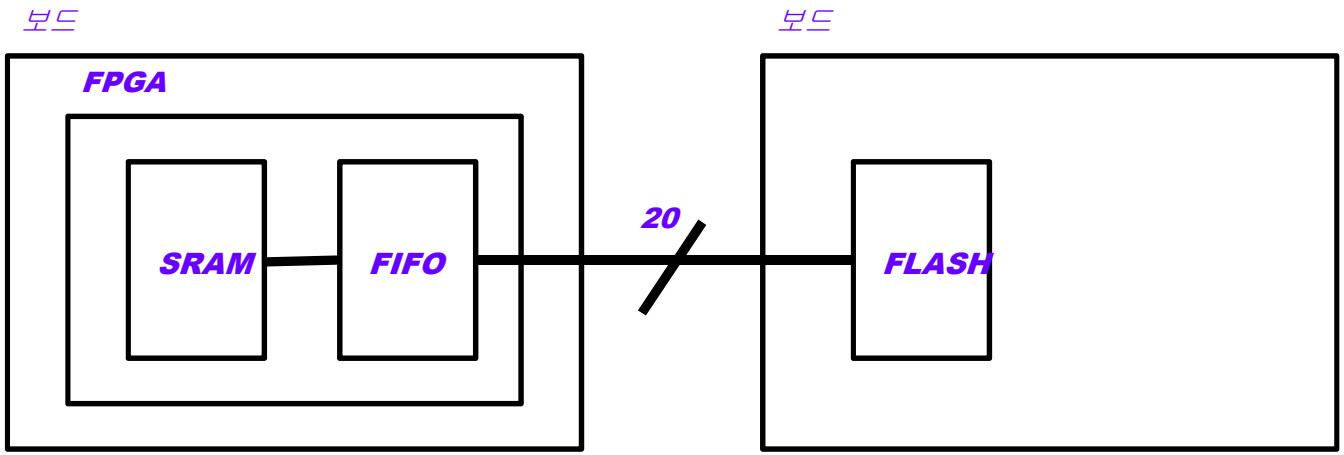
FIFO Ver.2
UART_FIFO
(PC to board)

FIFO 의 필요성

기존의 FIFO의 단점을 보완

- sw 를 FPGA 보드에 다운로드 해야 한다는 전제 조건이 있음
- SW bin 파일을 coe 파일 형식으로 준비를 해야 하는 불편
- SW 의 크기는 FPGA 매크로 블록 크기에 크게 의존적임
- 시스템에서 사용하는 메모리의 크기가 제한적임
- 전송시, 보드와 보드를 연결해야 하기 때문에 SW 구동 보드 이외에도 한 개의 보드가 추가로 준비되어야 함
- GPIO 가 많은 보드에서만 사용이 가능함



FIFO 의 필요성

FPGA 내부의 매크로 메모리의 크기

Device	RAM Columns	RAM Blocks Per Column	Total RAM Blocks	Total RAM Bits	Total RAM Kbits
XC3S50	1	4	4	73,728	72K
XC3S200	2	6	12	221,184	216K
XC3S400	2	8	16	294,912	288K
XC3S1000/L	2	12	24	442,368	432K
XC3S1500/L	2	16	32	589,824	576K
XC3S2000	2	20	40	737,280	720K
XC3S4000/L	4	24	96	1,769,472	1,728K
XC3S5000	4	26	104	1,916,928	1,872K

FPGA의 크기에 매우 의존적임
54K byte

보드 메모리의 크기

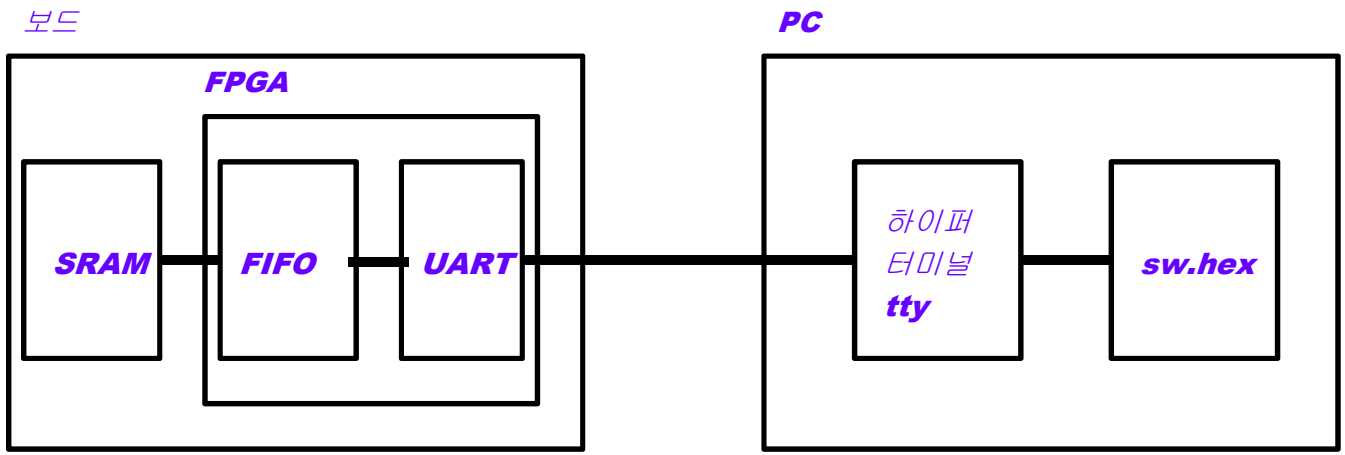
1M-byte of Fast Asynchronous SRAM (bottom side of board, see Figure 1-3)

- Two 256Kx16 ISSI IS61LV25616AL-10T 10 ns SRAMs
- Configurable memory architecture
  - Single 256Kx32 SRAM array, ideal for MicroBlaze code images
  - Two independent 256Kx16 SRAM arrays
- Individual chip select per device
- Individual byte enables

보드는 ***1M byte*** 이므로 이것을 이용하는 것이 효과적임

FIFO 요구사항

- 보통의 SoC 보드는 메모리만 붙어 있을 뿐, 컨트롤러가 제공되지 않는 경우가 대부분임
  - 따라서 fifo 는 보드 메모리 컨트롤러 역할도 해야 함
- 기존의 fifo 의 단점을 보완하기 위해, sw hex파일을 PC 의 UART 를 통해 입력 받고 보드의 메모리에 직접 write 하는 기능이 필요함
  - Uart 의 정확한 컨트롤을 해야 함



FIFO 요구사항

보드 메모리의 시뮬레이션 모델

- 정확한 fifo 의 구현 및 시뮬레이션을 위해서는 보드 메모리의 시뮬레이션 모델(.v) 이 반드시 필요함

하이퍼 터미널 모델

- Hex 파일을 터미널 프로그램을 통해서 전송하는 행동양식을 시뮬레이션 모델로 구현해야 함
- Instruction 이 터미널로 전송될 때 전송되는 형태를 구현해야 함

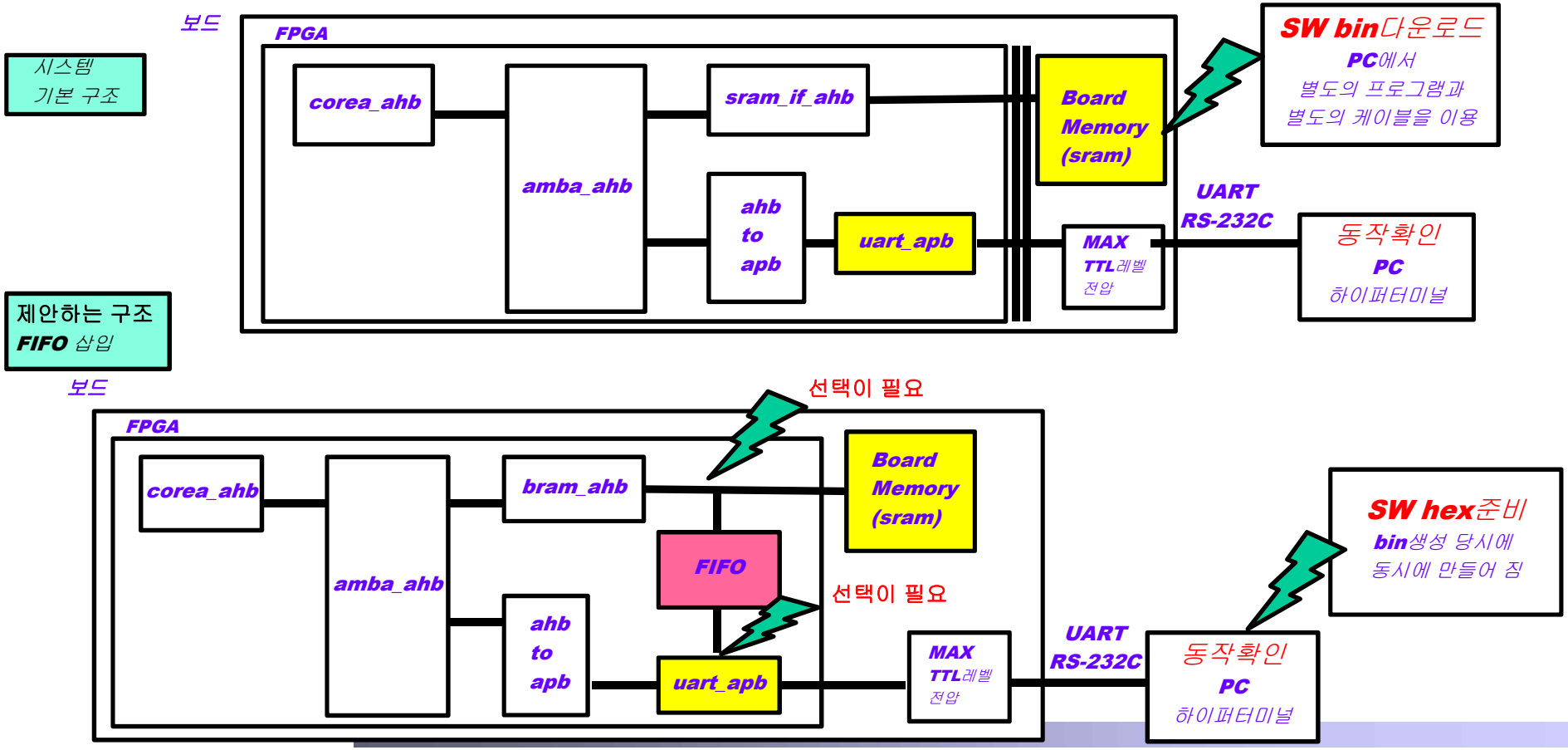
하이퍼 터미널 프로그램

- Windows의 일반적인 하이퍼 터미널은 파일 전송에 취약
- 범용적인 하이퍼 터미널을 이용
  - teraterm , Mttty 등

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 스펙

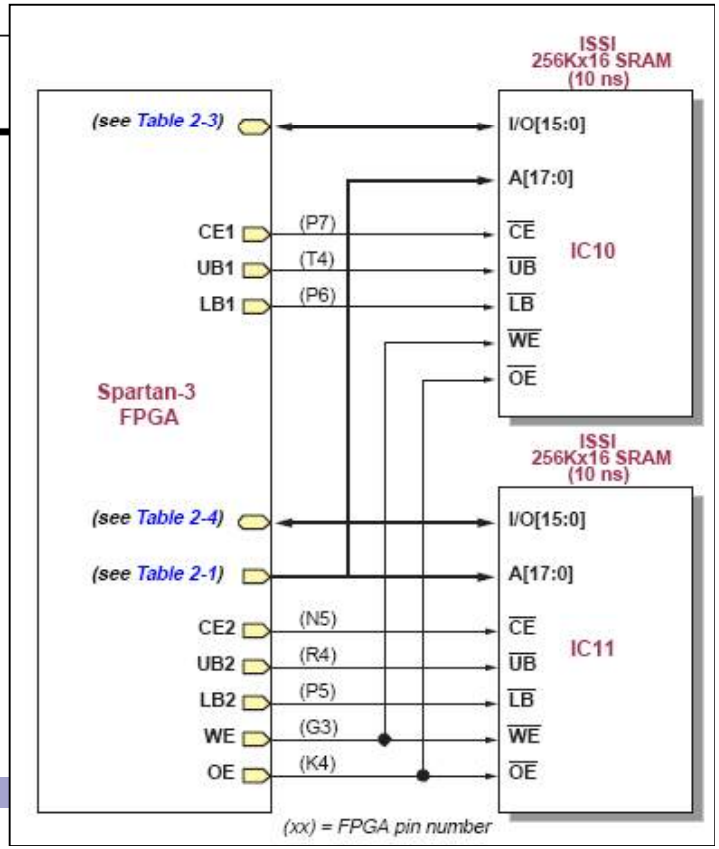
ISSI 사의 IS61LV25616AL

TRUTH TABLE

Mode	$\overline{WE}$	$\overline{CE}$	$\overline{OE}$	$\overline{LB}$	$\overline{UB}$	I/O PIN I/O0-I/O7	I/O PIN I/O8-I/O15	V_{DD} Current
Not Selected	X	H	X	X	X	High-Z	High-Z	I_{SB1}, I_{SB2}
Output Disabled	H	L	H	X	X	High-Z	High-Z	I_{CC}
	X	L	X	H	H	High-Z	High-Z	
Read	H	L	L	L	H	D_{OUT}	High-Z	I_{CC}
	H	L	L	H	L	High-Z	D_{OUT}	
	H	L	L	L	L	D_{OUT}	D_{OUT}	
Write	L	L	X	L	H	D_{IN}	High-Z	
	L	L	X	H	L	High-Z	D_{IN}	
	L	L	X	L	L	D_{IN}	D_{IN}	

PIN DESCRIPTIONS

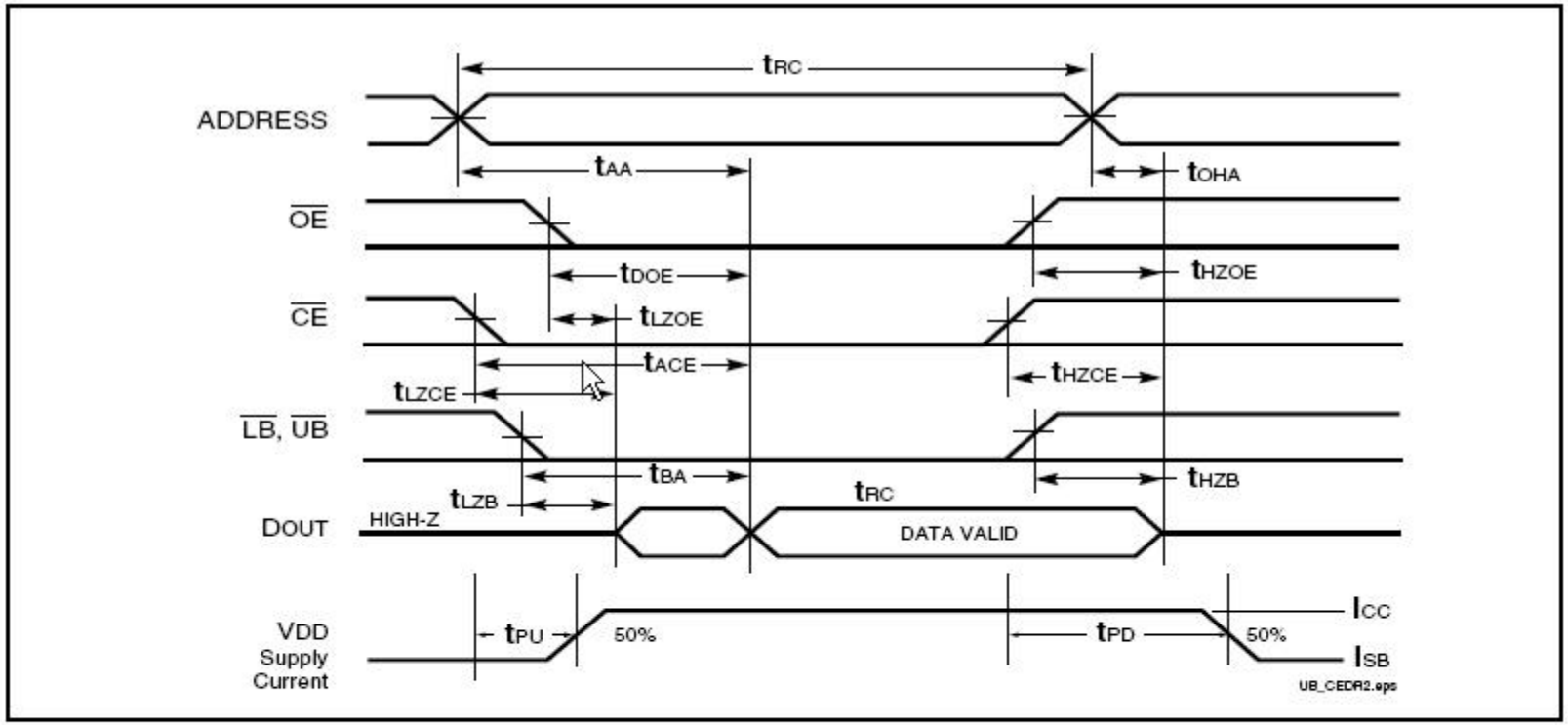
A0-A17	Address Inputs
I/O0-I/O15	Data Inputs/Outputs
$\overline{CE}$	Chip Enable Input
$\overline{OE}$	Output Enable Input
$\overline{WE}$	Write Enable Input
$\overline{LB}$	Lower-byte Control (I/O0-I/O7)
$\overline{UB}$	Upper-byte Control (I/O8-I/O15)
NC	No Connection
V_{DD}	Power
GND	Ground



메모리 스펙

READ

READ CYCLE NO. 2[1,3]

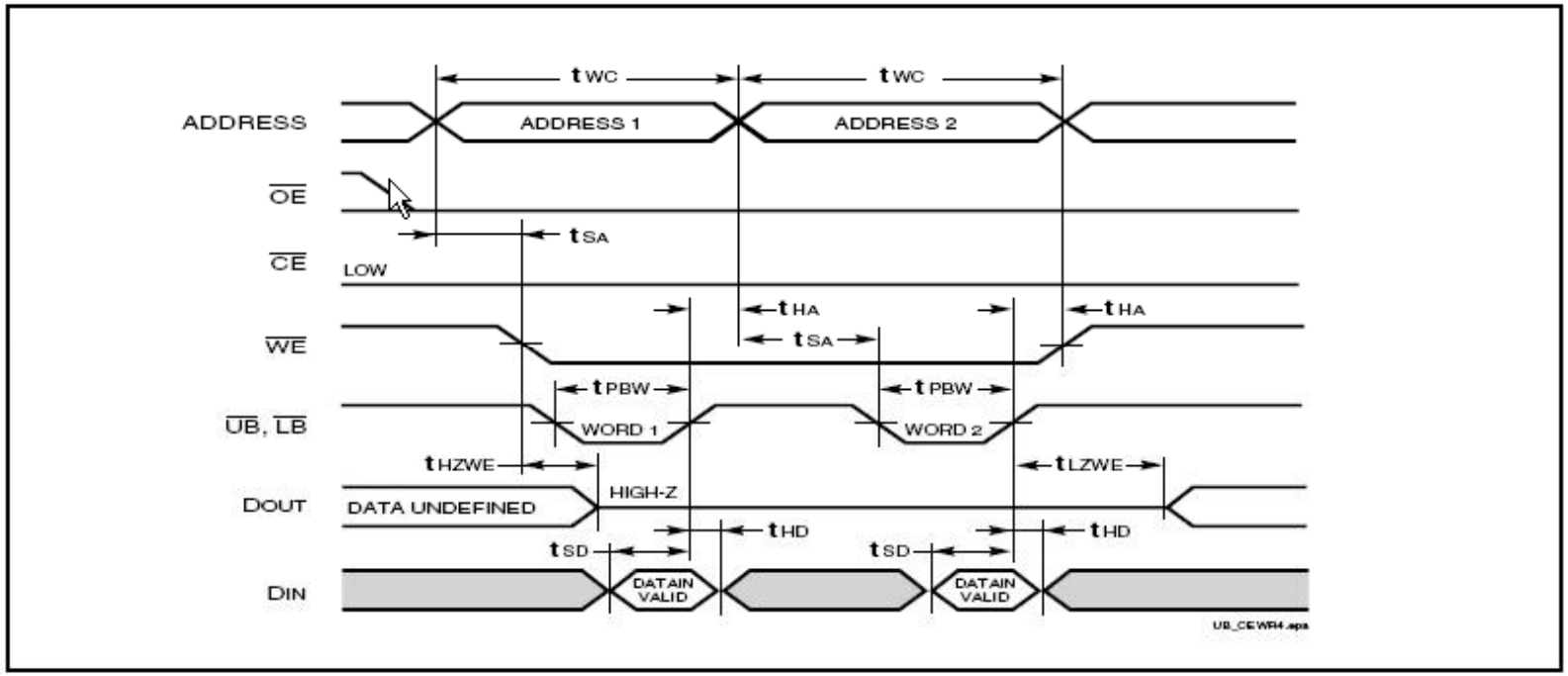


Notes:
1. $\overline{WE}$ is HIGH for a Read Cycle.
2. The device is continuously selected. $\overline{OE}$, $\overline{CE}$, $\overline{UB}$, or $\overline{LB}$ = V_{IL}.
3. Address is valid prior to or coincident with $\overline{CE}$ LOW transition.

