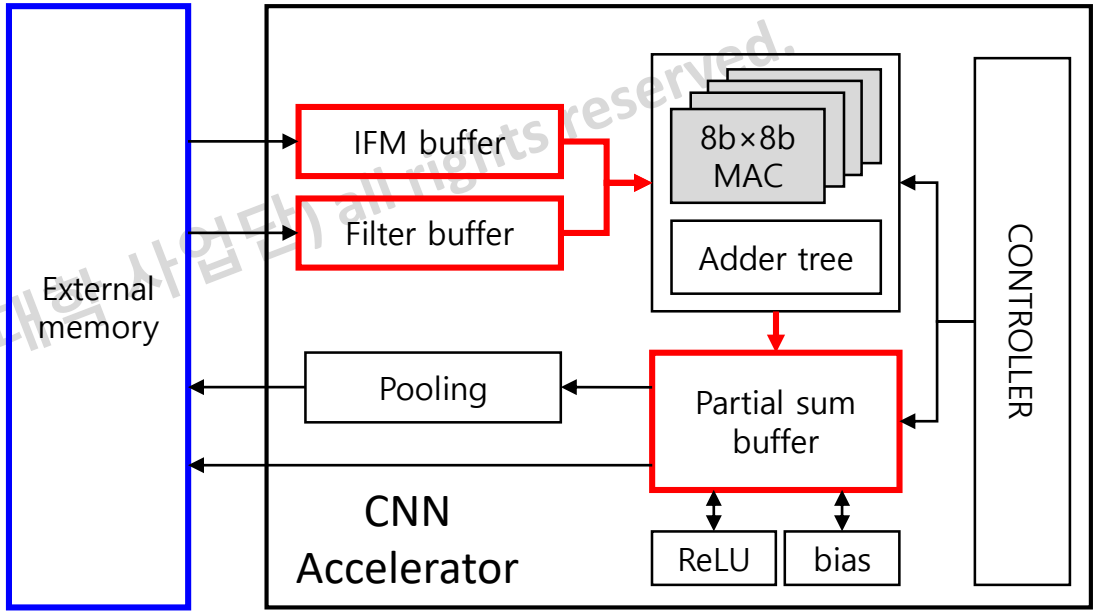
차세대 반도체 디지털 혁신공유대학 사업단
POLARIS

# On-chip buffers, Block RAM

2024.03.15 (Thursday)

# Motivation

- On-chip buffers (memory)
  - Input feature map (IFM)
  - Filters
  - Intermediate results or partial sum
- External memory



# Example of filter, bias, scale files

- Three types of buffers
  - Filter/weight: 16 lines, each line has 128 bits.
  - Bias/scale: 16 lines, each line has 16 bits

conv_weight_L1.hex

conv_weights_L1.hex | cnn_accel.v | spram.v

```
0000000000000071FFE5CDA37FD7978E
000000000000000879DD5F8013D1B545
0000000000000001EDFCF07FF1FFF40D
00000000000000D334305A2080EB8745
0000000000000033EF09D5FB8003CFF3
00000000000000FEEE02FC7FF6FAD2F8
00000000000000F880F7165702F4020D
00000000000000FB41CD8BB82B3480F3
0000000000000080AB00093CD20ED39F
00000000000000E2EE10AC4403C180CE
00000000000000FB0501FEBDF2F67F0D
0000000000000060EF6A77FF201FDFE
000000000000001F815BE67FD4F8800D
0000000000000193EDB928BAE1F3080
000000000000008BAB80F90D42F02400
000000000000000AF906E67FA8ED60C6
```

conv_bias_L1.hex

conv_biases_L1.hex

```
b744
1f82
0172
fe06
f911
f942
171d
f98c
066a
0199
f8b5
fd24
d332
e45d
0232
fe6b
```

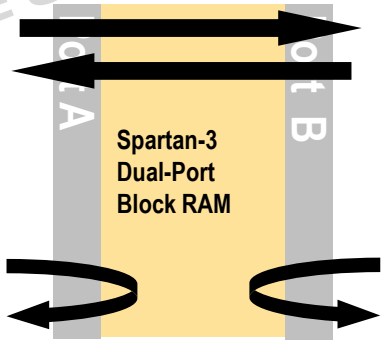
conv_scale_L1.hex

conv_scales_L1.hex

```
005f
0067
016c
00aa
007a
0136
00cb
0079
006b
00a0
0135
0107
004d
006a
00af
00ba
```

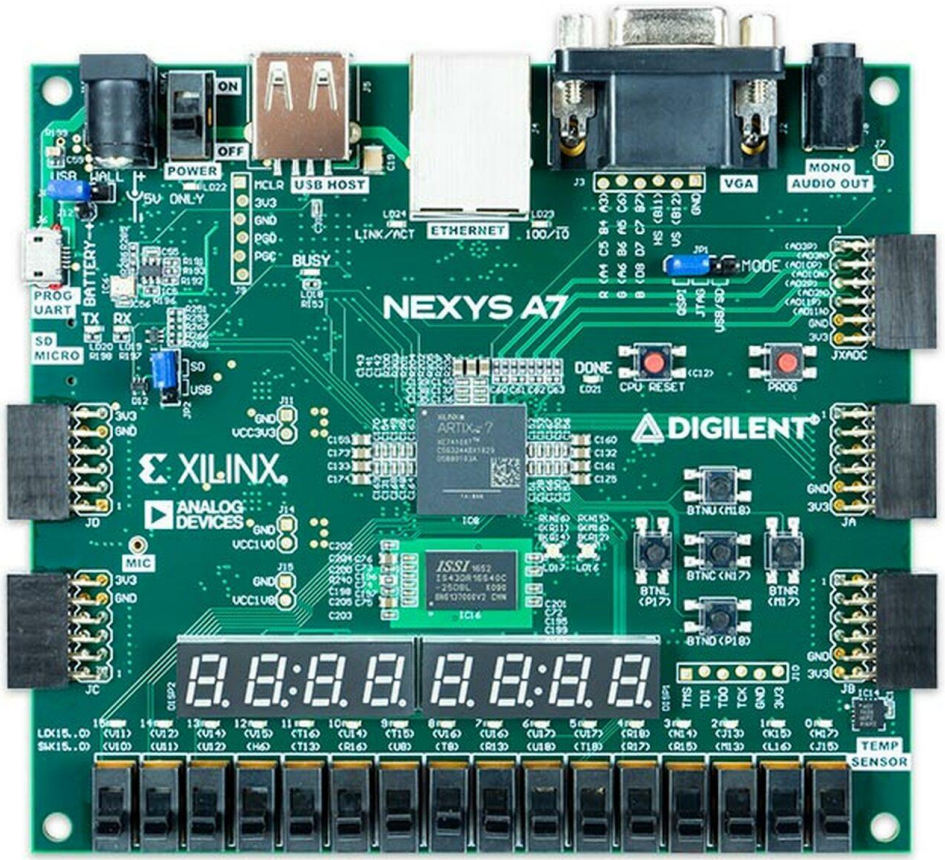
# Block RAM

- Most efficient memory implementation
  - Dedicated blocks of memory
- Ideal for most memory requirements
  - 4 to 104 memory blocks
    - **18 kbits = 18,432 bits per block (16 k without parity bits)**
  - Use multiple blocks for larger memories
- Builds both single and true dual-port RAMs
- Synchronous write and read (different from distributed RAM)



