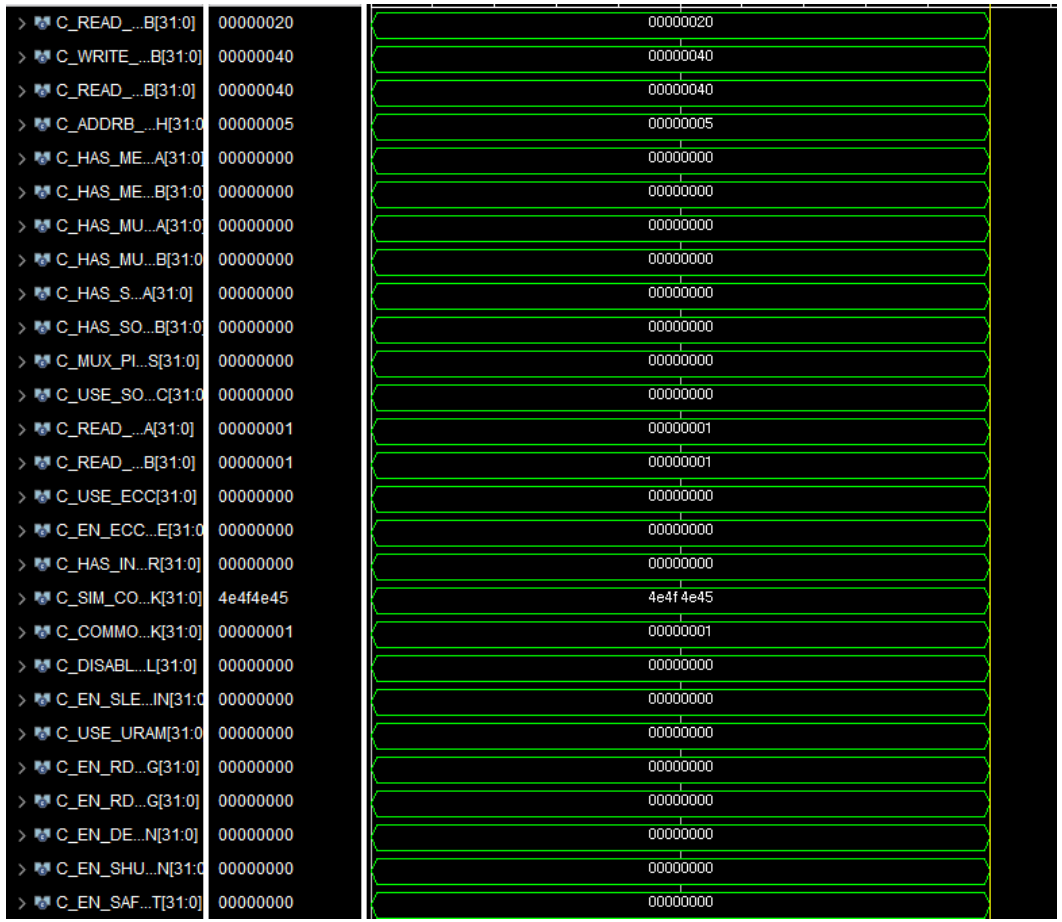
# 7-2) Simulation



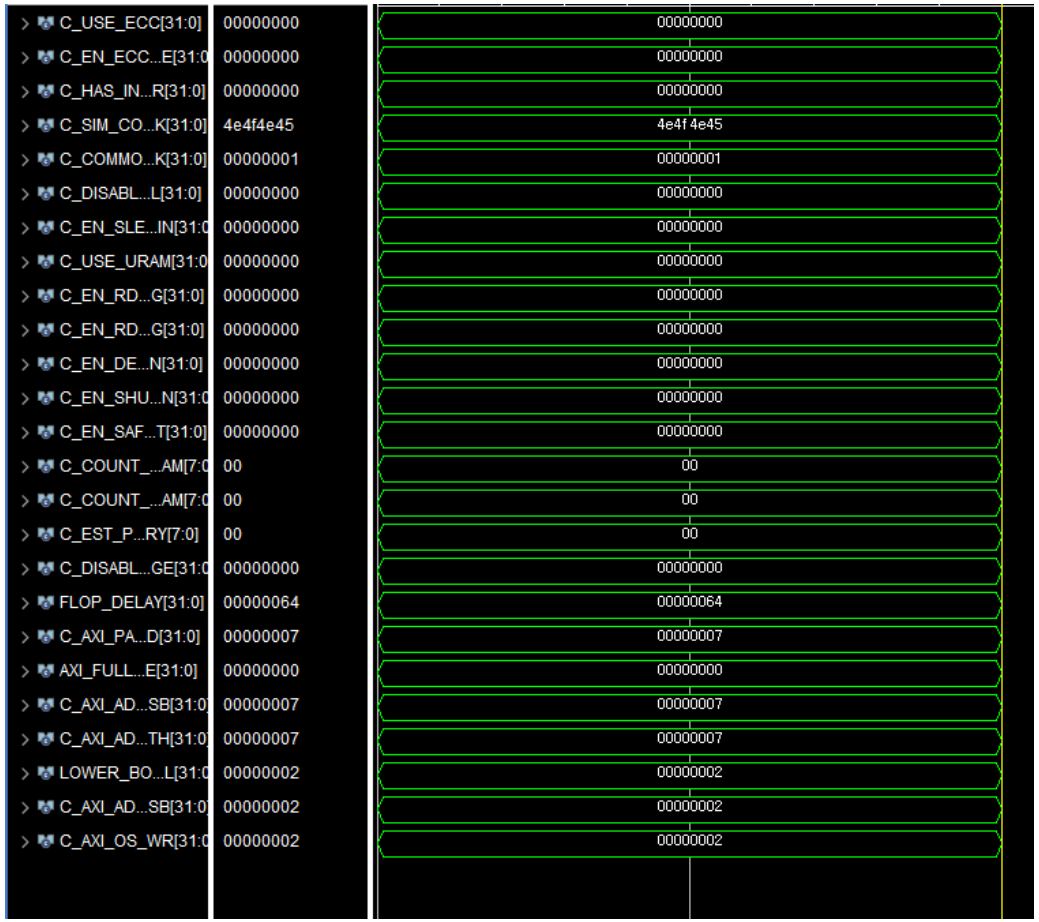
# 7-2) Simulation



# 7-2) Simulation



# 7-2) Simulation



# 7-3) Problems & Supplementations

1. BRAM의 파일이 .v 파일이 아니라 .xci 파일로 생성되는 원인 찾기

2. BRAM에 대해 더 많이 공부하여 [step.6]에서 설계한 모듈에 적합한 BRAM 디자인

3. 기본적으로 내장된 testbench 코드 대신 [step.6]에서 설계한 모듈에 맞는 testbench 코드 코딩

# The Goals

1. ~~Understanding the process of 'hit finding algorithm'~~
2. ~~Designing a hardware process~~
3. ~~Coding the hardware process without delay in Verilog~~
4. ~~SPACIROC : 'spaciroc_datasheet_20110510'~~
5. ~~Calibration for Photon Detector~~
6. ~~Demonstrating on FPGA~~
7. ~~Study BRAM~~
8. Input clock automatically
9. Optical sensor

➔ Find out the address of the *real hit*

# The Detailed Progress

8. Input clk automatically

8-1) Problems of BRAM

8-2) how to input clk automatically

# 8-1) Problems of BRAM

- Too many ports and still not enough time to study each port's role

→ It is difficult to add BRAM to existing modules

# 8-2) How to input clk automatically

## From FPGA User Manual

### 6.1. Clock

# Global Clock

| Signal Name | FPGA Pin | I/O | Description |
|---|---|---|---|
| FPGA_CLK | R4 | Input | FPGA Clock Input (100Mhz) |

**Different types of FPGA have different Pin List**, which must be mapped differently each time the FPGA changes

# The Detailed Progress

## 9. Optical Sensor

### 9-1) why do I need an optical sensor

### 9-2) problems

# 9-1) Why do I need an optical sensor

The module designed through Verilog is confirmed to work well through simulation and FPGA.

Thus, Optical sensors must be used to verify whether this hardware module serves to properly classify the intensity of light according to the threshold.