메모리 스펙

WRITE

WRITE CYCLE NO. 4 ($\overline{LB}$, $\overline{UB}$ Controlled, Back-to-Back Write) [1,3]

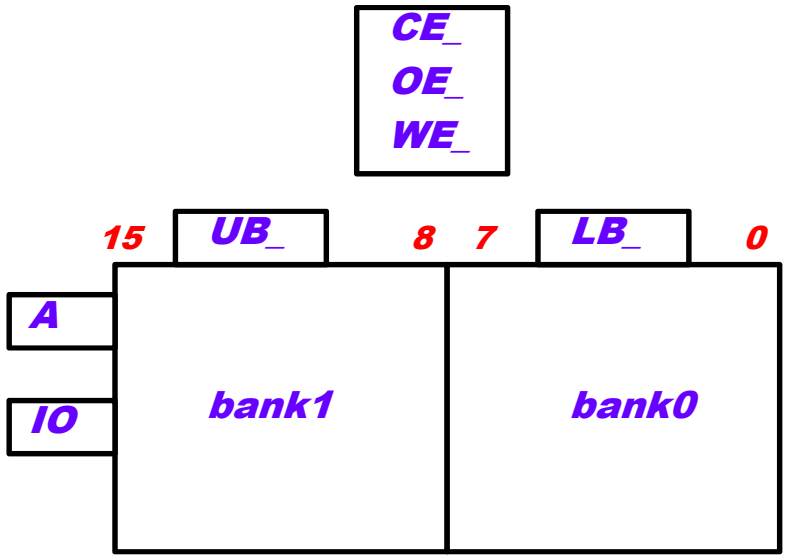


Notes:

1. The internal Write time is defined by the overlap of $\overline{CE}$ = LOW, $\overline{UB}$ and/or $\overline{LB}$ = LOW, and $\overline{WE}$ = LOW. All signals must be in valid states to initiate a Write, but any can be deasserted to terminate the Write. The t_{SA}, t_{HA}, t_{SD}, and t_{HD} timing is referenced to the rising or falling edge of the signal that terminates the Write.
2. Tested with $\overline{OE}$ HIGH for a minimum of 4 ns before $\overline{WE}$ = LOW to place the I/O in a HIGH-Z state.
3. $\overline{WE}$ may be held LOW across many address cycles and the $\overline{LB}$, $\overline{UB}$ pins can be used to control the Write function.

메모리 소스 (IS61LV25616AL.v)

```
`timescale 1ns/10ps

module IS61LV25616 (A, IO, CE_, OE_, WE_, LB_, UB_);

parameter dqbits = 16;
parameter memdepth = 262143;
parameter addbits = 18;
parameter Toha  = 2;

parameter Tsa   = 2;
input CE_, OE_, WE_, LB_, UB_;
input [(addbits - 1) : 0] A;
inout [(dqbits - 1) : 0] IO;

wire [      15      : 0] dout;
reg  [       7      : 0] bank0 [0 : memdepth];
reg  [       7      : 0] bank1 [0 : memdepth];
// wire [(dqbits - 1) : 0] memprobe = {bank1[A], bank0[A]};

wire r_en = WE_ & (~CE_) & (~OE_);
wire w_en = (~WE_) & (~CE_) & ((~LB_) | (~UB_));
assign #(r_en ? Taa : Thzce) IO = r_en ? dout : 16'bz;

initial
  $timeformat (-9, 0.1, " ns", 10);

assign dout [       7       : 0]        = LB_ ? 8'bz : bank0[A];
assign dout [      15       :      8]   = UB_ ? 8'bz : bank1[A];
//always @(A or w_en)
always @*
  begin
    #Tsa
    if (w_en)
      #Thzwe
      begin
        bank0[A] = LB_ ? bank0[A] : IO [       7       : 0];
        bank1[A] = UB_ ? bank1[A] : IO [      15       :      8    ];
      end
  end
```



메모리로부터
read

메모리에
write

Uart introduction

main.c

```
// main.c
#include "bsp.h"
#include "uart.h"

char msg[20]="Hello World!₩r₩n";

int main()
{
   uart_init(UART_BAUD, UART_FREQ);
   uart_put_string(msg);
   return(0);
}
```

Uart introduction

uart.c

```
// uart.c
#include "uart.h"
#define ADDR_UART_START    0x4000 0000
//-------------------------------------------------------
// Register access macros
#define REG32(add) *((volatile uint32_t *)(add))
#define REGR(add)  REG32(add)
//-------------------------------------------------------
#define UART_RB_THR   (ADDR_UART_START+0x00)  // read for RB, write for THR
#define UART_IER      (ADDR_UART_START+0x04)  // read/write
#define UART_IIR_FCR  (ADDR_UART_START+0x08)  // read for IIR, write for FCR
#define UART_LCR      (ADDR_UART_START+0x0C)  // read/write (0x03) -- line_control
#define UART_MCR      (ADDR_UART_START+0x10)  // write only (0x00) -- modem_control
#define UART_LSR      (ADDR_UART_START+0x14)  // read only -- line_status
#define UART_MSR      (ADDR_UART_START+0x18)  // read only -- modem_status

//-------------------------------------------------------
void uart_init(uint32_t baud, uint32_t freq) {
   uint32_t divisorH, divisorL;
   divisorH                = 0x00;
   divisorL                = 0x0e;     // 13.56   25Mhz / 115200 * 16
0C REGR(UART_LCR)          = 0x83;     // DLAB (7th bit of LCR) = 1
04 REGR(UART_IER)          =  0x00     // Divisor latch MSB
00 REGR(UART_RB_THR)       =  0x0e     // Divisor latch LSB
0C REGR(UART_LCR)          = 0x03;     // DLAB (7th bit of LCR) = 0
08 REGR(UART_IIR_FCR)      = 0x01;     // receiver FIFO trigger level
04 REGR(UART_IER)          = 0x01;     // receive-data available interrupt
}
```

Register access macros

UART CSR address definition

UART initialization with given frequency and baud rate

통신속도 정의

인터럽트 정의

Uart introduction

Uart.c

```
//-----------------------------------------------------
uint32_t uart_put_char(uint32_t c)
{
   while (!(REGR(UART_LSR)&0x20));
   REGR(UART_RB_THR) = c;
   return(c);
}

//-----------------------------------------------------
uint32_t uart_put_string(char* s)
{
   int i;
   char c;
   while ((c=*s++)!='₩0') {uart_put_char(c); i++; }
   return (i);
}


//-----------------------------------------------------
uint32_t uart_get_char(void)
{
   while (!( REGR(UART_LSR)&0x01));
   return REGR(UART_RB_THR);
}
```

Print Just one character

Print String
Call uart_put_char repeatly

Get one character

Uart introduction

Register list

Name	Address	Width	Access	Description
Receiver Buffer	0	8	R	Receiver FIFO output
Transmitter Holding Register (THR)	0	8	W	Transmit FIFO input
Interrupt Enable	1	8	RW	Enable/Mask interrupts generated by the UART
Interrupt Identification	2	8	R	Get interrupt information
FIFO Control	2	8	W	Control FIFO options
Line Control Register	3	8	RW	Control connection
Modem Control	4	8	W	Controls modem
Line Status	5	8	R	Status information
Modem Status	6	8	R	Modem Status

Interrupt Enable Register(IER)

This register allows enabling and disabling interrupt generation by the UART.

Bit #	Access	Description
0	RW	Received Data available interrupt '0' – disabled '1' – enabled
1	RW	Transmitter Holding Register empty interrupt '0' – disabled '1' – enabled
2	RW	Receiver Line Status Interrupt '0' – disabled '1' – enabled
3	RW	Modem Status Interrupt '0' – disabled '1' – enabled
7-4	RW	Reserved. Should be logic '0'.

Reset Value: 00h

0x00
0000 0000

0x01
0000 0001

데이터 수신하면 인터럽트로 사용할지 말지 결정

Uart introduction

Line Control Register(LCR)

The line control register allows the specification of the format of the asynchronous data communication used. A bit in the register also allows access to the Divisor Latches, which define the baud rate. Reading from the register is allowed to check the current settings of the communication.

Bit #	Access	Description
1-0	RW	Select number of bits in each character '00' – 5 bits '01' – 6 bits '10' – 7 bits '11' – 8 bits
2	RW	Specify the number of generated stop bits '0' – 1 stop bit '1' – 1.5 stop bits when 5-bit character length selected and 2 bits otherwise Note that the receiver always checks the first stop bit only.
3	RW	Parity Enable '0' – No parity '1' – Parity bit is generated on each outgoing character and is checked on each incoming one.
4	RW	Even Parity select '0' – Odd number of '1' is transmitted and checked in each word (data and parity combined). In other words, if the data has an even number of '1' in it, then the parity bit is '1'. '1' – Even number of '1' is transmitted in each word.
5	RW	Stick Parity bit. '0' – Stick Parity disabled '1' - If bits 3 and 4 are logic '1', the parity bit is transmitted and checked as logic '0'. If bit 3 is '1' and bit 4 is '0' then the parity bit is transmitted and checked as '1'.
6	RW	Break Control bit '1' – the serial out is forced into logic '0' (break state). '0' – break is disabled
7	RW	Divisor Latch Access bit. '1' – The divisor latches can be accessed '0' – The normal registers are accessed

Reset Value: 00000011b

0x83
1000 0011

전송 패킷 8bit
Divisor latch **에 접근**
→ 통신속도 설정

0x03
0000 0011

일반 노멀
레지스터에 접근

Uart introduction

Interrupt Identification Register(IIR)

The IIR enables the programmer to retrieve what is the current highest priority pending interrupt.
Bit 0 indicates that an interrupt is pending when it's logic '0'. When it's '1' – no interrupt is pending.
The following table displays the list of possible interrupts along with the bits they enable, priority, and their source and reset control.

Bit 3	Bit 2	Bit 1	Priority	Interrupt Type	Interrupt Source	Interrupt Reset Control
0	1	1	1st	Receiver Line Status	Parity, Overrun or Framing errors or Break Interrupt	Reading the Line Status Register
0	1	0	2nd	Receiver Data available	FIFO trigger level reached	FIFO drops below trigger level
1	1	0	2nd	Timeout Indication	There's at least 1 character in the FIFO but no character has been input to the FIFO or read from it for the last 4 Char times.	Reading from the FIFO (Receiver Buffer Register)
0	0	1	3rd	Transmitter Holding Register empty	Transmitter Holding Register Empty	Writing to the Transmitter Holding Register or reading IIR.
0	0	0	4th	Modem Status	CTS, DSR, RI or DCD.	Reading the Modem status register.

Bits 4 and 5: Logic '0'.
Bits 6 and 7: Logic '1' for compatibility reason.

Reset Value: C1h

Uart introduction

FIFO Control Register(FCR)

Bit #	Access	Description
0	W	Ignored (Used to enable FIFOs in NS16550D). Since this UART only supports FIFO mode, this bit is ignored.
1	W	Writing a '1' to bit 1 clears the Receiver FIFO and resets its logic. But it doesn't clear the shift register, i.e. receiving of the current character continues.
2	W	Writing a '1' to bit 2 clears the Transmitter FIFO and resets its logic. The shift register is not cleared, i.e. transmitting of the current character continues.
5-3	W	Ignored
7-6	W	Define the Receiver FIFO Interrupt trigger level '00' – 1 byte '01' – 4 bytes '10' – 8 bytes '11' – 14 bytes

Reset Value : 11000000b

0x01
0000 0001

1 byte 가 들어오면
Interrupt 로 인식

Uart introduction

Modem Control Register(MCR)

The modem control register allows transferring control signals to a modem connected to the UART.

Bit #	Access	Description
0	W	Data Terminal Ready (DTR) signal control '0' – DTR is '1' '1' – DTR is '0'
1	W	Request To Send (RTS) signal control '0' – RTS is '1' '1' – RTS is '0'
2	W	Out1. In loopback mode, connected Ring Indicator (RI) signal input
3	W	Out2. In loopback mode, connected to Data Carrier Detect (DCD) input.
4	W	Loopback mode '0' – normal operation '1' – loopback mode. When in loopback mode, the Serial Output Signal (STX_PAD_O) is set to logic '1'. The signal of the transmitter shift register is internally connected to the input of the receiver shift register. The following connections are made: DTR → DSR RTS → CTS Out1 → RI Out2 → DCD
7-5	W	Ignored

Reset Value: 0

Uart introduction

Line Status Register(LSR)
-- 읽기 전용 레지스터

0x20
0010 0000

송신 준비가 완료 되었음

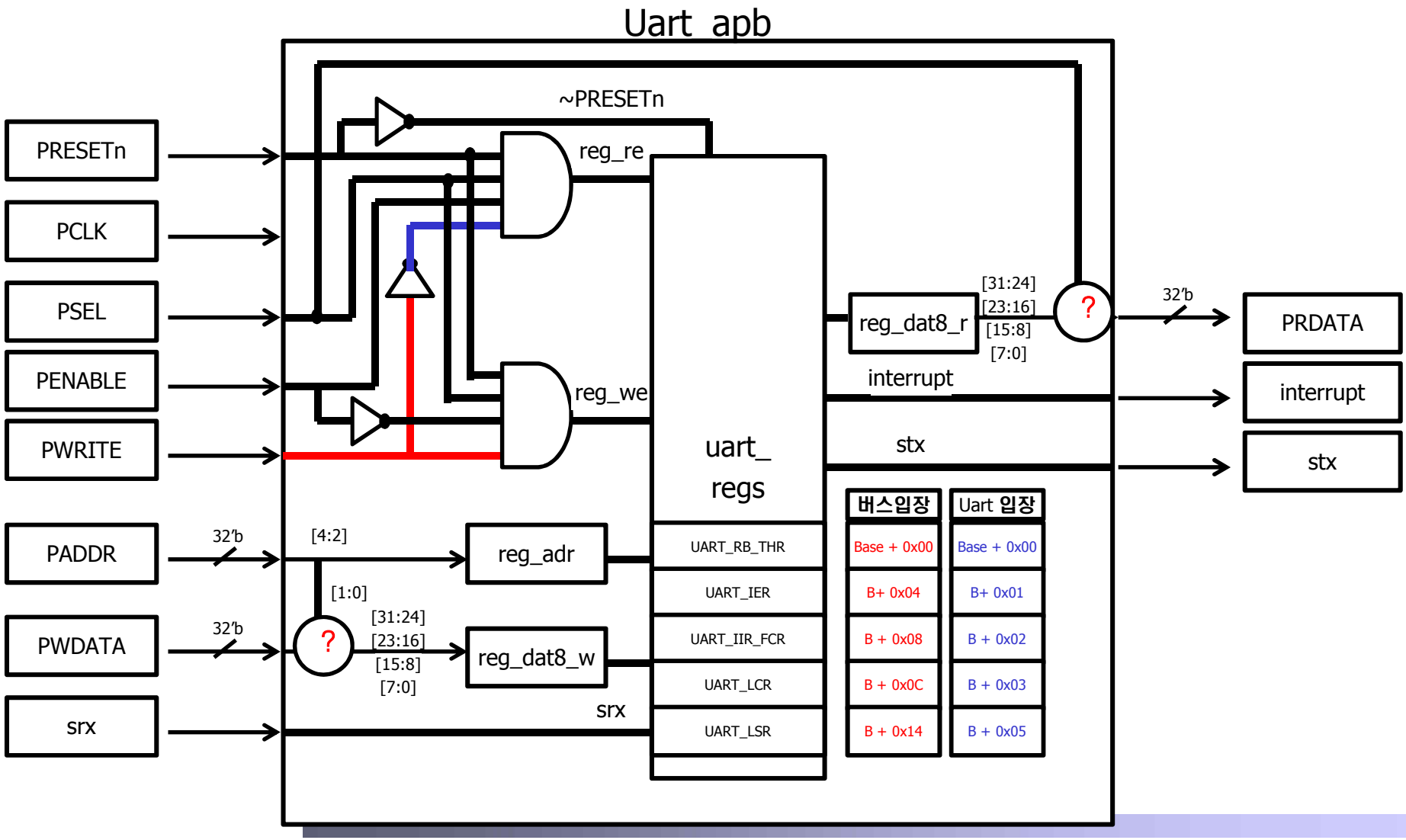
0x01
0000 0001

최소한 하나의 문자가 들어왔음

Bit #	Access	Description
0	R	Data Ready (DR) indicator. '0' – No characters in the FIFO '1' – At least one character has been received and is in the FIFO.
1	R	Overrun Error (OE) indicator '1' – If the FIFO is full and another character has been received in the receiver shift register. If another character is starting to arrive, it will overwrite the data in the shift register but the FIFO will remain intact. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '0' – No overrun state
2	R	Parity Error (PE) indicator '1' – The character that is currently at the top of the FIFO has been received with parity error. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '0' – No parity error in the current character
3	R	Framing Error (FE) indicator '1' – The received character at the top of the FIFO did not have a valid stop bit. Of course, generally, it might be that all the following data is corrupt. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '0' – No framing error in the current character
4	R	Break Interrupt (BI) indicator '1' –A break condition has been reached in the current character. The break occurs when the line is held in logic 0 for a time of one character (start bit + data + parity + stop bit). In that case, one zero character enters the FIFO and the UART waits for a valid start bit to receive next character. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '0' – No break condition in the current character
5	R	Transmit FIFO is empty. '1' – The transmitter FIFO is empty. Generates Transmitter Holding Register Empty interrupt. The bit is cleared when data is being been written to the transmitter FIFO. '0' – Otherwise
6	R	Transmitter Empty indicator. '1' – Both the transmitter FIFO and transmitter shift register are empty. The bit is cleared when data is being been written to the transmitter FIFO. '0' – Otherwise
7	R	'1' – At least one parity error, framing error or break indications have been received and are inside the FIFO. The bit is cleared upon reading from the register. '0' – Otherwise.