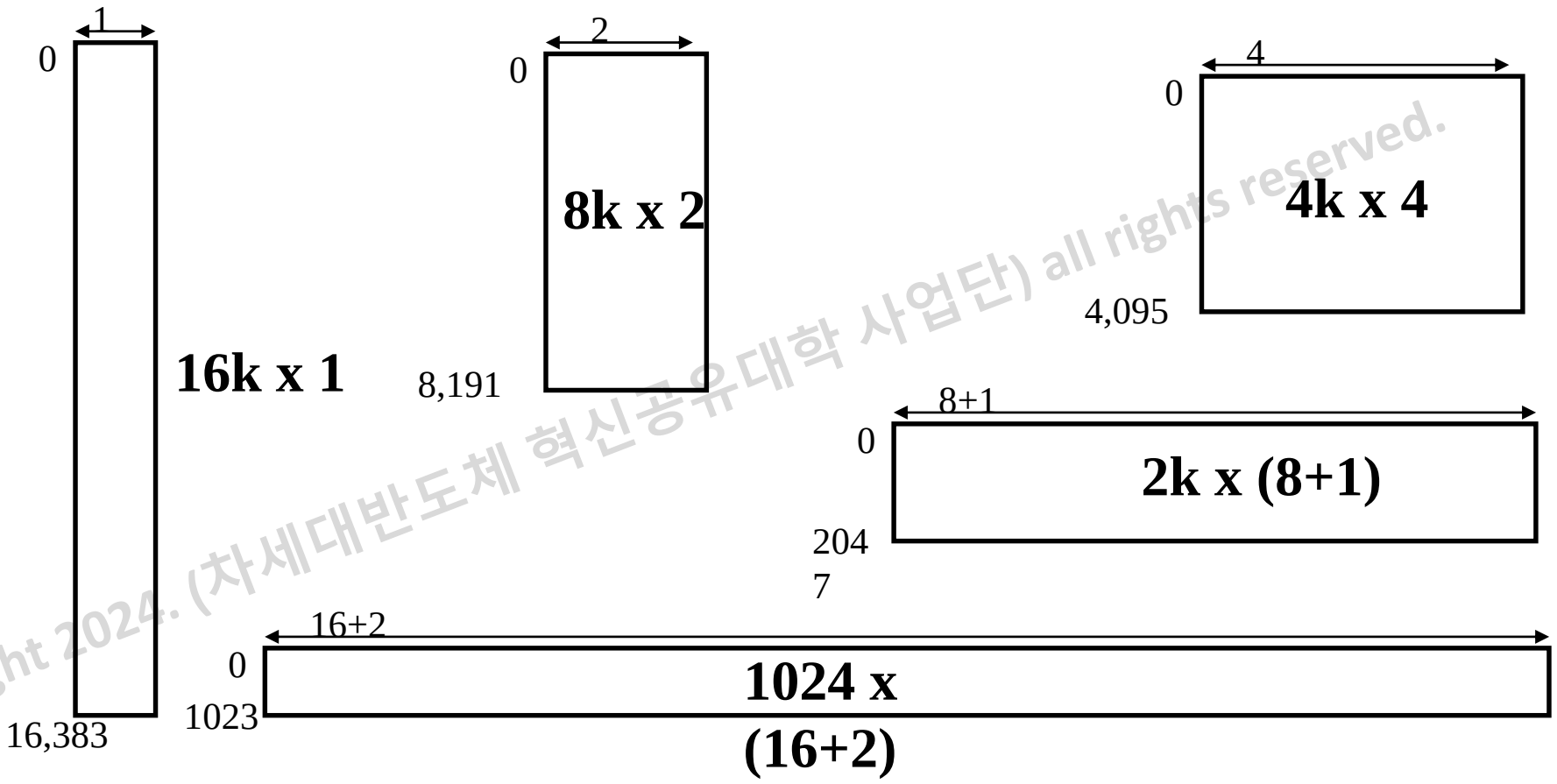
# Nexys A7 FPGA board

- Xilinx Artix-7 FPGA XC7A100T-1CSG324C
- 15,850 logic slices
  - Each with four 6-input LUTs and 8 FFs
- 4,860 Kbits of fast block RAM
  - 270 block RAMs
- 240 DSP slices (constrained to To, Ti)
  - Dedicated to multiplication and accumulation (MAC)
- Internal clock speeds exceeding 450 MHz
- 128 MB DDR2 Memory
- USB-JTAG port for FPGA programming and communication



# Block RAM Port Aspect Ratios

- Block RAM can have various configurations (port aspect ratios)



# Block RAM Port Aspect Ratios

- 18Kb BRAM can be configured
  - RAM_512×36 (512 words, each word has 36 bits).
  - RAM_1K×18 (1024 words, each word has 18 bits).
  - RAM_2K×9 (2048 words, each word has 9 bits).
- 16Kb BRAM can be configured
  - RAM_4K×4 (4096 words, each word has 4 bits).
  - RAM_8K×2 (8192 words, each word has 2 bits).
  - RAM_16K×1 (16384 words, each word has 1 bits).



| Organization | Memory Depth | Data Width | Parity Width | DI/DO | DIP/DOP | ADDR | Single-Port Primitive | Total RAM Kbits |
|---|---|---|---|---|---|---|---|---|
| 512x36 | 512 | 32 | 4 | (31:0) | (3:0) | (8:0) | RAMB16_S36 | 18K |
| 1Kx18 | 1024 | 16 | 2 | (15:0) | (1:0) | (9:0) | RAMB16_S18 | 18K |
| 2Kx9 | 2048 | 8 | 1 | (7:0) | (0:0) | (10:0) | RAMB16_S9 | 18K |
| 4Kx4 | 4096 | 4 | - | (3:0) | - | (11:0) | RAMB16_S4 | 16K |
| 8Kx2 | 8192 | 2 | - | (1:0) | - | (12:0) | RAMB16_S2 | 16K |
| 16Kx1 | 16384 | 1 | - | (0:0) | - | (13:0) | RAMB16_S1 | 16K |

# Lab 1: Single-port RAM

- Lab 1: Single-port RAM (spram): 16 bits per word, 6240 words
  - Use a single-port RAM
  - Generate a dedicated RAM using Vivado IP generator
  - Initialize a RAM using a coe file
  - Test bench

CONV20_W.coe

```
memory_initialization_radix=16;
memory_initialization_vector=
fe00fe01    ,
fd01ffff    ,
00000303    ,
00fdfe02    ,
0005ff00    ,
00fefd00    ,
0000fe00    ,
01000000    ,
fdff00fb    ,
01fefffb    ,
ffff0201    ,
00fa0001    ,
010400fd    ,
0500ff00    ,
```

# Spram and spram wrapper

- Ports
  - Clock (clka)
  - Read/Write enable (ena)
  - Write enable (we)
  - Address (addra)
  - Write data (dina)
  - Read data (douta)

- Two modes
  - Simulation
  - FPGA
    - Call a SRAM instance generated by IP generator (wrapper)
    - ***Optional: Initialize memory cell from a file***

