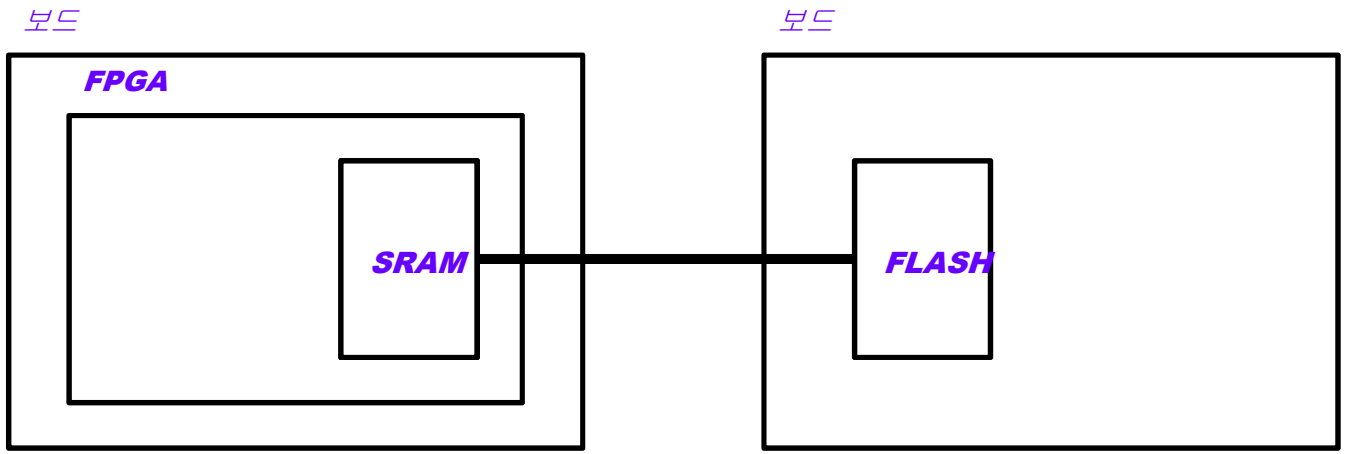
FIFO

FIFO 의 필요성

- MPW 를 진행할 때 중요한 이슈 중의 하나는 메모리
  - 시스템에 ROM 을 사용한 경우, 칩 동작 여부를 떠나서 특정 동작만 수행할 뿐임
  - 시스템의 다양한 동작을 위해서는 RAM 을 사용해야 함

- RAM 을 칩 안에 구성한 경우, 외부에서 w/r 이 가능해야 함

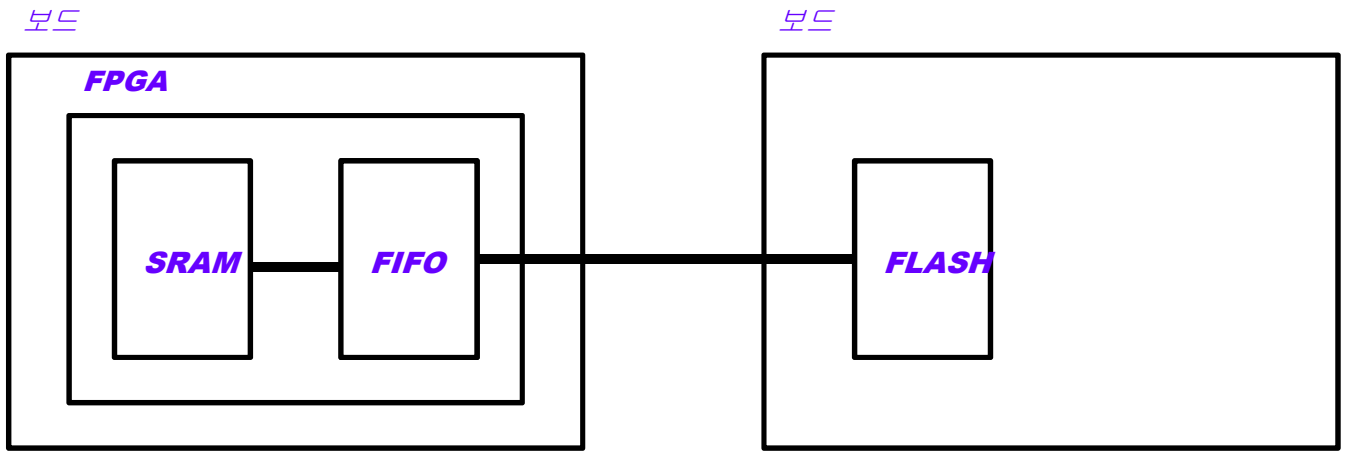
- 칩 외부에 Flash 개념의 메모리가 있고 구동 SW 가 들어있다면 칩 내부의 RAM 으로 옮겨다 놓을 수 있는 FIFO 가 필요함



FIFO 요구사항

- FIFO는 좌우 메모리 사이에서 데이터를 전달하는 역할을 함
  - 메모리의 내용 즉, sw 코드는 스트리밍 처럼 길게 존재함
  - Fifo 의 동작은 연속적으로 수행되어야 함

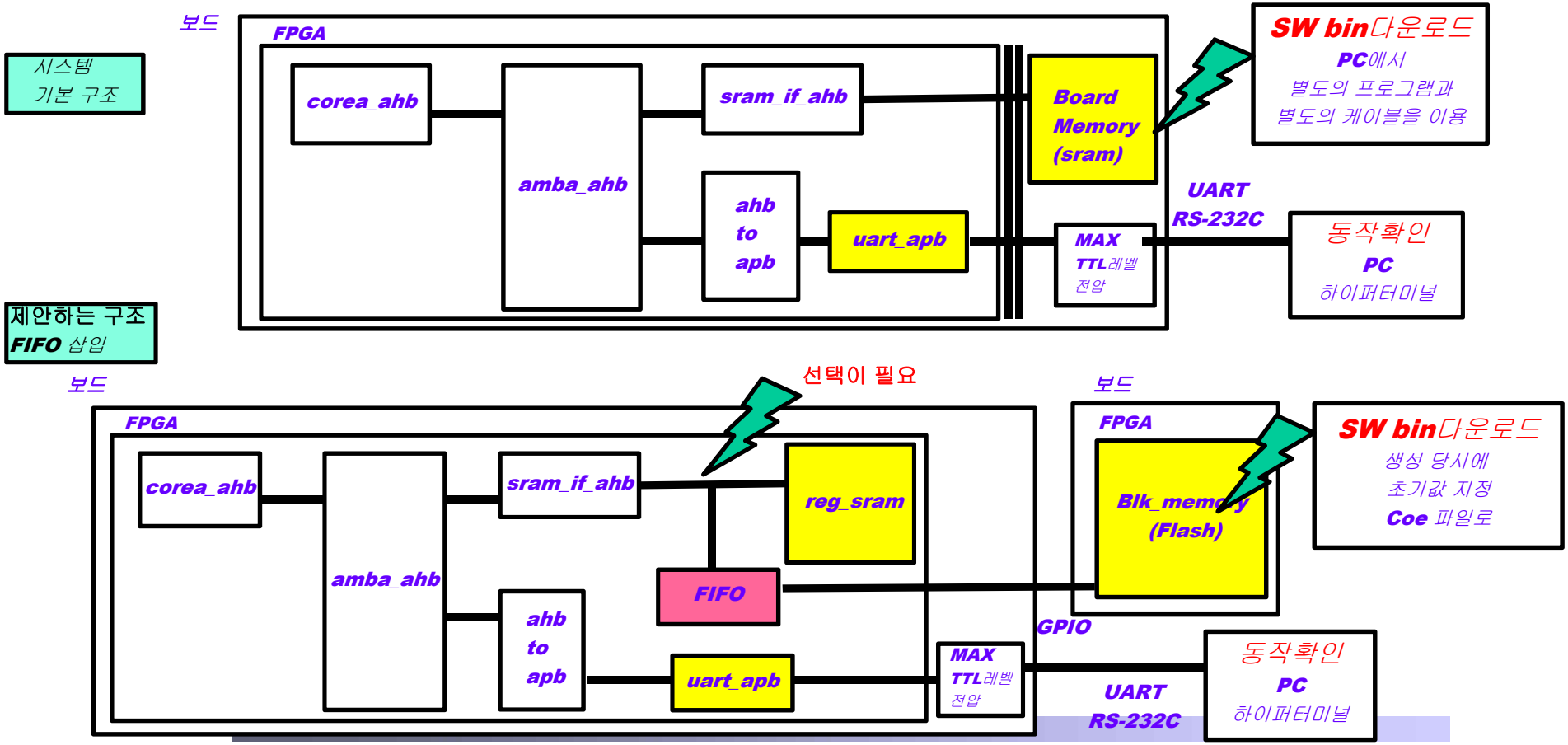
- 보통의 SoC 보드는 메모리만 붙어 있을 뿐, 컨트롤러가 제공되지 않는 경우가 대부분임
  - 따라서 fifo 는 양측의 메모리 컨트롤러 역할도 해야 함
  - 각 메모리의 동작 방법을 이해하는 것이 중요



FIFO 요구사항

▣ 시스템 레벨에서 FIFO 의 동작 방법

- ◆ FIFO 는 전체 시스템의 일부가 되어 존재해야 함
- ◆ 메모리 – 메모리 간의 전송이 끝난 후 FIFO 의 동작 OFF
- ◆ 유저가 FIFO 모듈의 Enable 여부를 외부 버튼으로 control



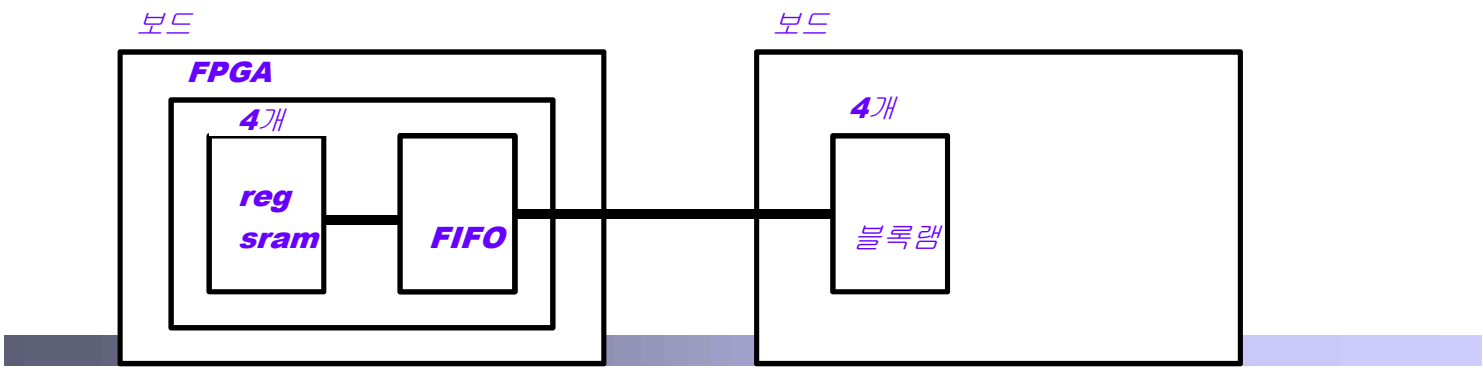
메모리 요구사항

Flash 개념의 메모리

- 메모리컨트롤러가 없다면 메모리 초기 데이터의 포팅이 용이하지 않음
- FPGA 내부의 블록 메모리를 일반적으로 사용
- UART 프로그램의 경우 블록램 1개로 만들수 있는 크기를 초과하므로 4개로 분할하여 구성함
- SW 결과물 bin 파일크기에 의존적

RAM 개념의 블록 램

- FPGA 디자인의 경우, 내부의 블록 메모리를 일반적으로 사용
- 칩 구현의 경우, 공정마다 각각의 메모리 심볼을 사용해야 하므로 변경의 가능성이 너무 많음
  - 따라서 reg 로 메모리 블록을 만드는 것이 효율적