Uart introduction

▣ Uart_apb 구조

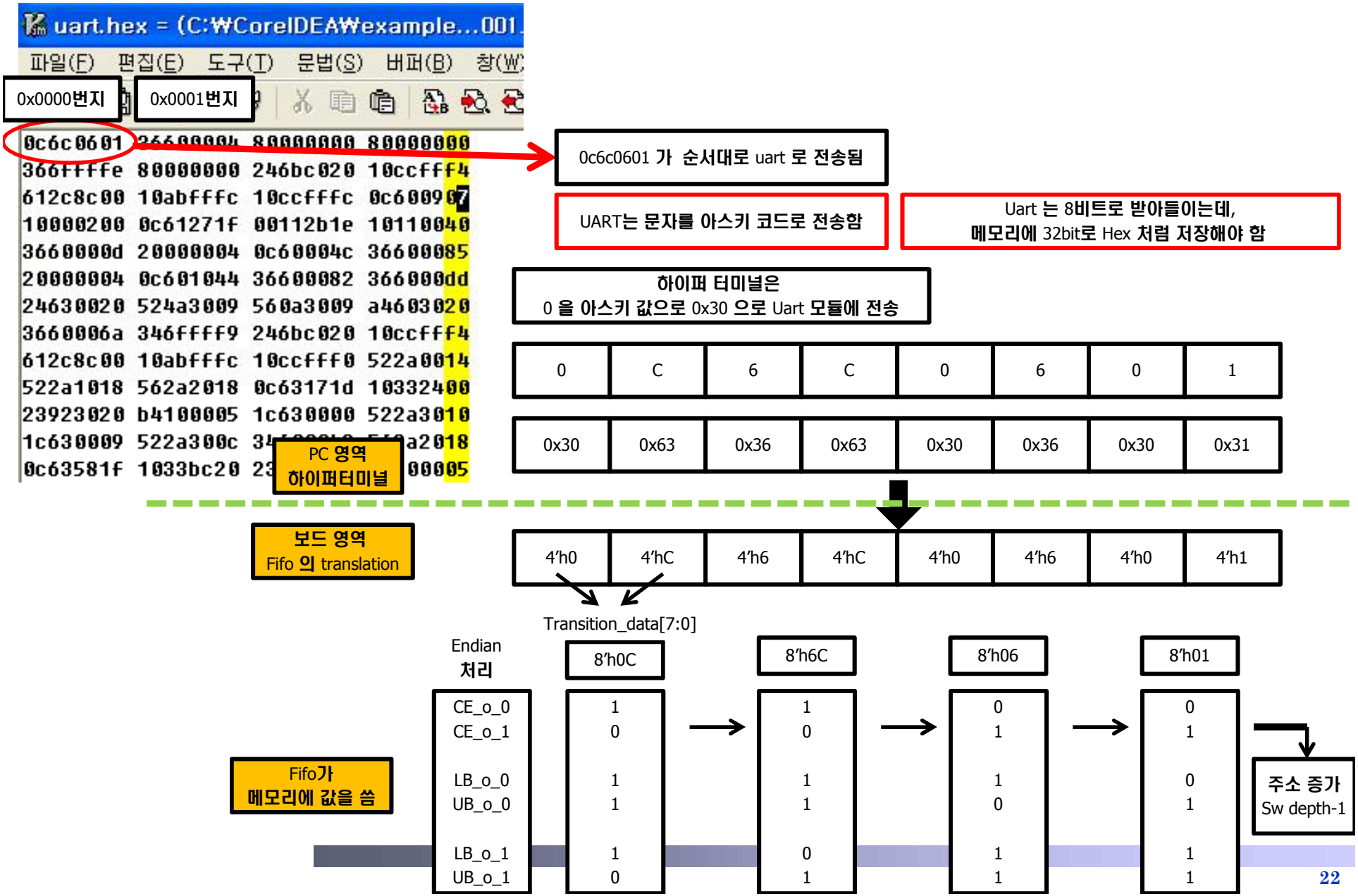


uart_apb.v

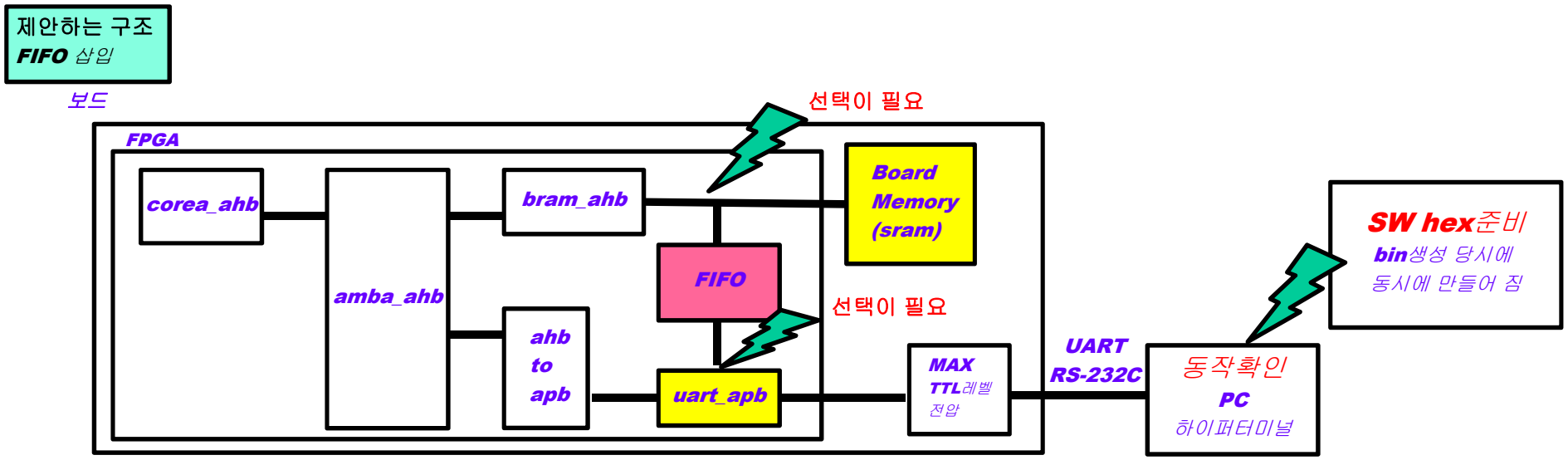
```
module uart_apb (
       PRESETn,
       PCLK,
       PSEL,
       PENABLE,
       PADDR,
       PWRITE,
       PWDATA,
       PRDATA,

       interrupt, // interrupt request (active-high)
       stx,   // serial output
       srx    // serial input
);
   input         PRESETn;    wire        PRESETn;
   input         PCLK;       wire        PCLK;
   input         PSEL;       wire        PSEL;
   input         PENABLE;    wire        PENABLE;
   input  [31:0] PADDR;      wire [31:0] PADDR;
   input         PWRITE;     wire        PWRITE;
   output [31:0] PRDATA;     wire [31:0] PRDATA;
   input  [31:0] PWDATA;     wire [31:0] PWDATA;
   output        interrupt;  wire        interrupt;
   input         srx;        wire        srx;
   output        stx;        wire        stx;
```

```
assign reg_we  = PRESETn & PSEL & ~PENABLE &  PWRITE;
assign reg_re  = PRESETn & PSEL &  PENABLE & ~PWRITE;
assign reg_adr = PADDR[4:2]; //assign adr_o   = PADDR[2:0];
assign PRDATA  = (PSEL) ? {4{reg_dat8_r}} : 'h0;
always @ (PADDR[1:0] or PWDATA) begin
          case (PADDR[1:0])
          `ifdef ENDIAN_BIG
          2'b00: #1 reg_dat8_w = PWDATA[31:24];
          2'b01: #1 reg_dat8_w = PWDATA[23:16];
          2'b10: #1 reg_dat8_w = PWDATA[15:8];
          2'b11: #1 reg_dat8_w = PWDATA[7:0];
          `else // little-endian -- default
          2'b00: #1 reg_dat8_w = PWDATA[7:0];
          2'b01: #1 reg_dat8_w = PWDATA[15:8];
          2'b10: #1 reg_dat8_w = PWDATA[23:16];
          2'b11: #1 reg_dat8_w = PWDATA[31:24];
          `endif
          endcase
end

uart_regs Uregs(
       .clk         (PCLK),
       .wb_rst_i    (~PRESETn),
       .wb_addr_i   (reg_adr),
       .wb_dat_i    (reg_dat8_w),
       .wb_dat_o    (reg_dat8_r),
       .wb_we_i     (reg_we),
       .wb_re_i     (reg_re),
       .modem_inputs({~ctsn, dsr_pad_i, ri_pad_i,  dcd_pad_i}),
       .stx_pad_o   (stx),
       .srx_pad_i   (srx),
       .rts_pad_o   (rts_internal),
       .dtr_pad_o   (dtr_pad_o),
       .int_o       (interrupt)
);
endmodule
```

하이퍼 터미널을 통한 데이터 포맷의 변환 과정

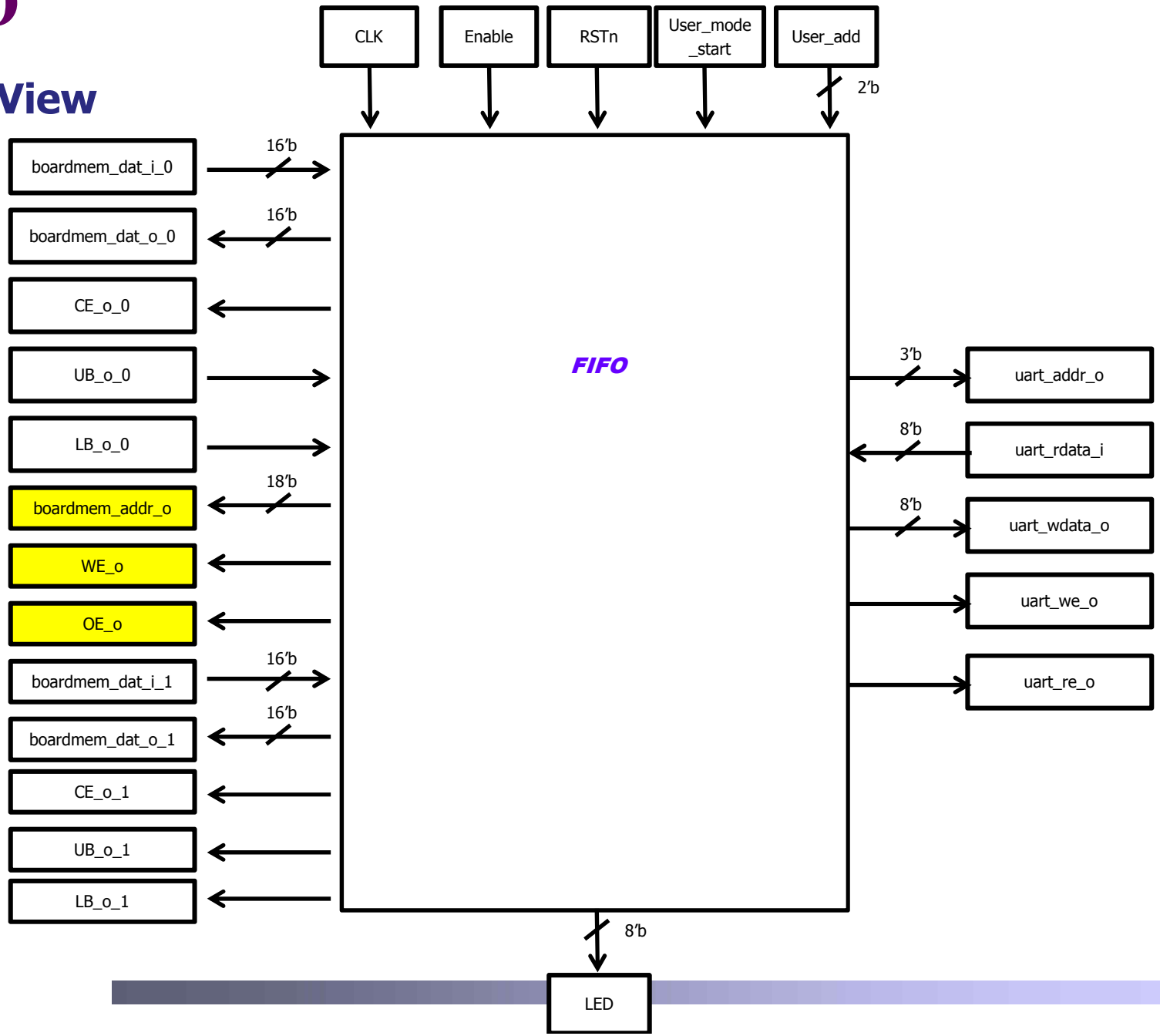


FIFO


제안하는 구조
FIFO 삽입
보드
선택이 필요
FPGA
corea_ahb
amba_ahb
bram_ahb
Board
Memory
(sram)
FIFO
선택이 필요
ahb
to
apb
uart_apb
MAX
TTL레벨
전압
UART
RS-232C
동작확인
PC
하이퍼터미널
SW hex준비
bin생성 당시에
동시에 만들어 짐

FIFO

Top View



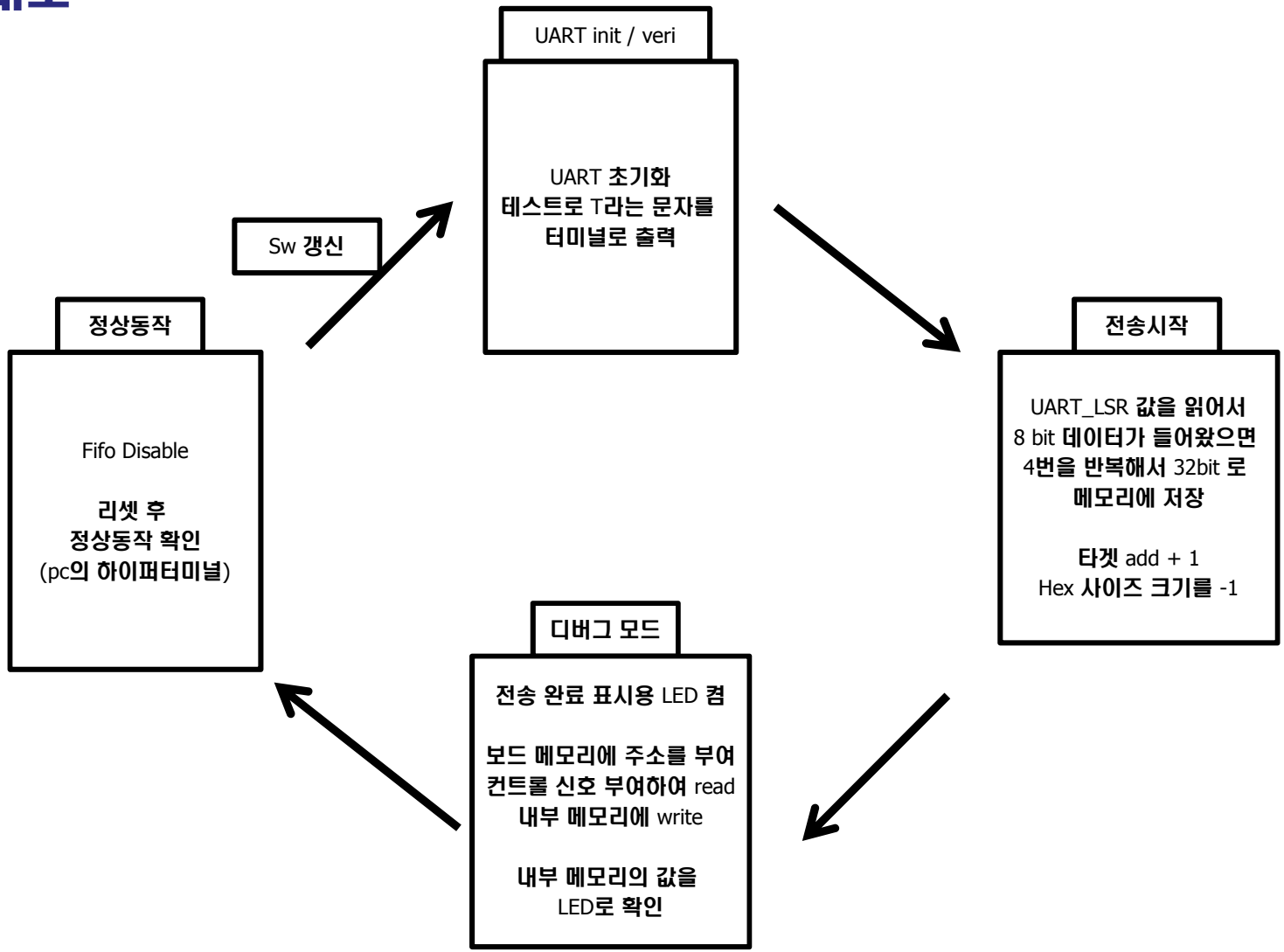
FIFO

▣ 개요

포트명	Width	Description
CLK	1	기준 클록
RSTn	1	입력 리셋
Enable	1	Fifo 의 동작 Enable
User_mode_start	1	유저모드의 시작 여부 (T : 1, F : 0)
User_add	2	유저모드일 경우 내부 램 블록으로 들어가는 주소 값
LED	8	Fifo 동작 완료 표시 또는 유저모드에서 instruction 값 출력
boardmem_dat_i_0, 1	16	보드 메모리로부터 입력되는 데이터 값
boardmem_dat_o_0, 1	16	보드 메모리로 출력하는 데이터 값
CE_o_0, 1	1	보드 메모리의 Chip Enable , low Active
UB_o_0, 1	1	보드 메모리의 상위 16비트 동작 Enable , low Active
LB_o_0, 1	1	보드 메모리의 하위 16비트 동작 Enable , low Active
boardmem_addr_o	18	보드 메모리로 출력하는 주소 값
WE_o	1	보드 메모리로 출력하는 Write Enable, low Active
OE_o	1	보드 메모리로 출력하는 Output Enable, low Active
uart_addr_o	3	Uart 로 출력하는 address 값
uart_rdata_i	8	Uart 에서 입력되는 데이터 값
uart_wdata_o	8	Uart 로 출력하는 데이터 값
uart_we_o	1	Uart 로 출력하는 Write Enable
uart_re_o	1	Uart 로 출력하는 Read Enable

FIFO

▣ 상태도


UART init / veri
UART 초기화
테스트로 T라는 문자를
터미널로 출력
전송시작
UART_LSR 값을 읽어서
8 bit 데이터가 들어왔으면
4번을 반복해서 32bit 로
메모리에 저장
타겟 add + 1
Hex 사이즈 크기를 -1
디버그 모드
전송 완료 표시용 LED 켬
보드 메모리에 주소를 부여
컨트롤 신호 부여하여 read
내부 메모리에 write
내부 메모리의 값을
LED로 확인
정상동작
Fifo Disable
리셋 후
정상동작 확인
(pc의 하이퍼터미널)
Sw 갱신

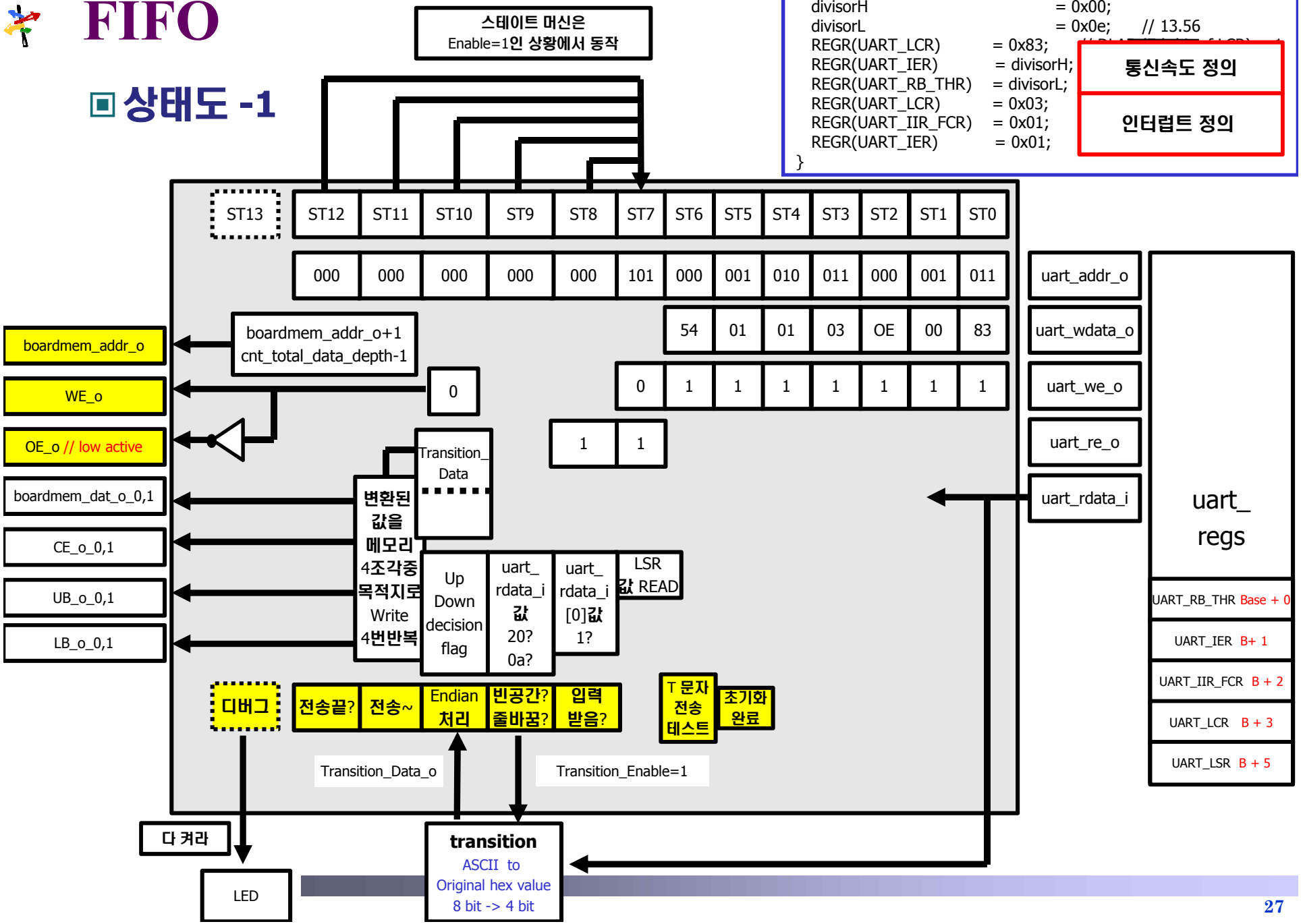
FIFO

▣ 상태도 -1

```
void uart_init(uint32_t baud, uint32_t freq) {
   uint32_t divisorH, divisorL;
   divisorH                          = 0x00;
   divisorL                          = 0x0e;      // 13.56
   REGR(UART_LCR)          = 0x83;
   REGR(UART_IER)           = divisorH;
   REGR(UART_RB_THR)     = divisorL;
   REGR(UART_LCR)          = 0x03;
   REGR(UART_IIR_FCR)    = 0x01;
   REGR(UART_IER)           = 0x01;
}
```