# SPRAM: Simulation

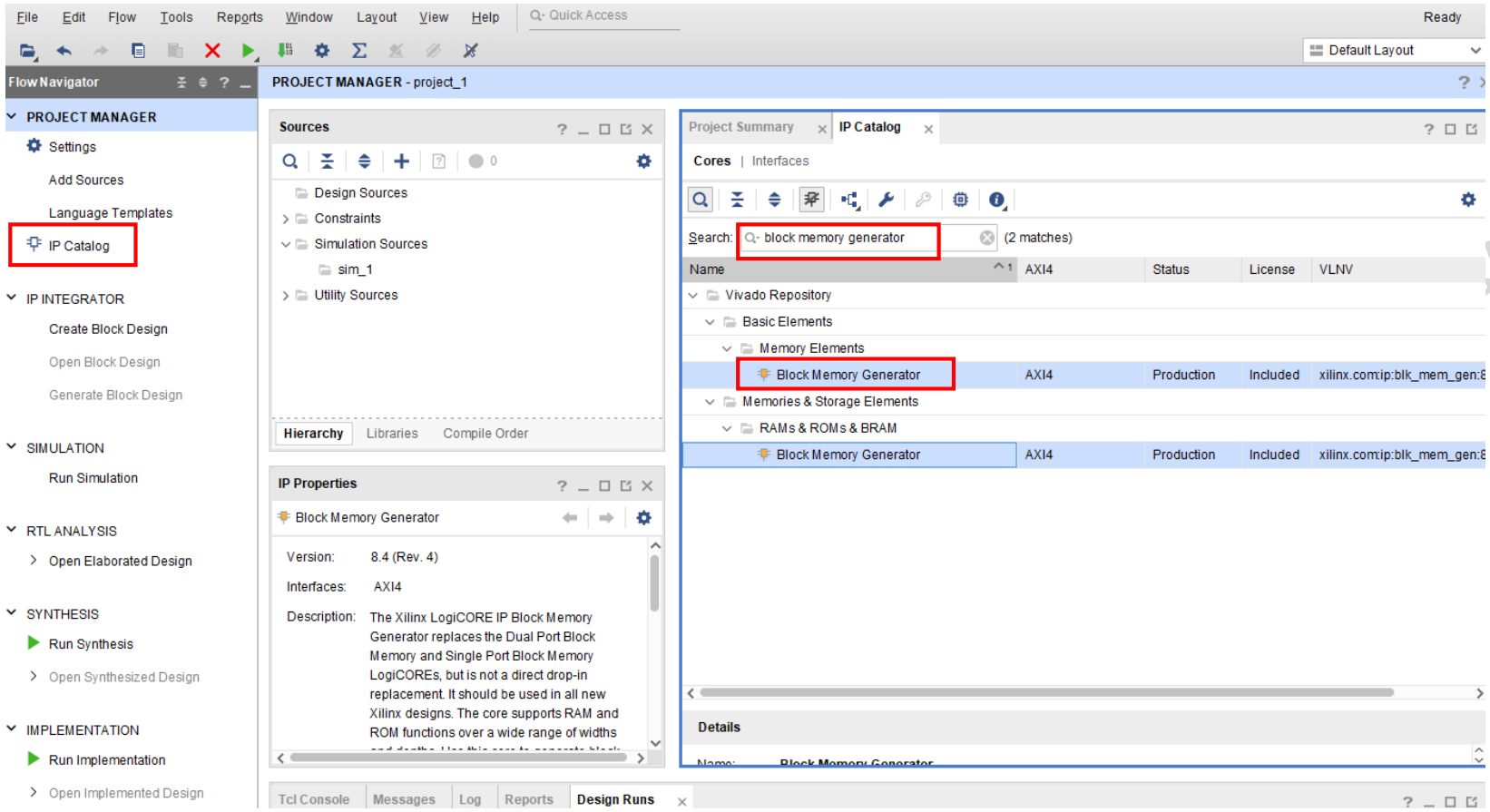
- Memory cell (mem)
  - Data width (DW)
    - Filter: 128, scale/bias: 16
  - Depth
    - Example: 16 lines
- Ports
  - Clock (clka)
  - Read/Write enable (ena)
  - Write enable (wea)
  - Address (addra)
  - Write data (dina)
  - Read data (douta)

- Read operation
  - If(ena)
    - douta <= mem[addra]

- Write operation
  - If(ena & wea)
    - mem[addra] <= dina

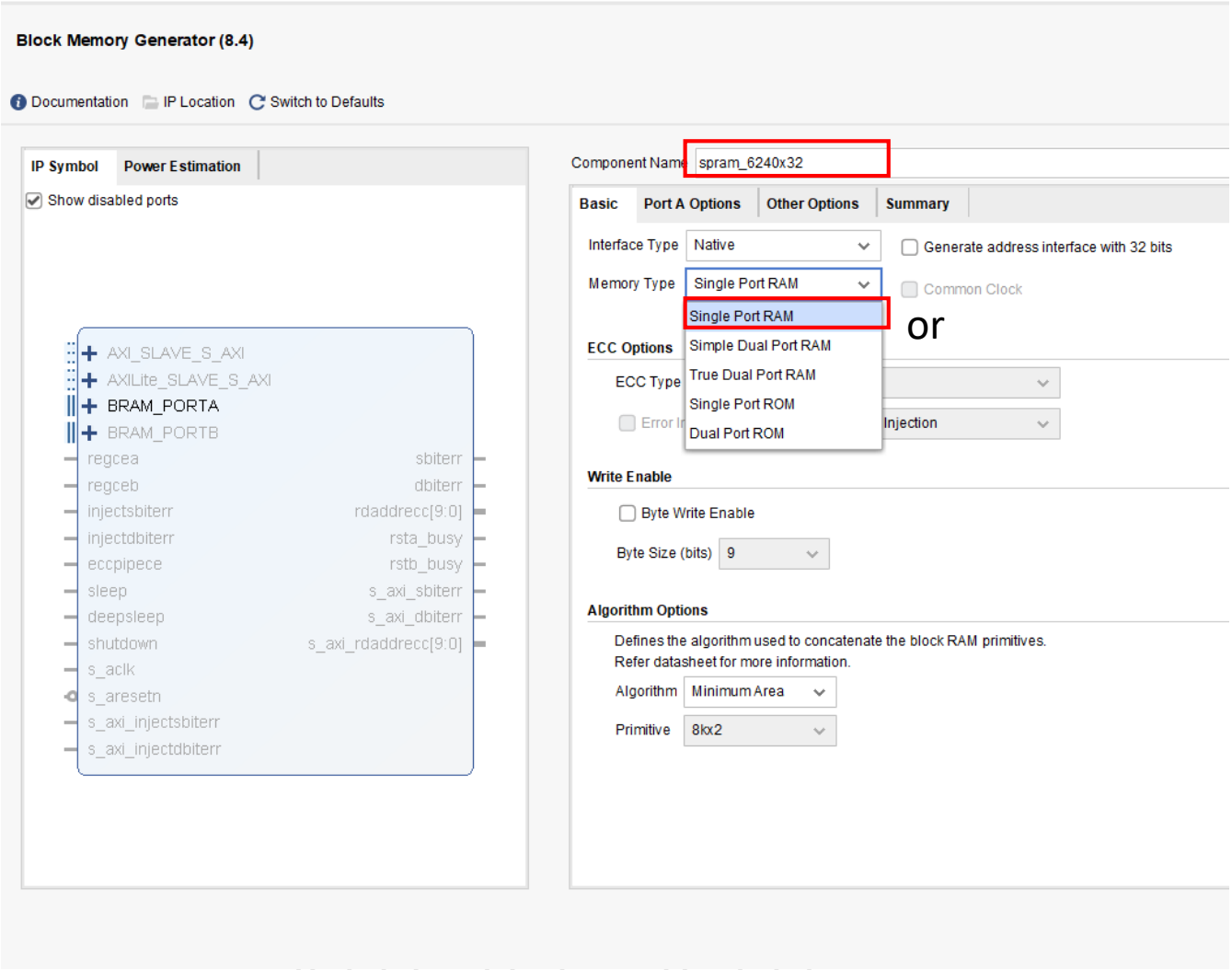
* Single-port
  - Read/wirte operations share "addra"
  - Only read or write occurs at a time

# FPGA: How to make BRAM IP?




- 1. click IP catalog
- 2. search the "block memory generator"
- 3. double click the "block memory generator"