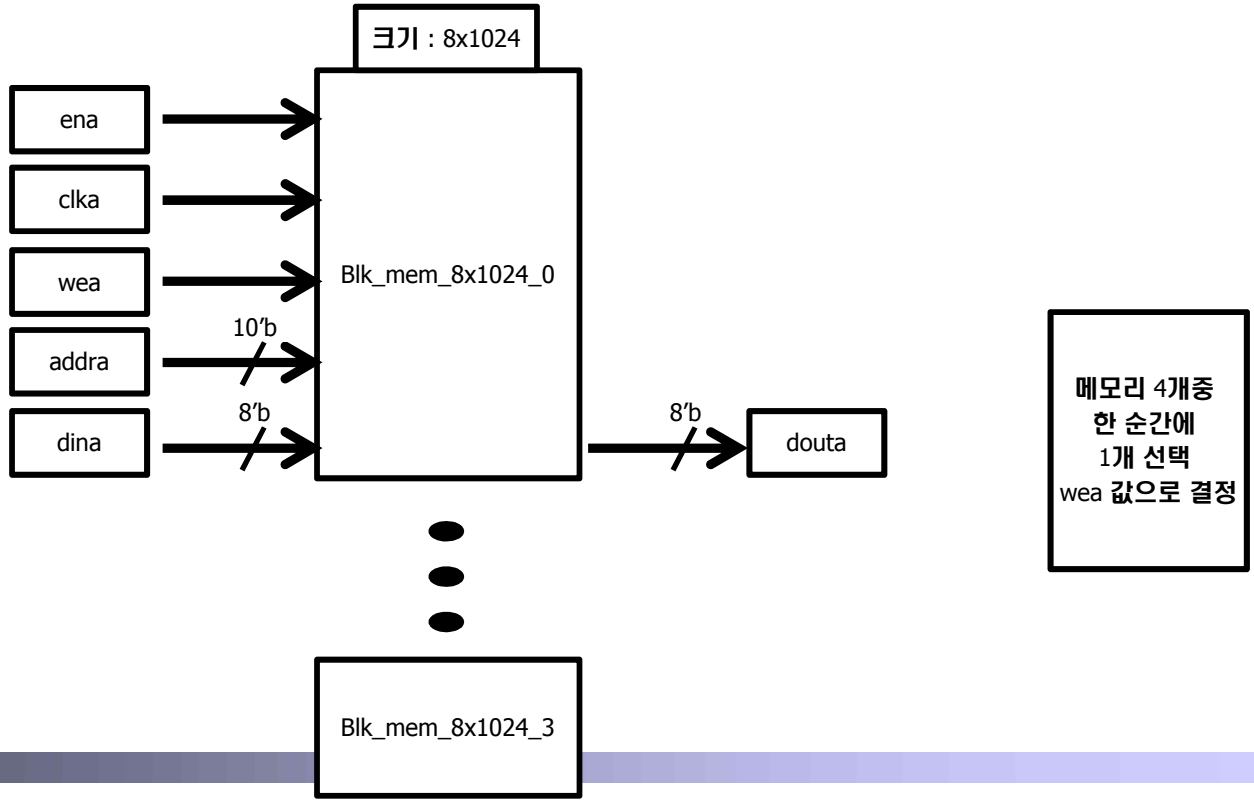


Flash 개념의 Block RAM

Flash 개념의 메모리

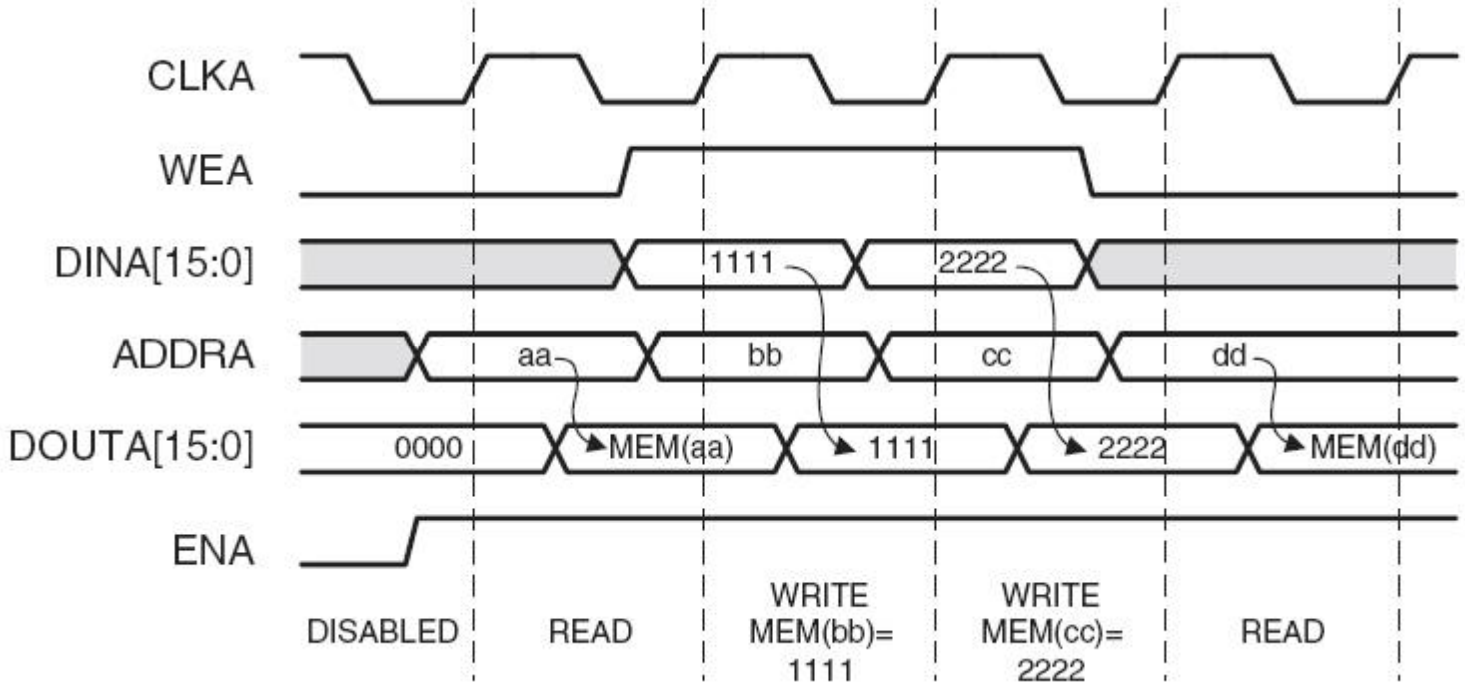
- FPGA 내부의 블록 메모리를 일반적으로 사용 ➔ 초기화는 coe 파일로 가능
- 메모리 크기는 SW 결과물인 bin 파일 크기에 따라 결정됨
- Xilinx Core generator 를 통해 생성

Flash 메모리의 구조



Flash 개념의 Block RAM

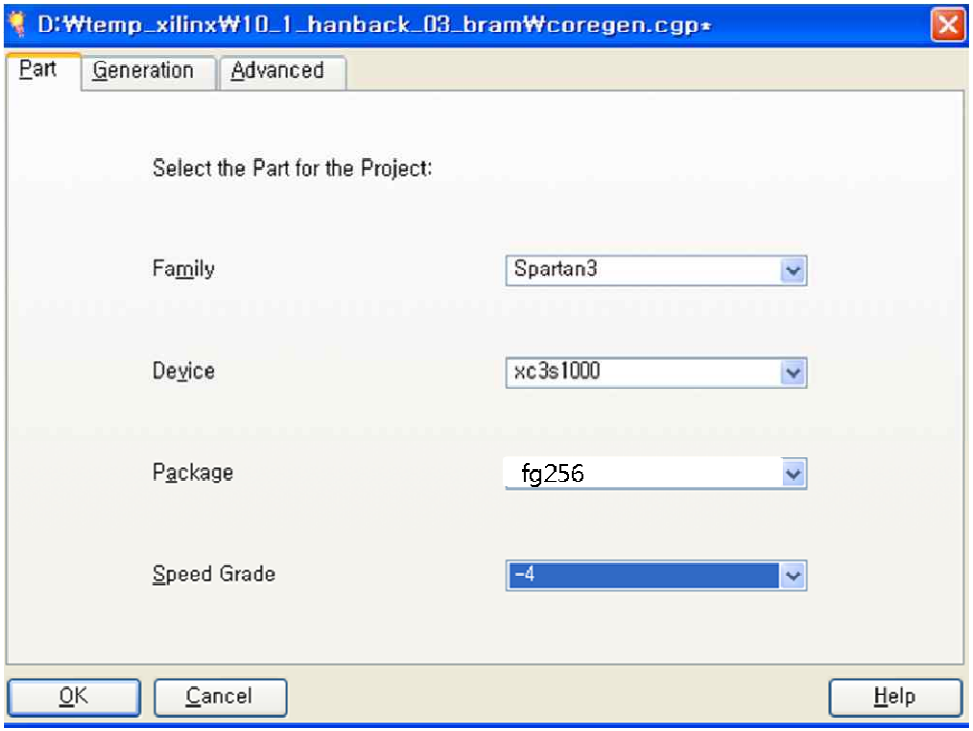
Write First Mode

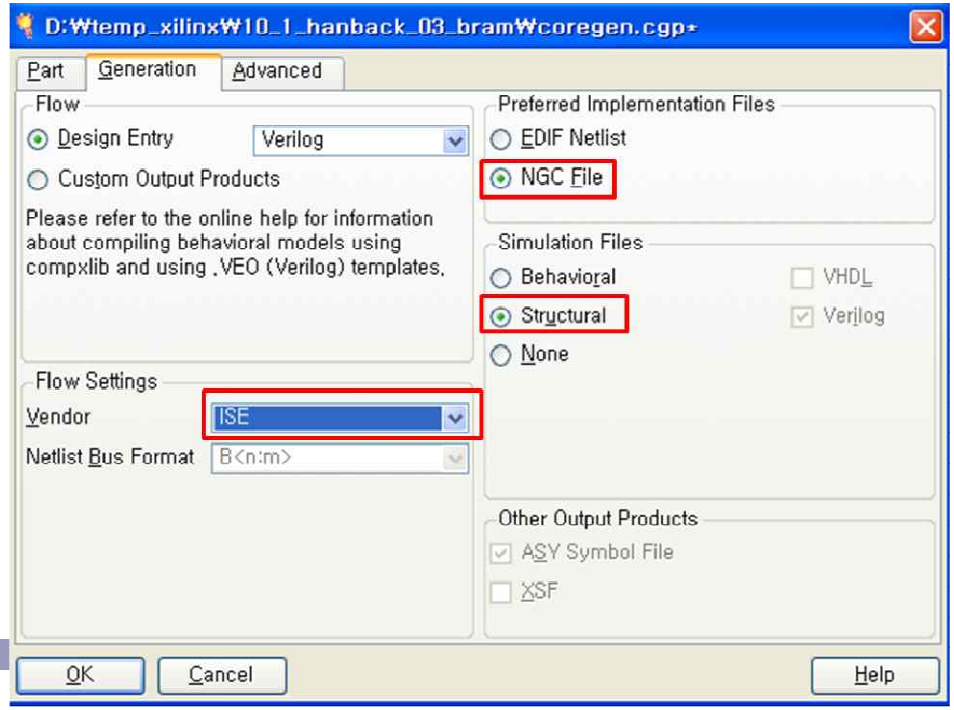


In WRITE FIRST mode, the input data is simultaneously written into memory and driven on the data output
This transparent mode offers the flexibility of using the data output bus during a write operation on the same port