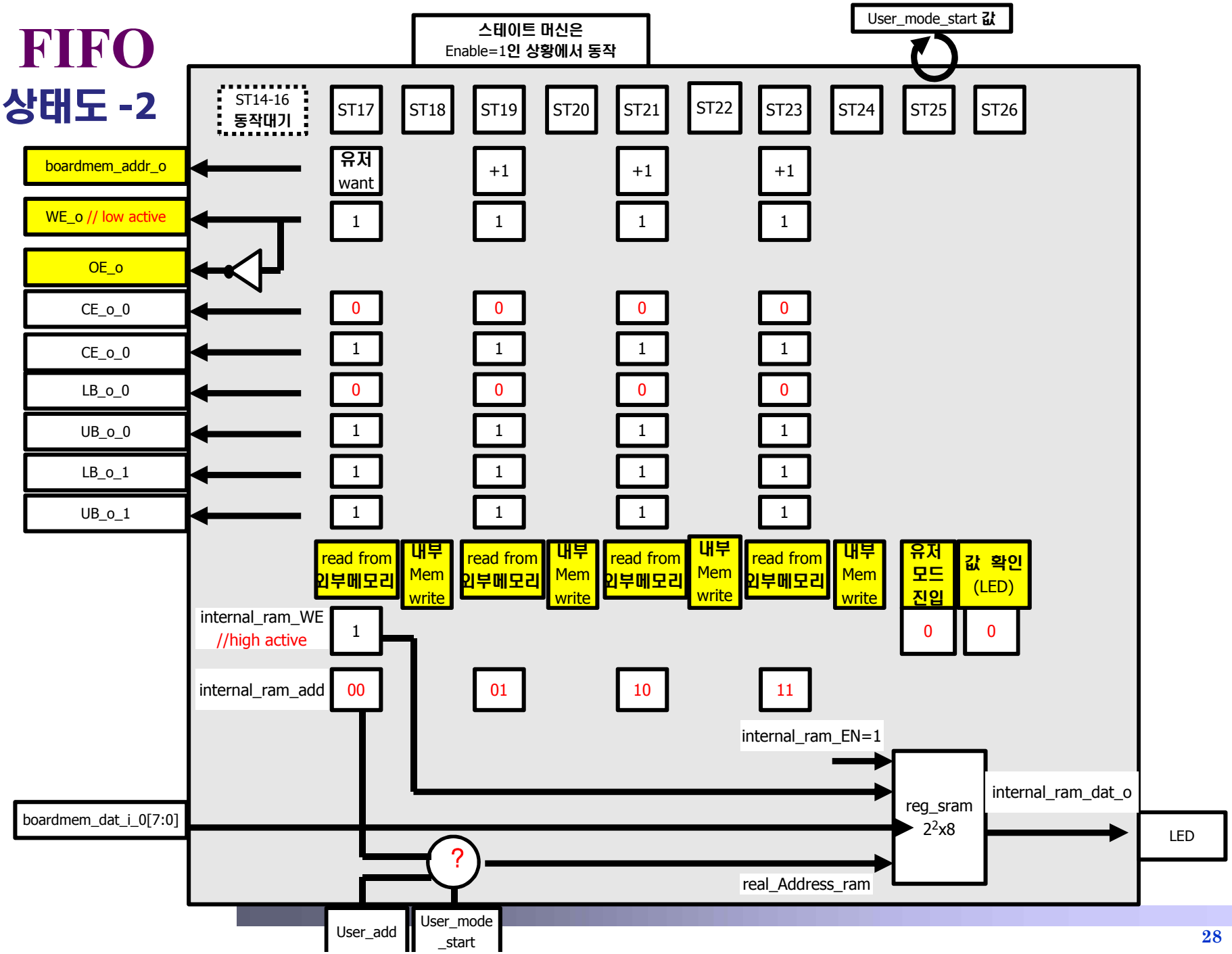
통신속도 정의

인터럽트 정의



FIFO

▣ 상태도 -2

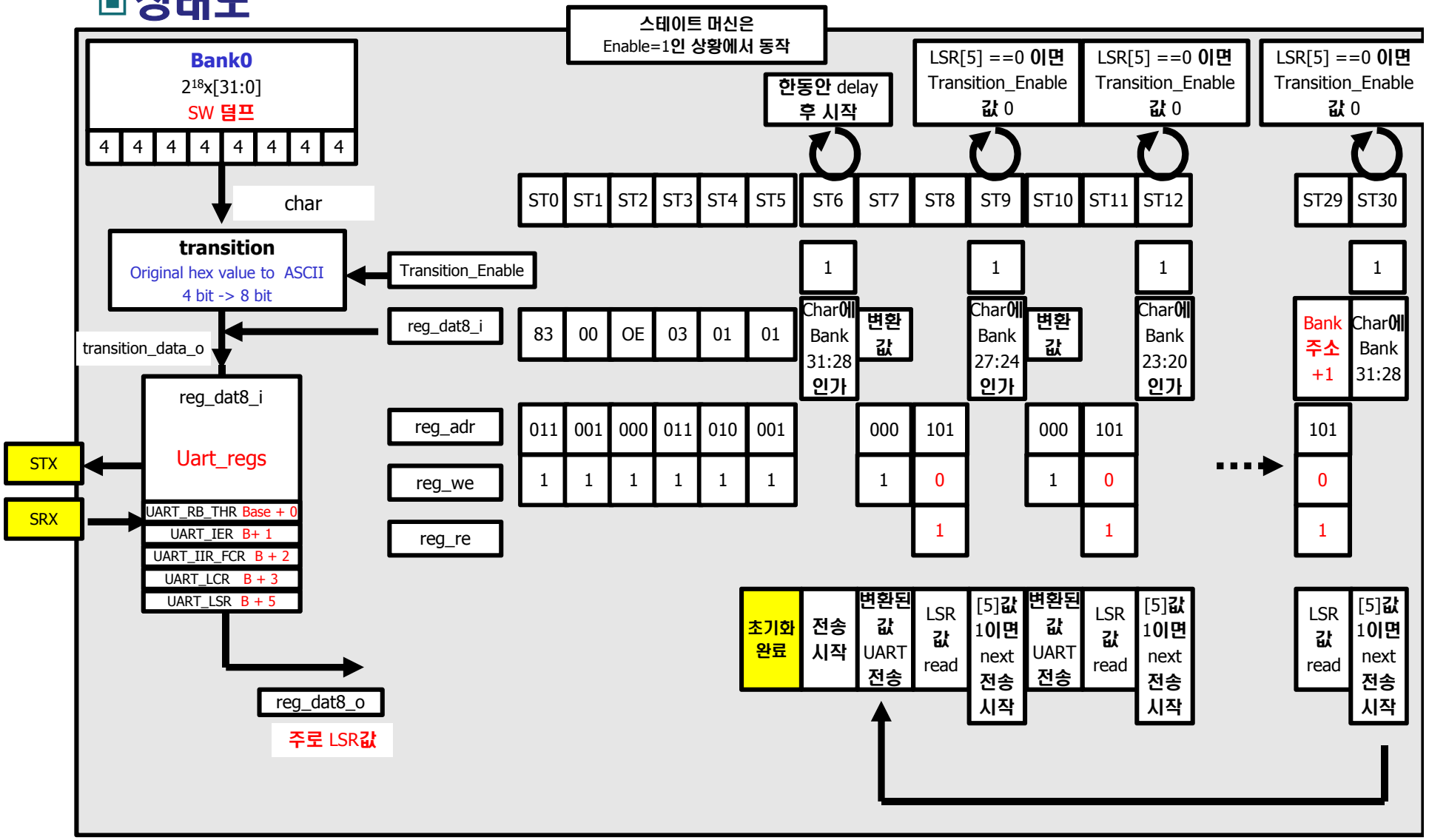

스테이트 머신은
Enable=1인 상황에서 동작
User_mode_start 값
ST14-16
동작대기
ST17
ST18
ST19
ST20
ST21
ST22
ST23
ST24
ST25
ST26
boardmem_addr_o
유저
want
+1
+1
+1
WE_o // low active
1
1
1
1
OE_o
CE_o_0
0
0
0
0
CE_o_0
1
1
1
1
LB_o_0
0
0
0
0
UB_o_0
1
1
1
1
LB_o_1
1
1
1
1
UB_o_1
1
1
1
1
read from 외부메모리
내부 Mem write
read from 외부메모리
내부 Mem write
read from 외부메모리
내부 Mem write
read from 외부메모리
내부 Mem write
유저 모드 진입
값 확인 (LED)
0
0
internal_ram_WE
//high active
1
internal_ram_add
00
01
10
11
internal_ram_EN=1
boardmem_dat_i_0[7:0]
reg_sram
2²x8
internal_ram_dat_o
LED
?
real_Address_ram
User_add
User_mode
_start

TTY 모듈

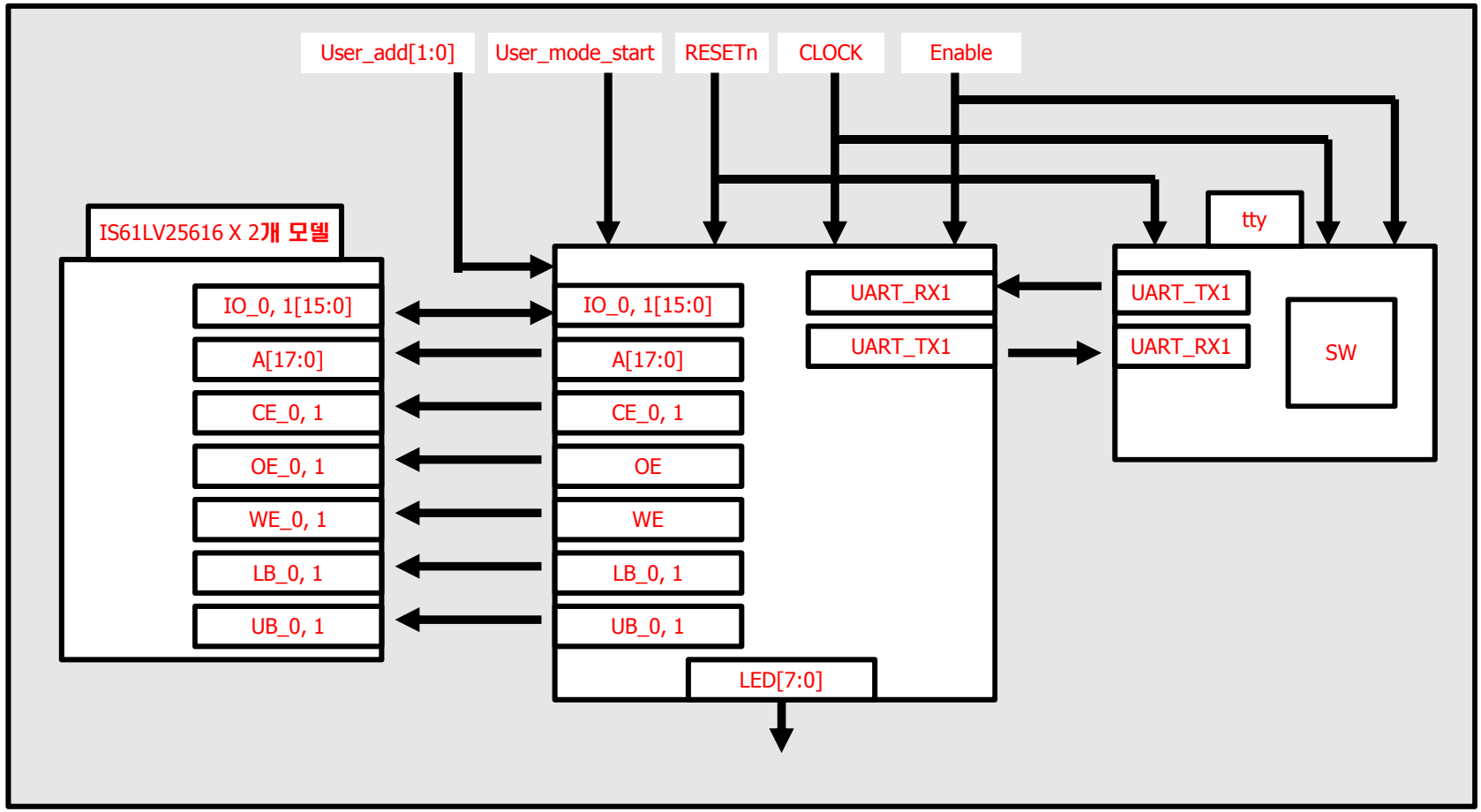
- PC의 하이퍼 터미널 모델
- PC의 SW 를 온-칩 환경의 Uart 모듈로 전송하기 위한 모듈
- Uart 모듈을 내부로 포함하며 FIFO 방식으로 SW 를 순차적으로 전송
- SW 가 담고 있는 코드의 개별 문자를 아스키 코드로 변환하여 전송

TTY 모듈

▣ 상태도


스테이트 머신은
Enable=1인 상황에서 동작
Bank0
2^18x[31:0]
SW 덤프
4 4 4 4 4 4 4 4
char
transition
Original hex value to ASCII
4 bit -> 8 bit
Transition_Enable
reg_dat8_i
transition_data_o
reg_dat8_i
Uart_regs
STX
SRX
UART_RB_THR Base + 0
UART_IER B+ 1
UART_IIR_FCR B + 2
UART_LCR B + 3
UART_LSR B + 5
reg_dat8_o
주로 LSR값
reg_adr
reg_we
reg_re
한동안 delay 후 시작
LSR[5] ==0 이면 Transition_Enable 값 0
LSR[5] ==0 이면 Transition_Enable 값 0
LSR[5] ==0 이면 Transition_Enable 값 0
ST0 ST1 ST2 ST3 ST4 ST5 ST6 ST7 ST8 ST9 ST10 ST11 ST12 ST29 ST30
83 00 OE 03 01 01
011 001 000 011 010 001
1 1 1 1 1 1
Char에 Bank 31:28 인가
변환 값
Char에 Bank 27:24 인가
변환 값
Char에 Bank 23:20 인가
Bank 주소 +1
Char에 Bank 31:28
000 101
1 0 1
000 101
1 0 1
101
0
1
초기화 완료
전송 시작
변환된 값 UART 전송
LSR 값 read
[5]값 1이면 next 전송 시작
변환된 값 UART 전송
LSR 값 read
[5]값 1이면 next 전송 시작
LSR 값 read
[5]값 1이면 next 전송 시작

테스트 벤치 환경



시뮬레이션 모델 (개발 보드의 외부 메모리)	합성될 영역 Fifo + uart	시뮬레이션 모델 (PC의 하이퍼터미널)

테스트 벤치 환경

▣ 테스트 시나리오

```
top_uart_fifo Utop_uart_fifo(
        .Enable(Enable),
        .User_add(User_add),
        .User_mode_start(User_mode_start),
        .LED(LED),
        .A(A),
        .IO_0(IO_0),
        .IO_1(IO_1),

        .CE_0(CE_0),
        .CE_1(CE_1),

        .WE(WE),
        .OE(OE),

        .LB_0(LB_0),
        .LB_1(LB_1),

        .UB_0(UB_0),
        .UB_1(UB_1),

        .RESETn(RESETn),        // active-lo
        .CLOCK(CLOCK),         // system clo
        .UART_TX1(UART_TX1),
        .UART_RX1(UART_RX1)
);
```

```
IS61LV25616 U_IS61LV25616(
        .A(A),
        .IO_0(IO_0),
        .IO_1(IO_1),
        .CE_0(CE_0),
        .CE_1(CE_1),
        .OE_0(~WE),
        .OE_1(~WE),
        .WE_0(WE),
        .WE_1(WE),
        .LB_0(LB_0),
        .LB_1(LB_1),
        .UB_0(UB_0),
        .UB_1(UB_1)
);

tty Utty(
        .RESETn(RESETn),
        .CLK(CLOCK),
        .STX(UART_RX1),
        .SRX(UART_TX1),
        .Enable(Enable)
);
```

```
initial begin


        CLOCK = 1'b0;
        Enable = 1'b1;
        RESETn = 1'b0;
        User_mode_start = 1'b0;
        User_add = 2'b00;

        #100 RESETn = 1'b1;

        #96348000 User_add = 2'b00;
        #20 User_mode_start = 1'b1;
        #200    User_add = 2'b01;
        #200    User_add = 2'b10;
        #200    User_add = 2'b11;

        #1000 Enable = 1'b0;
        #100    RESETn = 1'b0;
        #50     RESETn = 1'b1;
end

always begin
        #10 CLOCK = ~CLOCK;
end
```

리셋 / FIFO, tty Enable

리셋 해제 / sw **덤핑**

외부 메모리로부터
몇 개 코드 read **하여**
Fifo **내부의** reg_sram **으로**
Write **및** LED **로 값 확인**