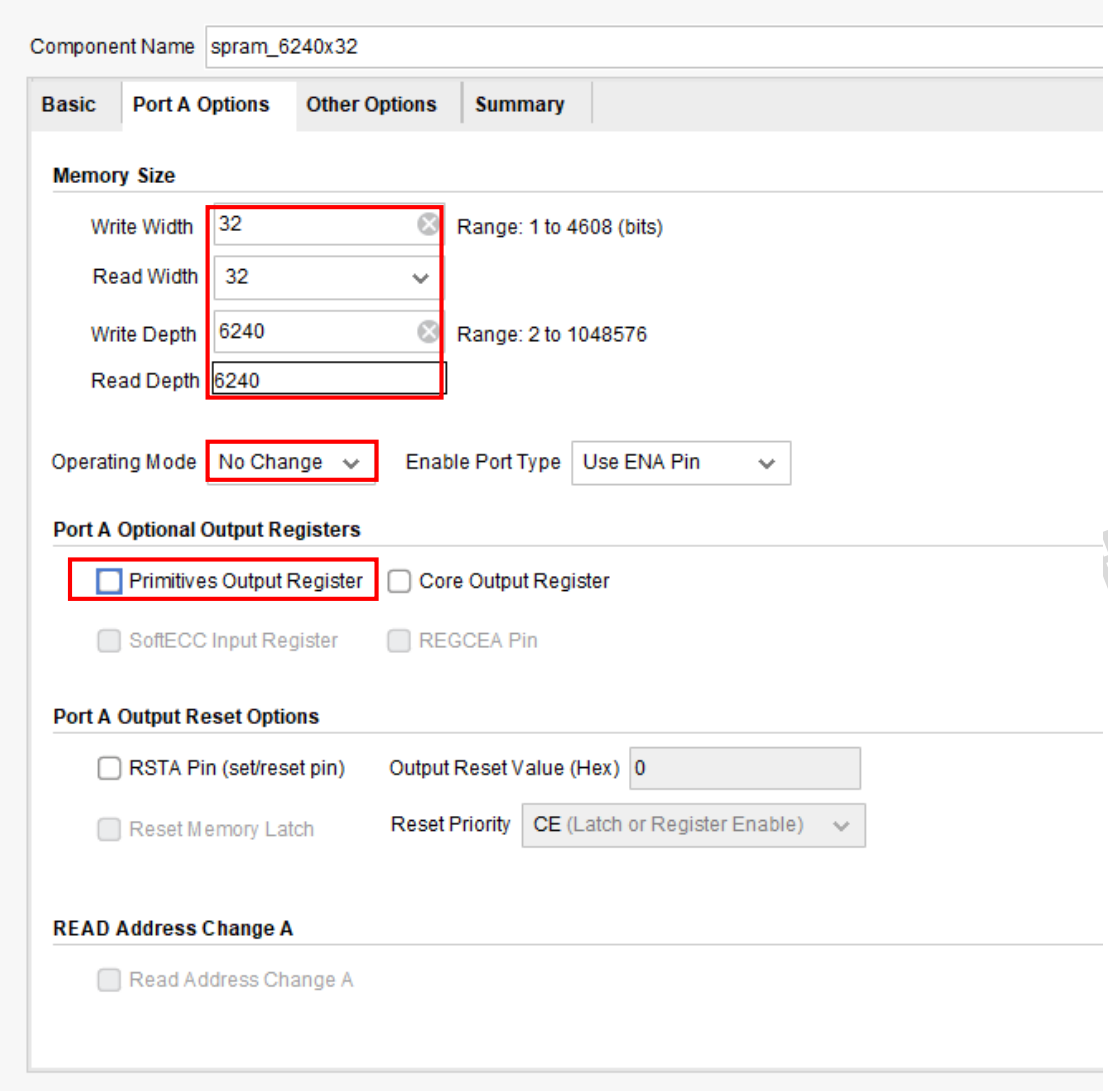
# How to make BRAM IP?




- 4. Enter the same name that you declare on your code
- 5. Select the memory type

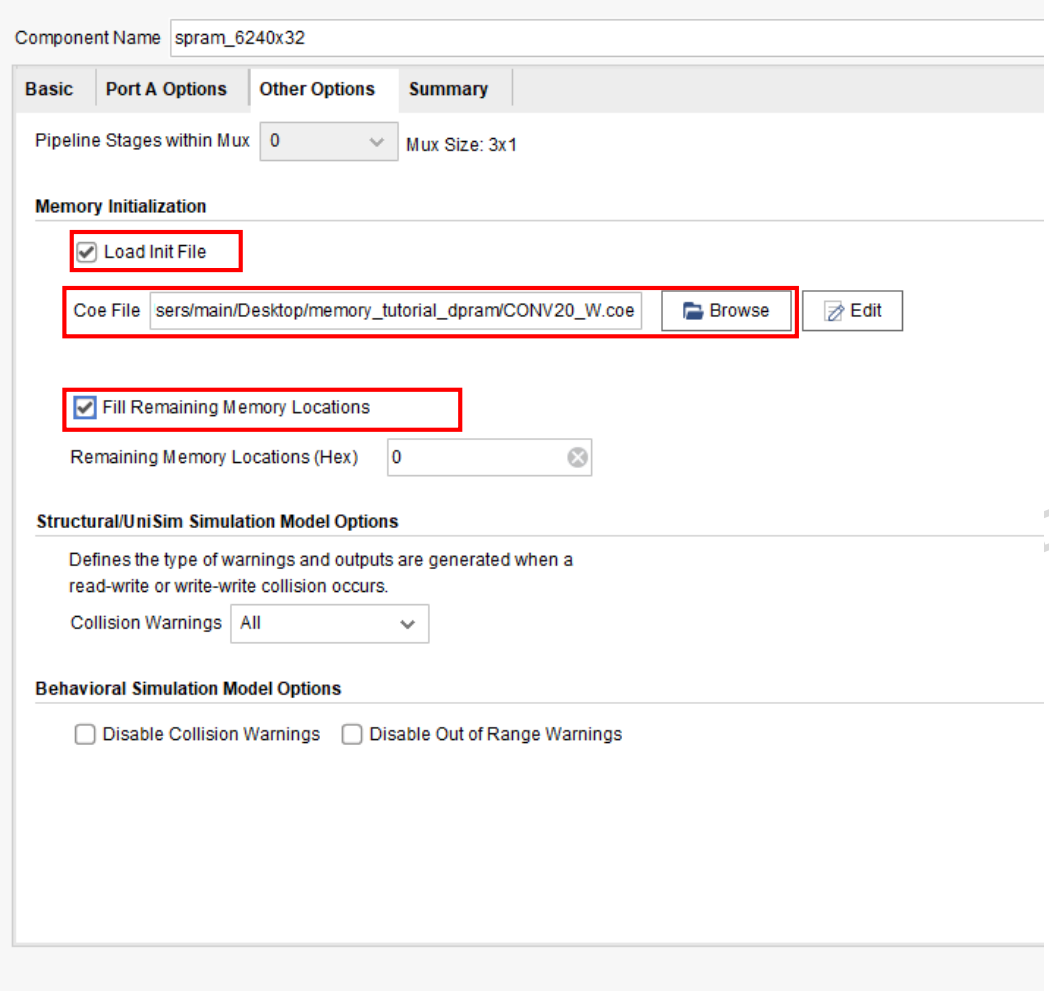
  For reference, in given file, you will use Single Port Ram (spram) and Dual Port Ram(dpram)

