SoC Report

-Register_Map-

1)Data_Pasing

2)Register_Map

3)load_tx_data

과목 : SOC수업

과제명 : Register Map

담당교수 : 박종성

학과 : 메카트로닉스공학과

학번 : 2015132039

이름 : 최영운

제출일 : 11월 18일

목차

1.주제 배경 이론

1)Register 란?

Register =1.저항기 2.하드웨어 레지스터 3.프로세서 레지스터 4.시프트 레지스터 5.상태 레지스터 6.더 레지스터로 분류된다.

흔히 말하는 Register는 저항기로써의 레지스터를 의미한다. 즉 회로에서 전류의 흐름을 억제하는 역할을 의미한다. 또한 공학도로써의 Register은 산술/연산적 연산이나 정보해석, 전송 등 을 할 수 있는 일정 길이의 정보를 저장하는 CPU 내부의 기억장치를 의미하는 하드웨어/프로세서 Register을 떠올린다. 하지만 SOC의 Register은 설계된 칩(SoC) 기능의 활성화 및 동작 조건 선택을 위한 제어 값을 저장하고 외부에서 내부 메모리로 접근이 가능한 프로토콜을 내장하는 역할을 뜻한다.

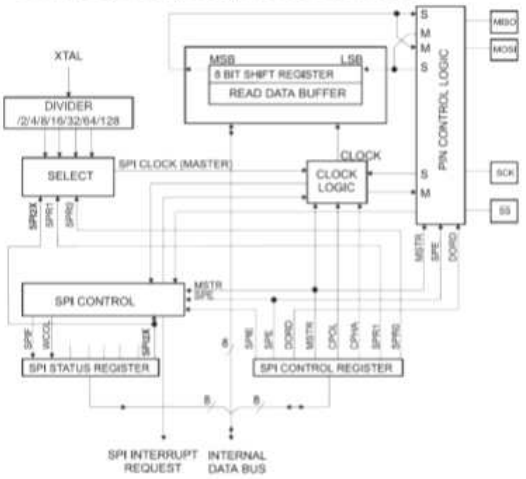
2)Register Map의 장점과 사례

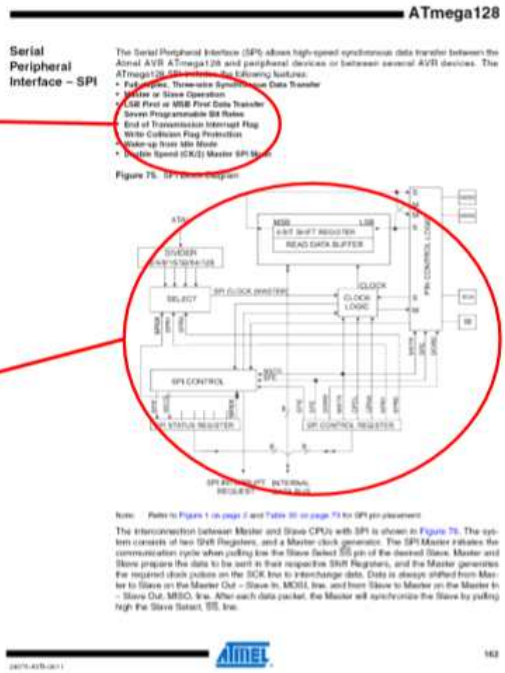
장점 : 간단한 프로세서에서는 Register Map을 사용하지 않고 직접 주소를 설정해서 쓴다. 이럴 때 필요한 Register를 하나하나 설정하여 저장하게 되는데 모듈이 늘어날수록 복잡해지며, 찾기 어려워진다. 또한 Register의 공간이 가득차게 되면 Register공간을 늘려야 하는데 그럴 때 기존 저장한 Register 하나하나 바뀐 Register호출 방식을 다 적용시켜야 하는 어려움이 있다. 이를 해결하기 위하여 Register_Map이라는 주소설정방식을 사용하게 된다. Register_Map은 여유있게 주소설정 공간을 확보해 놓은 다음 모듈하나를 호출할 때 다음 공간에 설정하도록 하는 방식이다. 특징으로는 1. 모듈의 주소를 알면 바로 호출이 가능하며, 2. 모듈을 추가하거나 제거하는 것에 용이하다는 점이다.

즉 모듈의 기능, 옵션을 설정하기위한 하나의 프로토콜로써 모듈 제어가 필요시 사용할 수 있는 한가지 방법(주소설정방식)이라는 것이다. 따라서 이것이 정답은 아니다.

ATmega128 SPI

- Full-duplex, Three-wire Synchronous Data Transfer
- Master or Slave Operation
- LSB First or MSB First Data Transfer
- Seven Programmable Bit Rates
- End of Transmission Interrupt Flag
- Write Collision Flag Protection
- Wake-up from Idle Mode
- Double Speed (CK/2) Master SPI Mode



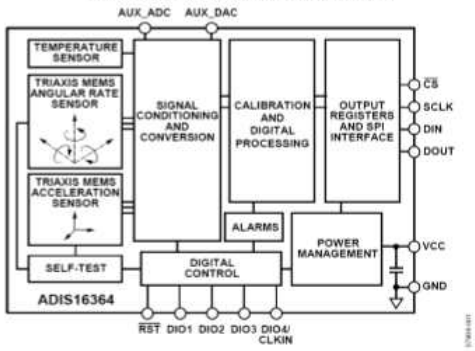


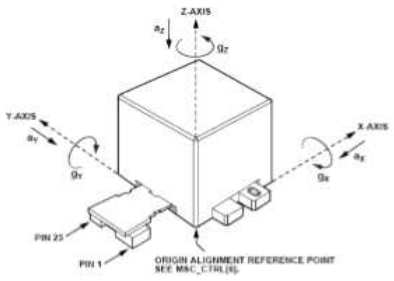
ADIS16364 IMU Sensor

FEATURES

- Triaxis digital gyroscope with digital range scaling
  - ±75°/sec, ±150°/sec, ±300°/sec settings
  - Tight orthogonal alignment: <0.05°
- Triaxis digital accelerometer: ±5 *g*
- Autonomous operation and data collection
  - No external configuration commands required
  - Start-up time: 180 ms
  - Sleep mode recovery time: 4 ms
- Factory-calibrated sensitivity, bias, and axial alignment
- Calibration temperature range: −20°C to +70°C
- SPI-compatible serial interface
- Wide bandwidth: 330 Hz
- Embedded temperature sensor
- Programmable operation and control
  - Automatic and manual bias correction controls
  - Bartlett window, FIR filter length, number of taps
  - Digital I/O: data ready, alarm indicator, general-purpose
  - Alarms for condition monitoring
  - Sleep mode for power management
  - DAC output voltage
  - Enable external sample clock input: up to 1.2 kHz
  - Single-command self-test
- Single-supply operation: 4.75 V to 5.25 V
- 2000 *g* shock survivability
- Operating temperature range: −40°C to +105°C

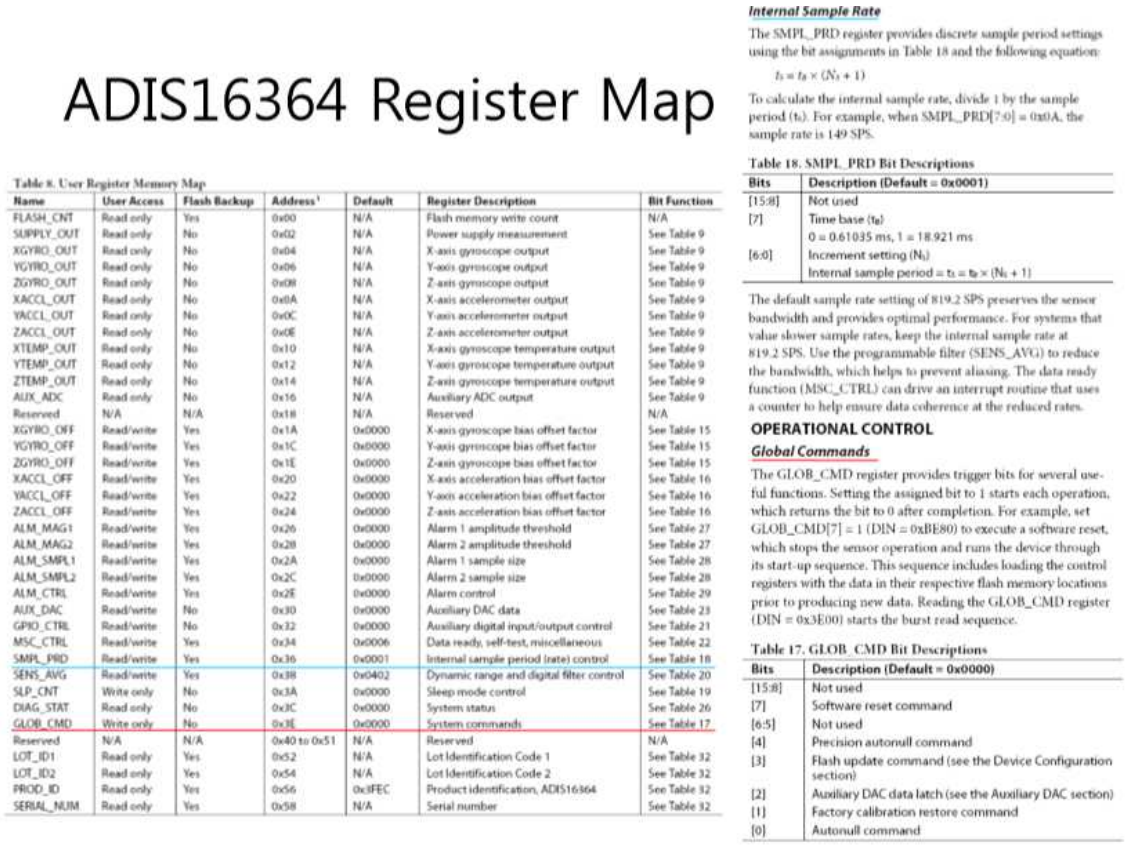
FUNCTIONAL BLOCK DIAGRAM





ATmega128 MCU에 있는 통신 모듈인 SPI와 가속도, 자이로 센서 모듈인 IMU sensor이다.

MCU에서 이 두 모듈을 설정한 주소이다



ADIS16364 Register Map

Table 8. User Register Memory Map

Name	User Access	Flash Backup	Address[1]	Default	Register Description	Bit Function
FLASH_CNT	Read only	Yes	0x00	N/A	Flash memory write count	N/A
SUPPLY_OUT	Read only	No	0x02	N/A	Power supply measurement	See Table 9
XGYRO_OUT	Read only	No	0x04	N/A	X-axis gyroscope output	See Table 9
YGYRO_OUT	Read only	No	0x06	N/A	Y-axis gyroscope output	See Table 9
ZGYRO_OUT	Read only	No	0x08	N/A	Z-axis gyroscope output	See Table 9
XACCL_OUT	Read only	No	0x0A	N/A	X-axis accelerometer output	See Table 9
YACCL_OUT	Read only	No	0x0C	N/A	Y-axis accelerometer output	See Table 9
ZACCL_OUT	Read only	No	0x0E	N/A	Z-axis accelerometer output	See Table 9
XTEMP_OUT	Read only	No	0x10	N/A	X-axis gyroscope temperature output	See Table 9
YTEMP_OUT	Read only	No	0x12	N/A	Y-axis gyroscope temperature output	See Table 9
ZTEMP_OUT	Read only	No	0x14	N/A	Z-axis gyroscope temperature output	See Table 9
AUX_ADC	Read only	No	0x16	N/A	Auxiliary ADC output	See Table 9
Reserved	N/A	N/A	0x18	N/A	Reserved	N/A
XGYRO_OFF	Read/write	Yes	0x1A	0x0000	X-axis gyroscope bias offset factor	See Table 15
YGYRO_OFF	Read/write	Yes	0x1C	0x0000	Y-axis gyroscope bias offset factor	See Table 15
ZGYRO_OFF	Read/write	Yes	0x1E	0x0000	Z-axis gyroscope bias offset factor	See Table 15
XACCL_OFF	Read/write	Yes	0x20	0x0000	X-axis acceleration bias offset factor	See Table 16
YACCL_OFF	Read/write	Yes	0x22	0x0000	Y-axis acceleration bias offset factor	See Table 16
ZACCL_OFF	Read/write	Yes	0x24	0x0000	Z-axis acceleration bias offset factor	See Table 16
ALM_MAG1	Read/write	Yes	0x26	0x0000	Alarm 1 amplitude threshold	See Table 27
ALM_MAG2	Read/write	Yes	0x28	0x0000	Alarm 2 amplitude threshold	See Table 27
ALM_SMPL1	Read/write	Yes	0x2A	0x0000	Alarm 1 sample size	See Table 28
ALM_SMPL2	Read/write	Yes	0x2C	0x0000	Alarm 2 sample size	See Table 28
ALM_CTRL	Read/write	Yes	0x2E	0x0000	Alarm control	See Table 29
AUX_DAC	Read/write	No	0x30	0x0000	Auxiliary DAC data	See Table 23
GPIO_CTRL	Read/write	No	0x32	0x0000	Auxiliary digital input/output control	See Table 21
MSC_CTRL	Read/write	Yes	0x34	0x0006	Data ready, self-test, miscellaneous	See Table 22
SMPL_PRD	Read/write	Yes	0x36	0x0001	Internal sample period (rate) control	See Table 18
SENS_AVG	Read/write	Yes	0x38	0x0402	Dynamic range and digital filter control	See Table 20
SLP_CNT	Write only	No	0x3A	0x0000	Sleep mode control	See Table 19
DIAG_STAT	Read only	No	0x3C	0x0000	System status	See Table 26
GLOB_CMD	Write only	No	0x3E	0x0000	System commands	See Table 17
Reserved	N/A	N/A	0x40 to 0x51	N/A	Reserved	N/A
LOT_ID1	Read only	Yes	0x52	N/A	Lot Identification Code 1	See Table 32
LOT_ID2	Read only	Yes	0x54	N/A	Lot Identification Code 2	See Table 32
PROD_ID	Read only	Yes	0x56	0x3FEC	Product identification, ADIS16364	See Table 32
SERIAL_NUM	Read only	Yes	0x58	N/A	Serial number	See Table 32

Internal Sample Rate

The SMPL_PRD register provides discrete sample period settings using the bit assignments in Table 18 and the following equation:

$$t_S = t_B \times (N_S + 1)$$

To calculate the internal sample rate, divide 1 by the sample period (t_S). For example, when SMPL_PRD[7:0] = 0x0A, the sample rate is 149 SPS.