Flash 개념의 Block RAM

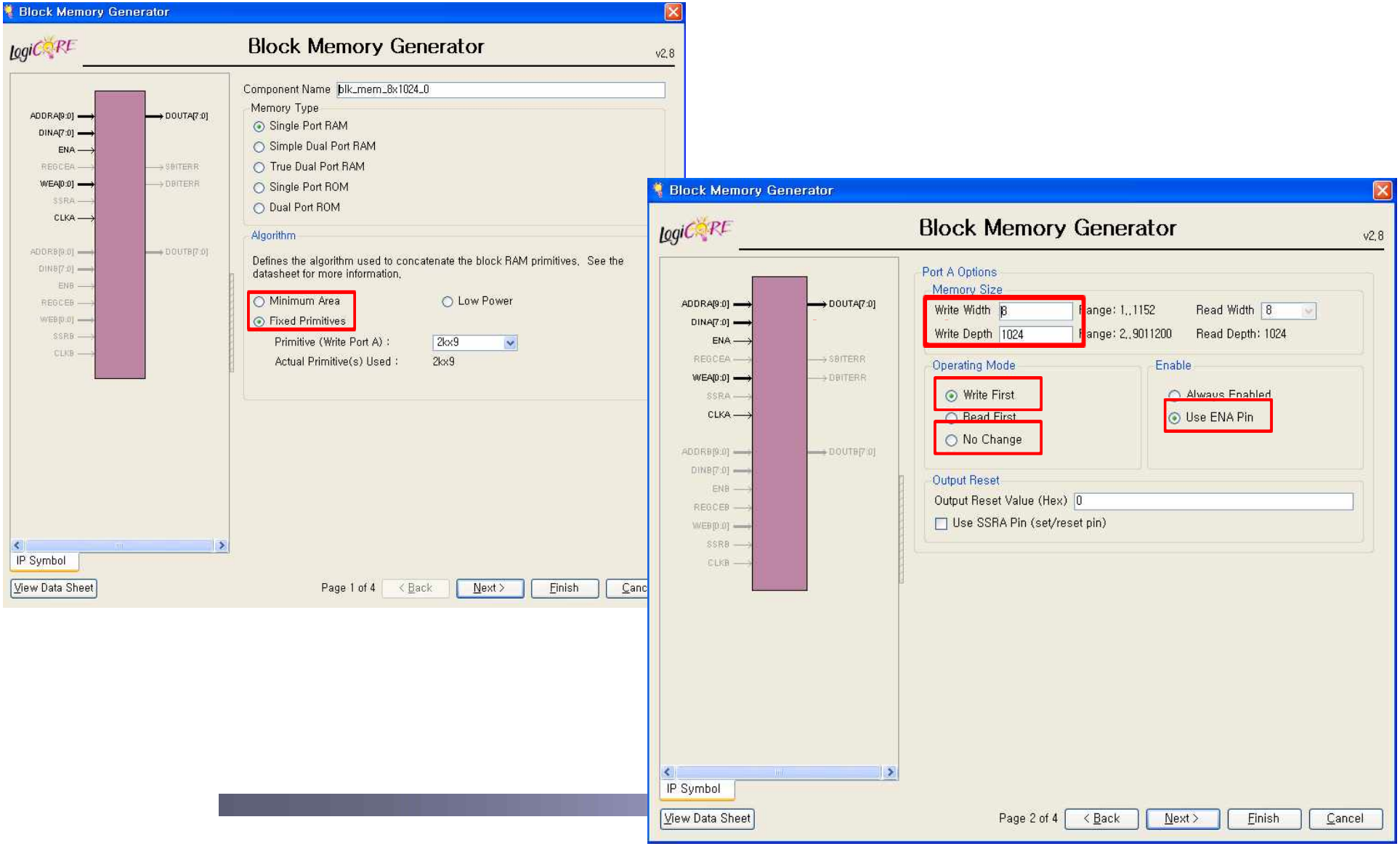
▣ 생성





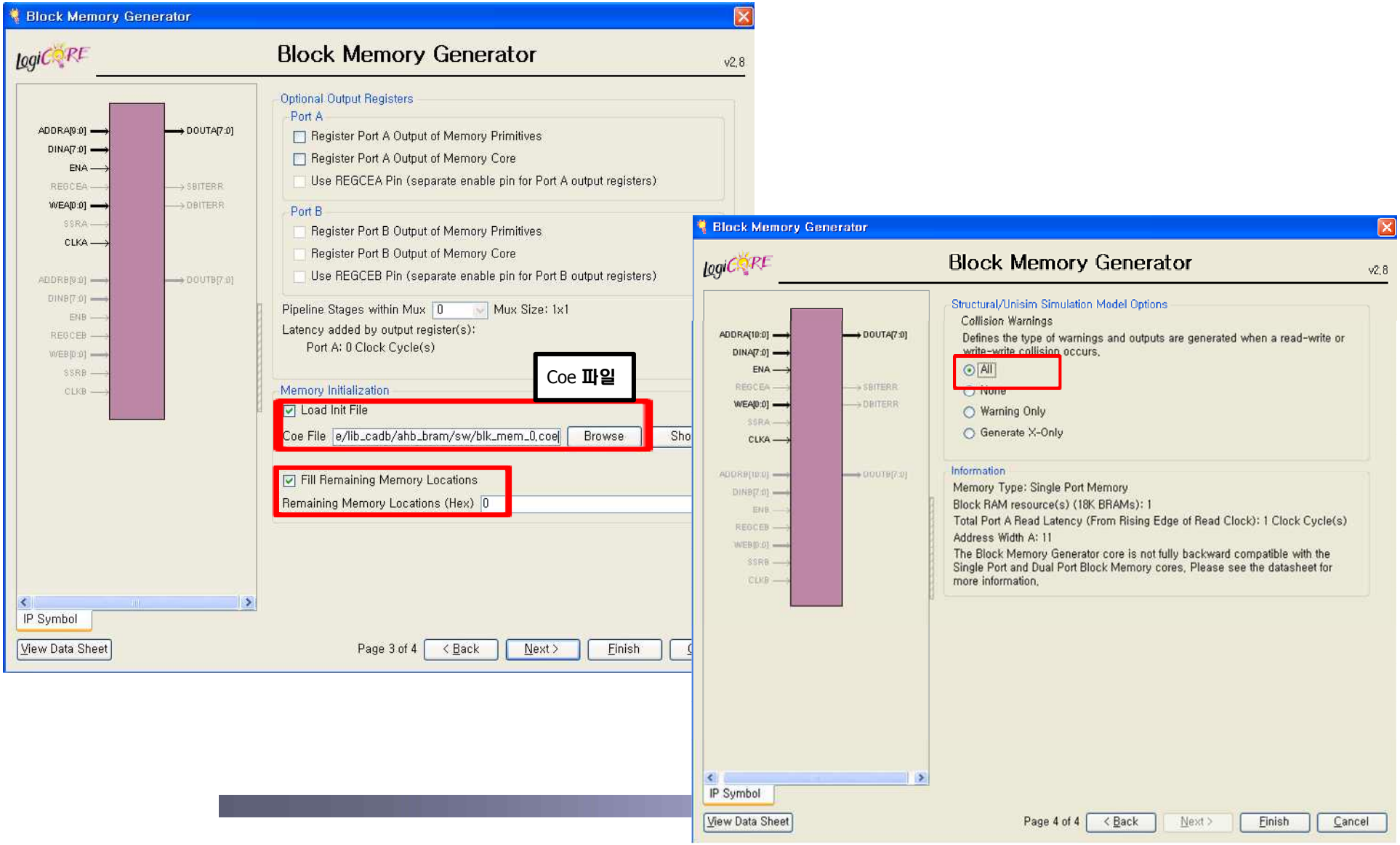
Flash 개념의 Block RAM

▣ 생성



Flash 개념의 Block RAM

▣생성

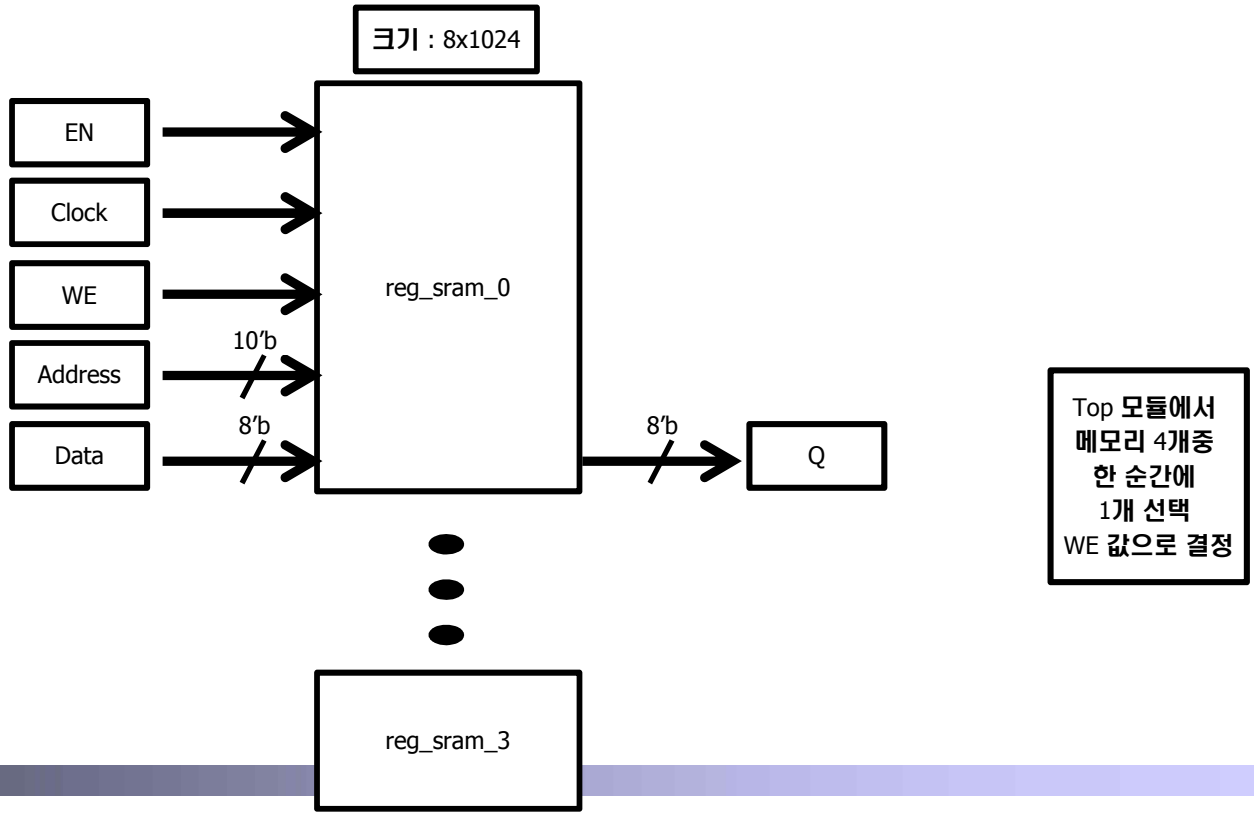


RAM 개념의 블록램

RAM 개념의 블록램

- 작은 크기의 메모리 구현을 통해 physical 면적의 부담을 감소시킴
- Reg 로 구현하여 공정에 관계없이 적용이 가능

레지스터 기반의 RAM 구조



RAM 개념의 블록램

reg_sram_1.v + (D:₩test...hb_bram₩ytkim_bram) - GVIM

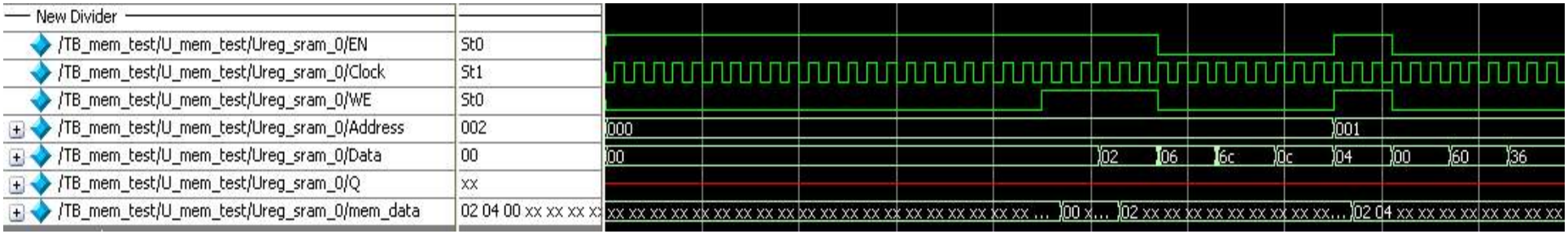
파일(F) 편집(E) 도구(T) 문법(S) 버퍼(B) 창(W) 도움말(H)

```
`timescale 1ns/1ps
module reg_sram(Data, Q, Clock, WE, Address, EN);

parameter width = 8;
parameter depth = 1024;
parameter addr = 10;

input EN;
input Clock;
input WE;
input [addr-1:0] Address;
input [width-1:0] Data;
output wire [width-1:0] Q;

reg [width-1:0] mem_data [0:depth-1];
reg [width-1:0] temp_reg;

integer i;

always @(posedge Clock)
begin
        if(EN==1'b1)
        begin
                if(WE==1'b1)
                        mem_data[Address] <= Data;
                else
                        temp_reg <= mem_data[Address];
        end
end
assign Q = temp_reg;

endmodule
```