# How to make BRAM IP?




- 6. Enter the proper width and depth of bram
- 7. Operating mode : No change
- 8. Primitives output register : deselect

# How to make BRAM IP?




- 9. click load init file
- 10. select the coe file
- 11. check the fill remaining memory locations
- 12. click ok button

* Memory initialization : initialize the bram data to selected "~.COE" file

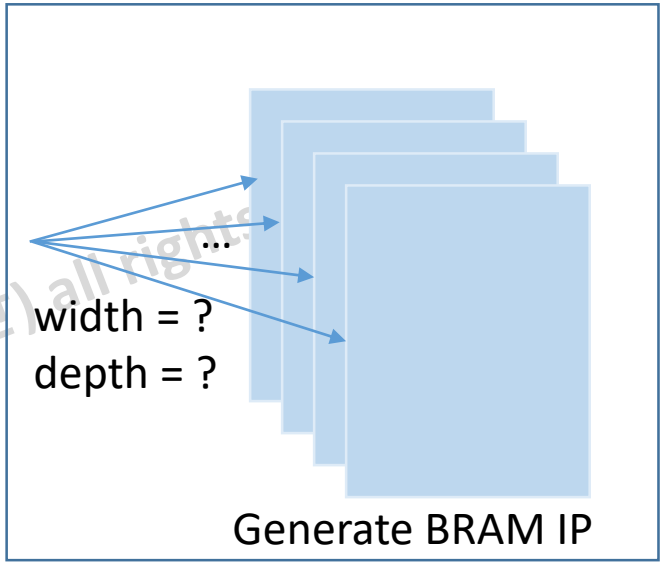
# Wrapper (spram_wrapper.v)

- Wrapper : wrapper chooses whether to make the BRAM IP or use the memory modeling (register)

```
`define FPGA 1
module spram_wrapper (
    clk,                // clock
    addr,               // input address
    we,                 // input write enable
    cs,                 // input chip-select
    wdata,              // input write data
    rdata               // output read-out data
);
```

wrapper

‘define FPGA = 1




// ‘define FPGA = 1

```
//`define FPGA 1
module spram_wrapper (
    clk,                // clock
    addr,               // input address
    we,                 // input write enable
    cs,                 // input chip-select
    wdata,              // input write data
    rdata               // output read-out data
);
```



Memory modeling

* memory modeling use “register”

# Wrapper (spram_wrapper.v) - parameter

- Parameter
  - DW : data bit width per word
  - AW : address bit width
  - DEPTH : word length

```
//-----------------------------------------------------------------------+
// Declare parameters
//-----------------------------------------------------------------------+
// output parameters
parameter DW = 64;          // data bit-width per word
parameter AW = 8;           // address bit-width
parameter DEPTH = 256;      // depth, word length
parameter N_DELAY = 1;
```

Ex)

DW = 4

DEPTH = 8

AW = $\log_2$DEPTH
= 3

| |
|---|
| 0 1 0 1 |
| 0 0 0 1 |
| 0 1 0 1 |
| 1 1 0 1 |
| 0 0 0 1 |
| 0 0 1 1 |
| 0 1 0 1 |
| 0 0 0 1 |

# Wrapper (spram_wrapper.v) - parameter

- Modify Parameter : you can customize the BRAM by modifying parameters

- 1. using BRAM IP (define FPGA = 1)
  - Set parameter
  - Add the proper "else if ~" code
  - Make BRAM IP

- 2. using memory modeling
  - Just set parameter

```
// output parameters
parameter DW = 32;
parameter AW = 13;
parameter DEPTH = 6240;
parameter N_DELAY = 1;
```

# Wrapper (spram_wrapper.v) - parameter

- Modify Parameter : you can customize the BRAM by modifying parameters

- 1. using BRAM IP (define FPGA = 1)
  - Set parameter
  - Add the proper "else if ~" code
  - Make BRAM IP

- 2. using memory modeling
  - Just set parameter

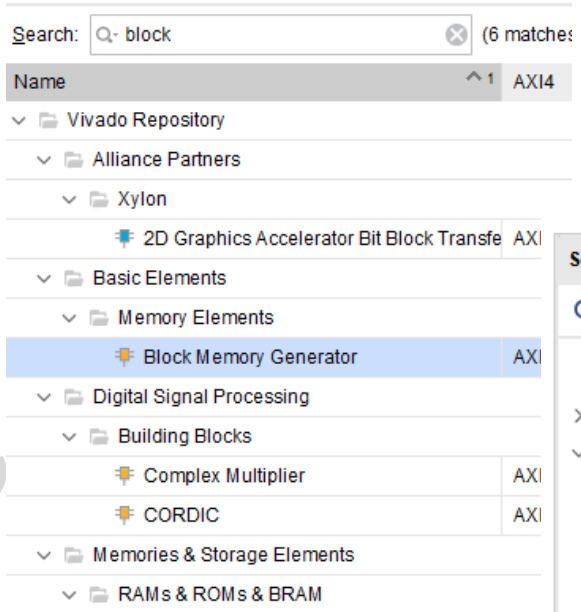
```
else if((DEPTH == 208) && (DW == 256)) begin: gen_spram_208x256
    spram_208x256 u_spram_208x256(
        // write
        .clka(clk),
        .ena(cs),
        .wea(we),
        .addra(addr),
        .dina(wdata),
        // read-out
        .douta(rdata)
    );
end
else if((DEPTH == 6240) && (DW == 32)) begin: gen_spram_6240x32
    spram_6240x32 u_spram_6240x32(
        // write
        .clka(clk),
        .ena(cs),
        .wea(we),
        .addra(addr),
        .dina(wdata),
        // read-out
        .douta(rdata)
    );
end
```

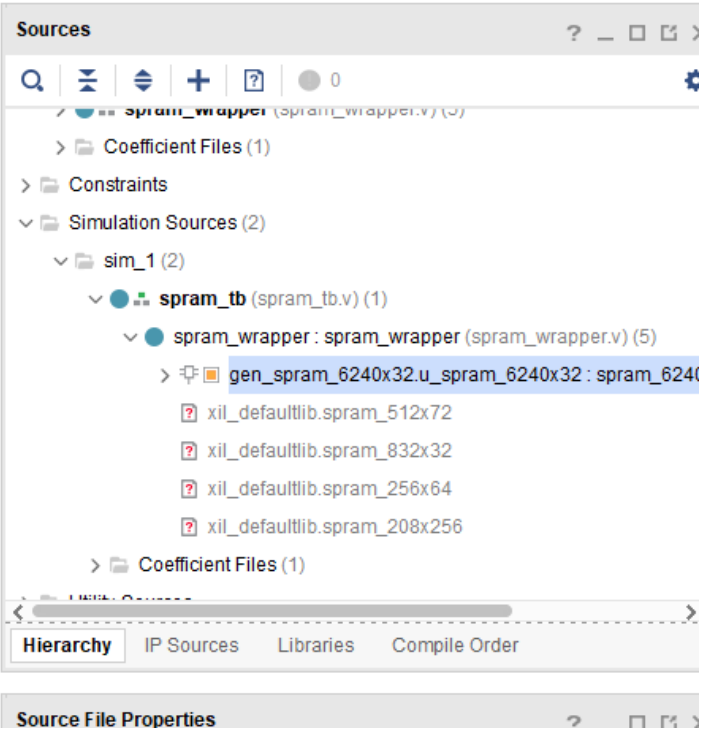
# Wrapper (spram_wrapper.v) - parameter

- Modify Parameter : you can customize the BRAM by modifying parameters

- 1. using BRAM IP (define FPGA = 1)
  - Set parameter
  - Add the proper "else if ~" code
  - Make BRAM IP