Table 18. SMPL_PRD Bit Descriptions

Bits	Description (Default = 0x0001)
[15:8]	Not used
[7]	Time base (t_B) 0 = 0.61035 ms, 1 = 18.921 ms
[6:0]	Increment setting (N_S) Internal sample period = $t_S = t_B \times (N_S + 1)$

The default sample rate setting of 819.2 SPS preserves the sensor bandwidth and provides optimal performance. For systems that value slower sample rates, keep the internal sample rate at 819.2 SPS. Use the programmable filter (SENS_AVG) to reduce the bandwidth, which helps to prevent aliasing. The data ready function (MSC_CTRL) can drive an interrupt routine that uses a counter to help ensure data coherence at the reduced rates.

OPERATIONAL CONTROL

Global Commands

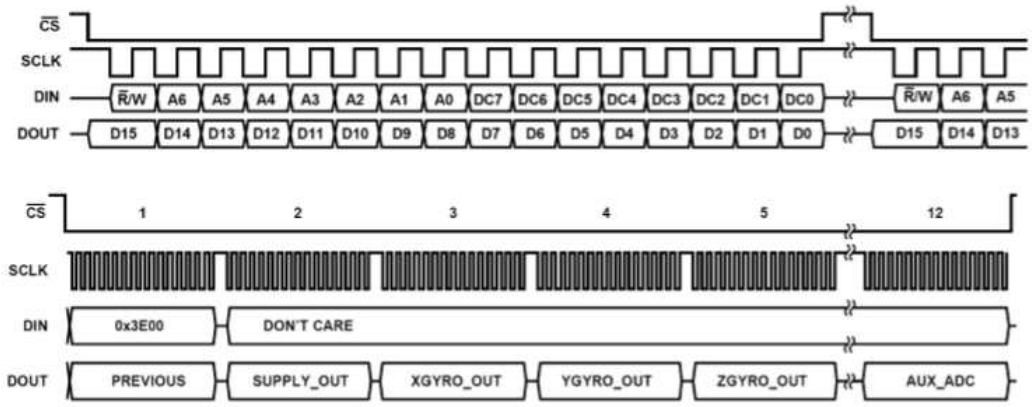
The GLOB_CMD register provides trigger bits for several useful functions. Setting the assigned bit to 1 starts each operation, which returns the bit to 0 after completion. For example, set GLOB_CMD[7] = 1 (DIN = 0xBE80) to execute a software reset, which stops the sensor operation and runs the device through its start-up sequence. This sequence includes loading the control registers with the data in their respective flash memory locations prior to producing new data. Reading the GLOB_CMD register (DIN = 0x3E00) starts the burst read sequence.

Table 17. GLOB_CMD Bit Descriptions

Bits	Description (Default = 0x0000)
[15:8]	Not used
[7]	Software reset command
[6:5]	Not used
[4]	Precision autonull command
[3]	Flash update command (see the Device Configuration section)
[2]	Auxiliary DAC data latch (see the Auxiliary DAC section)
[1]	Factory calibration restore command
[0]	Autonull command

위 Register_Map에서 각 모듈에 대한 address를 확인 할 수 있다. 해당 주소를 통해 해당 모듈을 컨트롤 하거나 데이터를 가져오는 것이다.

ADIS16364 Register Access Protocol



BURST READ DATA COLLECTION

Burst read data collection is a process-efficient method for collecting data from the ADIS16364. In a burst read, all output data registers are clocked out on DOUT, 16 bits at a time, in sequential data cycles (each separated by one SCLK period). To start a burst read sequence, set DIN = 0x3E00. The contents of each output data register are then shifted out on DOUT, starting with SUPPLY_OUT and ending with AUX_ADC (see Figure 13) in order by address (see Table 8).

MCU에서 모듈을 컨트롤 하기 위한 protocol이다.

모듈간에 제어할 때에도 정해진 규칙(프로토콜)이 필요하다.

Table 2. STM32F4xx register boundary addresses (continued)

Boundary address	Peripheral	Bus	Register map
0x4004 0000 - 0x4007 FFFF	USB OTG HS	AHB1	Section 31.12.6: OTG_HS register map on page 1248
0x4002 9000 - 0x4002 93FF	ETHERNET MAC		Section 29.8.5: Ethernet register maps on page 1017
0x4002 8C00 - 0x4002 8FFF			
0x4002 8800 - 0x4002 8BFF			
0x4002 8400 - 0x4002 87FF			
0x4002 8000 - 0x4002 83FF			
0x4002 6400 - 0x4002 67FF	DMA2		Section 9.5.11: DMA register map on page 245
0x4002 6000 - 0x4002 63FF	DMA1		
0x4002 4000 - 0x4002 4FFF	BKPSRAM		
0x4002 3C00 - 0x4002 3FFF	Flash interface register		Section 3.8: Flash interface registers
0x4002 3800 - 0x4002 3BFF	RCC		Section 6.3.32: RCC register map on page 181
0x4002 3000 - 0x4002 33FF	CRC		Section 4.4.4: CRC register map on page 88
0x4002 2000 - 0x4002 23FF	GPIOI		Section 7.4.11: GPIO register map on page 203
0x4002 1C00 - 0x4002 1FFF	GPIOH		
0x4002 1800 - 0x4002 1BFF	GPIOG		
0x4002 1400 - 0x4002 17FF	GPIOF		
0x4002 1000 - 0x4002 13FF	GPIOE		
0x4002 0C00 - 0x4002 0FFF	GPIOD		
0x4002 0800 - 0x4002 0BFF	GPIOC		
0x4002 0400 - 0x4002 07FF	GPIOB		
0x4002 0000 - 0x4002 03FF	GPIOA		
0x4001 5400 - 0x4001 57FF	SPI6	APB2	Section 27.5.10: SPI register map on page 845
0x4001 5000 - 0x4001 53FF	SPI5		
0x4001 4800 - 0x4001 4BFF	TIM11	APB2	Section 16.6.11: TIM10/11/13/14 register map on page 524
0x4001 4400 - 0x4001 47FF	TIM10		
0x4001 4000 - 0x4001 43FF	TIM9		Section 16.5.14: TIM9/12 register map on page 514
0x4001 3C00 - 0x4001 3FFF	EXTI		Section 10.3.7: EXTI register map on page 262
0x4001 3800 - 0x4001 3BFF	SYSCFG		Section 7.2.8: SYSCFG register maps on page 157
0x4001 3400 - 0x4001 37FF	SPI4	APB2	Section 27.5.10: SPI register map on page 845

이 사진은 메카트로닉스공학과에서 사용하는 Cortex32f4 MCU의 Register Map이다.

위와 같이 MCU에는 많은 모듈을 손쉽게 접근하기위해 Register_Map 방법을 쓴다. 다른말로 메모리 맵이라고도 부른다.

3)Register Map 설계 목표와 목적

-이번 실습에서 전체적인 목표는 PC에서 UART통신을 통해 DE2보드로 W/R신호와 해당 Register주소, 데이터값을 송신하면 DE2보드에서 수신하여 W/R 신호를 해석하여 해당 Register에 데이터를 쓰거나, 읽어서 다시 PC로 보내주는 회로를 설계해볼 것이다.

-직접 Register_Map을 생성, 설정해 보고 통신을 통해 해당 Register에 데이터를 쓰거나 읽어오는 과정을 실행함으로써 기본적인 Register_Map 설정을 익혀보는 것이 목표이다.
또한 주소 설정 프로토콜을 익혀보는 것 또한 포함된다.

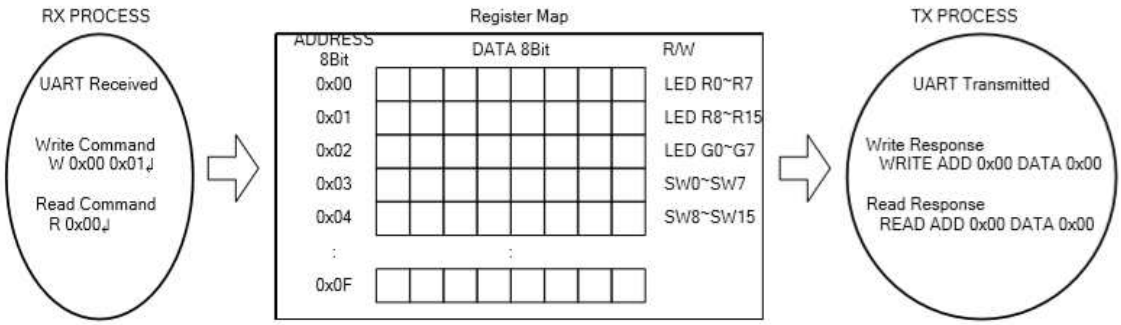
단. 이것 또한 정답이 아닌 방법의 하나일 뿐을 생각하면서 구상하는 것이다.

Register Map 설계 목표

- 레지스터 개수 : 16개 Address 0x00~0x0F로 구분
- DATA 8Bit
- UART를 이용한 Register Map Access Protocol
  - 터미널 명령어
  - 예약어(Space)0x(예약어)Add(hex)(Space)0x(예약어)data(hex)(Enter)
    - W 0x00 0x01 ↵ 0번지에 1값 쓰기
    - R 0x00 ↵ 0번지 읽기
  - FPGA Response : 해당 명령어에 대한 반응
    - WRITE ADD 0x00 DATA 0x01
    - READ ADD 0x00 DATA 0x01

Register Map 설계 목표

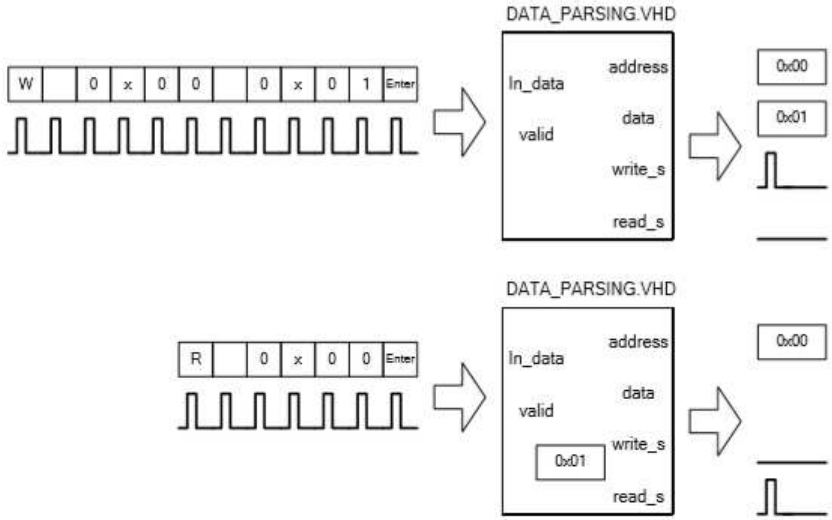
- 0x00~0x02 LED에 연결 : UART로 레지스터 값 설정
- 0x03~0x04 SW에 연결 : Board S/W로 레지스터 값 설정



1]Data_Pasing

data_parsing.vhd 설계

- UART_RX.vhd 를 통해 전달되는 명령어를 해석해 레지스터 맵에 쓰기 및 읽기를 위한 데이터 및 제어 신호 생성



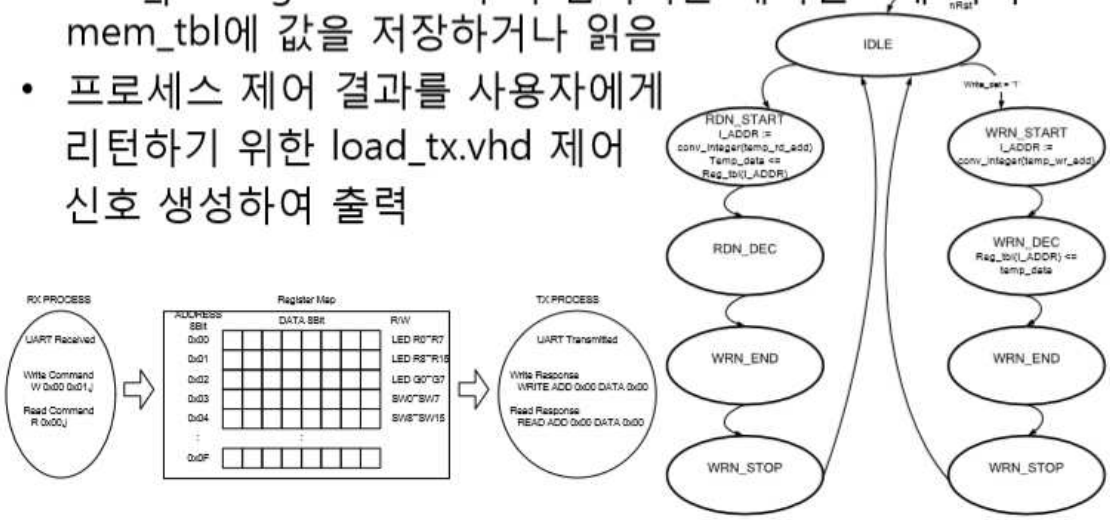
DE2보드로 UART통신을 통해 전해진 명령어를 해석하기 위한 로직이다.
또한 입력신호(아스키코드)을 16진법으로 바꿔서 송신한다.
쓰기 동작(write) = w 0x(add)(add) 0x(data)(data)(enter)
읽기 동작(read) = r 0x(add)(add)

일 때를 해석하여 write일 땐 address에 data값을 쓰기위한 제어신호를, read일 땐 address의 data값을 읽어오기위한 제어신호를 생성한다.
write => write_s 펄스 발생
read => read_s 펄스 발생
address => 주소 전달(아스키코드) -> 16진법)
data => 데이터 전달((아스키코드) -> 16진법)

2]Register_Map

register_map.vhd 설계

- Mem_tbl을 이용한 메모리 설계 응용
- data_parsing.vhd 으로부터 입력되는 제어신호에 따라 mem_tbl에 값을 저장하거나 읽음
- 프로세스 제어 결과를 사용자에게 리턴하기 위한 load_tx.vhd 제어 신호 생성하여 출력

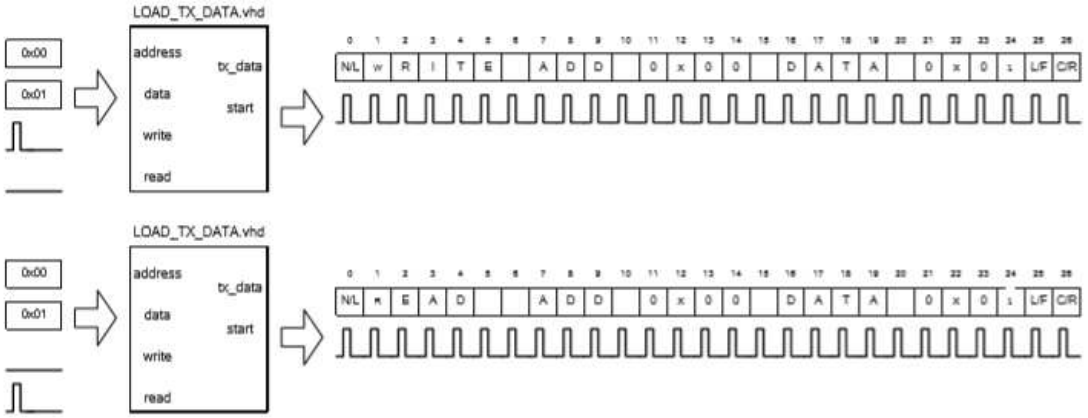


Data_Pasing로직으로부터 받은 신호를 처리하는 로직이다. 읽기와 쓰기 동작으로 나뉘어지며 해당 시퀀스에 따라 해당 Register에 값을 읽어주거나 써주는 로직 즉 Register_Map 로직이다. 읽기 쓰기 모두 IDLE=>START=>DEC=>END=>STOP=>IDLE 상태 순서로 진행되며 쓰기 동작 시 주소값과 데이터 두 개를 읽어와 저장하는 반면. 읽기동작에서는 주소 값만 읽어와 출력신호를 발생시킨다.