Fifo, tty **동작** Disable

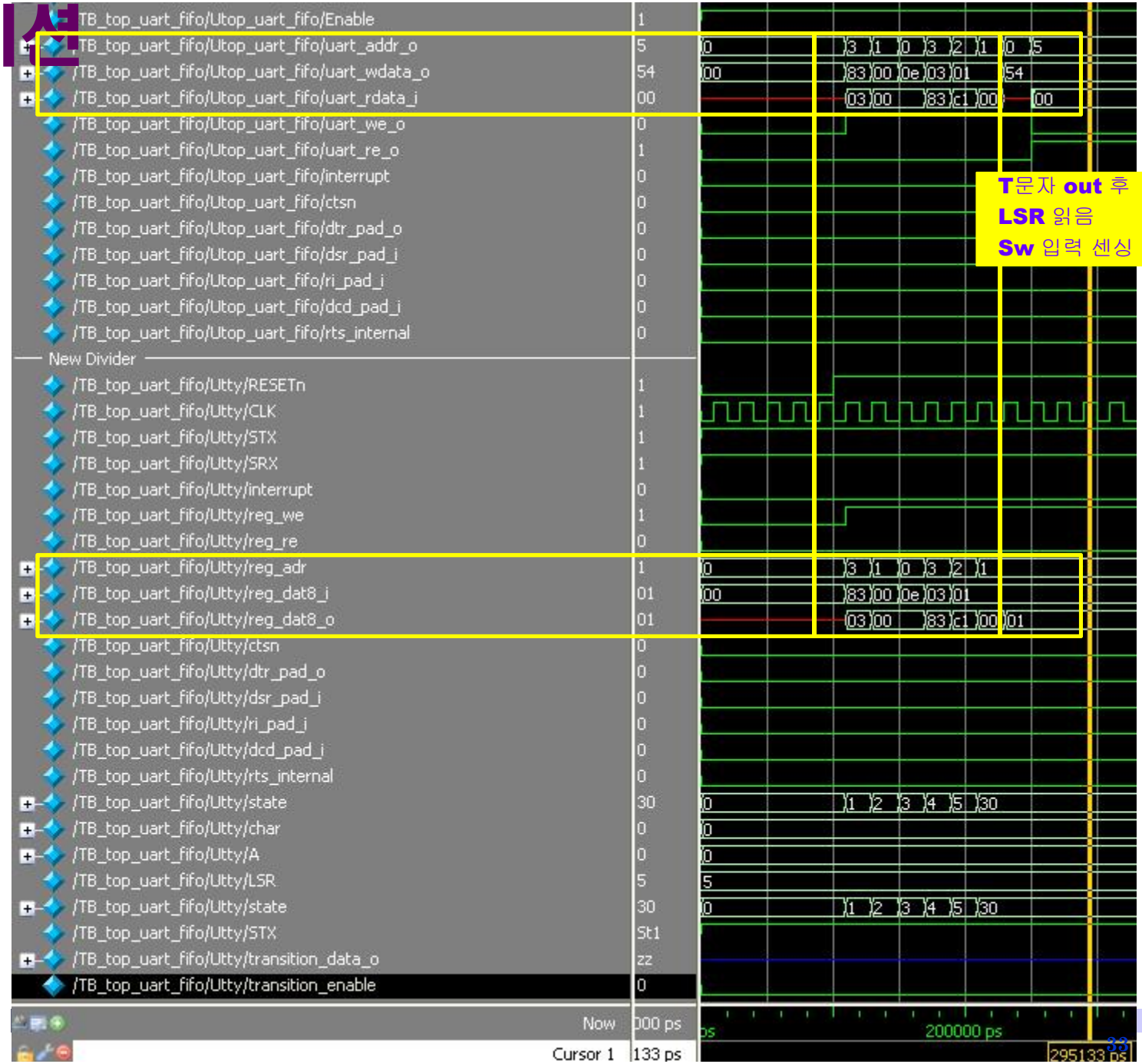
시뮬레이션

리셋 / FIFO, tty Enable

Uart **모듈 초기화**

Fifo 내부의 **Uart** 모듈

tty내부의 **Uart** 모듈



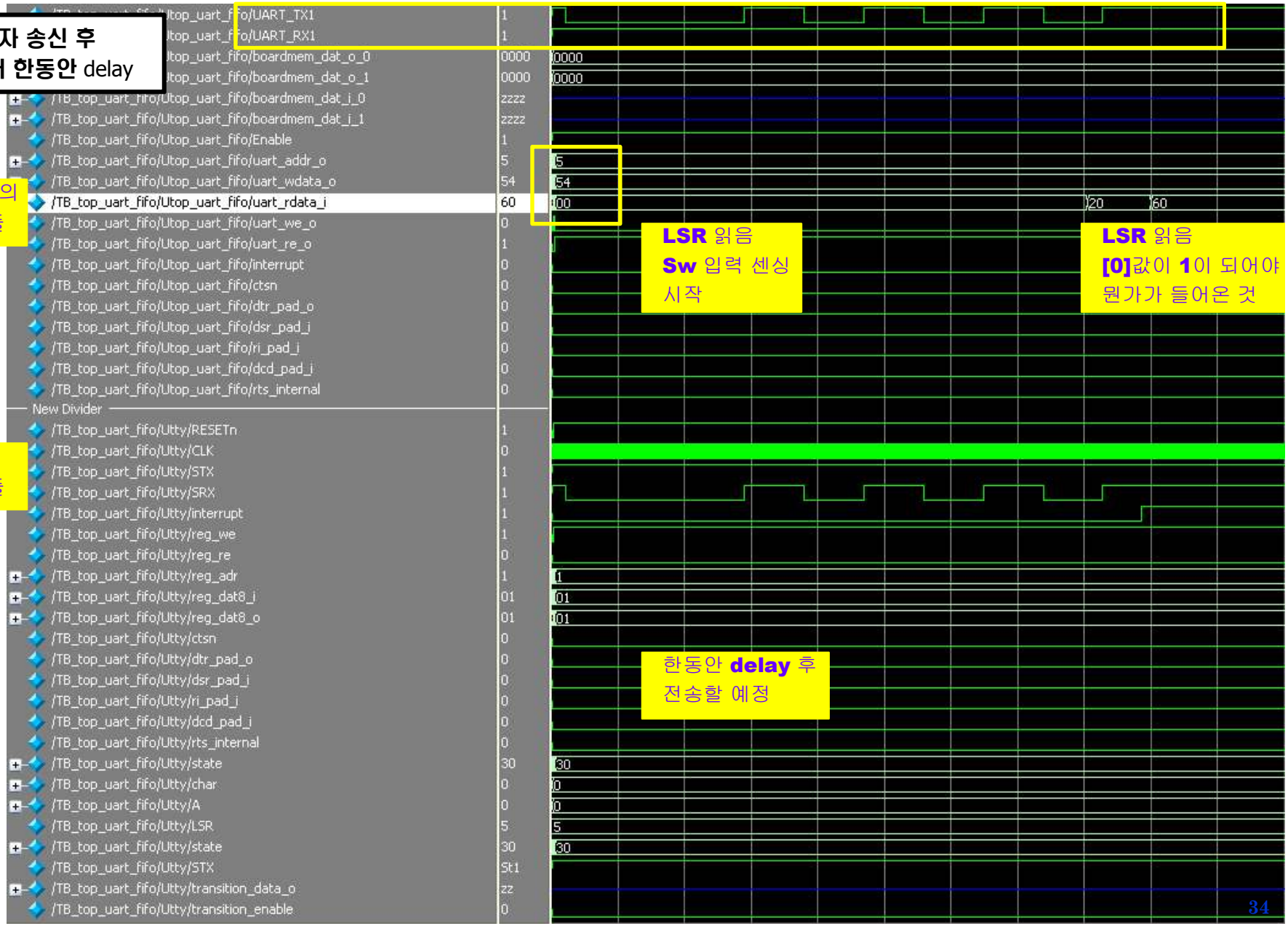
시뮬레이션

T문자 **out**

T **문자 송신 후**
TTY **에서 한동안** delay

Fifo 내부의 **Uart** 모듈

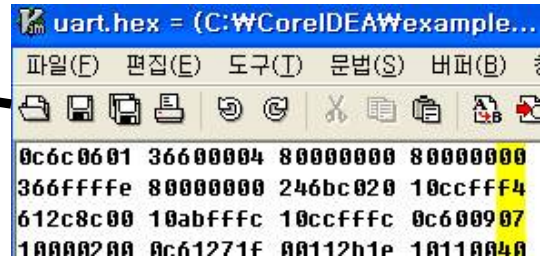
tty내부의 **Uart** 모듈

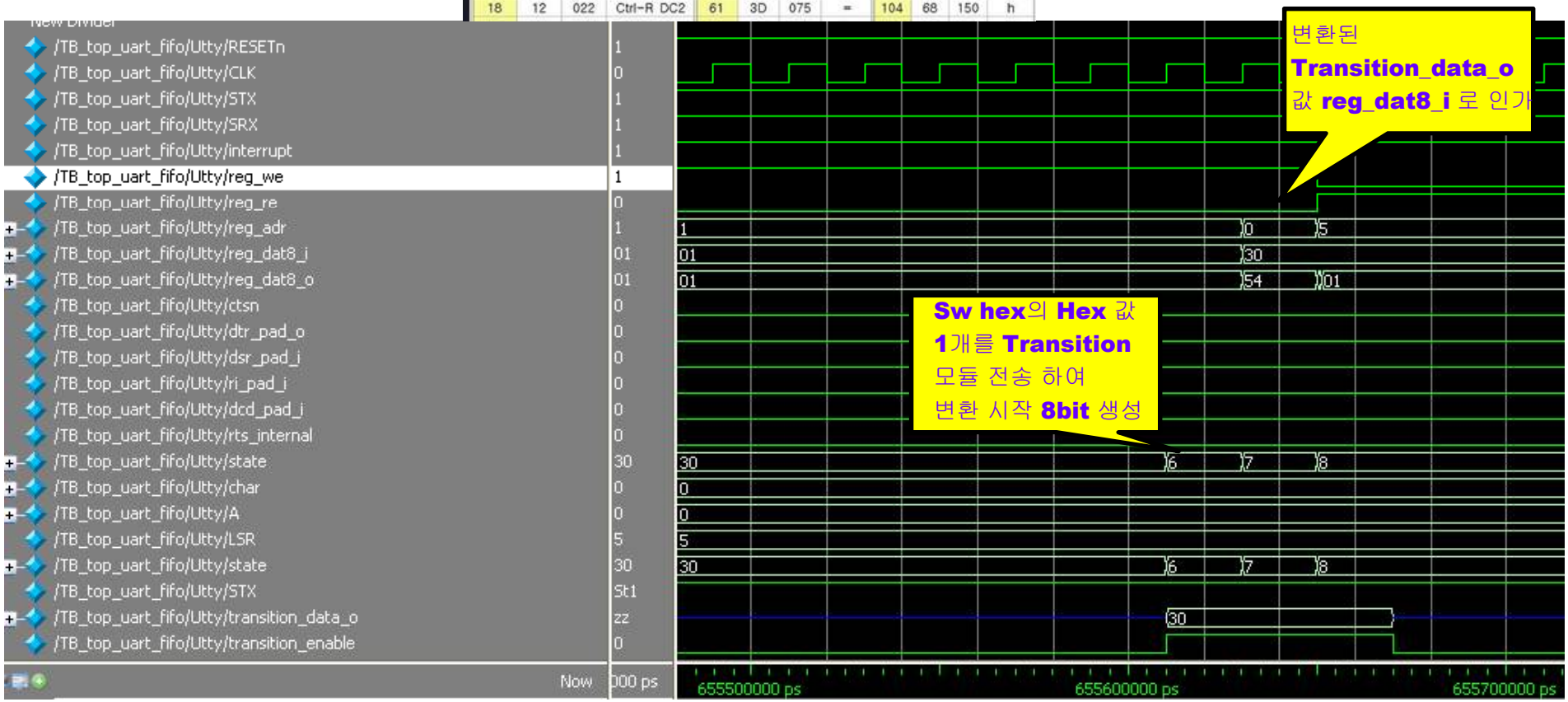


시뮬레이션

TTY 모듈의
Delay 종료 후 SW 전송 시작

DEC	HEX	OCT	Char	DEC	HEX	OCT	Char	DEC	HEX	OCT	Char
0	00	000	Ctrl-@ NUL	43	2B	053	+	86	56	126	V
1	01	001	Ctrl-A SOH	44	2C	054	,	87	57	127	W
2	02	002	Ctrl-B STX	45	2D	055	-	88	58	130	X
3	03	003	Ctrl-C ETX	46	2E	056	.	89	59	131	Y
4	04	004	Ctrl-D EOT	47	2F	057	/	90	5A	132	Z
5	05	005	Ctrl-E ENQ	48	30	060	0	91	5B	133	[
6	06	006	Ctrl-F ACK	49	31	061	1	92	5C	134	₩
7	07	007	Ctrl-G BEL	50	32	062	2	93	5D	135	]
8	08	010	Ctrl-H BS	51	33	063	3	94	5E	136	^
9	09	011	Ctrl-I HT	52	34	064	4	95	5F	137	_
10	0A	012	Ctrl-J LF	53	35	065	5	96	60	140	`
11	0B	013	Ctrl-K VT	54	36	066	6	97	61	141	a
12	0C	014	Ctrl-L FF	55	37	067	7	98	62	142	b
13	0D	015	Ctrl-M CR	56	38	070	8	99	63	143	c
14	0E	016	Ctrl-N SO	57	39	071	9	100	64	144	d
15	0F	017	Ctrl-O SI	58	3A	072	:	101	65	145	e
16	10	020	Ctrl-P DLE	59	3B	073	;	102	66	146	f
17	11	021	Ctrl-Q DC1	60	3C	074	<	103	67	147	g
18	12	022	Ctrl-R DC2	61	3D	075	=	104	68	150	h