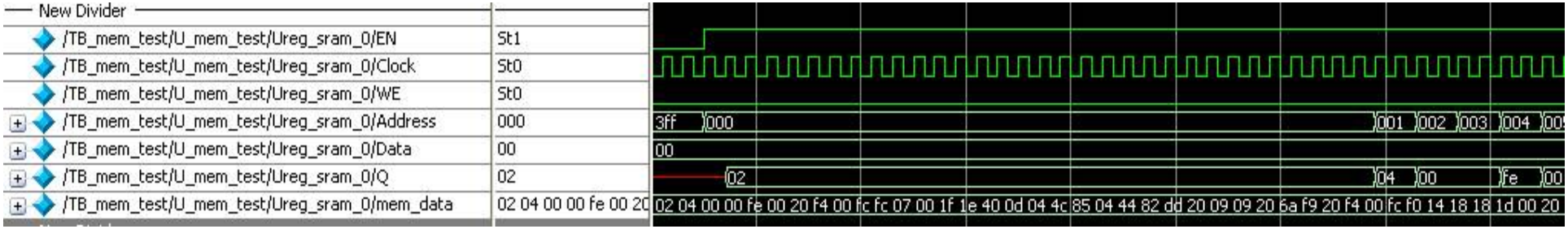
RAM 개념의 블록램

▣ 동작 원리

◆ WRITE

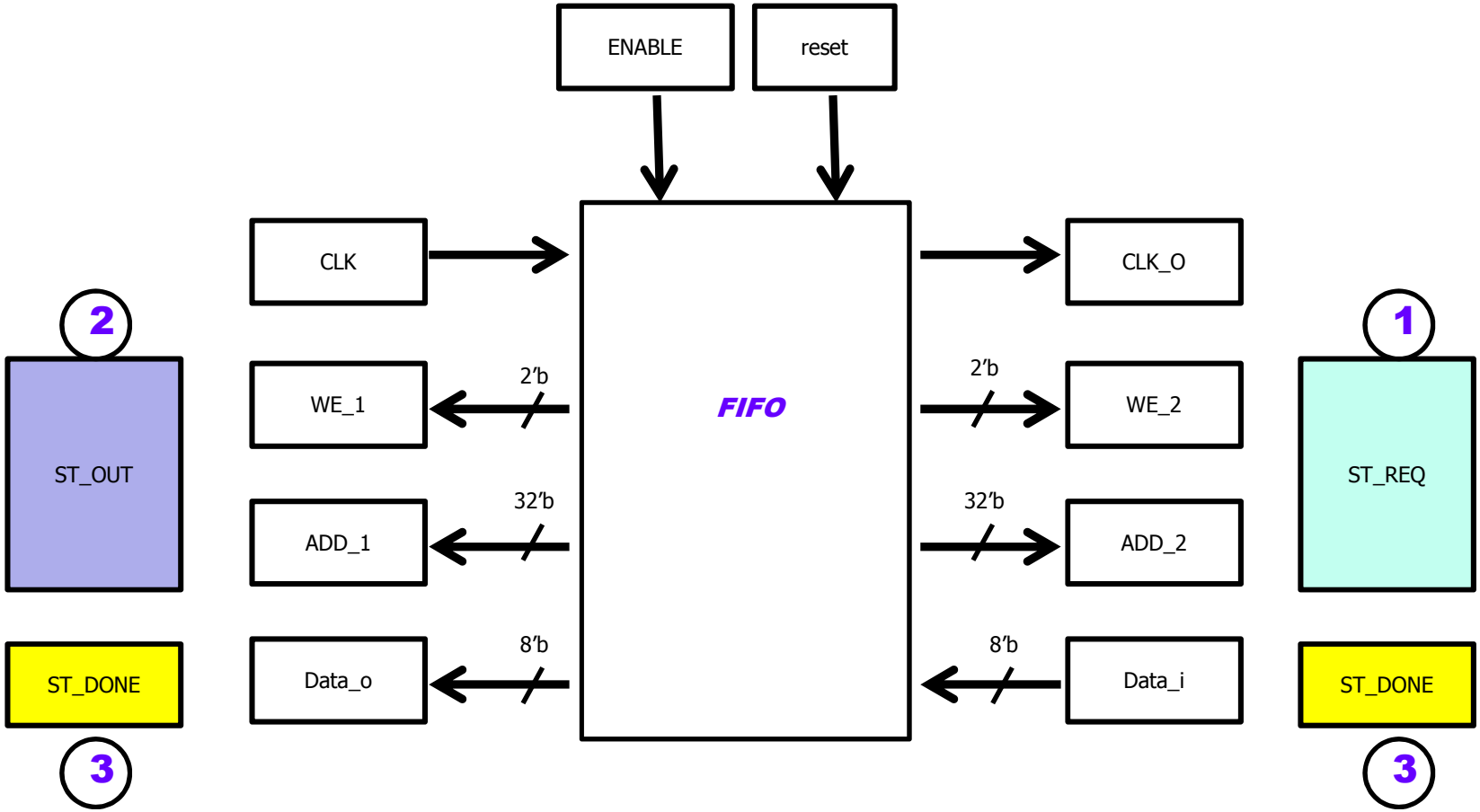


◆ READ



FIFO

Top View



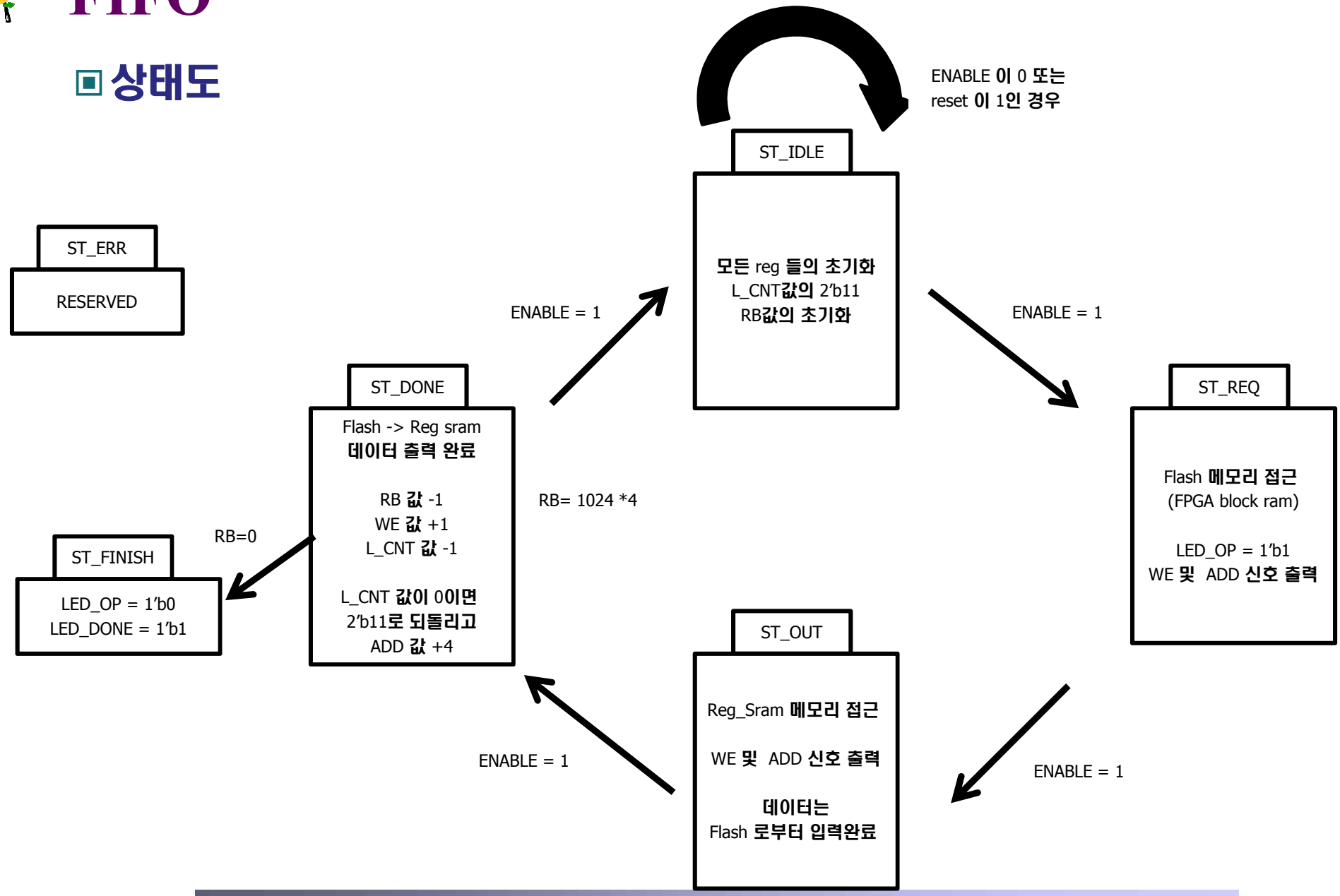
FIFO

▣ 개요

포트명	Width	Description
ENABLE	1	Enable 신호
reset	1	Reset 신호. Active High
CLK	1	입력 클럭
CLK_O	1	출력 출력. Flash 개념의 메모리에게 전달됨
WE_2	2	ST_REQ 에 동작. Flash 개념의 메모리에게 전달됨
ADD_2	32	ST_REQ 에 동작. Flash 개념의 메모리에게 전달됨
Data_i	8	ST_DONE 에 Flash 개념의 메모리로부터 입력됨
WE_1	2	ST_OUT 에 동작. Reg_sram 에게 전달됨
ADD_1	32	ST_OUT 에 동작. Reg_sram 에게 전달됨
Data_o	8	ST_DONE 에 Flash 에서 입력된 데이터를 Reg_sram으로 출력

FIFO

▣ 상태도


ENABLE 이 0 또는
reset 이 1인 경우
ST_IDLE
모든 reg 들의 초기화
L_CNT값의 2'b11
RB값의 초기화
ENABLE = 1
ST_REQ
Flash 메모리 접근
(FPGA block ram)
LED_OP = 1'b1
WE 및 ADD 신호 출력
ENABLE = 1
ST_OUT
Reg_Sram 메모리 접근
WE 및 ADD 신호 출력
데이터는
Flash 로부터 입력완료
ENABLE = 1
ST_DONE
Flash -> Reg sram
데이터 출력 완료
RB 값 -1
WE 값 +1
L_CNT 값 -1
L_CNT 값이 0이면
2'b11로 되돌리고
ADD 값 +4
RB= 1024 *4
RB=0
ST_FINISH
LED_OP = 1'b0
LED_DONE = 1'b1
ST_ERR
RESERVED

Fifo 소스

```
module fifo (CLK, reset, Data_o, WE_2, ADD_2, ENABLE, CLK_O, Data_i, ADD_1, WE_1, LED_OP, LED_DONE);
// local parameter
parameter LENGTH_ADD = 32;
parameter LENGTH_DATA = 8;

// STATE
parameter ST_IDLE = 3'b000;
parameter ST_REQ = 3'b001;
parameter ST_OUT = 3'b010;
parameter ST_DONE = 3'b011;
parameter ST_FINISH = 3'b100;
parameter ST_ERR = 3'b101;

// the number of read count
parameter Len = 12;

// input, output
input CLK;
input reset;
output [LENGTH_DATA-1:0] Data_o;
output reg [1:0] WE_2;
output reg [LENGTH_ADD-1:0] ADD_2;
//output reg [9:0] ADD_2;
input ENABLE;

output CLK_O;
input [LENGTH_DATA-1:0] Data_i;
output reg [LENGTH_ADD-1:0] ADD_1;
output reg [1:0] WE_1;
output reg LED_OP;
output reg LED_DONE;

// regs
reg [LENGTH_DATA-1:0] Data_tmp;
reg [LENGTH_ADD-1:0] L_ADD;
reg [1:0] L_WE;
reg [1:0] L_CNT;

reg [2:0] Cur_State;
reg [2:0]  Next_State;
reg ERR;

reg [2:0] ADD_INC;

reg [Len:0] RB;
```

ADD_INC 는 ADD 의 길이에 따라
주소 증가량을 결정하기 위해 만든
Reserved 레지스터

Fifo 소스

```
// clock out
assign CLK_O = CLK;

// Finite State Machine
always@ (posedge CLK or posedge reset)
begin
        if(reset==1'b1)
                Cur_State <= ST_IDLE;

        else
                Cur_State <= Next_State;
end
```

다음 상태 결정부

```
// Decision of next State
always@*
        begin
        if (RB!= 9'h00) begin
                casex(  {ENABLE,        reset,          ERR,            Cur_State})
                        {1'b0,          1'bx,           1'bx,           3'bxxx}:        Next_State <= ST_IDLE;
                        {1'b1,          1'b1,           1'bx,           3'bxxx}:        Next_State <= ST_IDLE;
                        {1'bx,          1'bx,           1'b1,           3'bxxx}:        Next_State <= ST_ERR;
                        {1'b1,          1'b0,           1'b0,           ST_IDLE}:       Next_State <= ST_REQ;
                        {1'b1,          1'b0,           1'b0,           ST_REQ}:        Next_State <= ST_OUT;
                        {1'b1,          1'b0,           1'b0,           ST_OUT}:        Next_State <= ST_DONE;
                        {1'b1,          1'b0,           1'b0,           ST_DONE}:       Next_State <= ST_REQ;
                        default : begin
                                        Next_State <= ST_IDLE;
                        end
                endcase
        end
        else begin
                Next_State <= ST_FINISH;
        end
end
```

Fifo 소스

```
always@(posedge CLK)
begin
        case (Cur_State)
                ST_FINISH : begin
                        LED_OP <= 1'b0;
                        LED_DONE <= 1'b1;
                end
                ST_ERR : begin
                        ERR <= 1'b1;
                end
                ST_DONE : begin
                        L_WE <= L_WE +1;
                        L_CNT <= L_CNT -1'b1;
                        if (L_CNT ==2'b00) begin
                                L_CNT <= 2'b11;
                                L_ADD <= L_ADD+3'b100;
                        end
                        RB<=RB-1;
                        Data_tmp <= Data_i;
                end
                ST_OUT : begin
                        WE_1 <= L_WE;
                        ADD_1 <= L_ADD;

                end
                ST_REQ : begin
                        LED_OP <= 1'b1;
                        WE_2 <= L_WE;
                        ADD_2 <= L_ADD;
                end
                ST_IDLE: begin
                        L_ADD <= 0;
                        L_CNT <= 2'b11;
                        L_WE <= 0;

                        ADD_1 <= 0;
                        ADD_2 <= 0;

                        WE_1 <= 0;
                        WE_2 <= 0;
                        RB <= {Len{1'b1}} + 1'b1;
                        Data_tmp <= 0;

                        ERR<=1'b0;
                        LED_OP <= 0;
                        LED_DONE <= 0;
                end
        endcase
end

assign Data_o = (Cur_State==ST_DONE)? Data_i : Data_tmp;
```