Before dealing with gamma ray, I want to verify them with visible ray since it is possible to visually check whether the module is operating properly

# What I am struggling with right now

- I don't know how the information of light measured in PMT flows to FPGA

# PLAN for Winter Vacation

- DAQ process of MAPMT with Dr. Gihan Hong

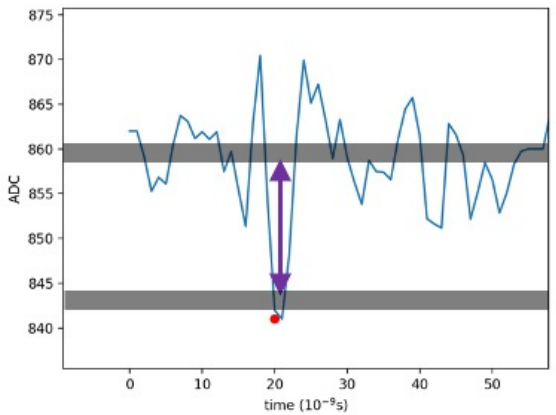
what I want to study :

→ the process of data flow

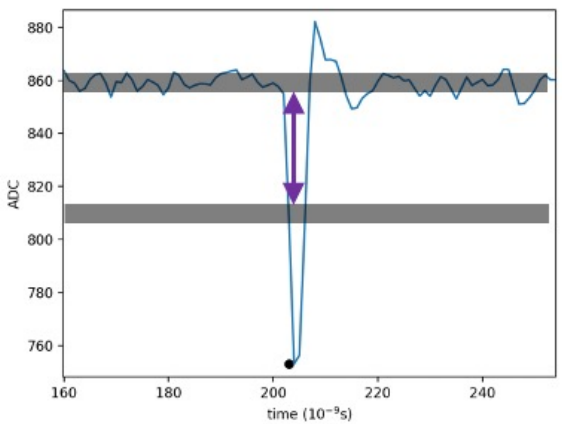
→ header structure

→ Visualizing the signa