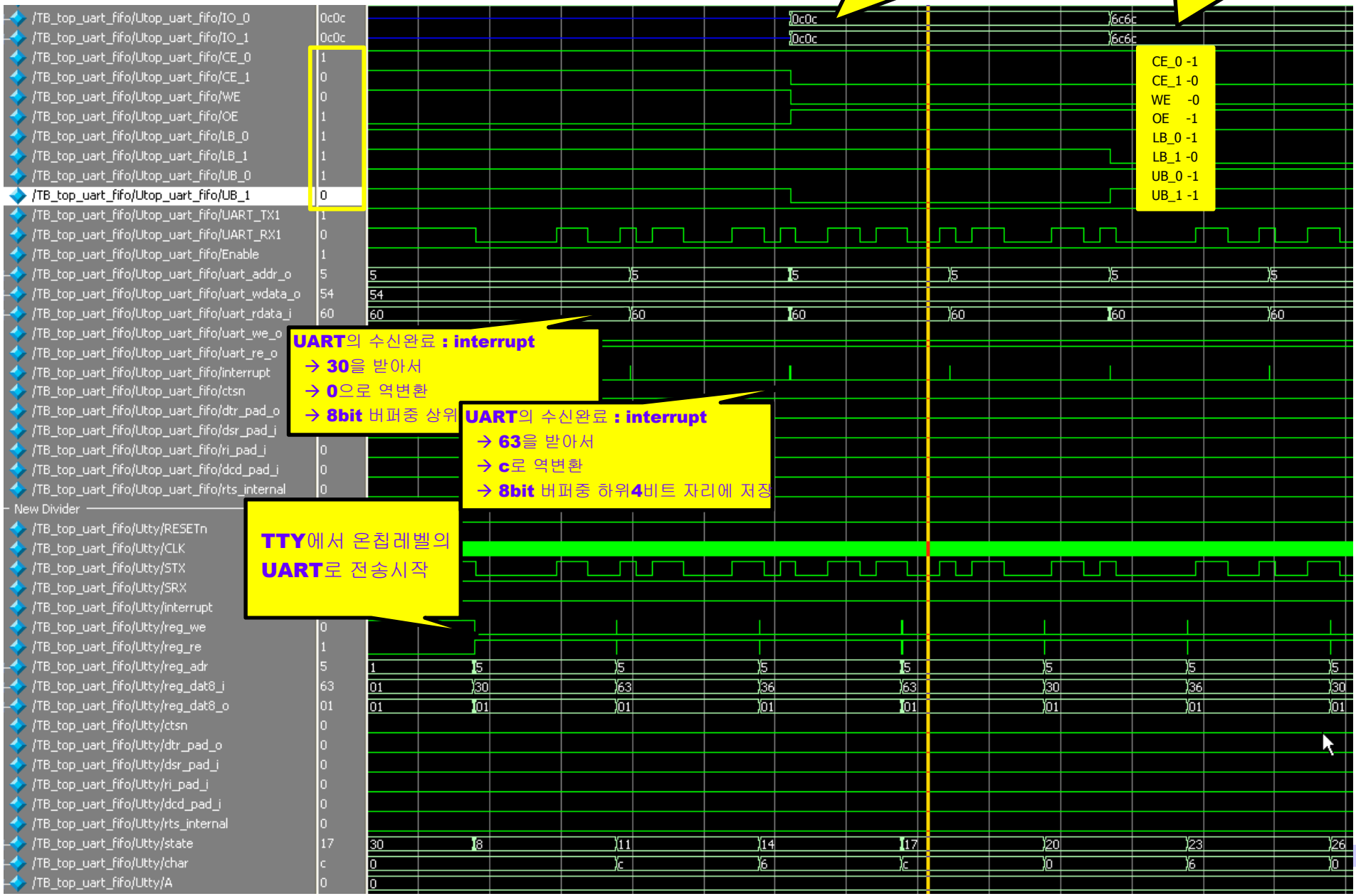


시뮬레이션

TTY 모듈의
SW 전송 중

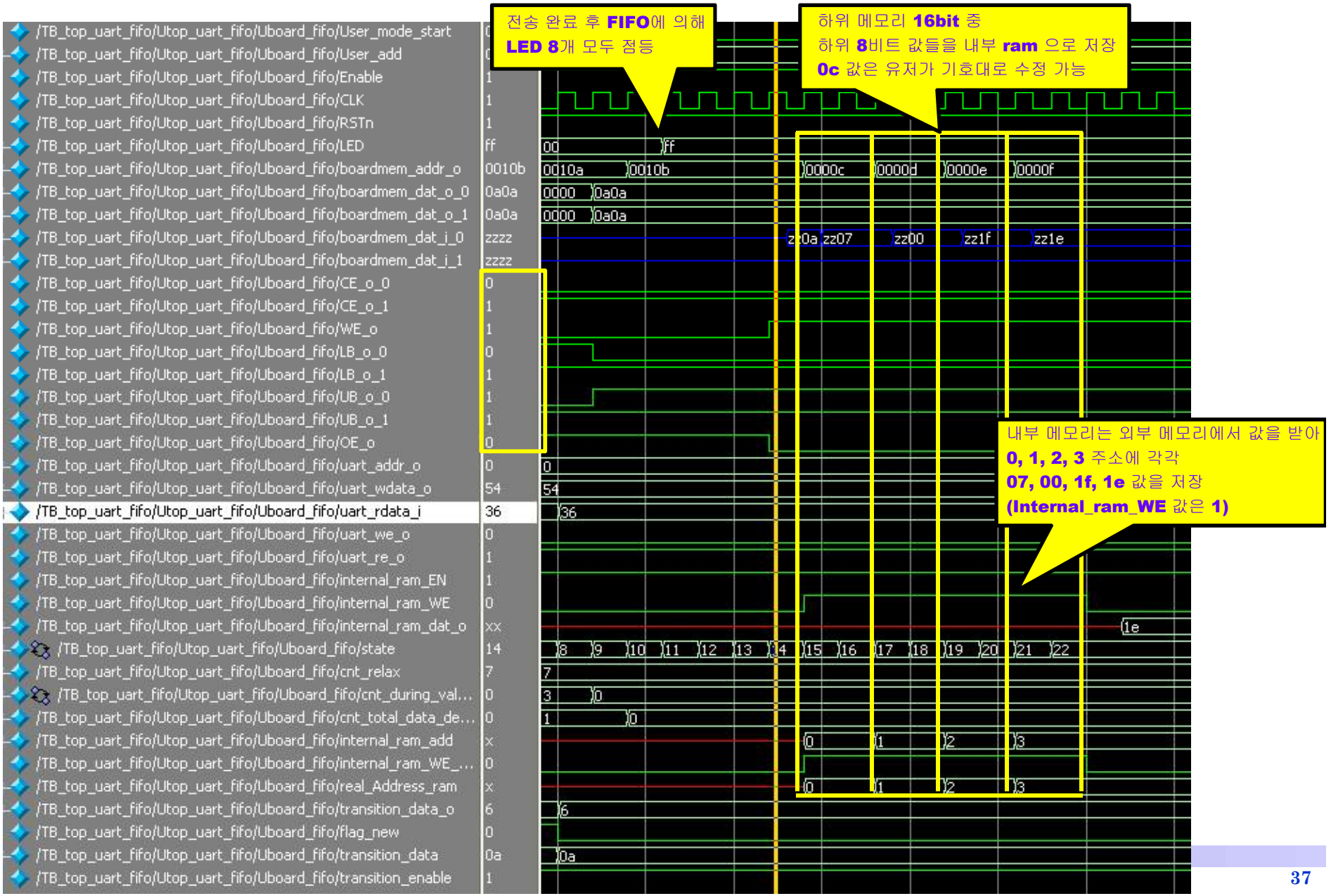
8비트 수신완료 후
상위 메모리(16bit) 에 저장

8비트 수신완료 후
하위 메모리(16bit)에 저장



시뮬레이션

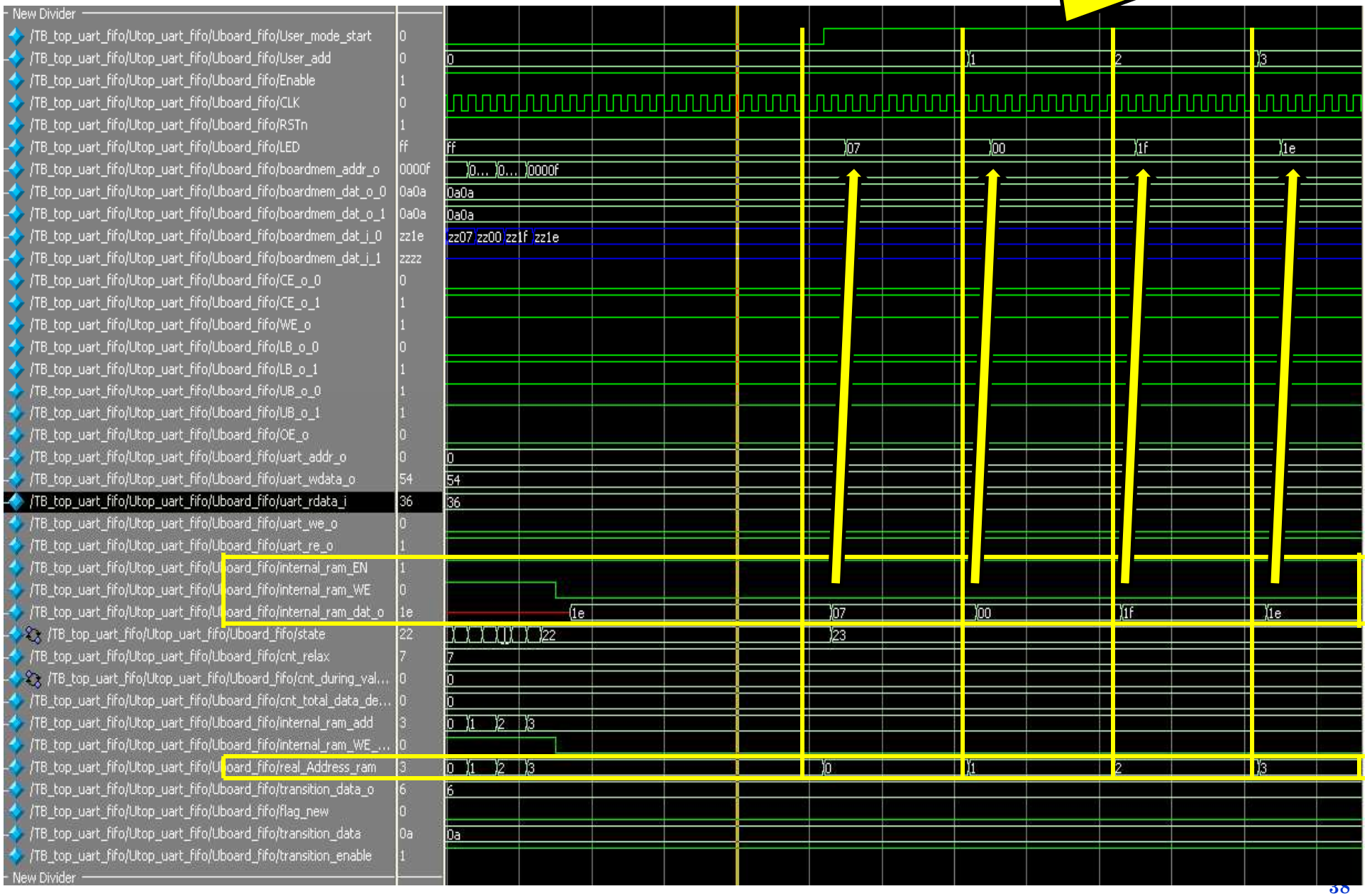
SW 전송 완료 후
디버깅 모드 준비 –내부 ram 에 값 받아오기



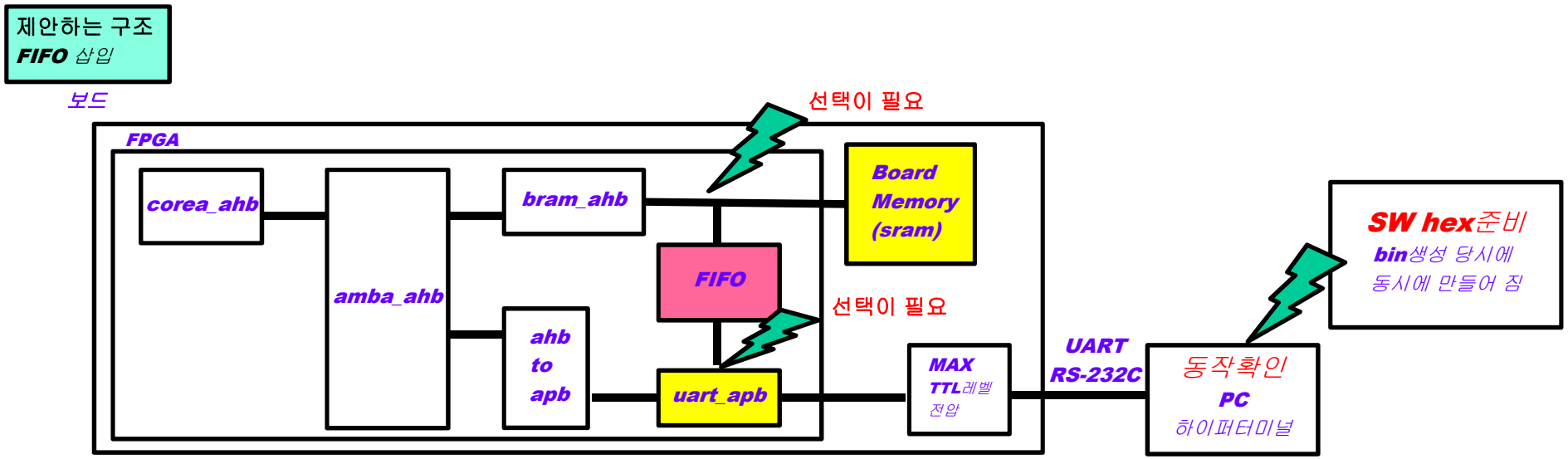
시뮬레이션

디버깅 모드 – 내부 ram 에 있는 값 LED 로 표시

User_mode_start 값이 **1**이 되면
User_add 값이 내부 메모리로 전달됨
→ 저장된 **SW**값을 **LED** 로 확인가능

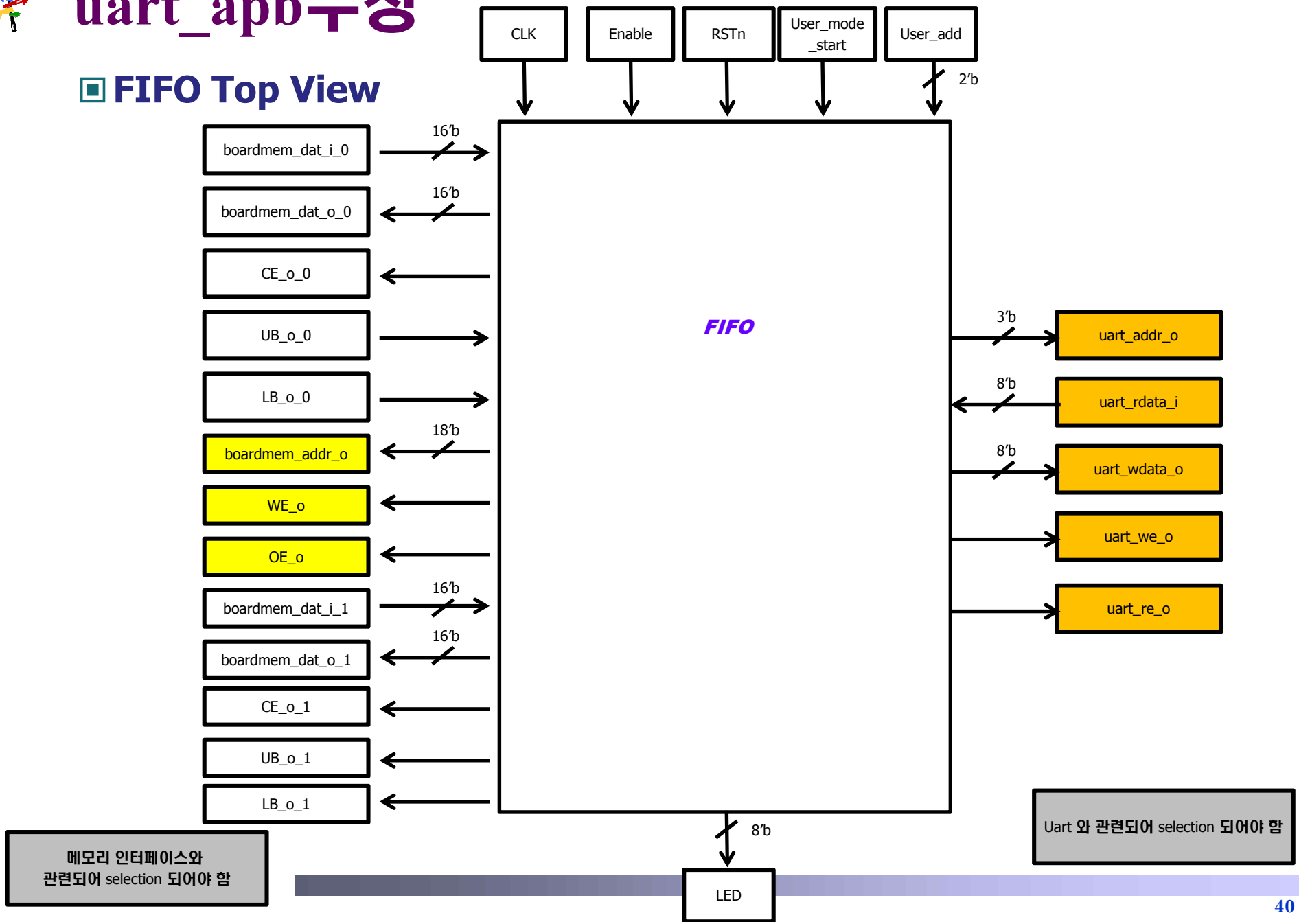


전체 total 시스템


제안하는 구조
FIFO 삽입
보드
선택이 필요
FPGA
corea_ahb
amba_ahb
bram_ahb
Board
Memory
(sram)
FIFO
선택이 필요
ahb
to
apb
uart_apb
MAX
TTL레벨
전압
UART
RS-232C
동작확인
PC
하이퍼터미널
SW hex준비
bin생성 당시에
동시에 만들어 짐

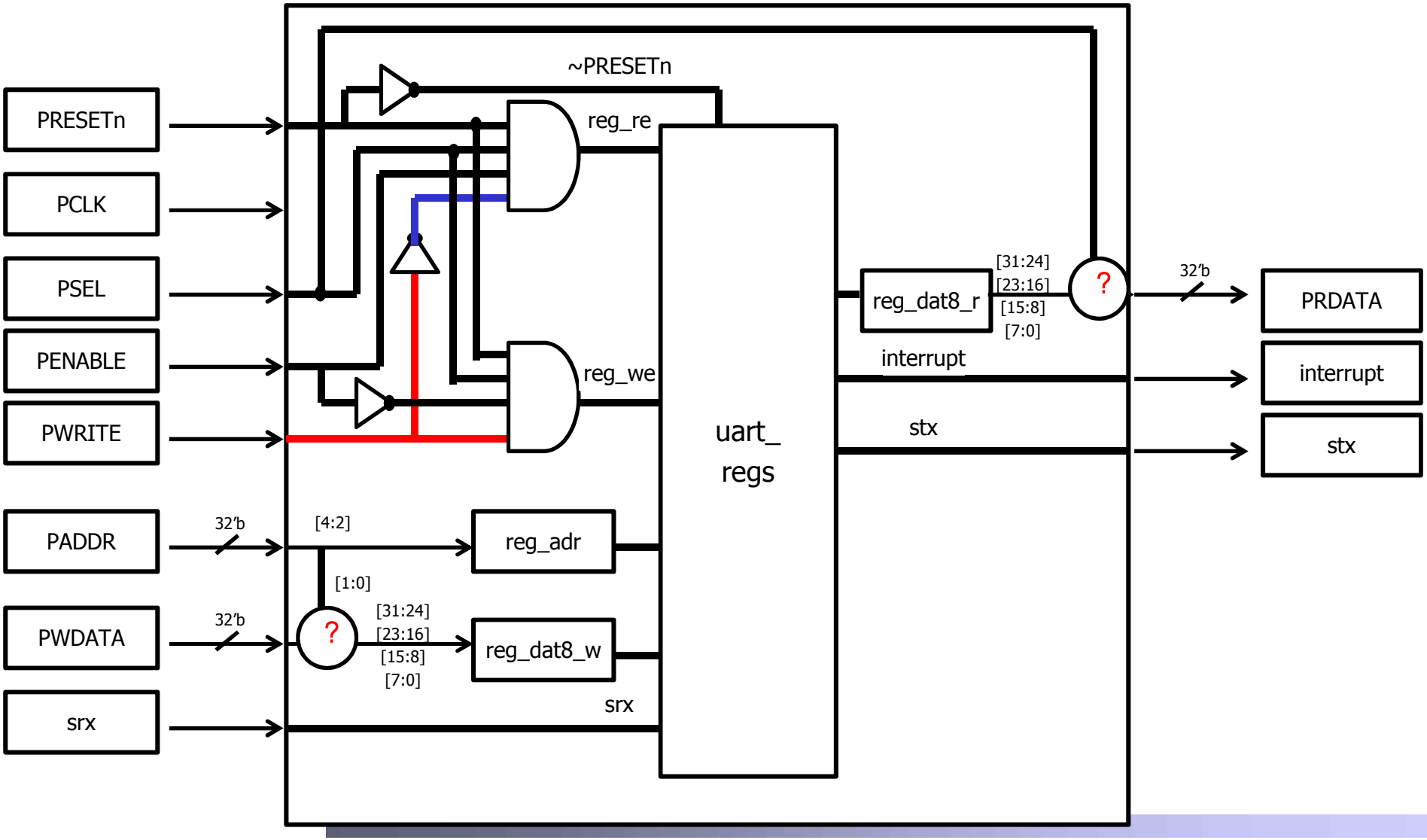
uart_apb수정

FIFO Top View



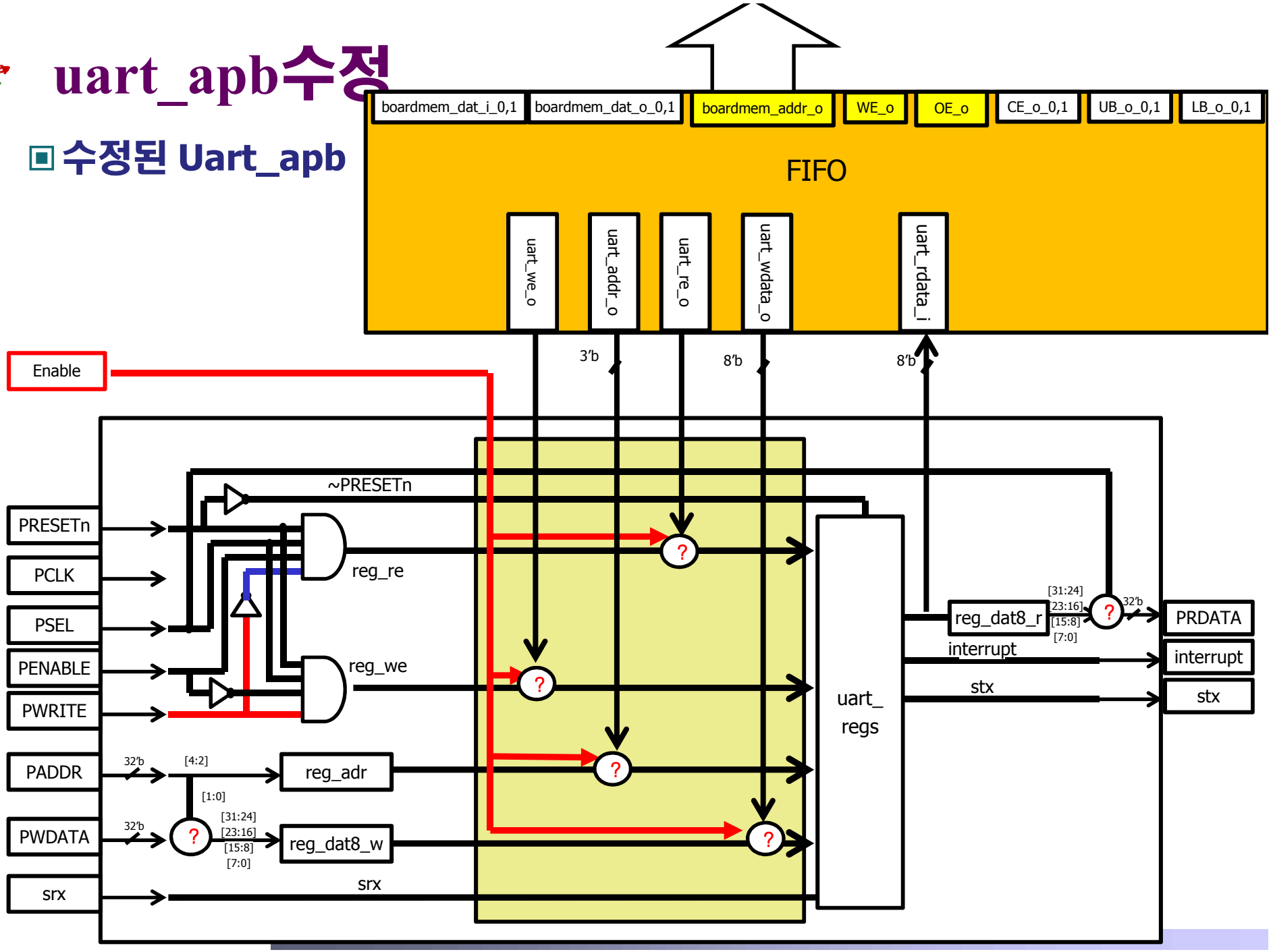
uart_apb수정

▣ 기존 Uart_apb



uart_apb수정

▣ 수정된 Uart_apb



수정된 uart_apb.v

```
module uart_apb (
       Enable,
      PRESETn,

      PCLK,
      PSEL,
      PENABLE,
      PADDR,
      PWRITE,
      PWDATA,
      PRDATA,

      interrupt, // interrupt request (active-high)
      stx,    // serial output
      srx,    // serial input

      uart_addr_o,
      uart_wdata_o, // write to reg
      uart_rdata_i, // read from reg
      uart_we_o, // Write enable for registers
      uart_re_o // Read enable for registers
);
   input         Enable;     wire        Enable;
   input         PRESETn;    wire        PRESETn;
   input         PCLK;       wire        PCLK;
   input         PSEL;       wire        PSEL;
   input         PENABLE;    wire        PENABLE;
   input  [31:0] PADDR;      wire [31:0] PADDR;
   input         PWRITE;     wire        PWRITE;
   output [31:0] PRDATA;     wire [31:0] PRDATA;
   input  [31:0] PWDATA;     wire [31:0] PWDATA;
   output        interrupt;  wire        interrupt;
   input         srx;        wire        srx;
   output        stx;        wire        stx;

input wire  [2:0] uart_addr_o;
input wire [7:0] uart_wdata_o; // write to reg
output wire [7:0] uart_rdata_i; // read from reg
input wire uart_we_o; // Write enable for registers
input wire uart_re_o; // Read enable for registers
```