RB 값이 0이 되면
LED_OP 소등
LED_DONE 점등

RB 값 -1, WE 값 +1, L_CNT 값 -1

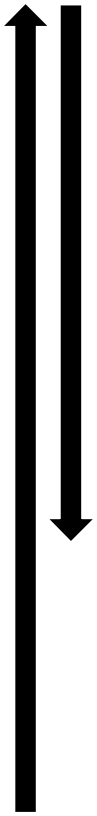
L_CNT 값이 0이면 2'b11로 되돌리고
ADD 값 +4

Flash -> Reg sram 데이터 Data_tmp 출력

Reg_Sram 메모리 접근
WE 및 ADD 신호 출력
데이터는 Flash 로부터 입력완료

Flash 메모리 접근 (FPGA block ram)
LED_OP = 1'b1
WE 및 ADD 신호 출력

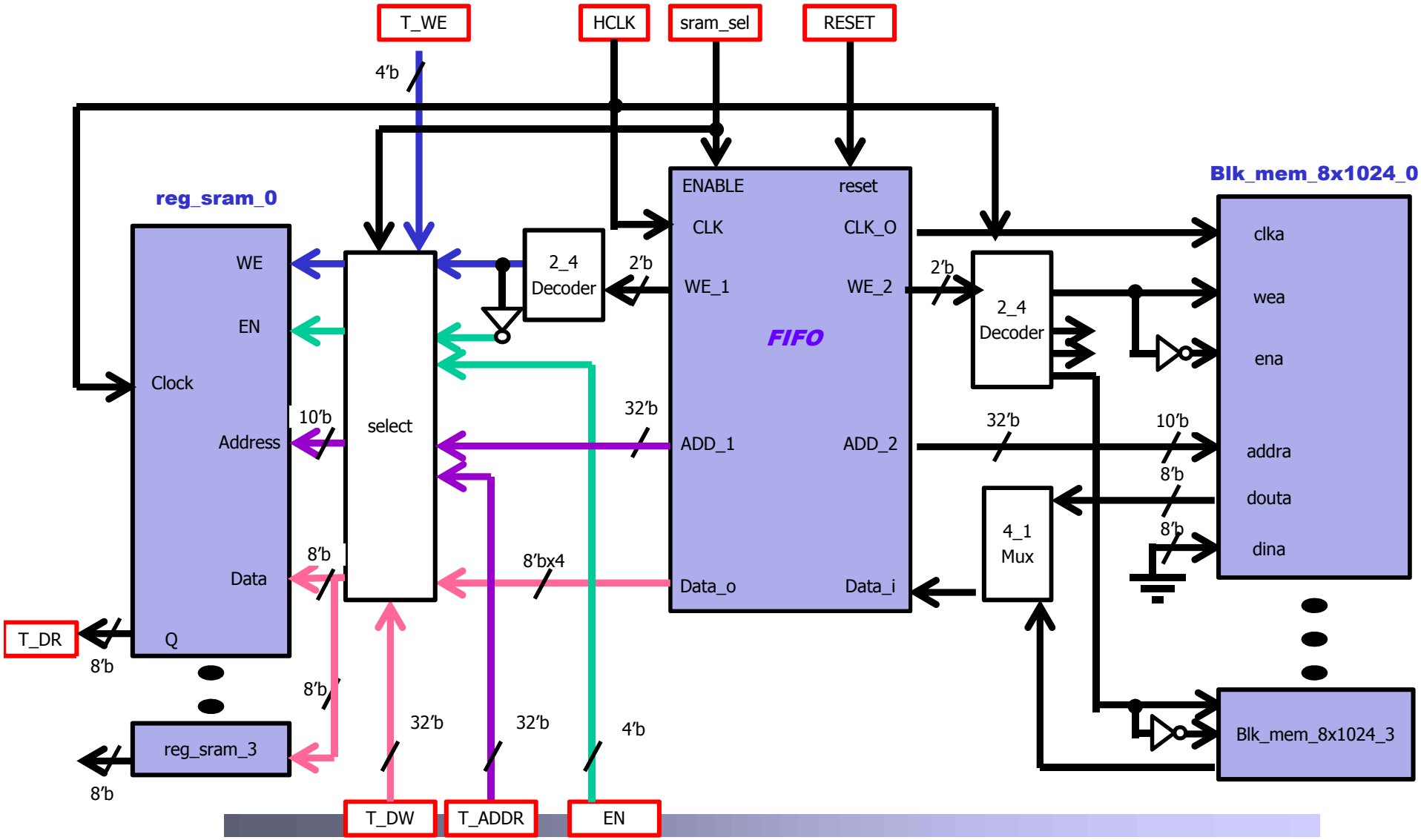
모든 reg 들의 초기화
L_CNT값의 2'b11
RB값의 초기화



출력 값 결정 부
현재 상태가 ST_DONE일 경우, 실시간 입력 데이터를 실시간 출력
그 외 경우는 저장된 Data_tmp 값 출력

전체 플랫폼 구조

Mem_test



테스트 벤치 환경

▣ 테스트 시나리오

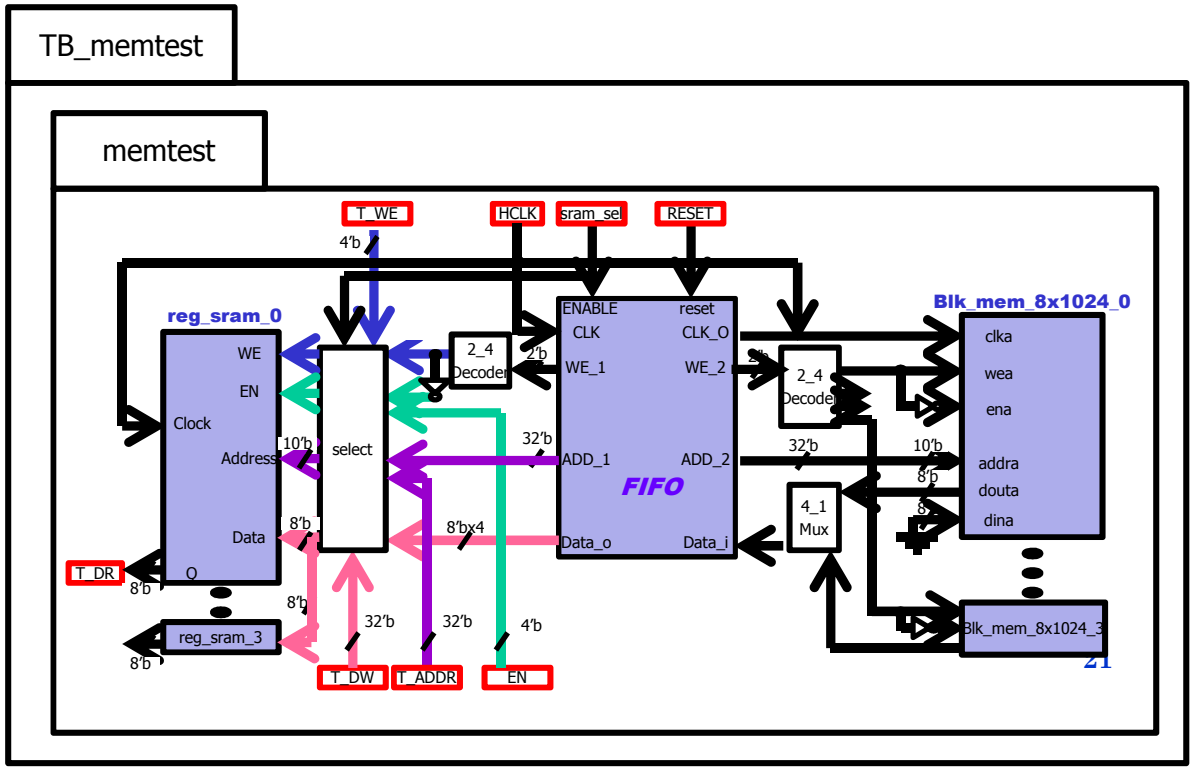
리셋 / 초기값

리셋 해제/ FIFO Enable

FIFO 동작 시간 고려
전송 완료 후 FIFO Disable

다시 리셋

Reg_sram 메모리에
주소를 올려가면서
Read 동작 테스트



테스트 벤치 환경

```
`timescale 1ns/1ps
module TB_mem_test();

reg HCLK, RESET;
reg [31:0] T_ADDR;
reg [31:0] T_DW;
reg [3:0] T_WE;
reg sram_sel;
reg [3:0] EN;

wire [31:0] T_DR;

wire LED_OP, LED_DONE;

integer i;

mem_test U_mem_test(
  .EN(EN),
        .HCLK(HCLK),
        .RESET(RESET),
        .T_ADDR(T_ADDR),
        .T_DW(T_DW),
        .T_WE(T_WE),
        .T_DR(T_DR),
        .sram_sel(sram_sel),
        .LED_OP(LED_OP),
        .LED_DONE(LED_DONE));
```

```
initial begin
  EN=4'b1111;
        HCLK = 1'b0;
        RESET = 1'b1;
        T_ADDR = 0;
        T_DW = 0;
        T_WE = 0;
        sram_sel = 1'b0;



        #150 RESET = 1'b0;
        #300 sram_sel =1'b1;

        #3000000 sram_sel = 1'b0;

        #500 RESET = 1'b1;
        #100 RESET = 1'b0;

        for(i=0 ; i!= 32768; i=i+1)
                #40 T_ADDR = T_ADDR + 4;



end

always begin
        #10 HCLK = ~HCLK;
end




endmodule
```