→I

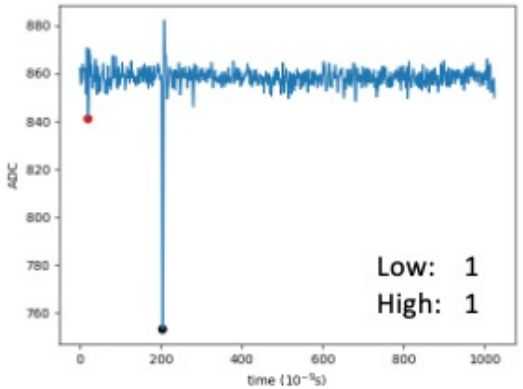


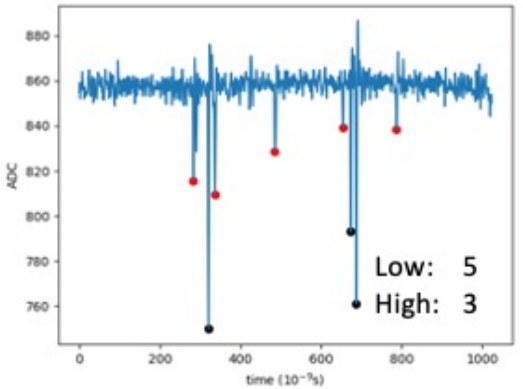

Low threshold
**15 ADC count**

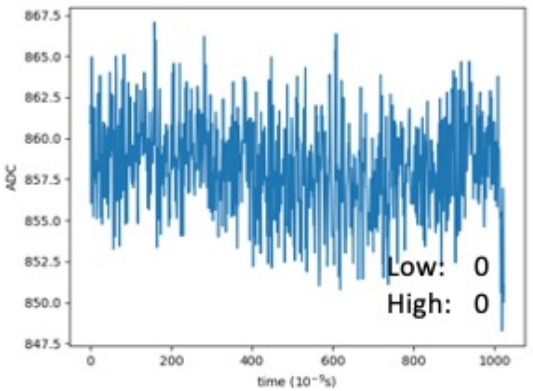


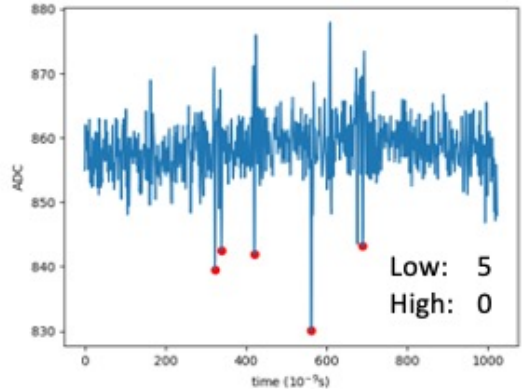

High threshold
**50 ADC count**

Signal counting



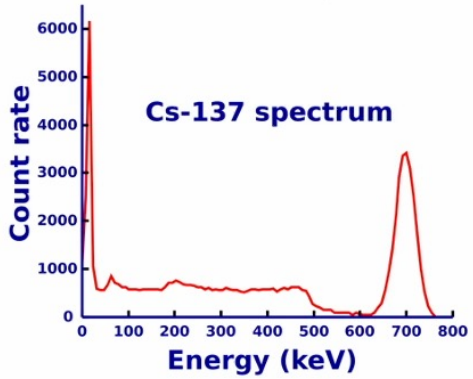



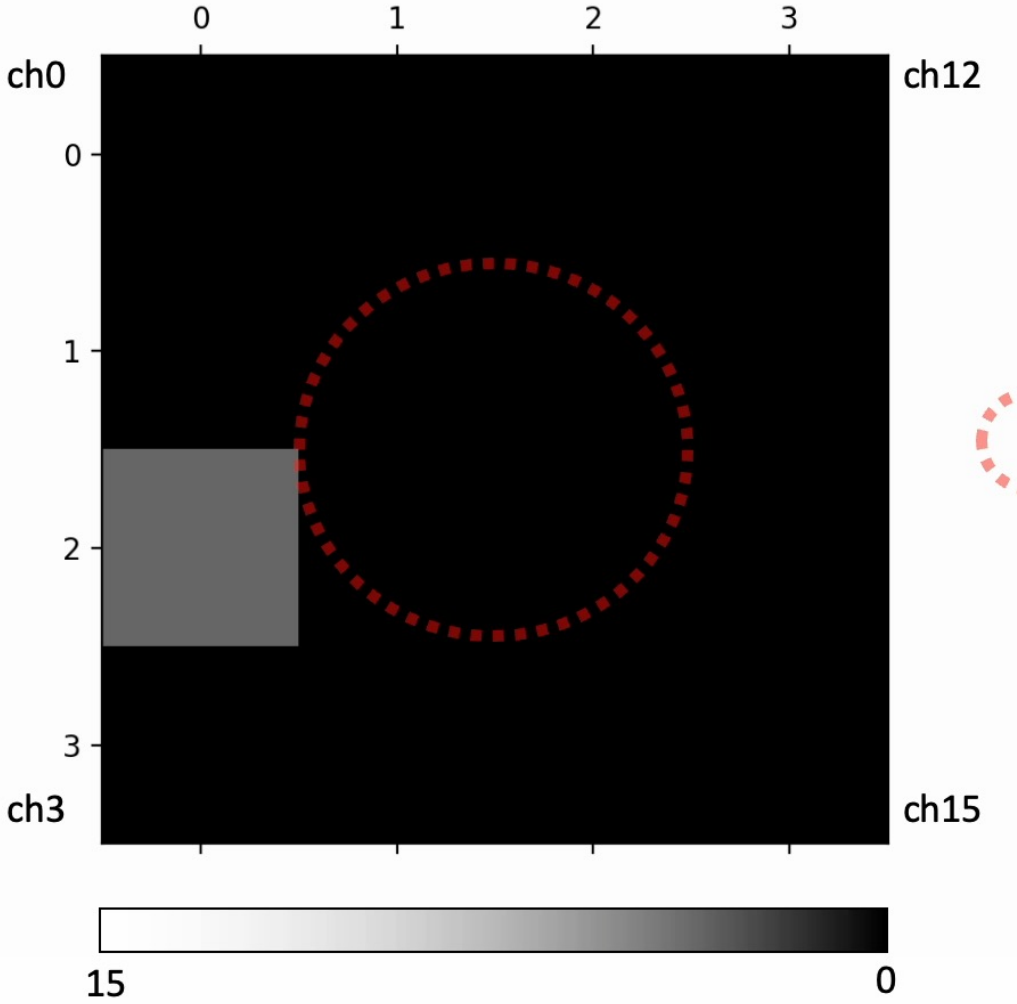




reference : *16chMAPMTreadout ,* Dr. Gihan Hong

# Multi-channel readout


$^{137}$Ce, ~0.7μCi




소스 위치

reference : *16chMAPMTreadout* , Dr. Gihan Hong

Thank you