3]Road_Tx_Data

load_tx_data.vhd 설계

- register_map.vhd로부터 입력되는 제어신호에 따라 uart_tx.vhd를 통해 리턴되는 명령어를 순차적으로 보내는 로직
- uart_tx.vhd의 busy 신호를 체크해 데이터를 전달



Register_Map로직이 명령을 수행 후 PC로 확인하는 메시지를 보내주는 로직이다.

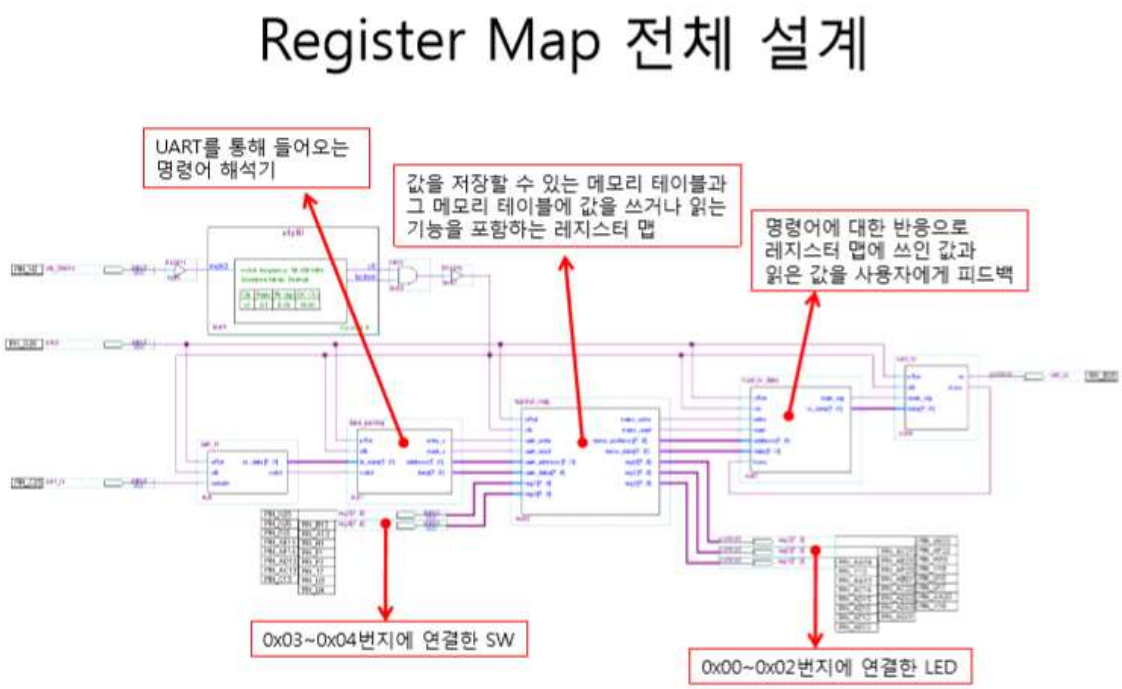
쓰기 동작 후에는

WRITE ADD 0x주소(아스키) 0x데이터(아스키)를 전송하며,

읽기 동작 후에는

READ ADD 0x주소(아스키) 0x데이터(아스키)를 전송해 준다.

4)Register map 전체 구상도(Block Diagram/Schematic)



전체적인 구상도 이다. 50MHz를 100MHz로 분주시키기 위한 clk분주기, UART통신을 통해 값을 받아오는 UART_RT로직, 받아온 값을 해석하는 Data_Pasing로직, 데이터를 읽거나 저장해주는 Register_Map, 작동 확인 할 수 있는 메시지를 보내주는 Load_Tx_Data로직, PC로 UART 송신을 하기위한 UART_TX로직으로 구성된다.

각 입출력 포트들은 DE2에 있는 LED, SW에 연결 맵핑 해줘서 DE2보드에서 확인 가능하도록 설정한다.

2. 소스코드 설명

1) Data_Pasing

```
1   library ieee;
2      use ieee.std_logic_1164.all;
3      use ieee.std_logic_arith.all;
4      use ieee.std_logic_unsigned.all;
5
6   entity data_parsing is
7     port(
8        nRst : in std_logic;
9        clk : in std_logic;
10       in_data : in std_logic_vector(7 downto 0);
11       valid : in std_logic;
12       write_s : out std_logic;
13       read_s : out std_logic;
14       address : out std_logic_vector(7 downto 0);
15       data : out std_logic_vector(7 downto 0)
16    );
17  end data_parsing;
18
19  architecture beh of data_parsing is
20
21     function data_decode(in_data : std_logic_vector(7 downto 0))return
22                                    std_logic_vector is
23                                    variable return_data : std_logic_vector(3 downto 0);
24        begin
25          case in_data is
26          when "00110000" => return_data := "0000";   --0 0x30
27          when "00110001" => return_data := "0001";   --1 0x31
28          when "00110010" => return_data := "0010";   --2 0x32
29          when "00110011" => return_data := "0011";   --3 0x33
30          when "00110100" => return_data := "0100";   --4 0x34
31          when "00110101" => return_data := "0101";   --5 0x35
32          when "00110110" => return_data := "0110";   --6 0x36
33          when "00110111" => return_data := "0111";   --7 0x37
34          when "00111000" => return_data := "1000";   --8 0x38
35          when "00111001" => return_data := "1001";   --9 0x39
36          when "01100001" => return_data := "1010";   --a 0x61
37          when "01101010" => return_data := "1011";   --b 0x62
38          when "01100011" => return_data := "1100";   --c 0x63
39          when "01100100" => return_data := "1101";   --d 0x64
40          when "01100101" => return_data := "1110";   --e 0x65
41          when "01100110" => return_data := "1111";   --f 0x66
42          when "01000001" => return_data := "1010";   --A 0x41
43          when "01000010" => return_data := "1011";   --B 0x42
44          when "01000011" => return_data := "1100";   --C 0x43
45          when "01000100" => return_data := "1101";   --D 0x44
46          when "01000101" => return_data := "1110";   --E 0x45
47          when "01000110" => return_data := "1111";   --F 0x46
48          when others     => return_data := "0000";
49          end case;
50        return(return_data);
51    end data_decode;
```

1~4 : IEEE라이브러리 참조선언

6~17 : IN = nRst, clk, in_data(uart 통신으로 받은 2진법 데이터), valid(데이터 받는 동안 확인신호)
OUT = write_s(쓰기동작 시작신호출력), read_s(읽기동작 시작신호출력), address(접근 주소 출력), data(데이터 출력)

21~23 : (아스키)->(2진법) 변환 함수 설정. [data_decode(in_data)->return_data]

25~51 : data_decode 내부 공식.

```
52
53      signal temp_data : std_logic_vector(7 downto 0);
54      signal data_0    : std_logic_vector(7 downto 0);
55      signal data_1    : std_logic_vector(7 downto 0);
56      signal data_2    : std_logic_vector(7 downto 0);
57      signal data_3    : std_logic_vector(7 downto 0);
58      signal data_4    : std_logic_vector(7 downto 0);
59      signal data_5    : std_logic_vector(7 downto 0);
60      signal data_6    : std_logic_vector(7 downto 0);
61      signal data_7    : std_logic_vector(7 downto 0);
62      signal data_8    : std_logic_vector(7 downto 0);
63      signal data_9    : std_logic_vector(7 downto 0);
64      signal data_10   : std_logic_vector(7 downto 0);
65      signal data_11   : std_logic_vector(7 downto 0);
66      signal valid_d   : std_logic;
67      signal valid_det   : std_logic;
68      signal valid_det_d   : std_logic;
69      signal valid_det_d2   : std_logic;
70
71    begin
72      process(nRst, clk)
73        begin
74          if(nRst = '0')then
75              valid_d <= '0';
76              valid_det <= '0';
77              valid_det_d <= '0';
78              valid_det_d2 <= '0';
79       elsif(rising_edge(clk)) then
80              valid_d <= valid;
81              valid_det_d <= valid_det;
82              valid_det_d2 <= valid_det_d;
83              if(valid_d = '0') and (valid = '1')then
84              valid_det <= '1';
85            else
86              valid_det <= '0';
87            end if;
88         end if;
89       end process;
90
91       process(nRst, clk)
92         begin
93           if(nRst = '0')then
94           temp_data <= (others => '0');
95           data_0    <= (others => '0');
96           data_1    <= (others => '0');
97           data_2    <= (others => '0');
98           data_3    <= (others => '0');
99           data_4    <= (others => '0');
100          data_5    <= (others => '0');
101          data_6    <= (others => '0');
102          data_7    <= (others => '0');
103          data_8    <= (others => '0');
104          data_9    <= (others => '0');
105          data_10   <= (others => '0');
106          data_11   <= (others => '0');
107          address   <= (others => '0');
108          data      <= (others => '0');
109          write_s   <= '0';
110          read_s    <= '0';
```

```
111     elsif (rising_edge(clk))then
112       temp_data <= in_data;
113       if(valid_det = '1')then
114         data_0 <= temp_data;  --enter     enter
115         data_1 <= data_0;     --data_lsb  data_lsb
116         data_2 <= data_1;     --data msb  data_msb
117         data_3 <= data_2;     --x         x
118         data_4 <= data_3;     --0         0
119         data_5 <= data_4;     --sp        sp
120         data_6 <= data_5;     --add lsb   r
121         data_7 <= data_6;     --add msb
122         data_8 <= data_7;     --x
123         data_9 <= data_8;     --0
124         data_10 <= data_9;    --sp
125         data_11 <= data_10;   --w
126       end if;
127
128       if(data_0 = x"0D") then  --enter
129       --if(data_0 = x"20") then --space
130               --  w                    sp                0
131         if((data_11 = x"77")and(data_10 = x"20")and (data_9 = x"30")
132                 --x                    sp                0              x
133         and(data_8 = x"78")and(data_5 = x"20")and(data_4 = x"30")and(data_3 = x"78"))then
134           write_s <= valid_det_d2;
135           address <= data_decode(data_7) & data_decode(data_6);
136           data    <= data_decode(data_2) & data_decode(data_1);
137         end if;
138         --        r               sp                0              x
139         if((data_6 = x"72")and(data_5 = x"20")and (data_4 = x"30")and(data_3 = x"78"))then
140           read_s <= valid_det_d2;
141           address <= data_decode(data_2) & data_decode(data_1);
142           data    <= (others => '0');
143         end if;
144
145       end if;
146     end if;
147   end process;
148 end beh;
```

53~69 : 내부 연결 data signal들. 각 data_x들은 8bit로 이뤄져 해당 자리의 아스키 코드를 가지고 있다. write통신시 12자리가 uart통신에 의해 전달되므로 data signal을 12개 선언해 준다.
valid_d, valid_det, valid_det_d, valid_det_d2의 경우 전달받는도중 데이터가 겹치지 않게 하는 시그널 들이나 det_d같이 더 선언해준 이유는 신호를 딜레이 시켜서 원하는 타이밍에 신호를 얻기 위해서 이다.(4clk딜레이)

72~89 : nRst 일 때 모두 '0'으로 초기화, clk의 raising edge에서 valid 신호를 valid_d에 입력하며 valid=1, valid_d=0 일 때 valid_det가 1이되며 한클락당 valid_det_d, valid_det_d2로 이동된다.

91~126 : nRst시 모든 신호 초기화, clk의 raising edge에서 valid_det신호가 '1'이 되면 내부 data_x를 상위 data_x로 이동시킨다.

128 : data_0번에 "0D" 즉 enter가 입력되면 작동을 시작한다.

131~143 : (write검사)data_11 = w, data_10 = space, data_9 = 0, data_8 = x, data_5 = space, data_4 = 0, data_3 = x 모두 만족할 시 write_s , address, data가 출력된다.
(read검사)data_6 = r, data_5 = space, data_4 = 0, data_3 = x 모두 만족할 시 read_s, address, data가 출력된다.