리셋 / 초기값

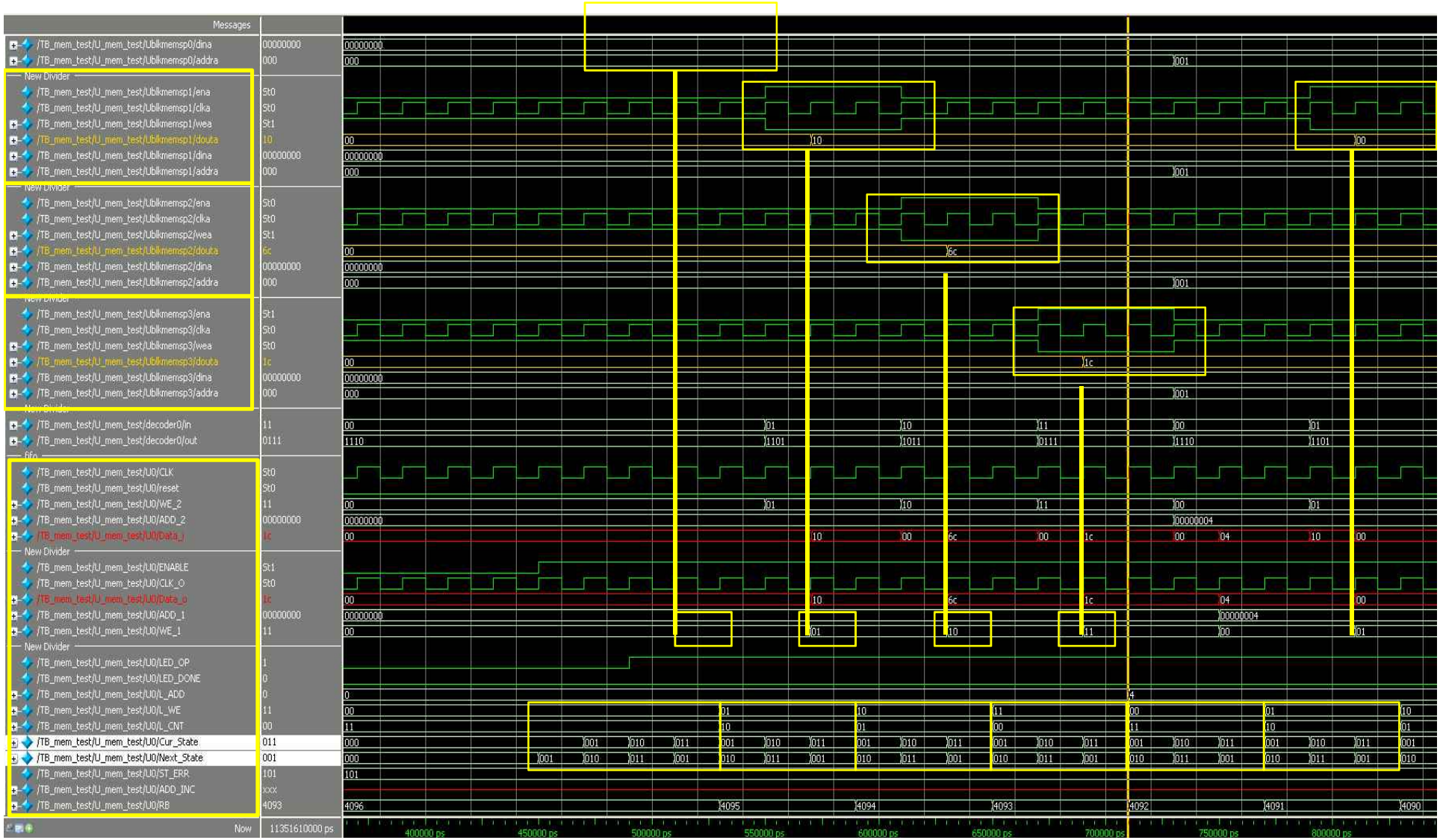
리셋 해제/ FIFO Enable

FIFO 동작 시간 고려
전송 완료 후 FIFO Disable

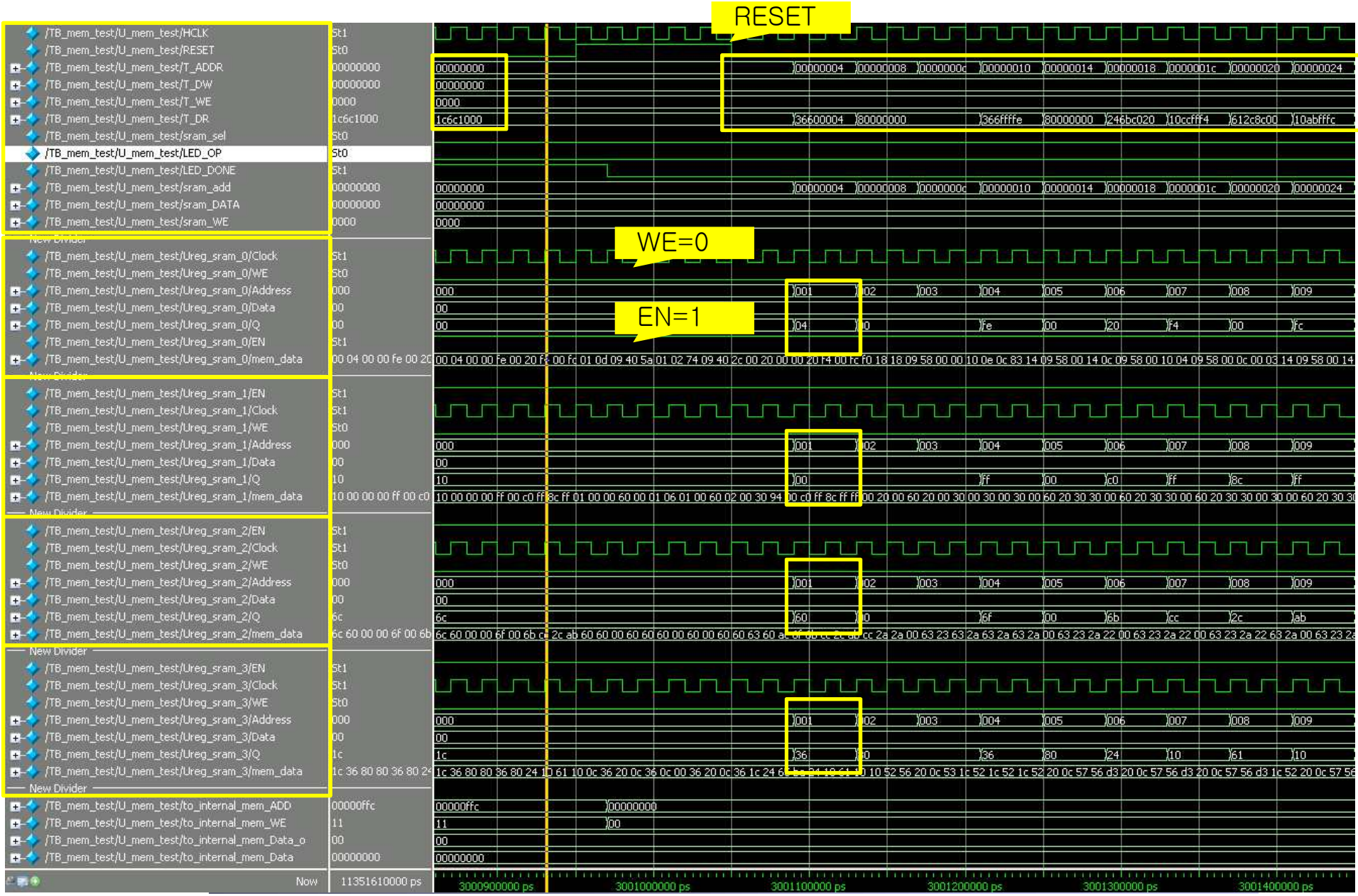
다시 리셋

Reg_sram 메모리에
주소를 올려가면서
Read 동작 테스트

메모리 시뮬레이션

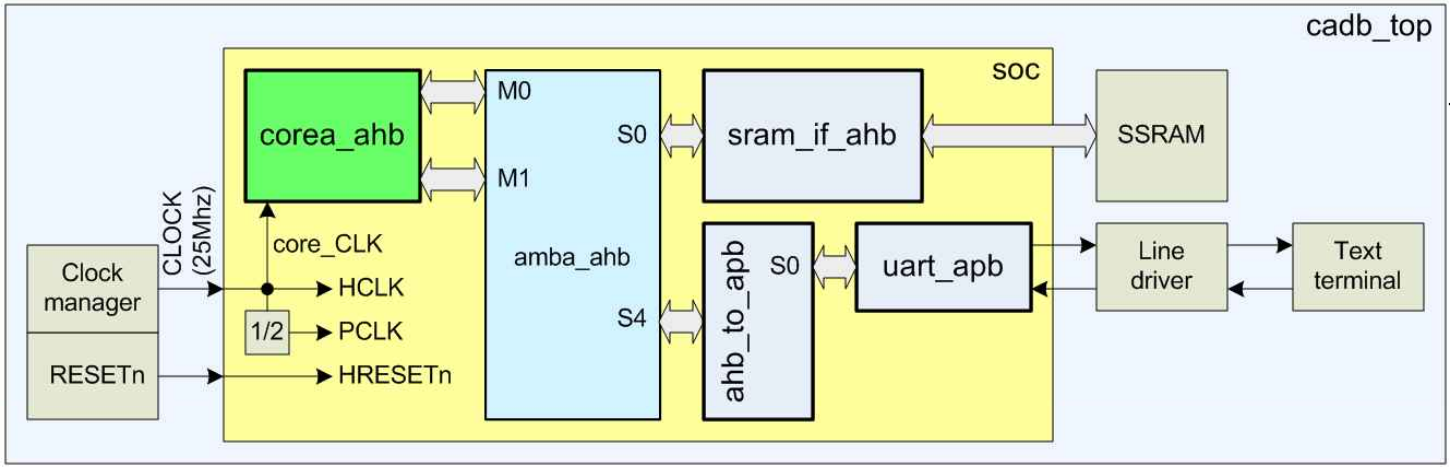

Messages
/TB_mem_test/U_mem_test/Ublkmemsp0/dina
/TB_mem_test/U_mem_test/Ublkmemsp0/addra
New Divider
/TB_mem_test/U_mem_test/Ublkmemsp1/ena
/TB_mem_test/U_mem_test/Ublkmemsp1/clka
/TB_mem_test/U_mem_test/Ublkmemsp1/wea
/TB_mem_test/U_mem_test/Ublkmemsp1/douta
/TB_mem_test/U_mem_test/Ublkmemsp1/dina
/TB_mem_test/U_mem_test/Ublkmemsp1/addra
New Divider
/TB_mem_test/U_mem_test/Ublkmemsp2/ena
/TB_mem_test/U_mem_test/Ublkmemsp2/clka
/TB_mem_test/U_mem_test/Ublkmemsp2/wea
/TB_mem_test/U_mem_test/Ublkmemsp2/douta
/TB_mem_test/U_mem_test/Ublkmemsp2/dina
/TB_mem_test/U_mem_test/Ublkmemsp2/addra
New Divider
/TB_mem_test/U_mem_test/Ublkmemsp3/ena
/TB_mem_test/U_mem_test/Ublkmemsp3/clka
/TB_mem_test/U_mem_test/Ublkmemsp3/wea
/TB_mem_test/U_mem_test/Ublkmemsp3/douta
/TB_mem_test/U_mem_test/Ublkmemsp3/dina
/TB_mem_test/U_mem_test/Ublkmemsp3/addra
New Divider
/TB_mem_test/U_mem_test/decoder0/in
/TB_mem_test/U_mem_test/decoder0/out
fifo
/TB_mem_test/U_mem_test/U0/CLK
/TB_mem_test/U_mem_test/U0/reset
/TB_mem_test/U_mem_test/U0/WE_2
/TB_mem_test/U_mem_test/U0/ADD_2
/TB_mem_test/U_mem_test/U0/Data_i
New Divider
/TB_mem_test/U_mem_test/U0/ENABLE
/TB_mem_test/U_mem_test/U0/CLK_O
/TB_mem_test/U_mem_test/U0/Data_o
/TB_mem_test/U_mem_test/U0/ADD_1
/TB_mem_test/U_mem_test/U0/WE_1
New Divider
/TB_mem_test/U_mem_test/U0/LED_OP
/TB_mem_test/U_mem_test/U0/LED_DONE
/TB_mem_test/U_mem_test/U0/L_ADD
/TB_mem_test/U_mem_test/U0/L_WE
/TB_mem_test/U_mem_test/U0/L_CNT
/TB_mem_test/U_mem_test/U0/Cur_State
/TB_mem_test/U_mem_test/U0/Next_State
/TB_mem_test/U_mem_test/U0/ST_ERR
/TB_mem_test/U_mem_test/U0/ADD_INC
/TB_mem_test/U_mem_test/U0/RB
Now
11351610000 ps
400000 ps
450000 ps
500000 ps
550000 ps
600000 ps
650000 ps
700000 ps
750000 ps
800000 ps