- 2. using memory modeling
  - Just set parameter



# Wrapper (spram_wrapper.v) - parameter

- Modify Parameter : you can customize the BRAM by modifying parameters

- 1. using BRAM IP (define FPGA = 1)
  - Set parameter
  - Add the proper "else if ~" code
  - Make BRAM IP

- 2. using memory modeling
  - Just set parameter

```
// output parameters
parameter DW = 32;
parameter AW = 13;
parameter DEPTH = 6240;
parameter N_DELAY = 1;
```

# Explanation - spram_wrapper.v

- Memory modeling : just operate like BRAM IP but it's actually register

```
//-----------------------------------------------------------------
// Memory modeling
//-----------------------------------------------------------------
reg [DW-1          :   0]      mem[0:DEPTH-1]; // Memory cell
// Write
always @(posedge clk) begin
    if(cs && we)            mem[addr] <= wdata;
end
// Read
generate
   if(N_DELAY == 1) begin: gen_delay_1
      always @(posedge clk)
         if (cs && !(|we)) rdata_o <= mem[addr];

      assign rdata = rdata_o;
   end
   else begin: gen_delay_n
      reg [N_DELAY*DW-1:0] rdata_r;

      always @(posedge clk)
         if (cs && !(|we)) rdata_r[0*DW+:DW] <= mem[addr];

      always @(posedge clk) begin: delay
         integer i;
         for(i = 0; i < N_DELAY-1; i = i+1)
            if(cs && !(|we))
               rdata_r[(i+1)*DW+:DW] <= rdata_r[i*DW+:DW];
      end
      assign rdata = rdata_r[(N_DELAY-1)*DW+:DW];
   end
endgenerate
```

Make "mem" register (data storage)

Data write to "mem"

Data read from "mem"
: 1 cycle delay

Data read from "mem"
: more than 1 cycle delay
(not used for this example! Just ignore)

Support N cycle data delay
: 1 cycle delay for this example

Assign "mem" output port

# Test bench (spram_tb.v)

- Test bench for bram operation
  - Operation
    - 1. Read the initialized value (initialized by COE file)
    - 2. Write the test data set (width = 32, depth = 16)
    - 3. Read the changed value

```
// test data set: width =  DW , depth = 16
test_data[0] = 32'h00000000;
test_data[1] = 32'h11111111;
test_data[2] = 32'h22222222;
test_data[3] = 32'h33333333;
test_data[4] = 32'h44444444;
test_data[5] = 32'h55555555;
test_data[6] = 32'h66666666;
test_data[7] = 32'h77777777;
test_data[8] = 32'h88888888;
test_data[9] = 32'h99999999;
test_data[10] = 32'haaaaaaaa;
test_data[11] = 32'hbbbbbbbb;
test_data[12] = 32'hcccccccc;
test_data[13] = 32'hdddddddd;
test_data[14] = 32'heeeeeeee;
test_data[15] = 32'hffffffff;
```

write

```
// ---------- read operation ---------- //
for(i=0;i<DEPTH;i=i+1) begin
    #(8*CLK_HALF_CYCLE)
    cs = 1'b1;
    addr = i;
    #(4*CLK_HALF_CYCLE)
    cs = 1'b0;
end

// ---------- write operation ---------- //
for(i=0;i<16;i=i+1) begin
    #(8*CLK_HALF_CYCLE)
    cs = 1'b1;
    we = 1'b1;
    addr = i;
    wdata = test_data[i];
    #(2*CLK_HALF_CYCLE)
    cs = 1'b0;
    we = 1'b0;
end
```

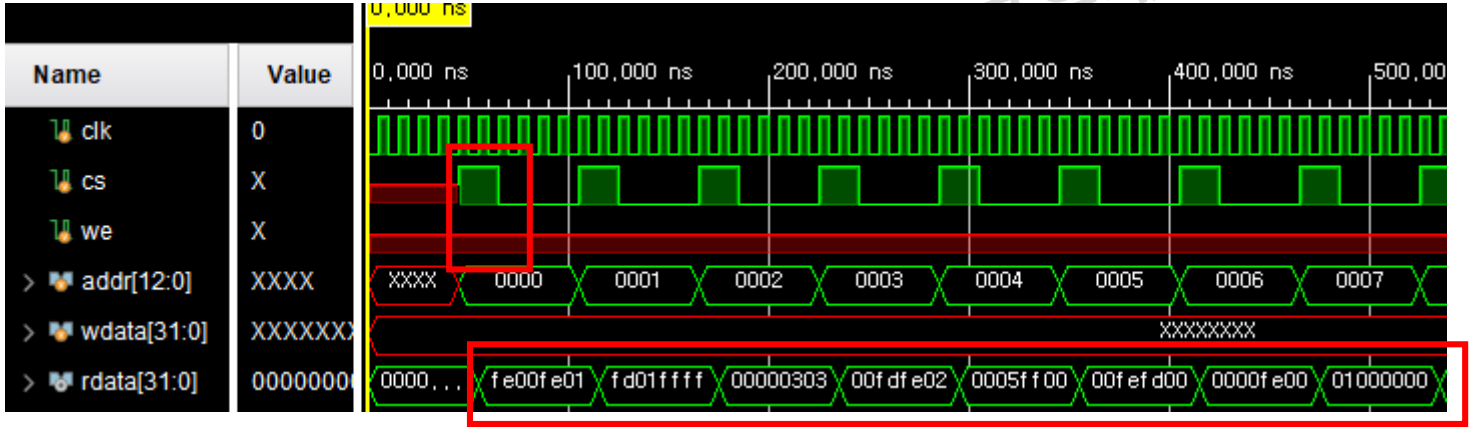
read

```
// ---------- read operation ---------- //
for(i=0;i<DEPTH;i=i+1) begin
    #(8*CLK_HALF_CYCLE)
    cs = 1'b1;
    addr = i;
    #(4*CLK_HALF_CYCLE)
    cs = 1'b0;
end
```

# Test bench (spram_tb.v) - waveform

- 1. Read the initialized value (initialized by COE file)

```
// ---------- read operation ---------- //
for(i=0;i<DEPTH;i=i+1) begin
    #(8*CLK_HALF_CYCLE)
    cs = 1'b1;
    addr = i;
    #(4*CLK_HALF_CYCLE)
    cs = 1'b0;
end
```




CONV20_W.coe

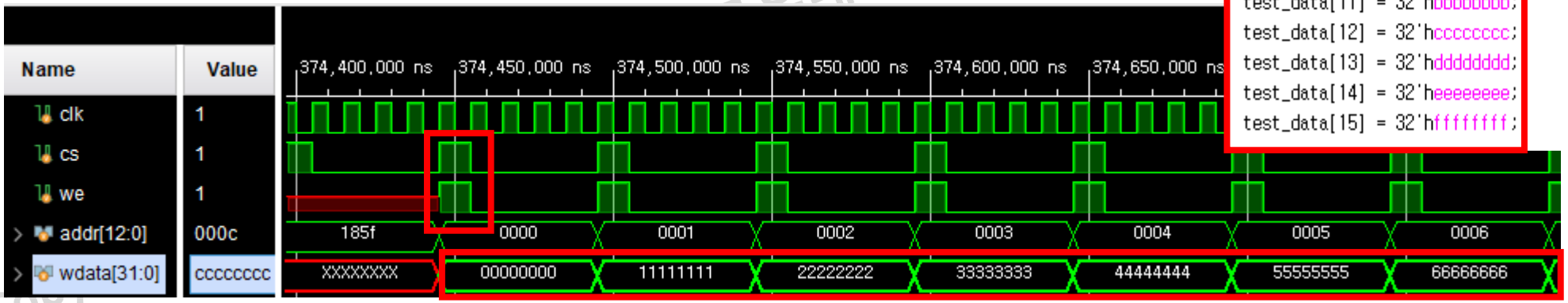
```
memory_initialization_radix=16;
memory_initialization_vector=
fe00fe01    ,
fd01ffff    ,
00000303    ,
00fdfe02    ,
0005ff00    ,
00fefd00    ,
0000fe00    ,
01000000    ,
fdff00fb    ,
01fefffb    ,
ffff0201    ,
00fa0001    ,
010400fd    ,
0500ff00    ,
```