2) Register_Map

```
1   library ieee;
2      use ieee.std_logic_1164.all;
3      use ieee.std_logic_arith.all;
4      use ieee.std_logic_unsigned.all;
5
6   entity register_map is
7      port(
8           nRst              : in std_logic;
9           clk               : in std_logic;
10
11          uart_write        : in std_logic;
12          uart_read         : in std_logic;
13          uart_address      : in std_logic_vector(7 downto 0);
14          uart_data         : in std_logic_vector(7 downto 0);
15
16          trans_write       : out std_logic;
17          trans_read        : out std_logic;
18          trans_address     : out std_logic_vector(7 downto 0);
19          trans_data        : out std_logic_vector(7 downto 0);
20
21          reg0              : out std_logic_vector(7 downto 0); -- led  r0~r7
22          reg1              : out std_logic_vector(7 downto 0); -- led  r8~r15
23          reg2              : out std_logic_vector(7 downto 0); -- led  g0~g7
24
25          reg3              : in std_logic_vector(7 downto 0); -- led sw0~ sw7
26          reg4              : in std_logic_vector(7 downto 0)  -- led sw8~ sw15
27          );
28  end register_map;
29
30  architecture beh of register_map is
31
32        type state_type is(IDLE, WRN_START, WRN_DEC, WRN_END, WRN_STOP, RDN_START, RDN_DEC, RDN_END, RDN_STOP);
33        signal state : state_type;
34
35        type mem_tbl is array(0 to 15) of std_logic_vector(7 downto 0);
36        signal reg_tbl : mem_tbl;
37
38        signal temp_wr_add : std_logic_vector(7 downto 0);
39        signal temp_rd_add : std_logic_vector(7 downto 0);
40        signal temp_address : std_logic_vector(7 downto 0);
41        signal temp_data : std_logic_vector(7 downto 0);
42        signal temp_mdi : std_logic_vector(7 downto 0);
43        signal temp_mdo : std_logic_vector(7 downto 0);
44
45        signal write_d : std_logic;
46        signal write_det : std_logic;
47        signal read_d : std_logic;
48        signal read_det : std_logic;
49
```

1~4 : IEEE 라이브러리 선언

6~28 : IN = nRst, clk, uart_write(쓰기 신호 입력), uart_read(읽기 신호 입력), uart_address(주소 입력), uart_data(데이터 입력), reg3(3번 register의 데이터 입력), reg4(4번 register 데이터 입력)

OUT = trans_write(쓰기 신호 출력), trans_read(읽기 신호 출력), trans_address(주소 신호 출력), trans_data(데이터 신호 출력), reg0~2(쓰기 상태 시 0~2번 register의 데이터 신호 출력)

30~49 : state = IDLE(대기), WRN_START(쓰기 신호 받기), WRN_DEC(쓰기 신호 출력), WRN_END(쓰기 완료 출력), WRN_STOP(쓰기 종료), RDN_START(읽기 신호 받기), RDN_DEC(읽기 신호 출력), RDN_END(읽기 완료 출력), RDN_STOP(읽기 종료)

temp_wr_add(쓰기 주소), temp_rd_add(읽기 주소), temp_address(주소 저장공간), temp_data(데이터 저장공간), temp_mdi(데이터 임시 저장), temp_mdo(필요없는 신호. 지워도 무방), write_d, write_det(쓰기 신호 입력), read_d, read_det(읽기 신호 입력)

```
50  begin
51      process(nRst, clk)
52      begin
53          if(nRst = '0') then
54              write_d <= '0';
55              write_det <= '0';
56              read_d <= '0';
57              read_det <= '0';
58              temp_wr_add <= (others => '0');
59              temp_rd_add <= (others => '0');
60              temp_mdi <= (others => '0');
61          elsif(rising_edge(clk))then
62              write_d <= uart_write;
63              read_d <= uart_read;
64              if(write_d = '0')and(uart_write = '1')then
65                  write_det <= '1';
66                  temp_wr_add <= uart_address;
67                  temp_mdi <= uart_data;
68              elsif(state = WRN_START) then
69                  temp_wr_add <= (others => '0');
70                  temp_mdi <= (others => '0');
71              else
72                  write_det <= '0';
73              end if;
74              if(read_d = '0')and (uart_read ='1')then
75                  read_det <= '1';
76                  temp_rd_add <= uart_address;
77              elsif(state = RDN_START)then
78                  temp_rd_add <= (others => '0');
79              else
80                  read_det <= '0';
81              end if;
82           end if;
83        end process;
84
85        process(nRst, clk)
86           variable I_ADDR : natural;
87          begin
88            if(nRst = '0')then
89              state <= IDLE;
90              temp_address <= (others => '0');
91              temp_data <= (others => '0');
92              I_ADDR := 0;
93              trans_address <= (others => '0');
94              trans_data <= (others => '0');
95              trans_write <= '0';
96              trans_read <= '0';
97            elsif (rising_edge(clk))then
98              case state is
99                 when IDLE =>
100                   if(write_det = '1')then
101                       state <= WRN_START;
102                   elsif(read_det = '1')then
103                       state <= RDN_START;
104                   else
105                       state <= IDLE;
106                   end if;
107                   temp_address <= (others => '0');
108                   temp_data <= (others => '0');
109                   I_ADDR := 0;
110                   trans_address <= (others => '0');
111                   trans_data <= (others => '0');
112                   trans_write <= '0';
113                   trans_read <= '0';
```

```
114                 when WRN_START =>
115                    temp_address <= temp_wr_add;
116                    temp_data <= temp_mdi;
117                    I_ADDR   := conv_integer(temp_wr_add);
118                    state  <= WRN_DEC;
119                 when WRN_DEC =>
120                    state    <= WRN_END;
121                    reg_tbl(I_ADDR)  <= temp_data;
122                    trans_address    <= temp_address;
123                    trans_data       <= temp_data;
124                 when WRN_END =>
125                    trans_write  <='1';
126                    state   <= WRN_STOP;
127                 when WRN_STOP =>
128                    trans_write <= '0';
129                    state <= IDLE;
130                 when RDN_START =>
131                    temp_address <= temp_rd_add;
132                    I_ADDR   := conv_integer(temp_rd_add);
133                    temp_data <= reg_tbl(I_ADDR);
134                    state  <= RDN_DEC;
135                 when RDN_DEC =>
136                    state  <= RDN_END;
137                    trans_address <= temp_address;
138                    trans_data <= temp_data;
139                 when RDN_END =>
140                    trans_read <= '1';
141                    state <= RDN_STOP;
142                 when RDN_STOP =>
143                    trans_read <= '0';
144                    state <= IDLE;
145                 when others =>
146                 state <= IDLE;
147              end case;
148              reg0 <= reg_tbl(0);
149              reg1 <= reg_tbl(1);
150              reg2 <= reg_tbl(2);
151              reg_tbl(3) <= reg3;
152              reg_tbl(4) <= reg4;
153            end if;
154          end process;
155       end beh;
156
```

53~60 : nRst 시 모든 신호 초기화

61~63 : rising edge clk에서 진행. uart에서 온 읽기 or 쓰기 신호 입력

64~73 : 읽기 신호 검출시 uart_address -> temp_wr_add, uart_data -> temp_mdi, 현재 state가 WRN_START라면 temp_wr_add, temp_mdi 초기화, 두 경우가 아니라면 write신호는 발생하지 않음.

74~83 : 읽기 신호 검출시 uart_address -> temp_rd_add , 현재 state가 RDN_START라면 temp_rd_add초기화

86 : I_ADDR = 자연수로 정의.

87~96 : nRst시 모든 신호 초기화

97 : clk의 rising edge에서 동작.

98~113 : case문 IDLE(대기) 상태시 모든신호 초기화상태이며, write_det = '1'이라면 WRN_START상태로 이동. read_det = '1'이라면 RDN_START로 이동, 두 신호다 아니라면 IDLE 상태

114~118 : WRN_START상태 동작. temp_wr_add에 있던 쓰기 주소를 temp_address로 읽어옴, temp_mdi에 있던 쓰기 데이터를 temp_data로 읽어옴, temp_wr_add의 데이터를 자연수로 바꿔서 I_ADDR에 저장. state를 WRN_DEC로 이동

119~123 : WRN_DEC상태 동작. temp_address를 trans_address로 출력, temp_data를 trans_data로 출력. temp_data를 reg_tb1의 선택된 주소에 씀. 선택된 주소란 START에서 I_ADDR에 저장된 주소를 뜻함. WRN_END상태로 이동

123~126 : WRN_END상태동작. trans_write 신호 '1'을 출력, WRN_STOP상태로 이동

127~129 : WRN_STOP상태 동작. trans_write 신호를 리셋시키고 IDLE(대기)상태로 이동

130~147 : 동일한 순서로 읽기(read) RDN_START -> RDN_DEC -> RDN_END -> RDN_STOP -> IDLE 순서로 동작함. 그 이외의 경우는 모두 IDLE상태.

148~150 : reg0~2 = reg_tb1(0~2)의 값을 해당 출력선으로 출력시킨다

151~152 : reg_tb1(3,4)에 입력받은 reg3,4를 의 값을 입력시킨다.

2) Load_Tx_Data

```
1   library ieee;
2     use ieee.std_logic_1164.all;
3     use ieee.std_logic_arith.all;
4     use ieee.std_logic_unsigned.all;
5
6     entity load_tx_data is
7     port(
8       nRst : in std_logic;
9       clk : in std_logic;
10      write : in std_logic;
11      read : in std_logic;
12      address : in std_logic_vector(7 downto 0);
13      data : in std_logic_vector(7 downto 0);
14      busy : in std_logic;
15      start_sig : out std_logic;
16      tx_data : out std_logic_vector(7 downto 0)
17    );
18  end load_tx_data;
19
20  architecture beh of load_tx_data is
21
22          function data_decode(in_data : std_logic_vector(3 downto 0))return
23                                      std_logic_vector is
24                                      variable return_data : std_logic_vector(7 downto 0);
25      begin
26        case in_data is
27          when "0000" => return_data := "00110000"; -- 0 0x30
28          when "0001" => return_data := "00110001"; -- 1 0x31
29          when "0010" => return_data := "00110010"; -- 2 0x32
30          when "0011" => return_data := "00110011"; -- 3 0x33
31          when "0100" => return_data := "00110100"; -- 4 0x34
32          when "0101" => return_data := "00110101"; -- 5 0x35
33          when "0110" => return_data := "00110110"; -- 6 0x36
34          when "0111" => return_data := "00110111"; -- 7 0x37
35          when "1000" => return_data := "00111000"; -- 8 0x38
36          when "1001" => return_data := "00111001"; -- 9 0x39
37          when "1010" => return_data := "01000001"; -- A 0x41
38          when "1011" => return_data := "01000010"; -- B 0x42
39          when "1100" => return_data := "01000011"; -- C 0x43
40          when "1101" => return_data := "01000100"; -- D 0x44
41          when "1110" => return_data := "01000101"; -- E 0x45
42          when "1111" => return_data := "01000110"; -- F 0x46
43          when others     => return_data := "00111111"; -- ?
44          end case;
45        return(return_data);
46    end data_decode;
```

1~4 : IEEE라이브러리 선언

6~18 : IN = nRst, clk, write(쓰기신호입력), read(읽기신호입력), address(주소신호입력), data(데이터신호입력), busy(작동중 상태신호입력)

OUT = start_sig(시작 신호 출력), tx_data(전송 데이터 출력)

22~24 : (2진법)->(아스키) 변환 함수 설정. [data_decode(in_data)->return_data]

25~46 : data_decode 내부 공식.

```
47
48  type state_type is (IDLE,WRN_LOAD, WRN_SEND, WRN_WAIT, RDN_LOAD, RDN_SEND, RDN_WAIT);
49  type mem_tbl is array(0 to 26) of std_logic_vector(7 downto 0);
50  signal state     : state_type;
51  signal reg_tbl   : mem_tbl;
52  signal write_d   : std_logic;
53  signal write_det : std_logic;
54  signal read_d    : std_logic;
55  signal read_det  : std_logic;
56  signal busy_d    : std_logic;
57  signal busy_det  : std_logic;
58  signal data_cnt  : std_logic_vector(4 downto 0);
59  signal temp_address : std_logic_vector(7 downto 0);
60  signal temp_data : std_logic_vector(7 downto 0);
61
62  signal data_sp  : std_logic_vector(7 downto 0) := x"20";
63  signal data_a   : std_logic_vector(7 downto 0) := x"41";
64  signal data_d   : std_logic_vector(7 downto 0) := x"44";
65  signal data_0   : std_logic_vector(7 downto 0) := x"30";
66  signal data_x   : std_logic_vector(7 downto 0) := x"78";
67  signal data_m   : std_logic_vector(7 downto 0) := x"4D";
68  signal data_g   : std_logic_vector(7 downto 0) := x"47";
69
70  signal data_w   : std_logic_vector(7 downto 0) := x"57";
71  signal data_r   : std_logic_vector(7 downto 0) := x"52";
72  signal data_i   : std_logic_vector(7 downto 0) := x"49";
73  signal data_t   : std_logic_vector(7 downto 0) := x"54";
74  signal data_e   : std_logic_vector(7 downto 0) := x"45";
75
76  signal data_cr   : std_logic_vector(7 downto 0) := x"0D";
77  signal data_lf   : std_logic_vector(7 downto 0) := x"0A";
78  signal data_ff   : std_logic_vector(7 downto 0) := x"0C";
79  signal data_nl   : std_logic_vector(7 downto 0) := x"00";
80  signal data_ds   : std_logic_vector(7 downto 0) := x"2D";
81  signal data_cm   : std_logic_vector(7 downto 0) := x"2C";
82
```

48~60 : 내부 시그널들 정의.

state = IDLE(대기), WRN_LOAD(전송될 register에 각 자리에 맞는 데이터 입력), WRN_SEND(register에 있는 데이터를 전송한다), WRN_WAIT(모든 register가 다 전송될때까지 개수를 샌다), RDN_LOAD(전송될 register에 각 자리에 맞는 데이터 입력), RDN_SEND(register에 있는 데이터를 전송한다), RDN_WAIT(모든 register가 다 전송될때까지 개수를 샌다).

reg_tbl = 27개의 register로써 pc로 전송하기 위한 아스키 코드를 저장하는 곳이다.

write, read, busy = 읽기, 쓰기, 전송중 신호로 각각 검출신호와 쌍을 이룬다.

temp_address, data= 주소와 데이터 임시 저장소

data_cnt = 전송된 글자 개수를 세기위한 신호.

62~82 : 각 아스키 글자에 대한 이진법을 저장해둔 곳

예) data_sp = space (아스키) = “20” => “0010, 0000” 입력됨

```
83      begin
84        process(nRst, clk)
85          begin
86            if(nRst = '0')then
87                write_d <= '0';
88                write_det <= '0';
89                read_d <= '0';
90                read_det <= '0';
91                busy_d <= '0';
92                busy_det <= '0';
93                temp_address <= (others => '0');
94                temp_data <= (others => '0');
95          elsif(rising_edge(clk))then
96                write_d <= write;
97                read_d <= read;
98                busy_d <= busy;
99                if(write_d = '0')and(write = '1')then
100                   write_det <= '1';
101                   temp_address <= address;
102                   temp_data <= data;
103               else
104                    write_det <= '0';
105               end if;
106               if(read_d = '0')and(read = '1')then
107                    read_det <= '1';
108                    temp_address <= address;
109                    temp_data <= data;
110               else
111                    read_det <= '0';
112               end if;
113               if(busy_d = '0')and(busy = '1')then
114                    busy_det <= '1';
115               else
116                   busy_det <= '0';
117               end if;
118         end if;
119       end process;
```

83~94 : nRst시 모든 신호 초기화

95~98 : clk rising edge시 write, read, busy신호를 받아서 저장한다.