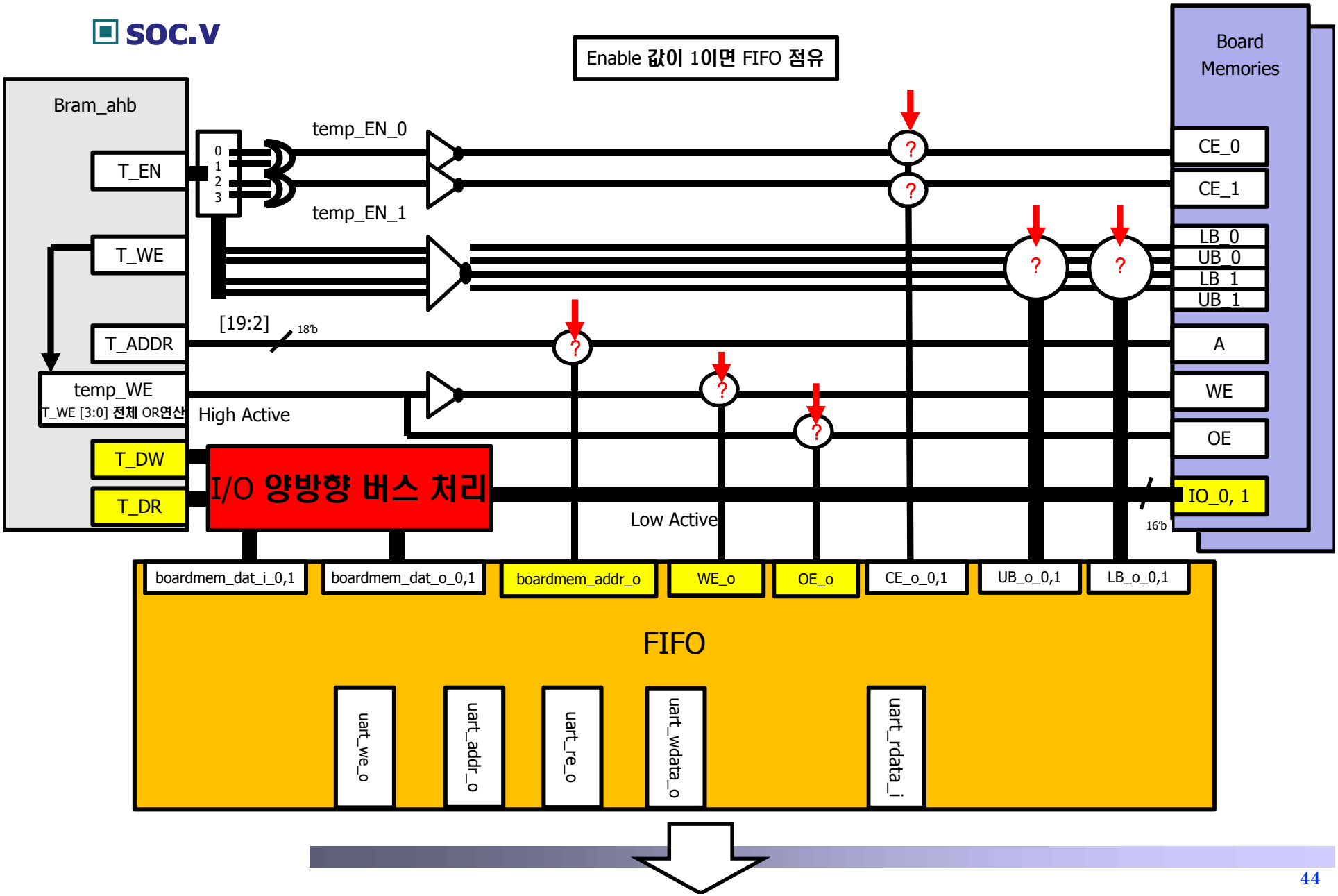
From FIFO

```
      wire [2:0] reg_adr_temp;
      wire [7:0] reg_dat8_w_temp;
      wire [7:0] reg_dat8_r_temp;
      wire reg_we_temp;
      wire reg_re_temp;
      wire real_clk;

      assign reg_adr_temp = Enable? uart_addr_o : reg_adr;
      assign reg_dat8_w_temp = Enable? uart_wdata_o : reg_dat8_w;

      assign uart_rdata_i = reg_dat8_r_temp;
      assign reg_dat8_r = reg_dat8_r_temp;

      assign reg_we_temp = Enable? uart_we_o : reg_we;
      assign reg_re_temp = Enable? uart_re_o : reg_re;

  uart_regs Uregs(
         .clk          (PCLK),
         .wb_rst_i     (~PRESETn),
         .wb_addr_i    (reg_adr_temp),
         .wb_dat_i     (reg_dat8_w_temp),
         .wb_dat_o     (reg_dat8_r_temp),
         .wb_we_i      (reg_we_temp),
         .wb_re_i      (reg_re_temp),
         .modem_inputs({~ctsn, dsr_pad_i, ri_pad_i,  dcd_pad_i}),
         .stx_pad_o    (stx),
         .srx_pad_i    (srx),
         .rts_pad_o    (rts_internal),
         .dtr_pad_o    (dtr_pad_o),
         .int_o        (interrupt)
  );
endmodule
```

input 라인끼리 모여 있으므로
RTL 코딩이 간편

Sram_if 수정

SOC.V



Sram_if 수정

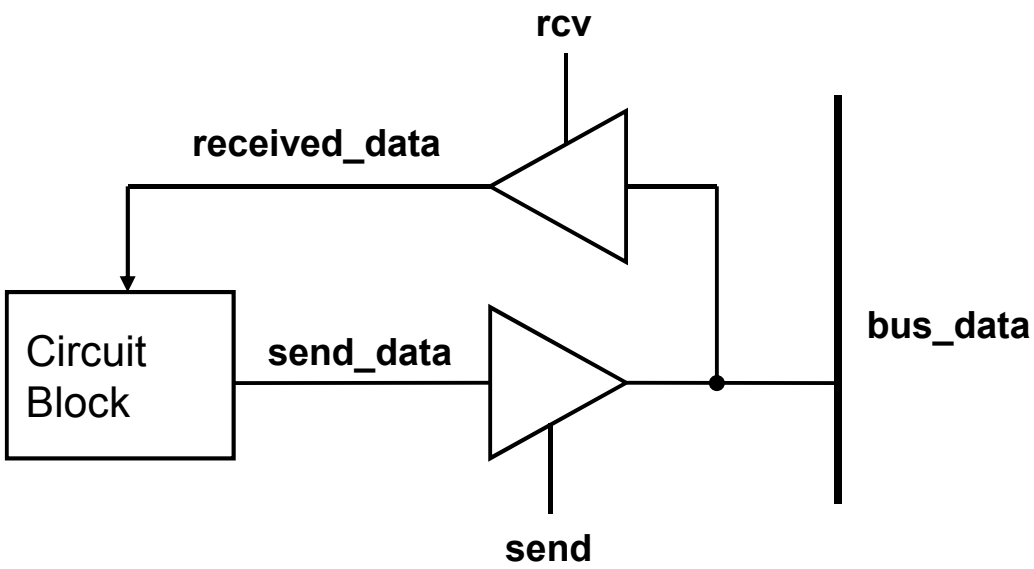
SOC.v

```
wire temp_EN_0 = T_EN[1] | T_EN[0];
wire temp_EN_1 = T_EN[3] | T_EN[2];
wire temp_WE;

assign A = Enable? A_temp : T_ADDR[19:2];

assign CE_0 = Enable? CE_0_temp : ~temp_EN_0;
assign CE_1 = Enable? CE_1_temp : ~temp_EN_1;

assign WE =  Enable? WE_temp : ~temp_WE;
assign OE =  Enable? OE_temp : temp_WE;

assign LB_0 =  Enable? LB_0_temp : ~T_EN[0];
assign UB_0 =  Enable? UB_0_temp: ~T_EN[1];

assign LB_1 = Enable? LB_1_temp: ~T_EN[2];
assign UB_1 = Enable? UB_1_temp: ~T_EN[3];
```

Sram_if 수정

▣ 참고) 양방향 버스 드라이버

- ◆ 보내기와 받기를 동시에 처리



Sram_if 수정

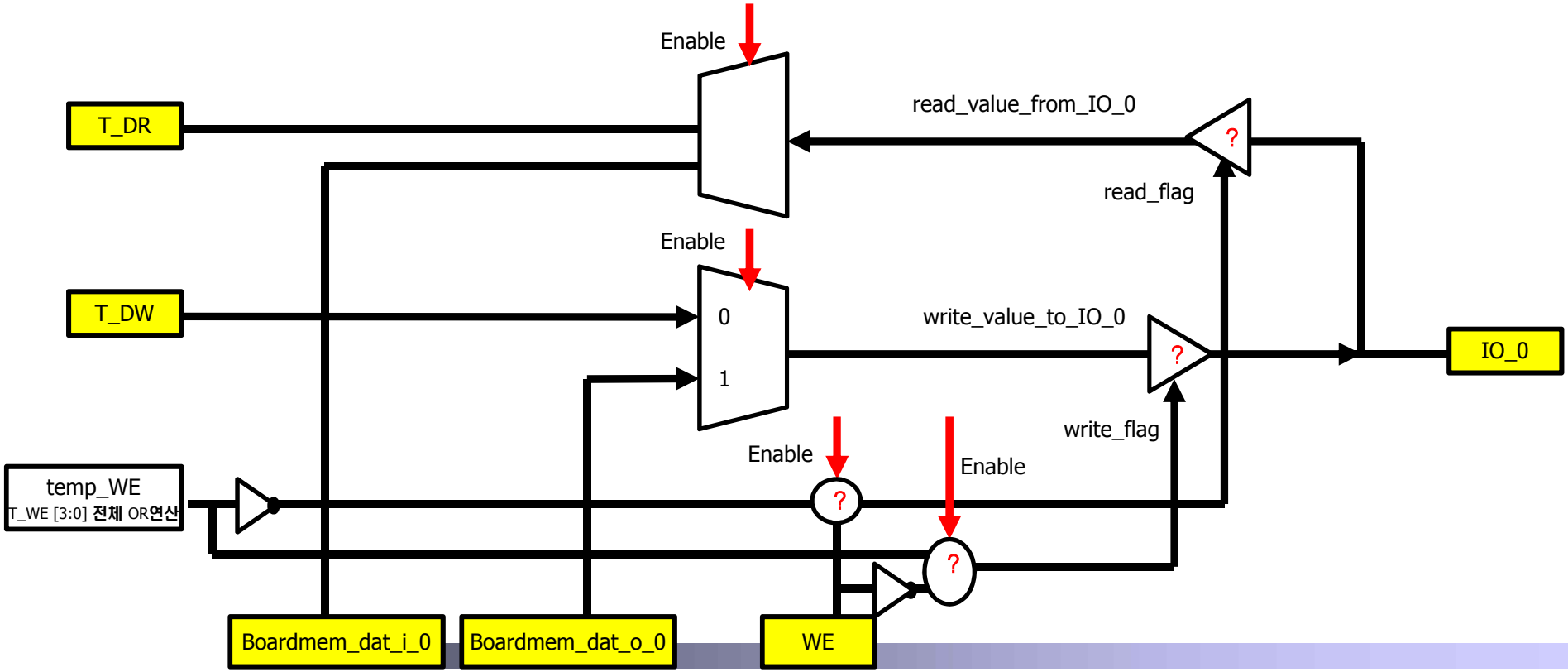
▣ 참고) 양방향 버스 드라이버

- ◆ 보내기와 받기를 동시에 처리

```
module bidir_bus(send, send_data, rcv, received_data, bus_data);
    input         send, rcv;
    input  [7:0] send_data;
    inout  [7:0] bus_data;
    output [7:0] received_data;

   assign received_data =(rcv) ? bus_data : 8'bz;
   assign bus_data =(send) ? send_data : bus_data;

endmodule
```

Sram_if 수정

SOC.V

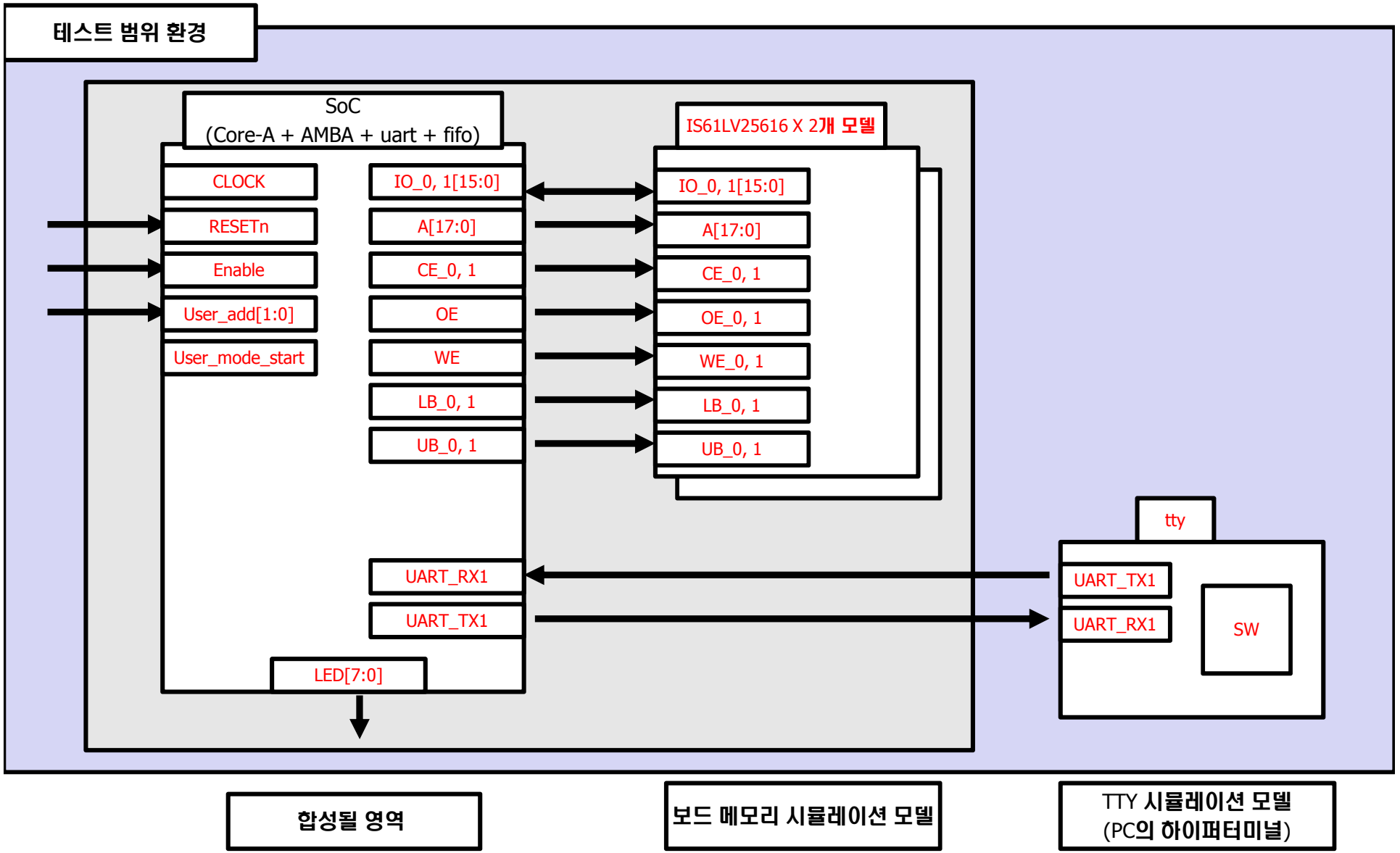


Sram_if 수정

SOC.V

```
assign read_flag = Enable? WE_temp:~temp_WE;  // '1' value means really read
assign write_flag = Enable? ~WE_temp : temp_WE ;  // '1' value means really write

assign read_value_from_IO_0 =  read_flag? IO_0 : 16'bz;
assign boardmem_dat_i_0 = Enable? read_value_from_IO_0 : 16'bz;
assign T_DR[15:0]             = Enable? 16'bz : read_value_from_IO_0 ;

assign read_value_from_IO_1 =  read_flag? IO_1 : 16'bz;
assign boardmem_dat_i_1 = Enable? read_value_from_IO_1 : 16'bz;
assign T_DR[31:16]       = Enable? 16'bz : read_value_from_IO_1 ;

assign IO_0 = write_flag? write_value_to_IO_0 : 16'bz;
assign write_value_to_IO_0 = Enable? boardmem_dat_o_0 : T_DW[15:0];

assign IO_1 = write_flag? write_value_to_IO_1 : 16'bz;
assign write_value_to_IO_1 = Enable? boardmem_dat_o_1 : T_DW[31:16];
```

테스트 벤치 환경


테스트 범위 환경
SoC
(Core-A + AMBA + uart + fifo)
CLOCK
RESETn
Enable
User_add[1:0]
User_mode_start
IO_0, 1[15:0]
A[17:0]
CE_0, 1
OE
WE
LB_0, 1
UB_0, 1
UART_RX1
UART_TX1
LED[7:0]
IS61LV25616 X 2개 모델
IO_0, 1[15:0]
A[17:0]
CE_0, 1
OE_0, 1
WE_0, 1
LB_0, 1
UB_0, 1
tty
UART_TX1
UART_RX1
SW
합성될 영역
보드 메모리 시뮬레이션 모델
TTY 시뮬레이션 모델
(PC의 하이퍼터미널)

테스트 벤치 환경

▣ 테스트 시나리오

동작 스피드 : 50MHz

```
soc Usoc(
        .User_mode_start(User_mode_start),
        .User_add(User_add),
        .Enable(Enable),
        .LED(LED),

        .A(A),
        .IO_0(IO_0),
        .IO_1(IO_1),

        .CE_0(CE_0),
        .CE_1(CE_1),

        .WE(WE),
        .OE(OE),

        .LB_0(LB_0),
        .LB_1(LB_1),

        .UB_0(UB_0),
        .UB_1(UB_1),
        .RESETn(RESETn),        // active-
        .CLOCK(CLOCK),         // system c
        .UART_TX1(UART_TX1),
        .UART_RX1(UART_RX1)
);
```

```
IS61LV25616 UIS61LV25616_0(
        .A(A),
        .IO(IO_0),
        .CE_(CE_0),
        .OE_(OE),
        .WE_(WE),
        .LB_(LB_0),
        .UB_(UB_0)
);


IS61LV25616 UIS61LV25616_1(
        .A(A),
        .IO(IO_1),
        .CE_(CE_1),
        .OE_(OE),
        .WE_(WE),
        .LB_(LB_1),
        .UB_(UB_1)
);


tty Utty(
        .RESETn(RESETn),
        .CLK(PCLK),
        .STX(UART_RX1),
        //.SRX(UART_TX1)
        .SRX(),
                .Enable(Enable)
);
```

```
initial begin
        CLOCK = 1'b0;
        RESETn = 1'b1;
        Enable = 1'b1;
        User_mode_start = 1'b0;
        User_add = 2'b00;

        #110 RESETn = 1'b0;
        #40 RESETn = 1'b1;

        #200000000 User_mode_start = 1'b1;
        #150 User_add = 2'b00;
        #300 User_add = 2'b01;
        #300 User_add = 2'b10;
        #300 User_add = 2'b11;

        #500 RESETn = 1'b0;
        #500 Enable = 1'b0;
        #150 RESETn = 1'b1;

end

always begin
        #10 CLOCK = ~CLOCK;
end
```

리셋 / FIFO, tty Enable

리셋 해제 / sw 덤핑

외부 메모리로부터 몇 개 코드 read **하여** Fifo **내부의** reg_sram **으로** Write **및** LED **로 값 확인**

Fifo, tty **동작** Disable

리셋 후 UART **로 메시지 확인**

시뮬레이션

RESET 후
FIFO 의 sw 덤핑
to Board mem

uart.hex = (C:₩CoreIDEA₩example...