# Test bench (spram_tb.v) - waveform

- 2. Write the test data set (width = 32, depth = 16) ;

```
// ---------- write operation ---------- //
for(i=0;i<16;i=i+1) begin
    #(8*CLK_HALF_CYCLE)
    cs = 1'b1;
    we = 1'b1;
    addr = i;
    wdata = test_data[i];
    #(2*CLK_HALF_CYCLE)
    cs = 1'b0;
    we = 1'b0;
end
```

```
// test_data set: width = DW, depth = 16
test_data[0] = 32'h00000000;
test_data[1] = 32'h11111111;
test_data[2] = 32'h22222222;
test_data[3] = 32'h33333333;
test_data[4] = 32'h44444444;
test_data[5] = 32'h55555555;
test_data[6] = 32'h66666666;
test_data[7] = 32'h77777777;
test_data[8] = 32'h88888888;
test_data[9] = 32'h99999999;
test_data[10] = 32'haaaaaaaa;
test_data[11] = 32'hbbbbbbbb;
test_data[12] = 32'hcccccccc;
test_data[13] = 32'hdddddddd;
test_data[14] = 32'heeeeeeee;
test_data[15] = 32'hffffffff;
```

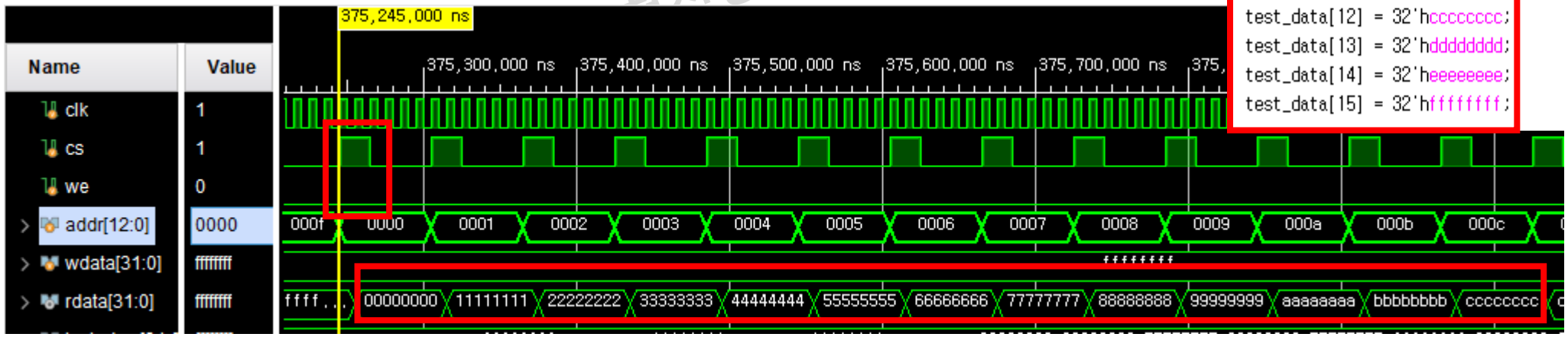


# Test bench (spram_tb.v) - waveform

- 3. Read the changed value

```
// ---------- read operation ---------- //
for(i=0;i<DEPTH;i=i+1) begin
    #(8*CLK_HALF_CYCLE)
    cs = 1'b1;
    addr = i;
    #(4*CLK_HALF_CYCLE)
    cs = 1'b0;
end
```

```
// test_data_set: width = DW, depth = 16
test_data[0] = 32'h00000000;
test_data[1] = 32'h11111111;
test_data[2] = 32'h22222222;
test_data[3] = 32'h33333333;
test_data[4] = 32'h44444444;
test_data[5] = 32'h55555555;
test_data[6] = 32'h66666666;
test_data[7] = 32'h77777777;
test_data[8] = 32'h88888888;
test_data[9] = 32'h99999999;
test_data[10] = 32'haaaaaaaa;
test_data[11] = 32'hbbbbbbbb;
test_data[12] = 32'hcccccccc;
test_data[13] = 32'hdddddddd;
test_data[14] = 32'heeeeeeee;
test_data[15] = 32'hffffffff;
```



# Lab 2: dual-port RAM

- Lab 2: Dual-port RAM (spram): 16 bits per word, 6240 words
  - Use a dual-port RAM
  - Generate a dedicated RAM using Vivado IP generator
  - Initialize a RAM using a coe file
  - Test bench

CONV20_W.coe

```
memory_initialization_radix=16;
memory_initialization_vector=
fe00fe01    ,
fd01ffff    ,
00000303    ,
00fdfe02    ,
0005ff00    ,
00fefd00    ,
0000fe00    ,
01000000    ,
fdff00fb    ,
01fefffb    ,
ffff0201    ,
00fa0001    ,
010400fd    ,
0500ff00    ,
```

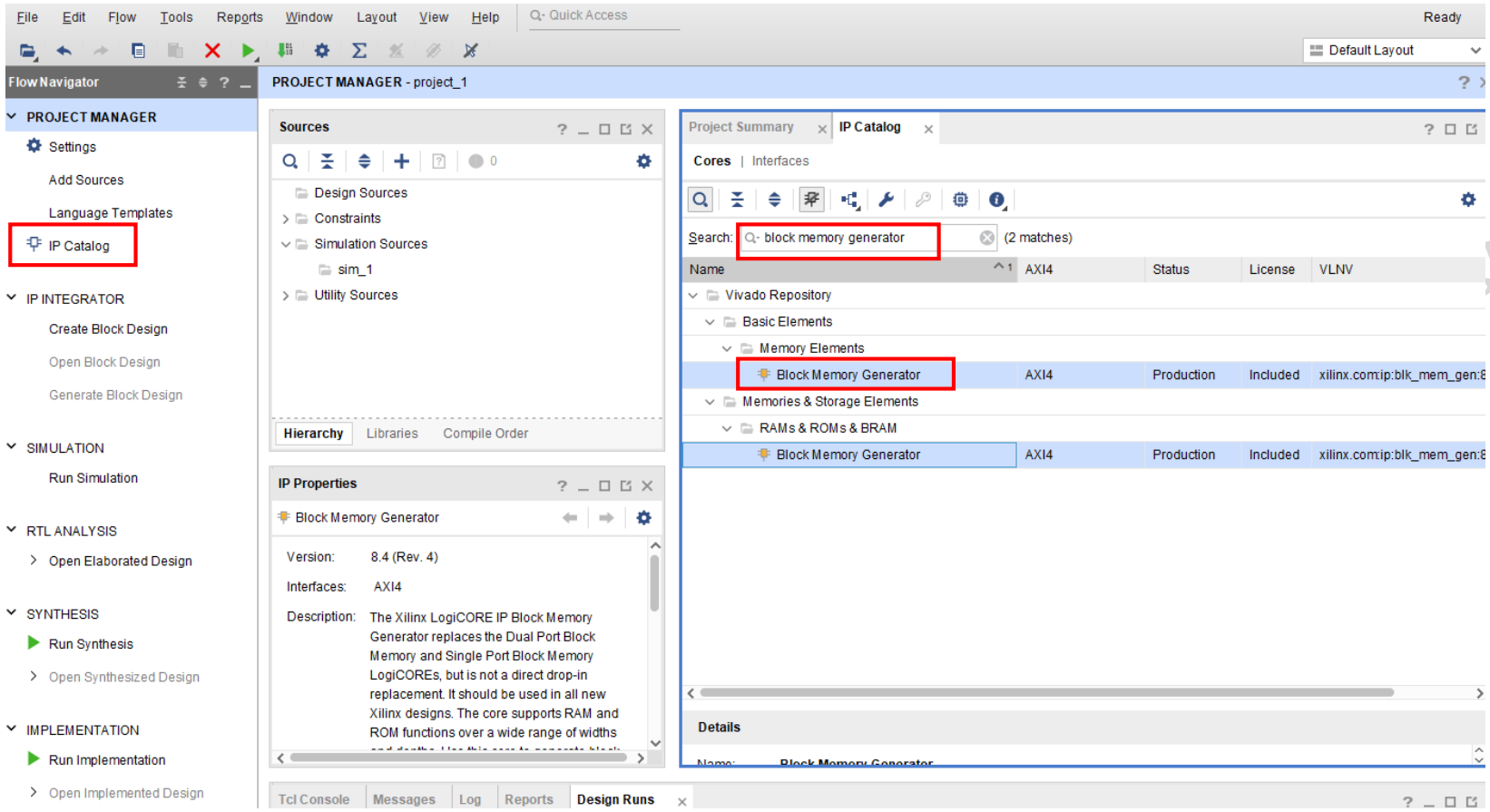
# Double-port block RAM (dpram)