99~105 : 쓰기신호 검출시 쓰기 주소와 쓰기 데이터를 저장한다.

106~112 : 읽기 신호 검출시 읽기 주소와 읽기 데이터를 저장한다.

113~119 : busy신호 검출시 busy검출 신호를 '1'로 출력한다.

```
121     process(nRst, clk)
122        variable ADDR : natural;
123       begin
124          if(nRst = '0')then
125              state          <= IDLE;
126              data_cnt       <= (others => '0');
127              reg_tbl(0)     <= (others => '0');
128              reg_tbl(1)     <= (others => '0');
129              reg_tbl(2)     <= (others => '0');
130              reg_tbl(3)     <= (others => '0');
131              reg_tbl(4)     <= (others => '0');
132              reg_tbl(5)     <= (others => '0');
133              reg_tbl(6)     <= data_sp;
134              reg_tbl(7)     <= data_a;
135              reg_tbl(8)     <= data_d;
136              reg_tbl(9)     <= data_d;
137              reg_tbl(10)    <= data_sp;
138              reg_tbl(11)    <= data_0;
139              reg_tbl(12)    <= data_x;
140              reg_tbl(13)    <= (others => '0');-- address msb
141              reg_tbl(14)    <= (others => '0');-- address lsb
142              reg_tbl(15)    <= data_sp;
143              reg_tbl(16)    <= data_d;
144              reg_tbl(17)    <= data_a;
145              reg_tbl(18)    <= data_t;
146              reg_tbl(19)    <= data_a;
147              reg_tbl(20)    <= data_sp;
148              reg_tbl(21)    <= data_0;
149              reg_tbl(22)    <= data_x;
150              reg_tbl(23)    <= (others => '0'); -- data msb
151              reg_tbl(24)    <= (others => '0'); -- data lsb
152              reg_tbl(25)    <= data_lf;
153              reg_tbl(26)    <= data_cr;
154              tx_data        <= (others => '0');
155              start_sig      <= '0';
156          elsif(rising_edge(clk))then
157             case state is
158               when IDLE =>
159                  if(write_det = '1')then
160                     state     <= WRN_LOAD;
161                  elsif(read_det = '1')then
162                     state <= RDN_LOAD;
163                  else
164                     state <= IDLE;
165                  end  if;
166                    data_cnt    <= (others => '0');
167                    tx_data     <= (others => '0');
168                    start_sig   <= '0';
169               when WRN_LOAD =>
170                  state          <= WRN_SEND;
171                  reg_tbl(1)     <= data_w;
172                  reg_tbl(2)     <= data_r;
173                  reg_tbl(3)     <= data_i;
174                  reg_tbl(4)     <= data_t;
175                  reg_tbl(5)     <= data_e;
176                  reg_tbl(13)    <= data_decode(temp_address(7 downto 4));--address msb
177                  reg_tbl(14)    <= data_decode(temp_address(3 downto 0));--address lsb
178                  reg_tbl(23)    <= data_decode(temp_data(7 downto 4));--data msb
179                  reg_tbl(24)    <= data_decode(temp_data(3 downto 0));--data lsb
180               when WRN_SEND =>
181                  state          <= WRN_WAIT;
182                  ADDR           := conv_integer(data_cnt);
183                  tx_data        <= reg_tbl(ADDR);
184                  start_sig      <= '1';
```

```
183                ADDR          := conv_integer(data_cnt);
184                tx_data       <= reg_tbl(ADDR);
185                start_sig     <= '1';
186             when WRN_WAIT =>
187                if(busy_det = '1')then
188                   if(data_cnt = 26)then
189                      data_cnt <= (others => '0');
190                      state    <= IDLE;
191                   else
192                      state    <= WRN_SEND;
193                      data_cnt <= data_cnt + 1;
194                   end if;
195                else
196                   state <= WRN_WAIT;
197                end if;
198                start_sig <= '0';
199             when RDN_LOAD =>
200                state        <= RDN_SEND;
201                reg_tbl(1)   <= data_r;
202                reg_tbl(2)   <= data_e;
203                reg_tbl(3)   <= data_a;
204                reg_tbl(4)   <= data_d;
205                reg_tbl(5)   <= data_sp;
206                reg_tbl(13)  <= data_decode(temp_address(7 downto 4));--address msb
207                reg_tbl(14)  <= data_decode(temp_address(3 downto 0));--address lsb
208                reg_tbl(23)  <= data_decode(temp_data(7 downto 4));--data msb
209                reg_tbl(24)  <= data_decode(temp_data(3 downto 0));--data lsb
210             when RDN_SEND =>
211                state        <= RDN_WAIT;
212                ADDR         := conv_integer(data_cnt);
213                tx_data      <= reg_tbl(ADDR);
214                start_sig    <= '1';
215             when RDN_WAIT =>
216                if(busy_det = '1')then
217                   if(data_cnt = 26)then
```

```
217                   if(data_cnt = 26)then
218                      state    <= IDLE;
219                      data_cnt <= (others => '0');
220                   else
221                      state    <= RDN_SEND;
222                      data_cnt <= data_cnt + 1;
223                   end if;
224                else
225                  state <= RDN_WAIT;
226                end if;
227                start_sig <= '0';
228              when others =>
229                state <= IDLE;
230            end case;
231          end if;
232        end process;
233      end beh;
```

122 : ADDR을 자연수로 선언

123~155 : nRst 신호시 register에 서 고정된 주소를 제외한 register값을 초기화 시킨다.(write 0x00 0x00에서 write 00(address) 00(data)을 제외한 부분.)

156~168 : IDLE(대기)상태 시 쓰기 신호 검출 시 WRN_LOAD상태로, 읽기 신호 검출시 RDN_LOAD상태로 이동한다. 아닐시 계속 대기.

169~179 : WRN_LOAD상태로써 pc로 보낼 데이터를 쓰는 구역이다. write = reg_tb1(1~5)에 쓰며 reg_tb1(13,14)에는 주소에 해당하는 값을 적으며, reg_tb1(23,24)에는 데이터 값을 적는다. 이때 8bit를 4bit씩 쪼개서 적는다.

180~185 : WRN_SEND상태로써 reg_tb1(0~26)에 있는 데이터 들을 차례로 전송하는 부분이다. 하나 보내질 때 마다 start_sig를 '1'로 출력한다.

186~198 : WRN_WAIT상태로써 27개의 reg_tb1이 다 보내지는걸 검사하는 부분이다.

199~233 : RDN_LOAD -> RDN_SEND -> RDN_WAIT 상태로써 쓰기 동작과 동일한 일을 수행한다.

4)Tast_Banch

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity TB_UART is  end ;

architecture BEH of TB_UART is

   component  UART_TX is
  port (
    nRst      : in std_logic;
    clk       : in std_logic;
    start_sig : in std_logic;
    data      : in std_logic_vector(7 downto 0);
    tx        : out std_logic;
    busy      : out std_logic
  );
   end component;

   component  UART_RX is
  port (
    nRst      : in std_logic;
    clk       : in std_logic;
    serialin  : in std_logic;
   rx_data   : out std_logic_vector(7 downto 0);
    valid     : out std_logic
  );
   end component;

   component data_parsing is
   port(
      nRst : in std_logic;
      clk : in std_logic;
      in_data : in std_logic_vector(7 downto 0);
      valid : in std_logic;
      write_s : out std_logic;
      read_s : out std_logic;
      address : out std_logic_vector(7 downto 0);
      data : out std_logic_vector(7 downto 0)
   );
 end component;

   component register_map is
    port(
        nRst              : in std_logic;
        clk               : in std_logic;

        uart_write        : in std_logic;
        uart_read         : in std_logic;
        uart_address      : in std_logic_vector(7 downto 0);
        uart_data         : in std_logic_vector(7 downto 0);

        trans_write       : out std_logic;
        trans_read        : out std_logic;
        trans_address     : out std_logic_vector(7 downto 0);
        trans_data        : out std_logic_vector(7 downto 0);

        reg0              : out std_logic_vector(7 downto 0); -- led  r0~r7
        reg1              : out std_logic_vector(7 downto 0); -- led  r8~r15
        reg2              : out std_logic_vector(7 downto 0); -- led  g0~g7

        reg3              : in std_logic_vector(7 downto 0); -- led sw0~ sw7
        reg4              : in std_logic_vector(7 downto 0)  -- led sw8~ sw15
        );
 end component;
```

```
68      component load_tx_data is
69       port(
70          nRst : in std_logic;
71          clk : in std_logic;
72          write : in std_logic;
73          read : in std_logic;
74          address : in std_logic_vector(7 downto 0);
75          data : in std_logic_vector(7 downto 0);
76          busy : in std_logic;
77          start_sig : out std_logic;
78          tx_data : out std_logic_vector(7 downto 0)
79       );
80    end component;
81
82          signal signRst, sigclk, sigstart_sig : std_logic;
83          signal sigdata : std_logic_vector(7 downto 0);
84          signal sig_data : std_logic;
85          signal int_cnt : std_logic_vector(99 downto 0);
86          signal sigvalid : std_logic;
87          signal sigbusy : std_logic;
88          signal sigrx_data : std_logic_vector(7 downto 0);
89
90          signal rx_data_pasing : std_logic_vector(7 downto 0);
91          signal valid_data_pasing : std_logic;
92          signal write_regi : std_logic;
93          signal read_regi : std_logic;
94          signal address_regi : std_logic_vector(7 downto 0);
95          signal data_regi : std_logic_vector(7 downto 0);
96
97          signal reg0_map : std_logic_vector(7 downto 0);
98          signal reg1_map : std_logic_vector(7 downto 0);
99          signal reg2_map : std_logic_vector(7 downto 0);
100         signal reg3_map : std_logic_vector(7 downto 0);
101         signal reg4_map : std_logic_vector(7 downto 0);
102         signal trans_w  : std_logic;
103         signal trans_r  : std_logic;
104         signal trans_add  : std_logic_vector(7 downto 0);
105         signal trans_data : std_logic_vector(7 downto 0);
106
107         signal start_sig_load : std_logic;
108         signal tx_data_load   : std_logic_vector(7 downto 0);
109
110         signal tx_end         : std_logic;
111         signal busy_end       : std_logic;
112
113   begin
114
```

```
113     begin
114
115       process
116       begin
117         if(NOW = 0 ns) then
118          signRst <= '0', '1' after 10 us;
119        end if;
120          wait for 1 sec;
121          end process;
122
123          process
124       begin
125          sigclk <= '0', '1' after 5 ns;
126          wait for 10 ns;
127          end process;
128
129          process(signRst, sigclk)
130            begin
131              if(signRst = '0') then
132              int_cnt <= (others => '0');
133            elsif(rising_edge(sigclk)) then
134            int_cnt <= int_cnt + 1;
135          end if;
136        end process;
137
```

```
139         sigstart_sig <= '1'when int_cnt = 500 else     --w
140                         '1'when int_cnt = 11000 else   --sp
141                         '1'when int_cnt = 21500 else   --0
142                         '1'when int_cnt = 32000 else   --x
143                         '1'when int_cnt = 42500 else   --address
144                         '1'when int_cnt = 53000 else   --address
145                         '1'when int_cnt = 63500 else   --sp
146                         '1'when int_cnt = 74000 else   --0
147                         '1'when int_cnt = 84500 else   --x
148                         '1'when int_cnt = 95000 else   --data
149                         '1'when int_cnt = 105500 else   --data
150                         '1'when int_cnt = 116000 else   --enter
151
152                         '1'when int_cnt = 130000 else   --r
153                         '1'when int_cnt = 140500 else   --sp
154                         '1'when int_cnt = 151000 else   --0
155                         '1'when int_cnt = 161500 else   --x
156                         '1'when int_cnt = 172000 else   --address
157                         '1'when int_cnt = 182500 else   --address
158                         '1'when int_cnt = 193000 else   --enter
159
160                         '0';
161
162         sigdata <= "01110111" when ((int_cnt > 450)and(int_cnt < 550)) else     --w
163                    "00100000" when ((int_cnt > 10950)and(int_cnt <11050))else   --sp
164                    "00110000" when ((int_cnt > 21450)and(int_cnt <21550))else   --0
165                    "01111000" when ((int_cnt > 31950)and(int_cnt <32050))else   --x
166                    "00110000" when ((int_cnt > 42450)and(int_cnt <42550))else   --address
167                    "00110001" when ((int_cnt > 52950)and(int_cnt <53050))else   --address
168                    "00100000" when ((int_cnt > 63450)and(int_cnt <63550))else   --sp
169                    "00110000" when ((int_cnt > 73950)and(int_cnt <74050))else   --0
170                    "01111000" when ((int_cnt > 84450)and(int_cnt <84550))else   --x
171                    "00110000" when ((int_cnt > 94950)and(int_cnt <95050))else   --data
172                    "00110010" when ((int_cnt > 105450)and(int_cnt <105550))else   --data
173                    "00001101" when ((int_cnt > 115950)and(int_cnt <116050))else   --enter
174
175                    "01110010" when ((int_cnt > 129950)and(int_cnt < 130050)) else     --w
176                    "00100000" when ((int_cnt > 140450)and(int_cnt < 140550))else   --sp
177                    "00110000" when ((int_cnt > 150950)and(int_cnt < 151050))else   --0
178                    "01111000" when ((int_cnt > 161450)and(int_cnt < 161550))else   --x
179                    "00110000" when ((int_cnt > 171950)and(int_cnt < 172050))else   --address
180                    "00110011" when ((int_cnt > 182450)and(int_cnt < 182550))else   --address
181                    "00001101" when ((int_cnt > 192950)and(int_cnt < 193050))else   --enter
182                    "00000000";
183
184          reg3_map <= "00010011" when int_cnt > 4000 else (others => '0');
185          reg4_map <= "00010100" when int_cnt > 4000 else (others => '0');
186
```