메모리 시뮬레이션

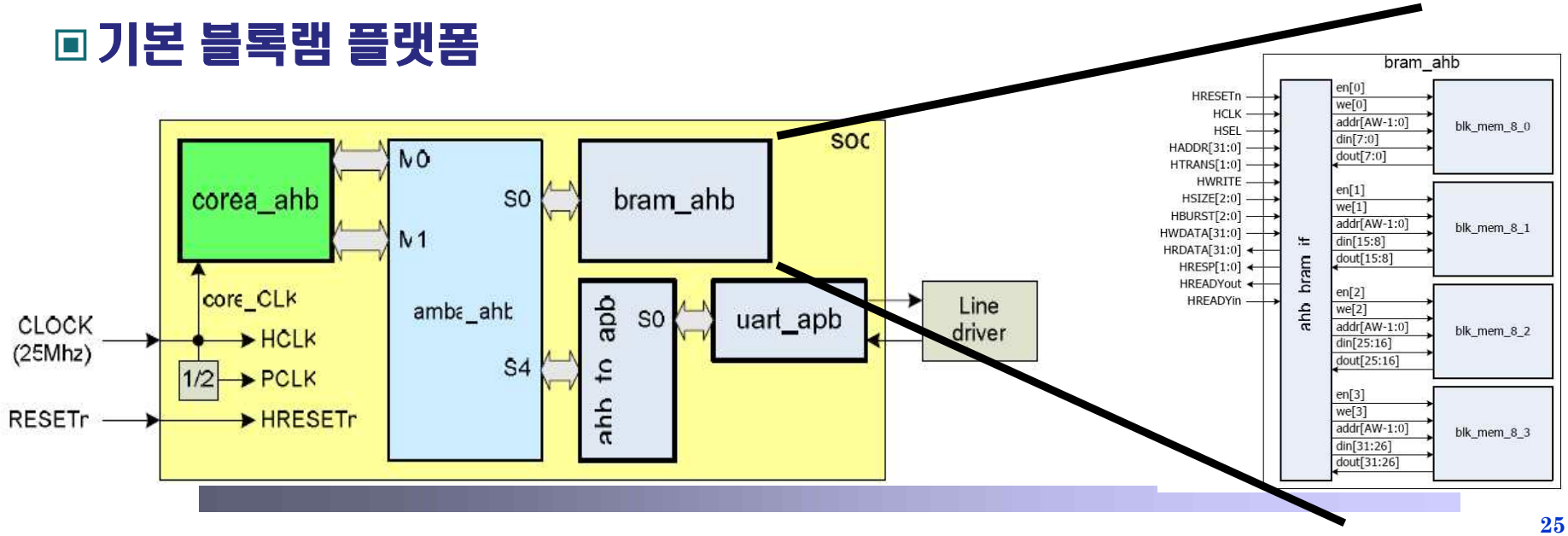

RESET
WE=0
EN=1
/TB_mem_test/U_mem_test/HCLK
/TB_mem_test/U_mem_test/RESET
/TB_mem_test/U_mem_test/T_ADDR
/TB_mem_test/U_mem_test/T_DW
/TB_mem_test/U_mem_test/T_WE
/TB_mem_test/U_mem_test/T_DR
/TB_mem_test/U_mem_test/sram_sel
/TB_mem_test/U_mem_test/LED_OP
/TB_mem_test/U_mem_test/LED_DONE
/TB_mem_test/U_mem_test/sram_add
/TB_mem_test/U_mem_test/sram_DATA
/TB_mem_test/U_mem_test/sram_WE
/TB_mem_test/U_mem_test/Ureg_sram_0/Clock
/TB_mem_test/U_mem_test/Ureg_sram_0/WE
/TB_mem_test/U_mem_test/Ureg_sram_0/Address
/TB_mem_test/U_mem_test/Ureg_sram_0/Data
/TB_mem_test/U_mem_test/Ureg_sram_0/Q
/TB_mem_test/U_mem_test/Ureg_sram_0/EN
/TB_mem_test/U_mem_test/Ureg_sram_0/mem_data
/TB_mem_test/U_mem_test/Ureg_sram_1/EN
/TB_mem_test/U_mem_test/Ureg_sram_1/Clock
/TB_mem_test/U_mem_test/Ureg_sram_1/WE
/TB_mem_test/U_mem_test/Ureg_sram_1/Address
/TB_mem_test/U_mem_test/Ureg_sram_1/Data
/TB_mem_test/U_mem_test/Ureg_sram_1/Q
/TB_mem_test/U_mem_test/Ureg_sram_1/mem_data
/TB_mem_test/U_mem_test/Ureg_sram_2/EN
/TB_mem_test/U_mem_test/Ureg_sram_2/Clock
/TB_mem_test/U_mem_test/Ureg_sram_2/WE
/TB_mem_test/U_mem_test/Ureg_sram_2/Address
/TB_mem_test/U_mem_test/Ureg_sram_2/Data
/TB_mem_test/U_mem_test/Ureg_sram_2/Q
/TB_mem_test/U_mem_test/Ureg_sram_2/mem_data
/TB_mem_test/U_mem_test/Ureg_sram_3/EN
/TB_mem_test/U_mem_test/Ureg_sram_3/Clock
/TB_mem_test/U_mem_test/Ureg_sram_3/WE
/TB_mem_test/U_mem_test/Ureg_sram_3/Address
/TB_mem_test/U_mem_test/Ureg_sram_3/Data
/TB_mem_test/U_mem_test/Ureg_sram_3/Q
/TB_mem_test/U_mem_test/Ureg_sram_3/mem_data
/TB_mem_test/U_mem_test/to_internal_mem_ADD
/TB_mem_test/U_mem_test/to_internal_mem_WE
/TB_mem_test/U_mem_test/to_internal_mem_Data_o
/TB_mem_test/U_mem_test/to_internal_mem_Data
Now
11351610000 ps
3000900000 ps
3001000000 ps
3001100000 ps
3001200000 ps
3001300000 ps
3001400000 ps

FIFO 를 포함한 전체 플랫폼 구조

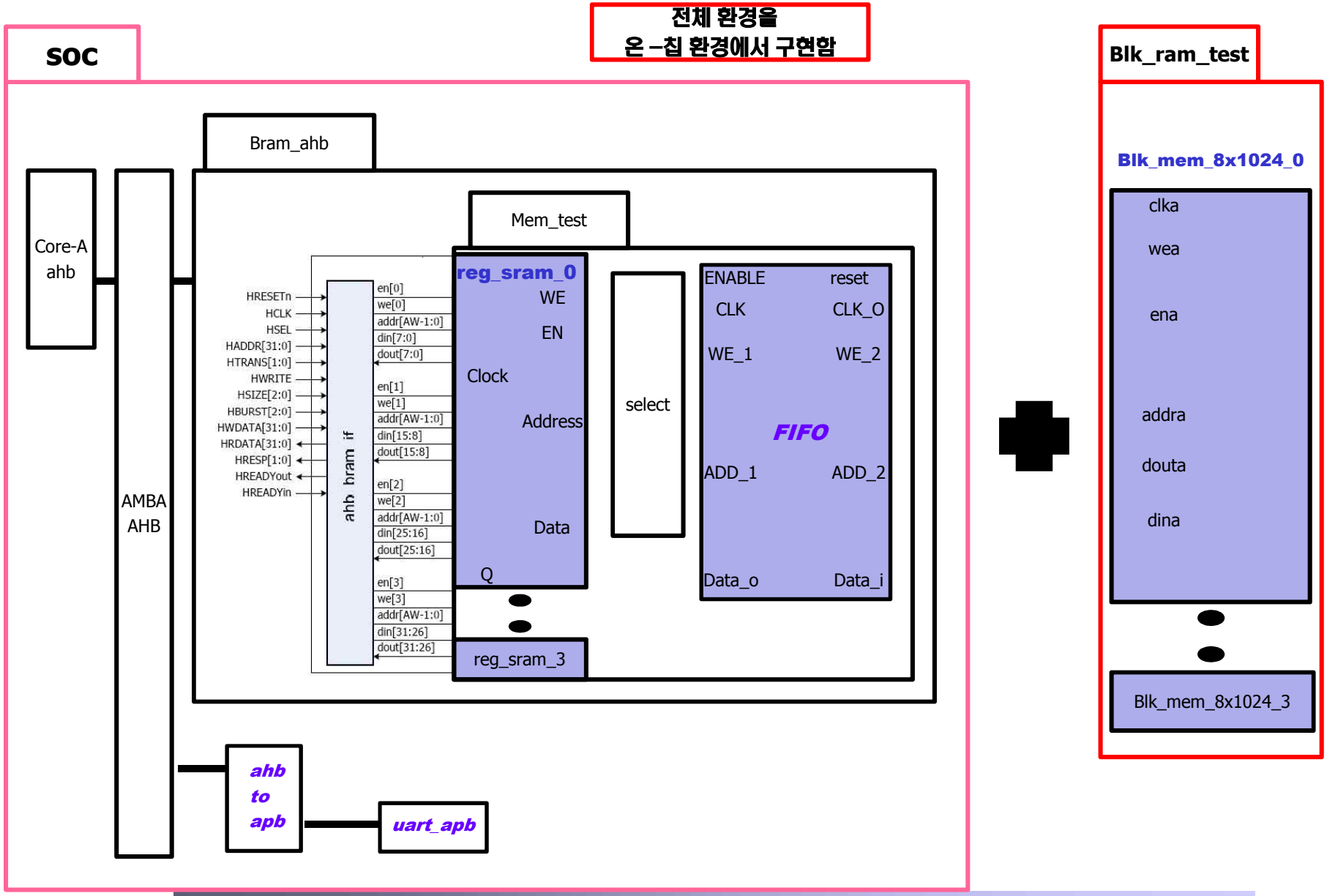
▣ 기본 플랫폼



▣ 기본 블록램 플랫폼



FIFO 를 포함한 전체 플랫폼 구조(기능적 분리)



FIFO 를 포함한 전체 플랫폼 구조(기능적 분리)

Top 모듈

```
module total_soc_smkcow_with_blkram (CLOCK, RESETn, UART_RX1, UART_TX1, sram_sel, LED_OP, LED_DONE);
input wire CLOCK;
input wire RESETn;

input wire UART_RX1;
output wire UART_TX1;

input sram_sel; wire sram_sel;
output wire LED_OP, LED_DONE;

wire  [1:0] to_bram_WE;
wire [31:0] to_bram_ADD;
wire [7:0] from_bram_Data_o;
wire CLK_O;

soc Usoc(
     .RESETn(RESETn),
     .CLOCK(CLOCK),


     .sram_sel(sram_sel),
     .LED_OP(LED_OP),
     .LED_DONE(LED_DONE),
     .UART_TX1(UART_TX1),
     .UART_RX1(UART_RX1),

     .to_bram_WE(to_bram_WE),
     .to_bram_ADD(to_bram_ADD),
     .from_bram_Data_i(from_bram_Data_o),
     .CLK_O(CLK_O)
);


blk_ram_test ublk_ram_test (
        .to_bram_WE(to_bram_WE),
        .to_bram_ADD(to_bram_ADD[11:2]),
        .from_bram_Data_o(from_bram_Data_o),
        .CLK_i(CLK_O)
);


endmodule
```

FIFO 를 포함한 전체 플랫폼 구조(기능적 분리)

▣ 테스트 벤치

```
`timescale 1ns/1ps
module TB_total_soc_smkcow_with_blkram ();

reg sram_sel;
reg RESETn, CLOCK;
reg UART_RX1;


wire UART_TX1;
wire LED_OP, LED_DONE;

total_soc_smkcow_with_blkram Utotal_soc_smkcow_with_blkram(
        .CLOCK(CLOCK),
        .RESETn(RESETn),
        .UART_RX1(UART_RX1),
        .UART_TX1(UART_TX1),
        .sram_sel(sram_sel),
        .LED_OP(LED_OP),
        .LED_DONE(LED_DONE)

);

initial begin
        CLOCK = 1'b0;
        RESETn = 1'b0;

        UART_RX1 = 1'b0;
        sram_sel = 1'b1;

        #150 RESETn = 1'b1;
        #500000 sram_sel = 1'b0;
        #100 RESETn = 1'b0;
        #100 RESETn = 1'b1;
end

always begin
        #10 CLOCK = ~CLOCK;
end


endmodule
```