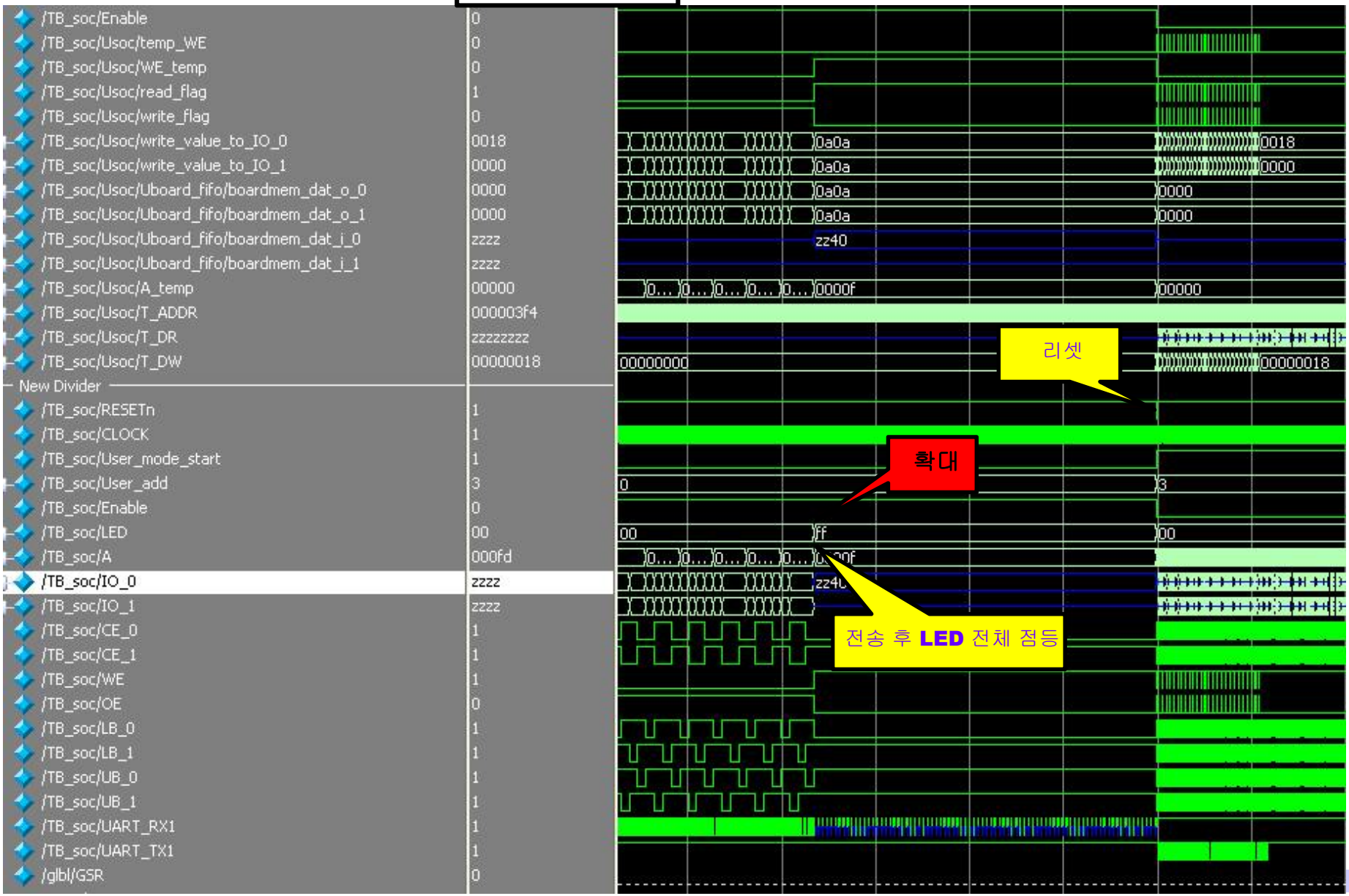
파일(F) 편집(E) 도구(T) 문법(S) 버퍼(B)

```
0c6c0601 36600004 80000000 80000000
366ffffe 80000000 246bc020 10ccfff4
612c8c00 10abfffc 10ccfffc 0c600907
10000200 0c61271f 00112b1e 10110040
```

write_flag 값 1, Enable 값 1
→ write value 는 fifo 에서 온 값

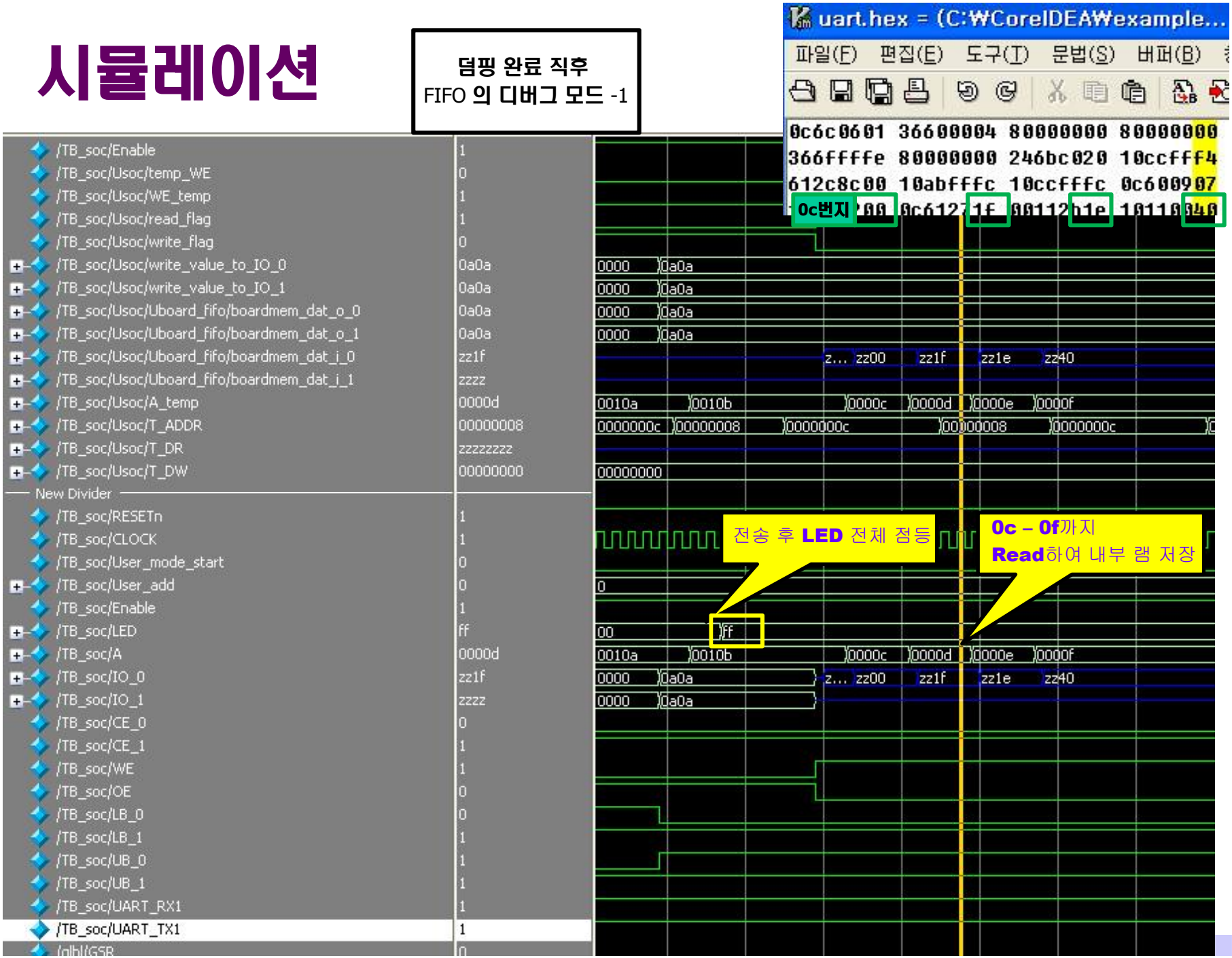
시뮬레이션

덤핑 완료 후
LED 점등 확인 및 리셋



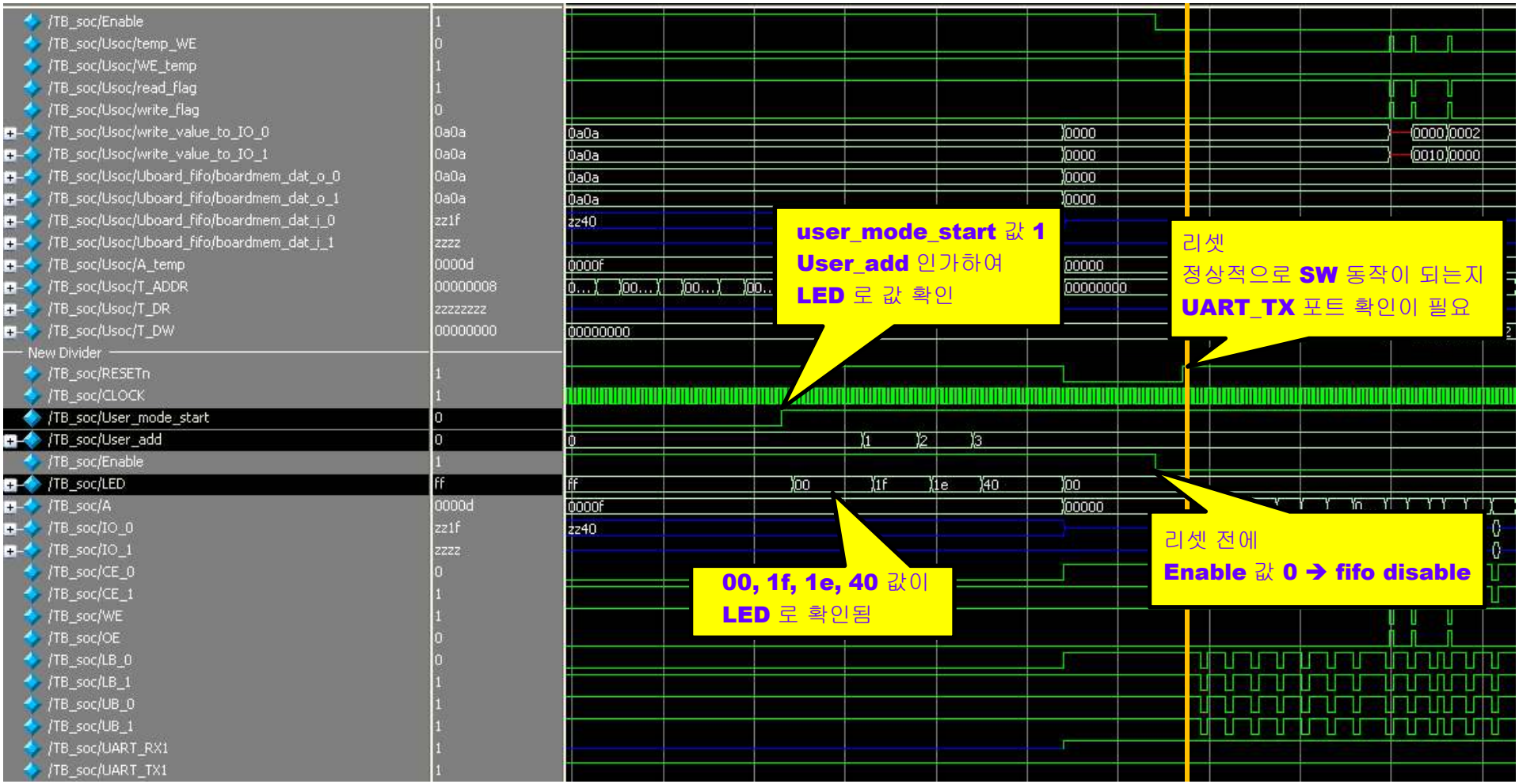
시뮬레이션

덤핑 완료 직후
FIFO 의 디버그 모드 -1

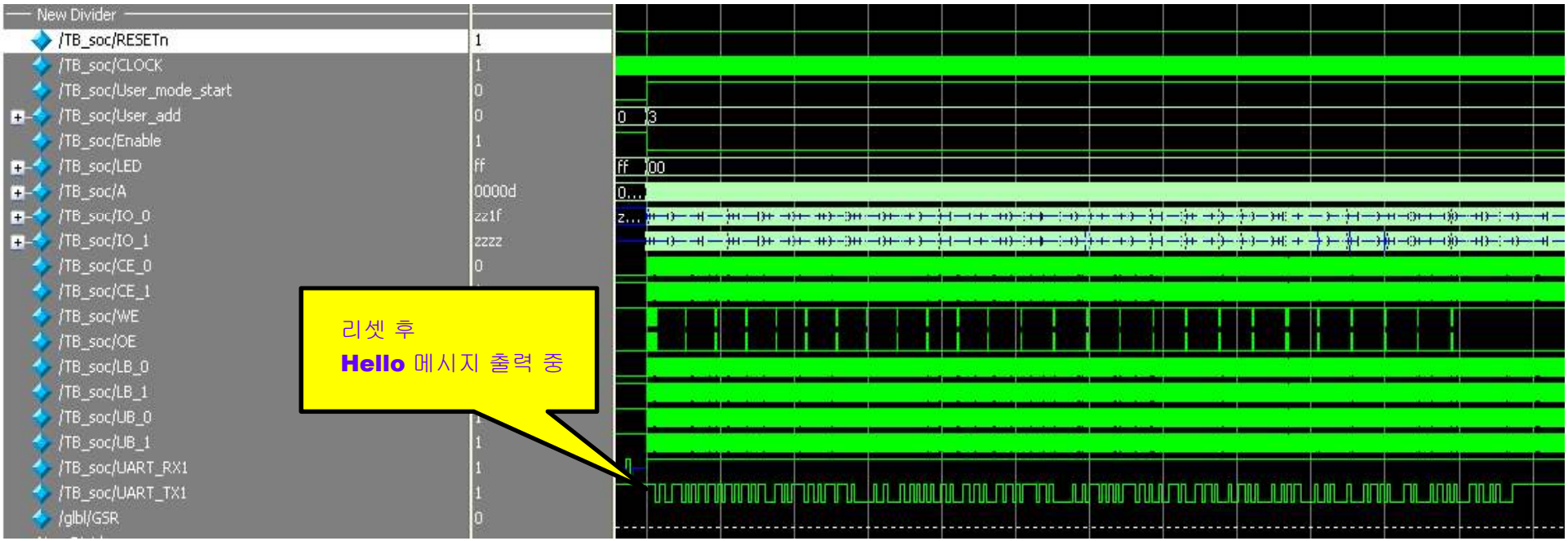


시뮬레이션

덤핑 완료 직후
FIFO **의 디버그 모드** -2



시뮬레이션


New Divider
/TB_soc/RESETn
/TB_soc/CLOCK
/TB_soc/User_mode_start
/TB_soc/User_add
/TB_soc/Enable
/TB_soc/LED
/TB_soc/A
/TB_soc/IO_0
/TB_soc/IO_1
/TB_soc/CE_0
/TB_soc/CE_1
/TB_soc/WE
/TB_soc/OE
/TB_soc/LB_0
/TB_soc/LB_1
/TB_soc/UB_0
/TB_soc/UB_1
/TB_soc/UART_RX1
/TB_soc/UART_TX1
/glbl/GSR
1
1
0
0
1
ff
0000d
zz1f
zzzz
0
1
1
1
0
0 3
ff 00
0...
z...
리셋 후
Hello 메시지 출력 중

데모를 위한 보드 스위치

Slide switches

Ucf 파일

```
##################################################
## User switches
##################################################
NET "RESETn" LOC="F12";
NET "RESETn" IOSTANDARD=LVCMOS33;
NET "Enable" LOC="K13";
NET "Enable" IOSTANDARD=LVCMOS33;


NET "User_mode_start" LOC="J14";
NET "User_mode_start" IOSTANDARD=LVCMOS33;


NET "User_add[0]" LOC="G12";
NET "User_add[0]" IOSTANDARD=LVCMOS33;

NET "User_add[1]" LOC="H14";
NET "User_add[1]" IOSTANDARD=LVCMOS33;
```

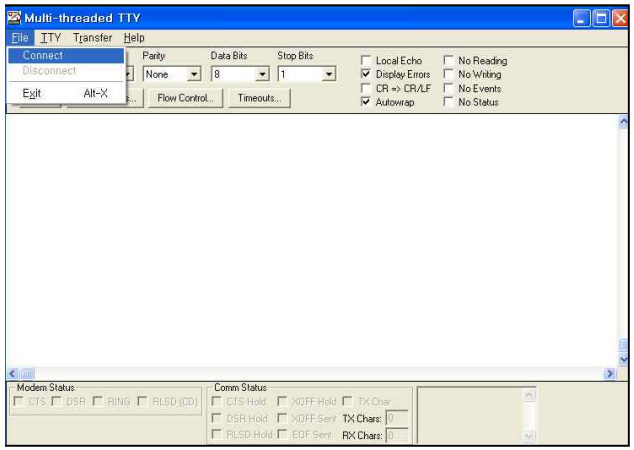
FIFO Enable | User Mode start | User Add[1] | User Add[0] | 리셋

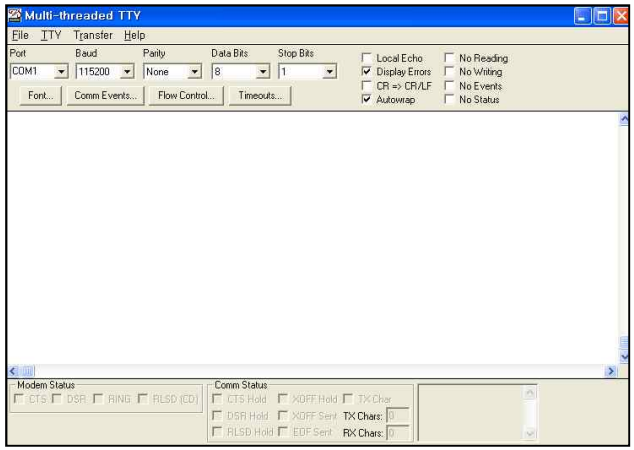
Switch	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
FPGA Pin	K13	K14	J13	J14	H13	H14	G12	F12

버튼 조작 시나리오

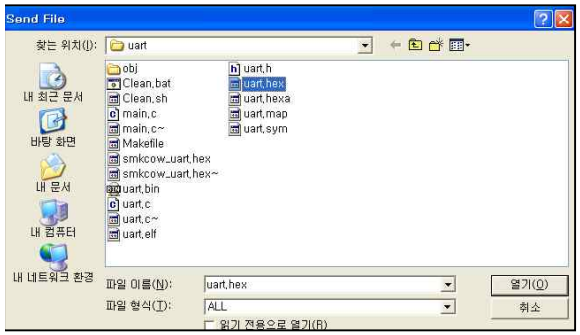
	리셋 / 초기값	FIFO Enable	리셋해제	파일전송 및 디버그 모드	FIFO Disable	리셋	리셋해제 정상동작
리셋스위치	0	0	**1**	User mode Start 버튼 1 및 user add 조작	1	**0**	**1**
FIFO Enable 스위치	0	**1**	1		0	0	0

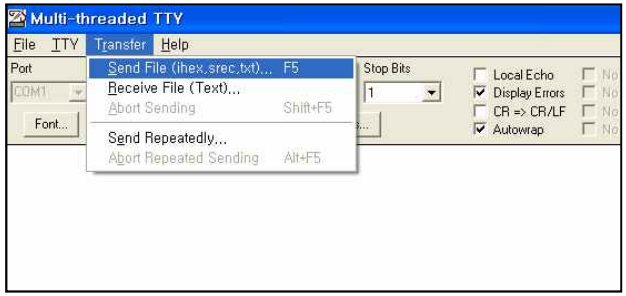
보드 레벨 검증

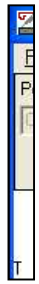




Enable **스위치** on
리셋 해제 후 T **문자 확인**
➔ FIFO **초기화 후** sw **전송 기다림**

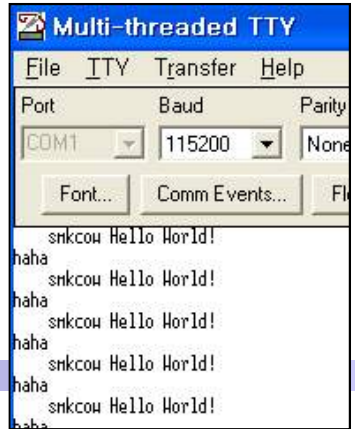






보드 LED 8**개 모두 점등 확인**
User mode start **버튼** on
User add 2**비트 조작으로** LED **로 값 확인**

Enable **스위치** off
다시 한번 리셋 후 터미널에 출력되는
문자 확인



출처

- 다이나릿, 'Core_A_BasicPlatform',Dynalith Systems, 2009.1
- Xilinx, Spartan-3 Starter Kit Board User Guide, 2005.5
- IS61LV25616AL, ISSI, 2011.12
- OPEN CORES, UART IP Core Specification, 2002.8