```
187
188     u_TB_UART_TX : UART_TX
189     port map(
190       nRst      => signRst,
191       clk       => sigclk,
192       start_sig => sigstart_sig,
193       data      => sigdata,
194       tx        => sig_data,
195       busy      => sigbusy);
196
197     u_TB_UART_RX : UART_RX
198      port map (
199       nRst      => signRst,
200      clk       => sigclk,
201      serialin  => sig_data,
202       rx_data   => sigrx_data,
203       valid     => sigvalid );
204
205     u_TB_data_pasing :  data_parsing
206      port map(
207         nRst      => signRst,
208         clk       => sigclk,
209         in_data   => sigrx_data,
210         valid     => sigvalid,
211         write_s   => write_regi,
212         read_s    => read_regi,
213         address   => address_regi,
214         data      => data_regi);
215
216      u_TB_register_map : register_map
217       port map(
218           nRst             => signRst,
219           clk              => sigclk,
220
221           uart_write       => write_regi,
222           uart_read        => read_regi,
223           uart_address     => address_regi,
224           uart_data        => data_regi,
225
226           trans_write      => trans_w,
227           trans_read       => trans_r,
228           trans_address    => trans_add,
229           trans_data       => trans_data,
230
231           reg0             => reg0_map,
232           reg1             => reg1_map,
233           reg2             => reg2_map,
234           reg3             => reg3_map,
235           reg4             => reg4_map);
236
```