리셋 / 초기값

FIFO Enable

리셋 해제

FIFO 동작 시간 고려하여 전송 완료 후 FIFO Disable

다시 리셋

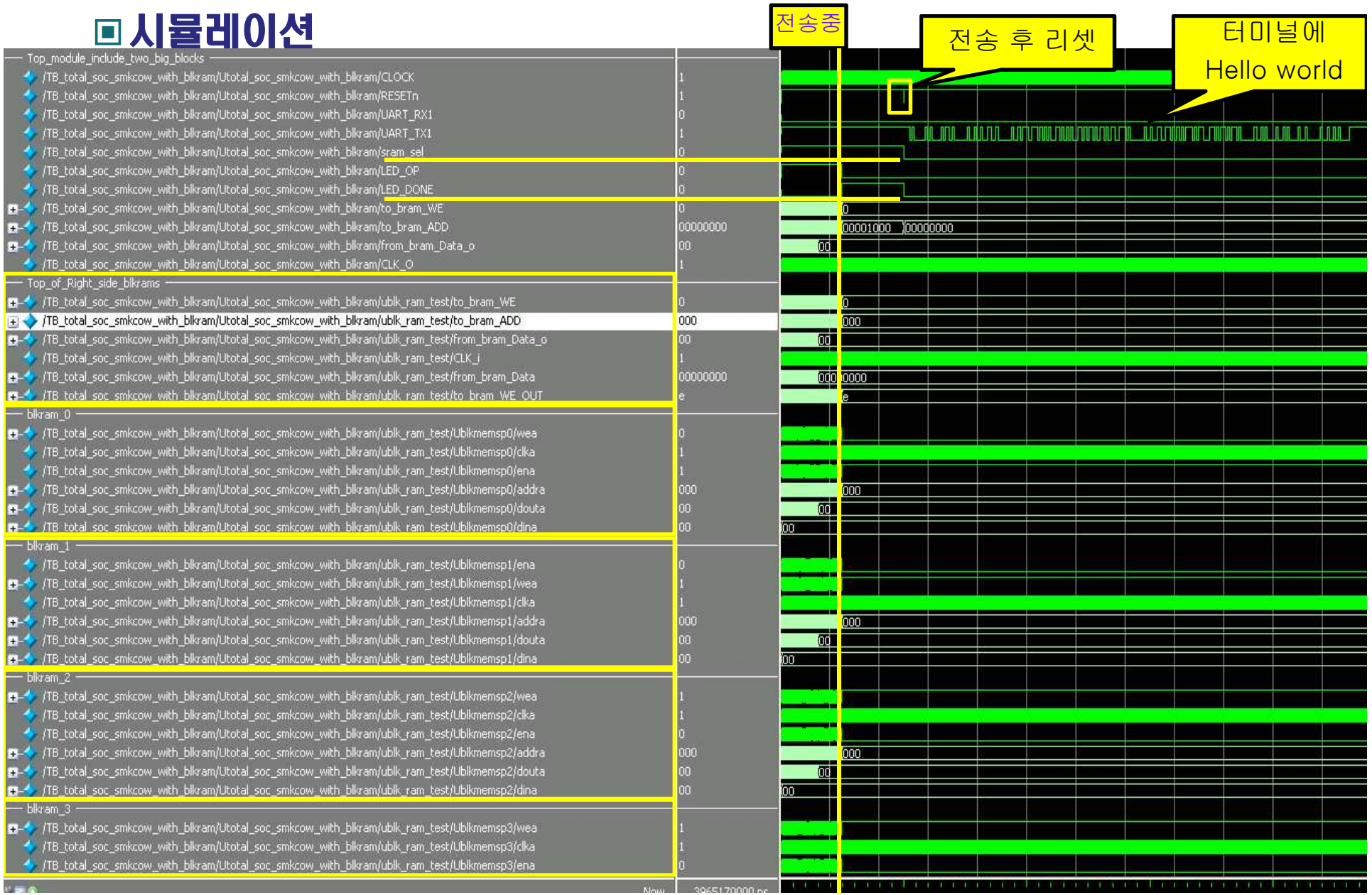
불필요하여 삭제한 내역

버스의 시그널과
메모리를 향한
주소, 컨트롤 시그널이 사라짐

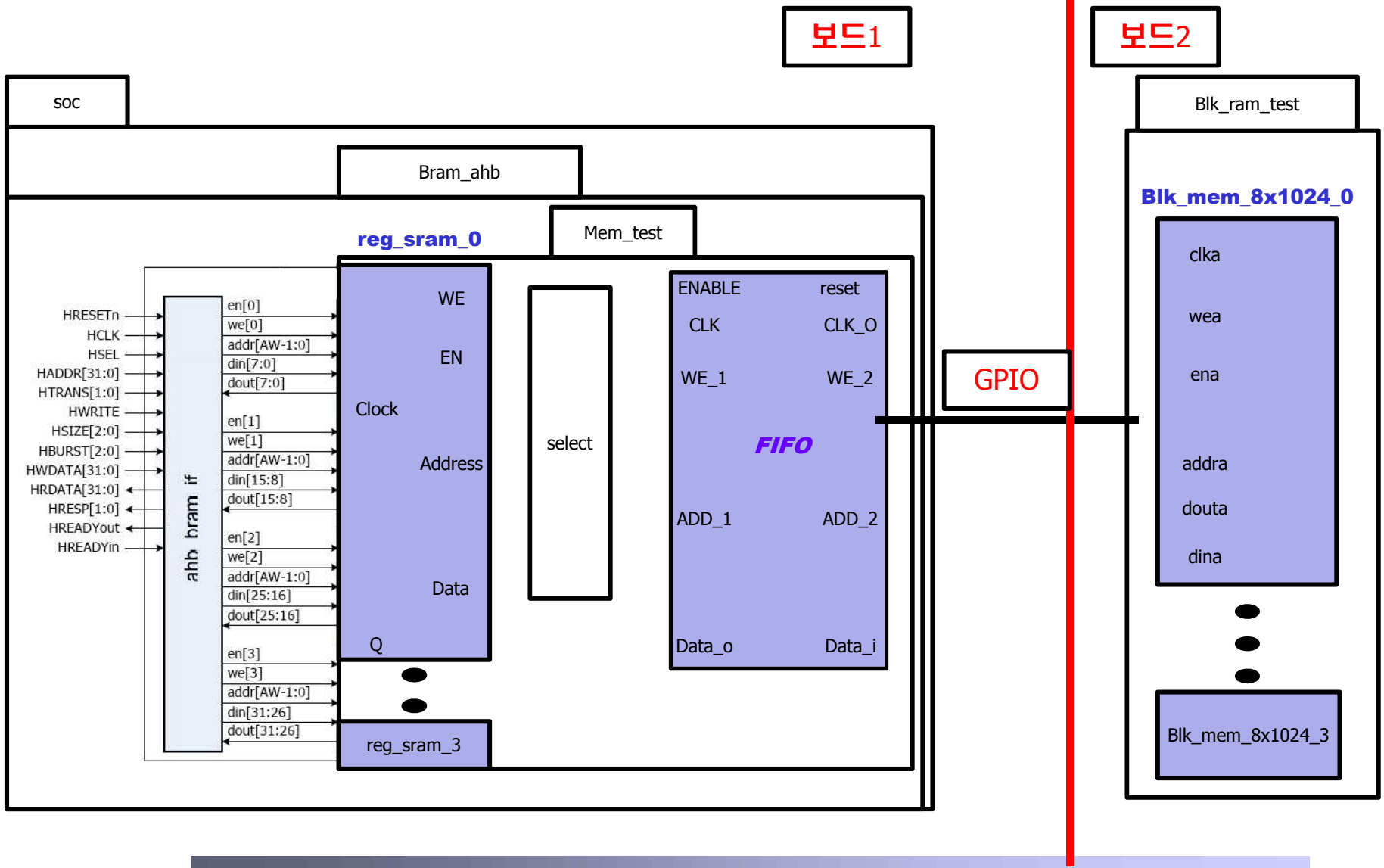
Reg_sram 메모리에
주소를 올려가면서
Read 동작 테스트

FIFO 를 포함한 전체 플랫폼 구조(기능적 분리)

▣ 시뮬레이션

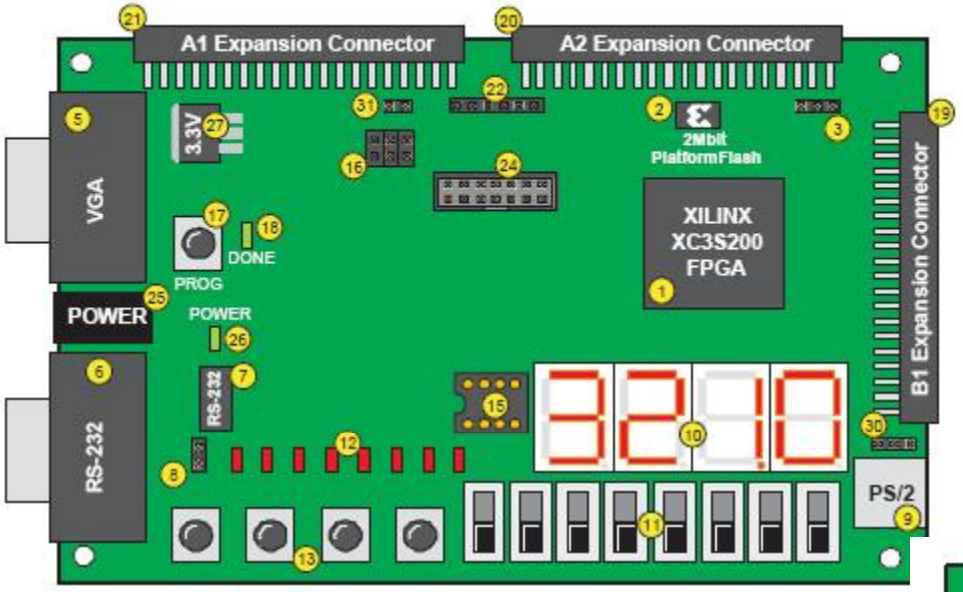


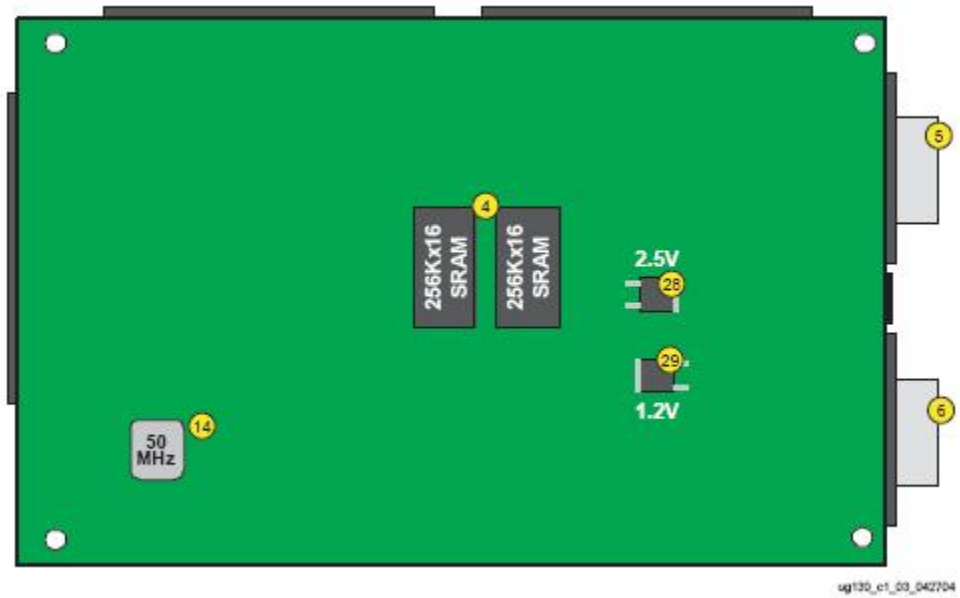
FIFO 를 포함한 전체 플랫폼 구조(물리적 분리)



FIFO 를 포함한 전체 플랫폼 구조(물리적 분리)

▣ 보드 구조

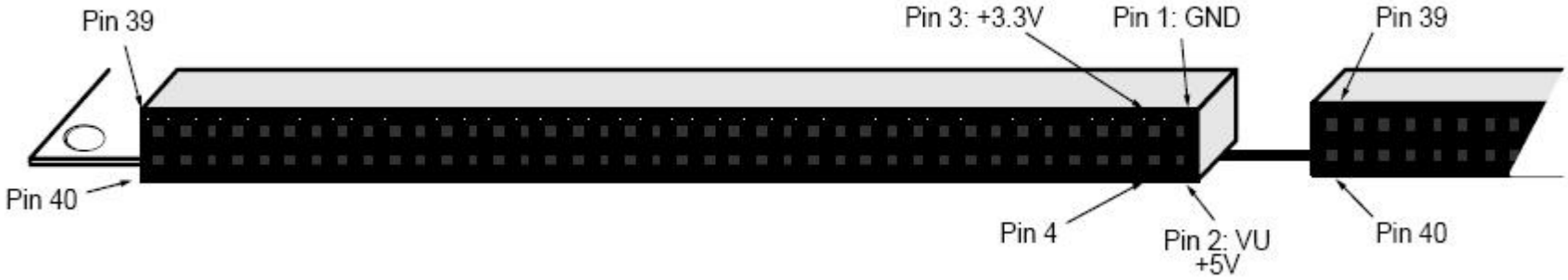




FIFO 를 포함한 전체 플랫폼 구조(물리적 분리)

GPIO 40 PIN header 구성

Connector	User I/O	SRAM	JTAG	Serial Configuration	Parallel Configuration
A1	32	Address OE#, WE# Data[7:0] to IC10 only	√		
A2	34			√	
B1	34			√	√



FIFO 를 포함한 전체 플랫폼 구조(물리적 분리)

▣ A2 확장 커넥터 구성

커넥터를 눈으로 봤을때 윗줄 / 아래줄

보드 1, 2용 프로젝트에 Ucf 파일로 정리되어 있음

Schematic Name	FPGA Pin	Connector		FPGA Pin	Schematic Name
GND		1	2		VU (+5V)
V_{CCO} (+3.3V)	V_{CCO} (all banks)	3	4	(E6)	PA-IO1
PA-IO2	(D5)	5	6	(C5)	PA-IO3
PA-IO4	(D6)	7	8	(C6)	PA-IO5
PA-IO6	(E7)	9	10	(C7)	PA-IO7
PA-IO8	(D7)	11	12	(C8)	PA-IO9
PA-IO10	(D8)	13	14	(C9)	PA-IO11
PA-IO12	(D10)	15	16	(A3)	PA-IO13
PA-IO14	(B4)	17	18	(A4)	PA-IO15
PA-IO16	(B5)	19	20		PA-IO17
PA-IO18	(B6)	21	22	(B7)	MA2-DB0
MA2-DB1	(A7)	23	24	(B8)	MA2-DB2
MA2-DB3	(A8)	25	26	(A9)	MA2-DB4
MA2-DB5	(B10)	27	28	(A10)	MA2-DB6
MA2-DB7	(B11)	29	30	(B12)	MA2-ASTB
MA2-DSTB	(A12)	31	32	(B13)	MA2-WRITE
MA2-WAIT	(A13)	33	34	(B14)	MA2-RESET
MA2-INT/GCK4	(D9) Oscillator socket	35	36	(B3) FPGA PROG_B	PROG-B
DONE	(R14) FPGA DONE	37	38	(N9) FPGA INIT_B	INIT
CCLK	(T15) FPGA CCLK Connects to (A14) via 390Ω resistor	39	40	(M11)	DIN

연결됨

데모를 위한 보드 스위치

Slide switches

FIFO Enable

리셋

Switch	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
FPGA Pin	K13	K14	J13	J14	H13	H14	G12	F12

Button switches

Push Button	BTN3 (User Reset)	BTN2	BTN1	BTN0
FPGA Pin	L14	L13	M14	M13

Ucf 파일

```
##################################################
## User switches
##################################################
NET "RESETn" LOC="F12";
NET "RESETn" IOSTANDARD=LVCMOS33;

NET "sram_sel" LOC="K13";
NET "sram_sel" IOSTANDARD=LVCMOS33;

NET "LED_OP" LOC="P11";
NET "LED_OP" IOSTANDARD=LVCMOS33;

NET "LED_DONE" LOC="P12";
NET "LED_DONE" IOSTANDARD=LVCMOS33;

NET "UART_TX1" LOC="R13";
NET "UART_TX1" IOSTANDARD=LVCMOS33;

NET "UART_RX1" LOC="T13";
NET "UART_RX1" IOSTANDARD=LVCMOS33;
```

버튼 조작 시나리오

	리셋 / 초기값	FIFO Enable	리셋해제	전송완료후 FIFO Disable	리셋	리셋해제 정상동작
리셋스위치	0	0	**1**	1	**0**	**1**
FIFO Enable 스위치	0	**1**	1	**0**	0	0

출처

- 다이나릿, 'Core_A_BasicPlatform',Dynalith Systems, 2009.1