- Ports
  - Port A : Write only
    - Clock (clka)
    - Read/Write enable (ena)
    - Write enable (wea)
    - Address (addra)
    - Write data (dina)
  - Port B : Read only
    - Clock (clkb)
    - Read/write enable (enb)
    - Address (addrb)
    - Read data (doutb)

- Read operation
  - If(enb)
    - doutb <= mem[addrb]
- Write operation
  - If(ena & wea)
    - mem[addra] <= dina

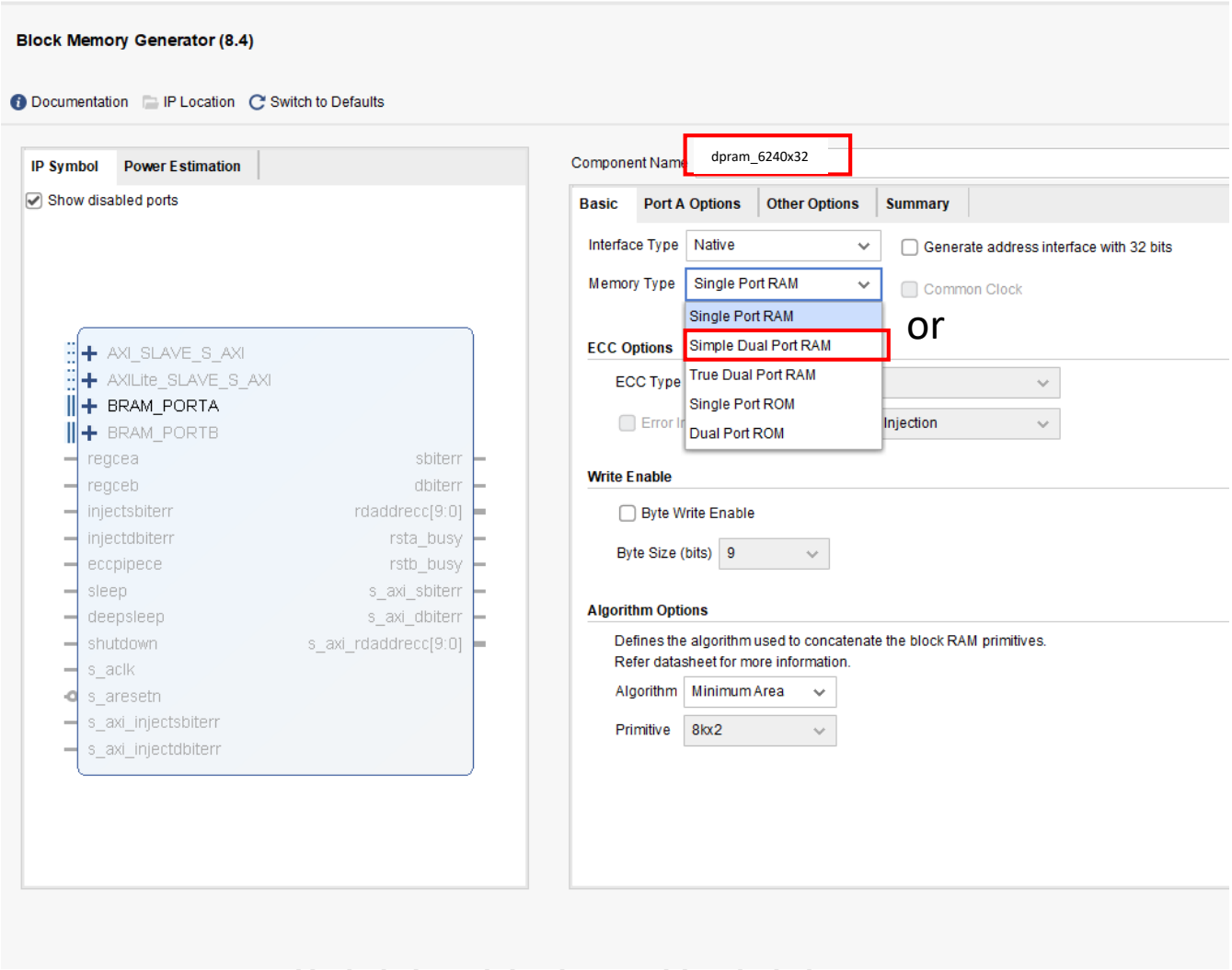
* Double-port
  - Allow read/write operations at the same time

# FPGA: How to make BRAM IP?




- 1. click IP catalog
- 2. search the "block memory generator"
- 3. double click the "block memory generator"

# How to make BRAM IP?




- 4. Enter the same name that you declare on your code
- 5. Select the memory type

  For reference, in given file, you will use Single Port Ram (spram) and Dual Port Ram(dpram)

# Double-port block RAM (dpram_tb.v)

- About wrapper, test bench… : It's almost same with spram's

- Point
  - dpram allow read/write operations at the same time!
  - dpram write the data through port A!
  - dpram read the data through port B!

These read & write can operate
at the same time, but what happen?

```
// ---------- read operation ---------- //
for(i=0;i<DEPTH;i=i+1) begin
    #(8*CLK_HALF_CYCLE)
    enb = 1'b1;
    addrb = i;
    #(4*CLK_HALF_CYCLE)
    enb = 1'b0;
end

// ---------- write operation ---------- //
for(i=0;i<16;i=i+1) begin
    #(8*CLK_HALF_CYCLE)
    ena = 1'b1;
    wea = 1'b1;
    addra = i;
    dia = test_data[i];
    #(2*CLK_HALF_CYCLE)
    ena =1'b0;
    wea = 1'b0;
end

// ---------- read operation ---------- //
for(i=0;i<DEPTH;i=i+1) begin
    #(8*CLK_HALF_CYCLE)
    enb = 1'b1;
    addrb = i;
    #(4*CLK_HALF_CYCLE)
    enb = 1'b0;
end
```