```
237     u_TB_load_tx_data : load_tx_data
238     port map(
239         nRst            => signRst,
240         clk             => sigclk,
241         write           => trans_w,
242         read            => trans_r,
243         address         => trans_add,
244         data            => trans_data,
245         busy            => busy_end,
246         start_sig       => start_sig_load,
247         tx_data         => tx_data_load);
248
249      u_TB_uart_tx_end : uart_tx
250      port map(
251         nRst            => signRst,
252         clk             => sigclk,
253         start_sig       => start_sig_load,
254         data            => tx_data_load,
255         tx              => tx_end,
256         busy            => busy_end );
257
258 end BEH;
259
```

코드 설명에 앞서 TB는 Uart_Tx-> Uart_Rx -> Data_Pasing -> Register_Map -> Load_Tx_Data -> Uart_Tx 순서로 이어져 있다. 첫 데이터 입력을 컴퓨터에서 주었다는 가정으로 Tx로직부터 시작하였다.

1~4 : IEEE라이브러리 선언

10~19 : UART_TX로직 component

21~29 : UART_RX로직 component

31~42 : Data_Pasing로직 component

44~66 : Register_Map로직 component

68~80 : Load_Tx_Data로직 component

82~88: 모듈과 모듈 사이 연결 로직을 순서대로 정의함.
UART_TX - UART_RX간 연결 로직 선언. 정의함

90~95 : Data_Pasing모듈의 입출력 부분에 연결될 신호선을 정의함.

97~105 : Data_Pasing모듈의 신호선과 중복되는 부분을 제외한 Register_Map모듈의 입출력 신호선을 정의함

107~108 : Register_Map모듈의 신호선과 중복되는 부분을 제외한 Load_Tx_Data모듈의 입출력 신호선을 정의함

110~111 : Load_Tx_Data모듈의 신호선과 중복되는 부분을 제외한 UART_TX모듈의 입출력 신호선을 정의함

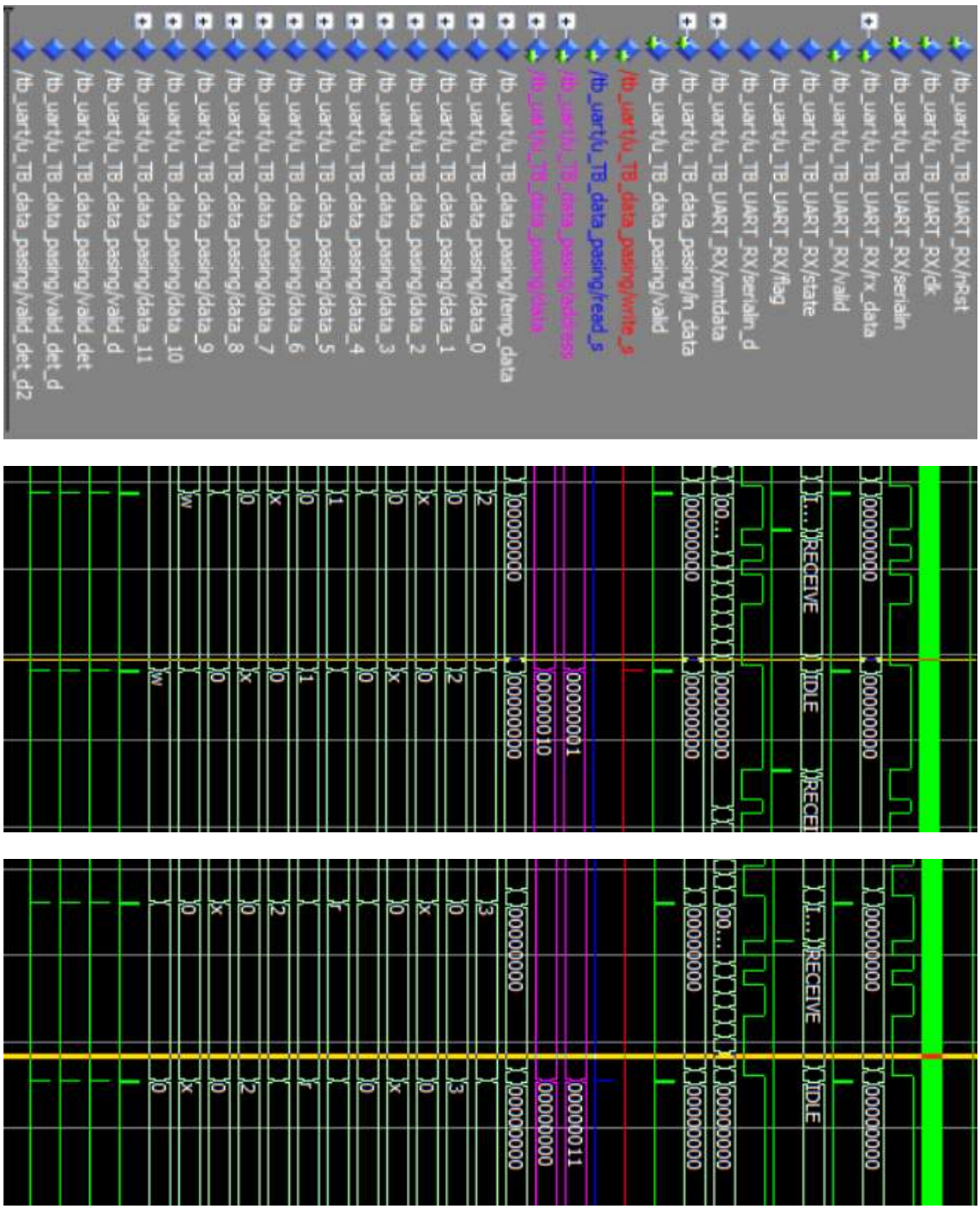
115~121 : nRst 신호가 0~10us까지 '0', 10us~1sec까지 '1'이 되도록 정의함

123~127 : 100MHZ의 주파수 클럭을 생성함.

129~136 : 100MHZ에 따른 clk의 개수를 세어주는 부분.
139~150 : TX로직을 가동 시키기 위한 start_sig를 정의 한 부분. 각 클럭의 사이를 10500x 10ns = 105us마다 신호를 보내주게 되어있다. 총 12번으로 (w 0x00 0x00)의 개수이다.
152~158 : TX로직을 가동 시키기 위한 start_sig를 정의 한 부분. 각 클럭의 사이를 10500x 10ns = 105us마다 신호를 보내주게 되어있다. 총 7번으로 (r 0x00)의 개수이다.
162~173 : start_sig를 보낼 때 마다 해당하는 값을 넣어주기 위한 것이다. (w 0x00 0x00)을 2진법 아스키 값으로 넣어준다
175~182 : start_sig를 보낼 때 마다 해당하는 값을 넣어주기 위한 것이다. (r 0x00)을 2진법 아스키 값으로 넣어준다.
184~185 : 초기 reg3,4의 데이터 값을 설정해준다.
188~195 : Uart_Tx로직을 port_map으로 연결
197~203 : Uart_Rx로직을 port_map으로 연결
205~214 : Data_Pasing로직을 port_map으로 연결
216~235 : Register_Map로직을 port_map으로 연결
237~247 : Load_Tx_Data로직을 port_map으로 연결
249~256 : Uart_Tx로직을 port_map으로 연결

3. 시뮬레이션 결과 및 설명

1)Uart_Rx to Data_Pasing



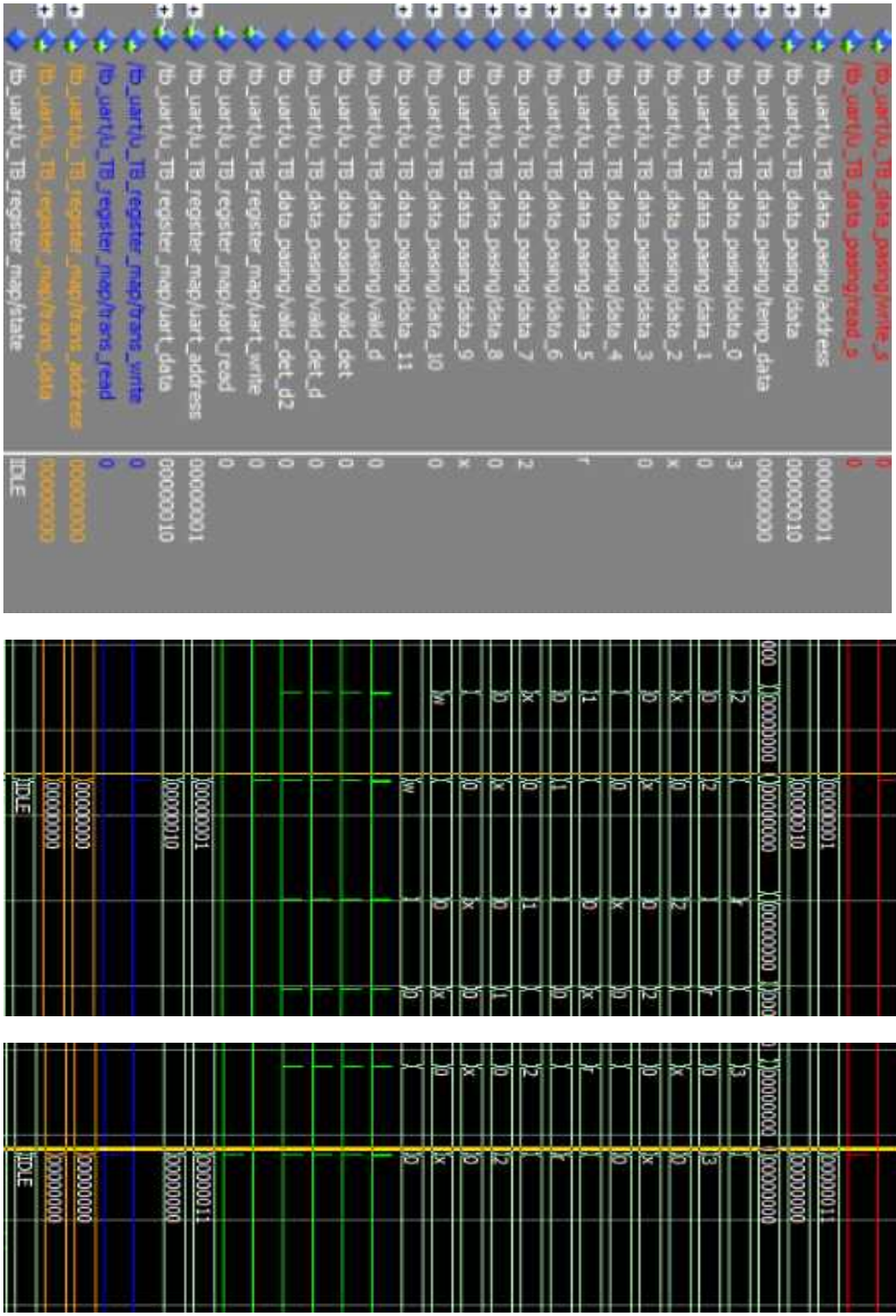
위에서부터 그림 1, 2, 3이다.

그림 1은 Uart_Rx와 Data_Pasing의 입출력, 내부 신호들을 나열한 사진이다.
빨간색은 write신호를 나타내며, 파란색은 read신호를 나타낸다.
분홍색은 address주소와 data데이터를 나타낸다.
Data_Pasing로직은 Uart_Rx로 입력된 데이터들을 분석하는 역할이다. 즉 값이 입력되었을 때 write 혹은 read를 구별하여 신호를 내보낼 수 있어야 한다. 그렇기에 중점적으로 봐야 할 부분은 write신호, read신호, address신호, data신호이다

그림 2에서는 읽어들인 데이터를 분석하여 write신호라는 것을 판단한 후 write_s신호를 출력하고, 읽은 데이터 속 address와 data를 판단하여 출력하는 모습이다.
입력받은 데이터가 (w 0x00 0x00)형식일 때 쓰기 주소라는 것을 판별하며 첫 0x00의 값을 주소로, 두 번째 0x00의 값을 데이터라는 것을 알아낸다.

그림 3에서는 읽어들인 데이터를 분석하여 read신호라는 것을 판단한 후 read_s신호를 출력하고, 읽은 데이터 속 address를 판단하여 출력하는 모습이다.
입력받은 데이터가 (r 0x00)형식일 때 읽기 주소라는 것을 판별하며 0x00의 값을 주소로 알아낸다.

2)Data_Pasing to Register_Map



위에서부터 그림 1, 2, 3이다.

그림1에서는 Data_Pasing로직과 Register_Map로직의 입출력 신호들을 정의 하였다.
빨간색 부분은 Data_Pasing에서 write인지 read인지 판단한 결과를 write_s 혹은 read_s로 출력하는데 이 부분이 Register_Map의 동작을 결정하게 되므로 중요한 파형이다.
파란색 부분은 Register_Map에서 출력하는 파형이다. write 혹은 read을 전달 받은 Register_Map은 주소와 데이터를 뒷단으로 출력하며 같이 write인지 read인지를 출력하는데 이부분이 trans_write 와 trans_read부분이다.

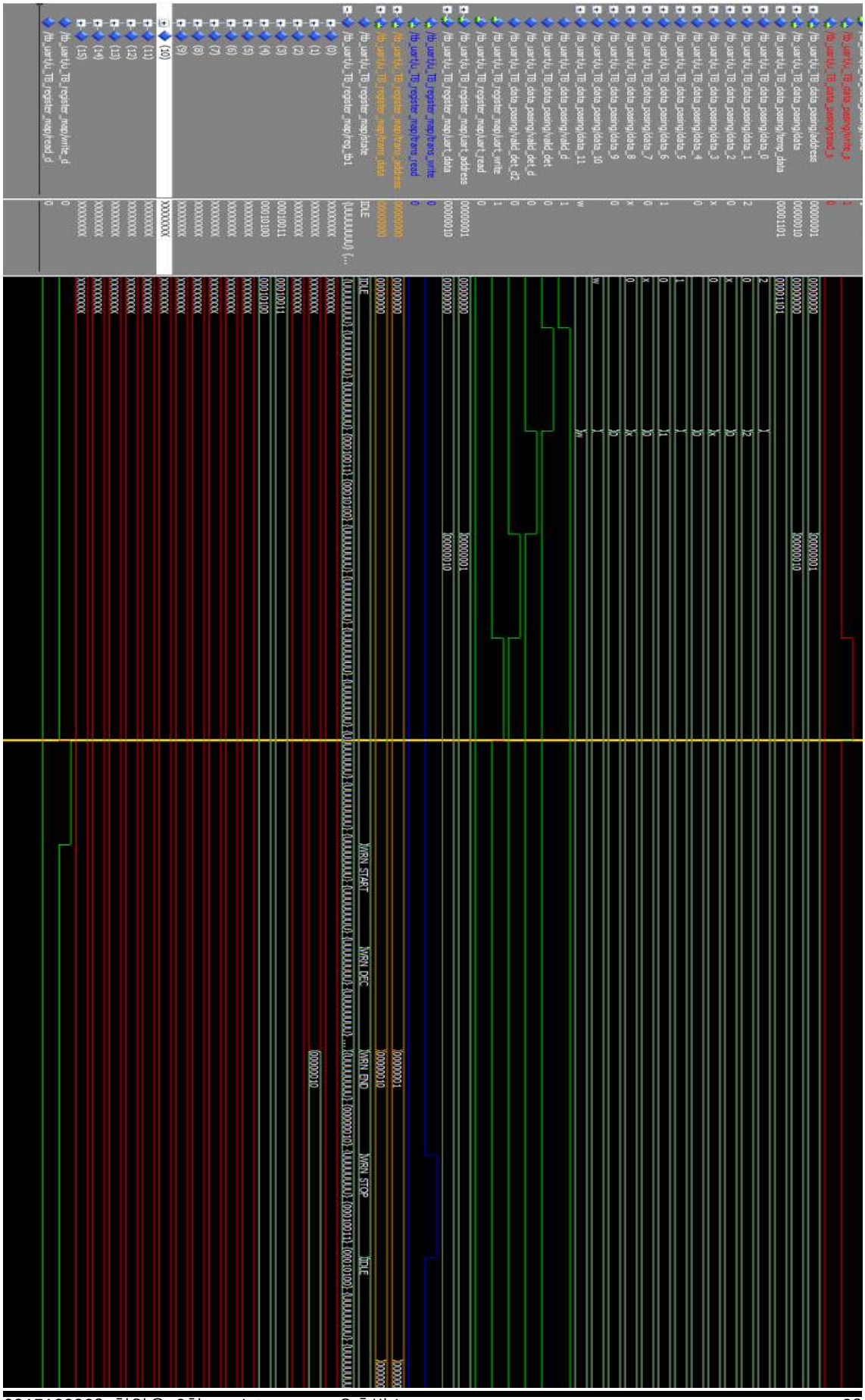
주황색 부분은 Register_Map에 저장된, 혹은 저장한 데이터 값과 해당 주소를 뒷단으로 출력해 주는 부분으로 trans_address와 trans_data이다.

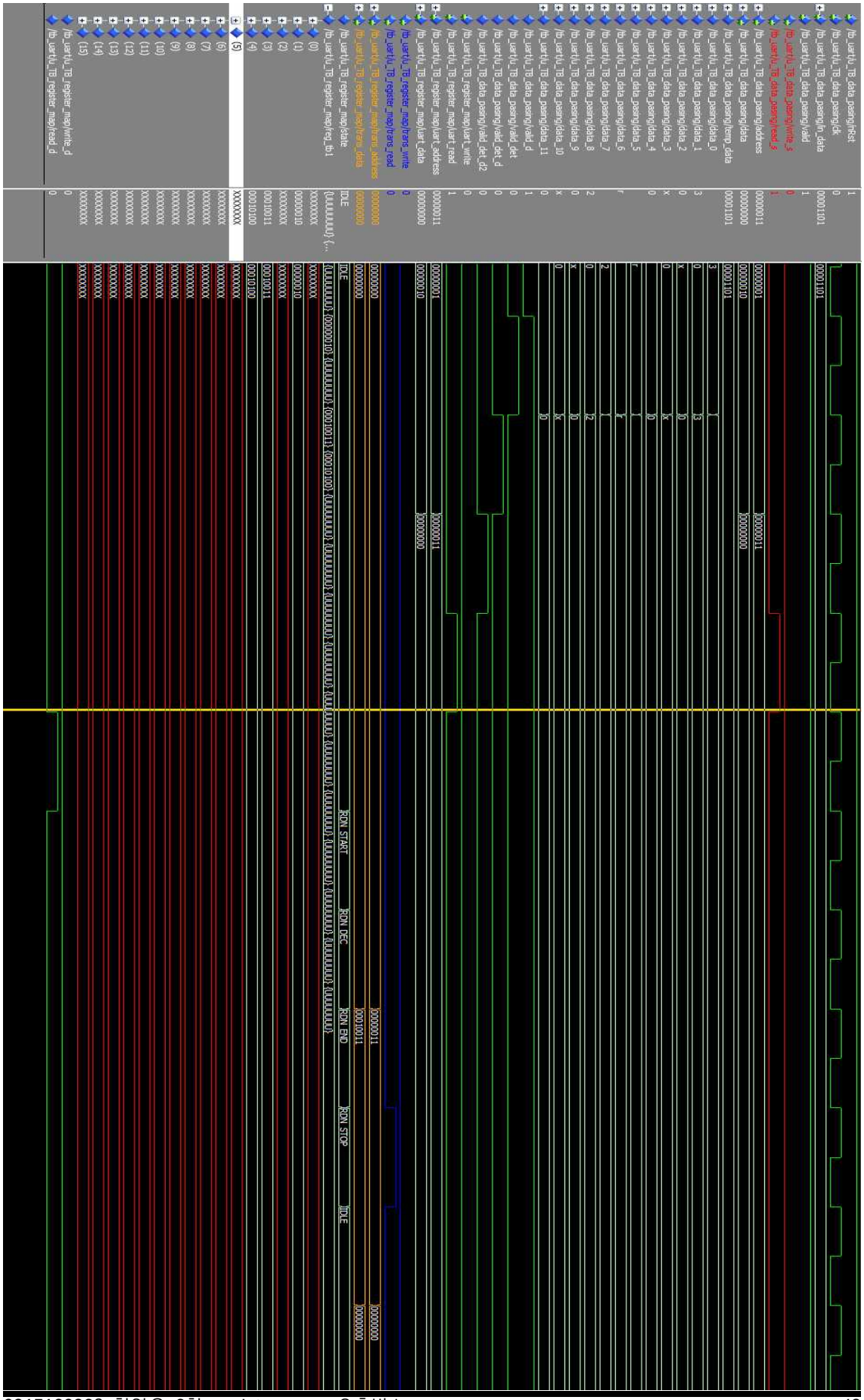
그림 2에서는 앞에 Data_Pasing의 write_s의 영향으로 write동작이 작동된 모습니다. (w 0x01 0x02) = 1번 주소에 '2'란 값을 저장한다.
주황색 부분은 순간적으로 나왔다 사라지므로 뒷부분에서 확대해서 다루기로 한다.

그림 3에서는 앞에 Data_Pasing의 read_s의 영향으로 read동작이 작동된 모습니다.
(r 0x03) = 3번 주소에 값을 읽어온다.
주황색 부분은 순간적으로 나왔다 사라지므로 뒷부분에서 확대해서 다루기로 한다.

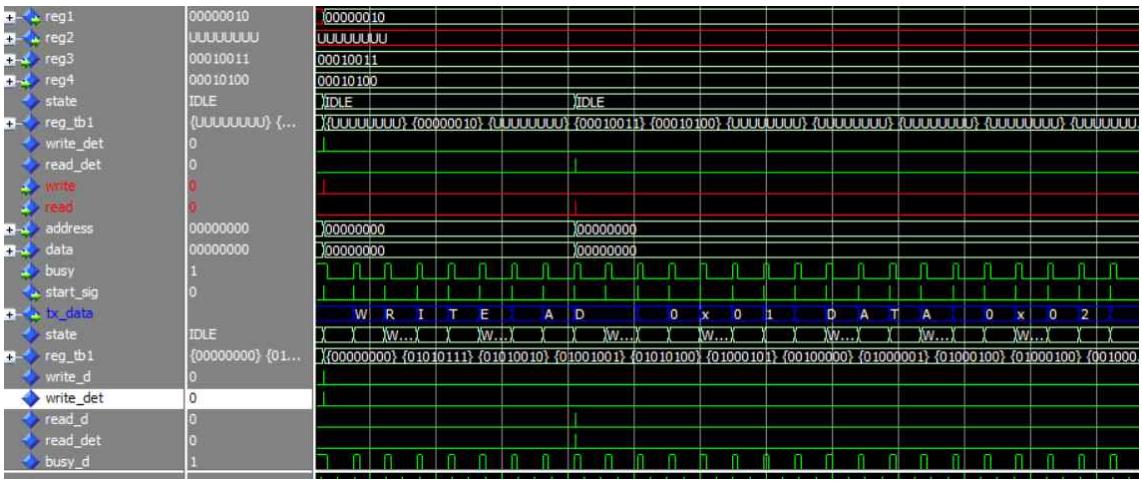
39페이지 그림은 write 신호일 때 파형을 확대해 놓은 것이다.
입력받은 신호가 write이므로 WRN_START를 시작으로 상태를 진행한다.

40페이지 그림은 read 신호일 때 파형을 확대해 놓은 것이다.
입력받은 신호가 read이므로 RDN_START를 시작으로 상태를 진행한다.

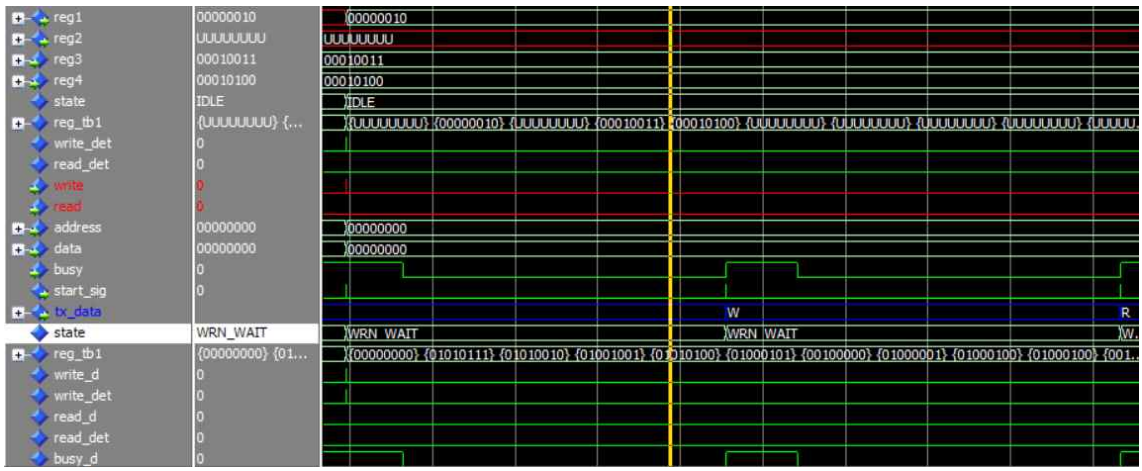

WRN_START
WRN_DEC
WRN_END
WRN_STOP
IDLE


/tb_uart4u_TB/data_paseng/RST
/tb_uart4u_TB/data_paseng/clk
/tb_uart4u_TB/data_paseng/in_data
/tb_uart4u_TB/data_paseng/valid
/tb_uart4u_TB/data_paseng/write_s
/tb_uart4u_TB/data_paseng/read_s
/tb_uart4u_TB/data_paseng/address
/tb_uart4u_TB/data_paseng/data
/tb_uart4u_TB/data_paseng/temp_data
/tb_uart4u_TB/data_paseng/data_0
/tb_uart4u_TB/data_paseng/data_1
/tb_uart4u_TB/data_paseng/data_2
/tb_uart4u_TB/data_paseng/data_3
/tb_uart4u_TB/data_paseng/data_4
/tb_uart4u_TB/data_paseng/data_5
/tb_uart4u_TB/data_paseng/data_6
/tb_uart4u_TB/data_paseng/data_7
/tb_uart4u_TB/data_paseng/data_8
/tb_uart4u_TB/data_paseng/data_9
/tb_uart4u_TB/data_paseng/data_10
/tb_uart4u_TB/data_paseng/data_11
/tb_uart4u_TB/data_paseng/valid_d
/tb_uart4u_TB/data_paseng/valid_det
/tb_uart4u_TB/data_paseng/valid_det_d
/tb_uart4u_TB/data_paseng/valid_det_d2
/tb_uart4u_TB/register_map/uart_write
/tb_uart4u_TB/register_map/uart_read
/tb_uart4u_TB/register_map/uart_address
/tb_uart4u_TB/register_map/uart_data
/tb_uart4u_TB/register_map/trans_write
/tb_uart4u_TB/register_map/trans_read
/tb_uart4u_TB/register_map/trans_address
/tb_uart4u_TB/register_map/trans_data
/tb_uart4u_TB/register_map/state
/tb_uart4u_TB/register_map/reg_tb1
/tb_uart4u_TB/register_map/write_d
/tb_uart4u_TB/register_map/read_d
IDLE
RDN_START
RDN_DEC
RDN_END
RDN_STOP
IDLE

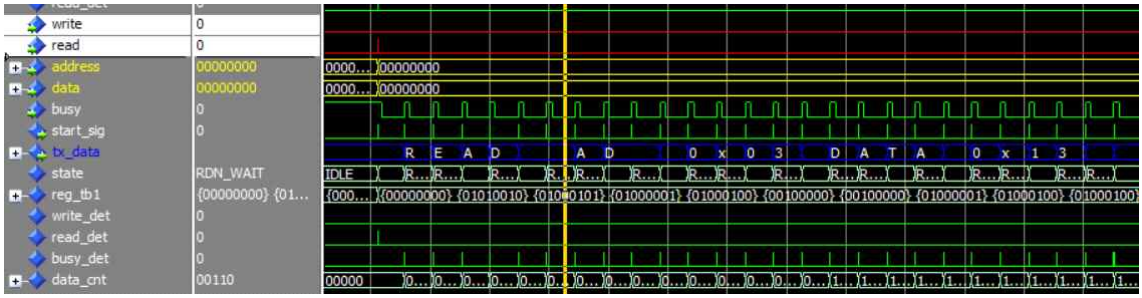
3)Register_Map to Load_Tx_Data



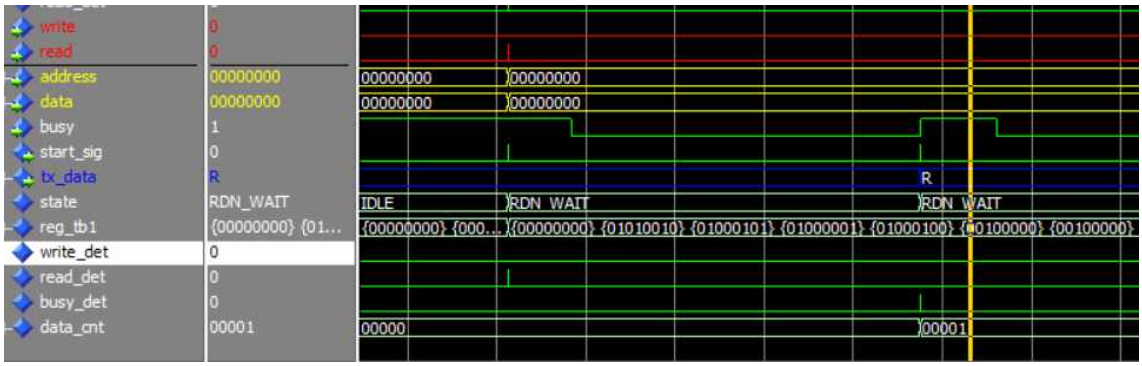
Load_Tx_Data로직의 write 과정이다. 1번지에 2라는 데이터를 쓴 후 pc로 확인 메시지를 전송하기 위한 것이다. 파란 줄을 주의 깊게 보면 WRITE ADD 0x01 0x02라는 값이 출력되는 것을 확인 할 수 있다. 각 문자를 보낼 때 마다 start_sig가 보내진다.



쓰기 작업을 할 때 한 문자를 보내는 과정을 확대해 놓은 것이다.



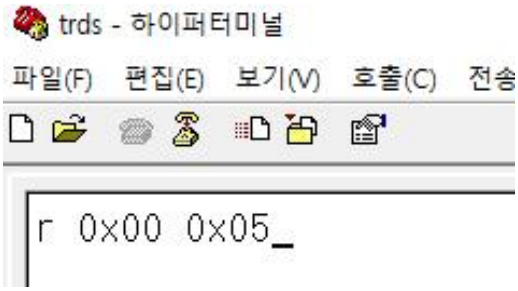
Load_Tx_Data로직의 read 과정이다. 3번지에 값을 pc로 확인 메시지를 전송하기 위한 것이다. 파란 줄을 주의깊게 보면 READ ADD 0x03 0x13라는 값이 출력되는 것을 확인 할 수 있다. 즉 3번지에는 13이라는 값이 있던 것이다. 각 문자를 보낼 때 마다 start_sig가 보내진다.

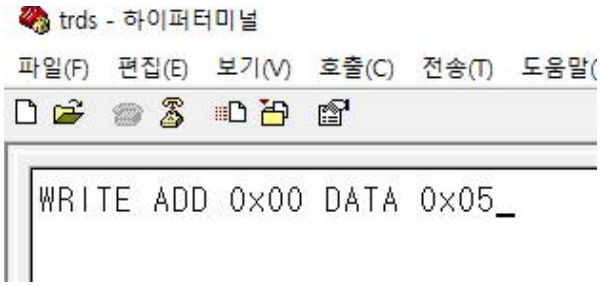


읽기 작업을 할 때 한 문자를 보내는 과정을 확대해 놓은 것이다.

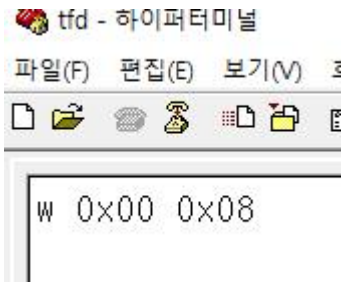
4. DE2 실습

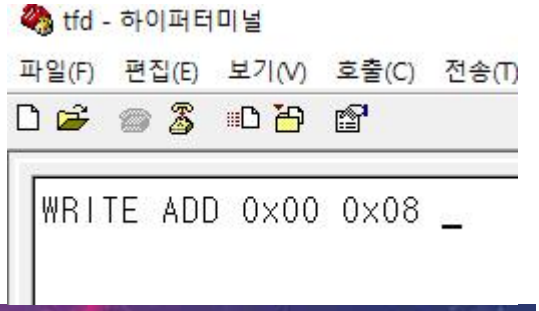
1)Reg0 = 0x00 번지 test



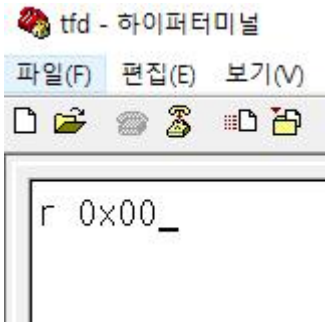


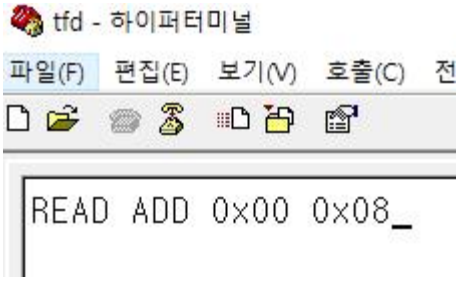
(0x00 번지에 0x05를 쓰고 led 켜짐 확인)





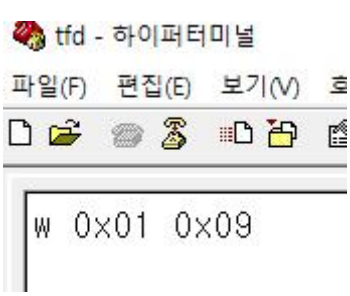
(0x00 번지에 0x08를 쓰고 led 켜짐 확인)

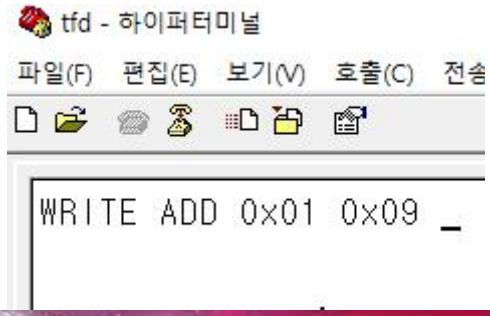


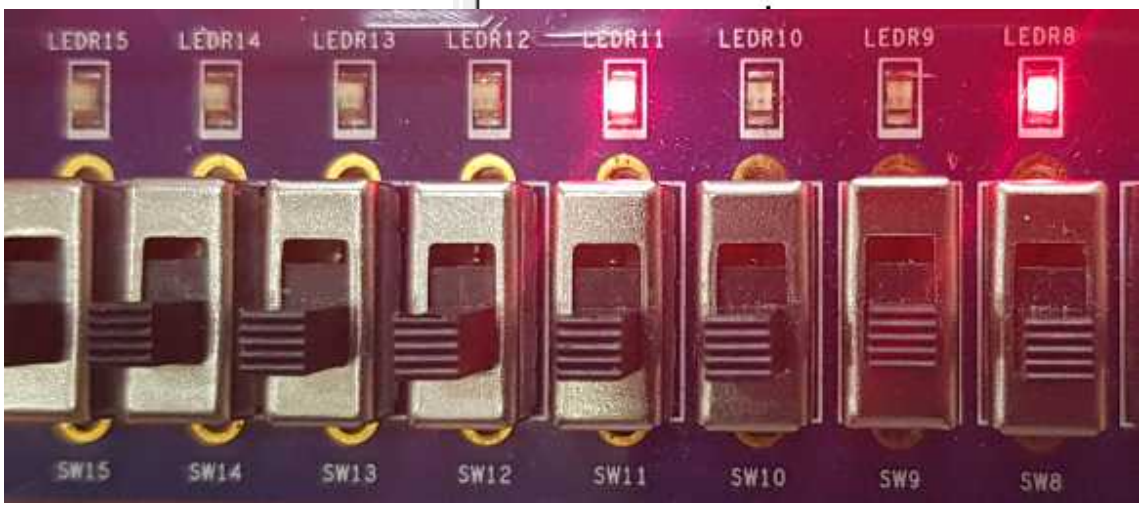


(0x00 번지에 있는 데이터 값을 읽어옴. 즉 방금 쓰인 0x08이 읽혀옴.)

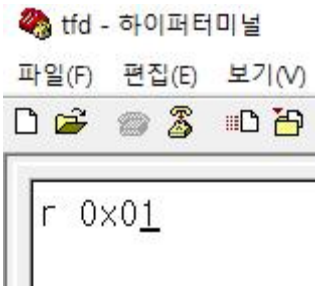
2)Reg1 = 0x01 번지 test

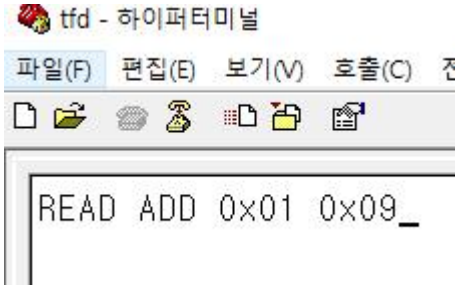






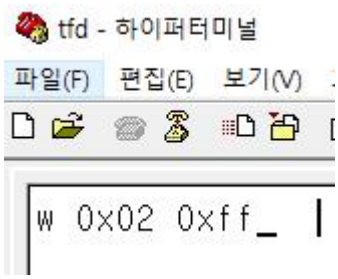
(0x01 번지에 0x09를 쓰고 led 켜짐 확인)

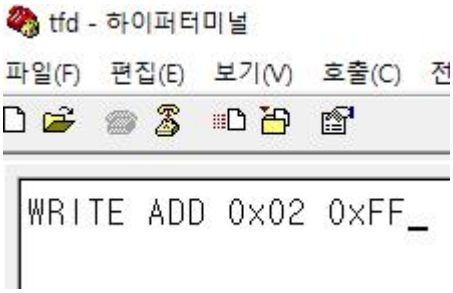




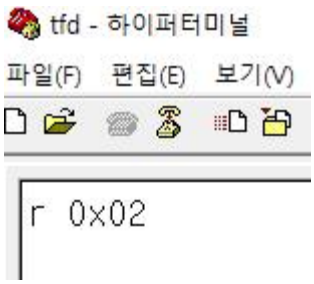
(0x01 번지에 있는 데이터 값을 읽어옴. 즉 방금 쓰인 0x09가 읽혀옴.)

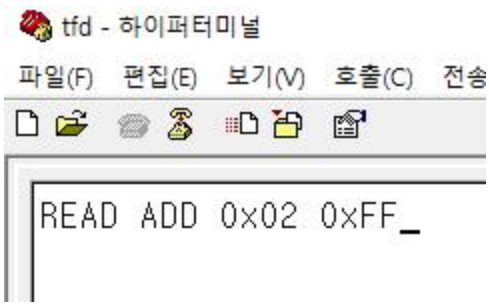
3)Reg2 = 0x02 번지 test





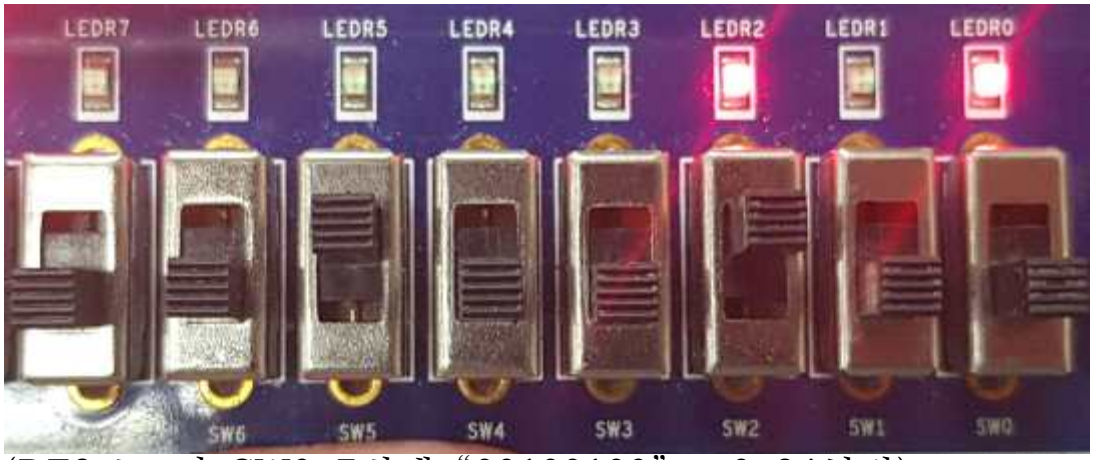
(0x02 번지에 0xFF를 쓰고 led 켜짐 확인)



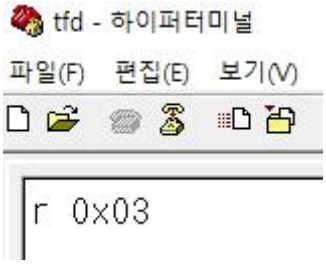


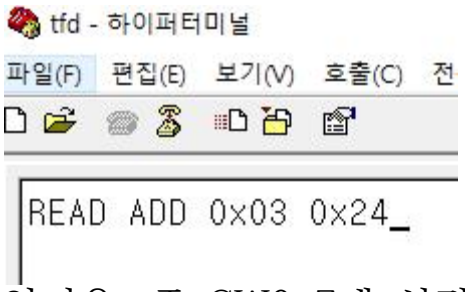
(0x02 번지에 있는 데이터 값을 읽어옴. 즉 방금 쓰인 0xFF가 읽혀옴.)

4)Reg3 = 0x03 번지 test



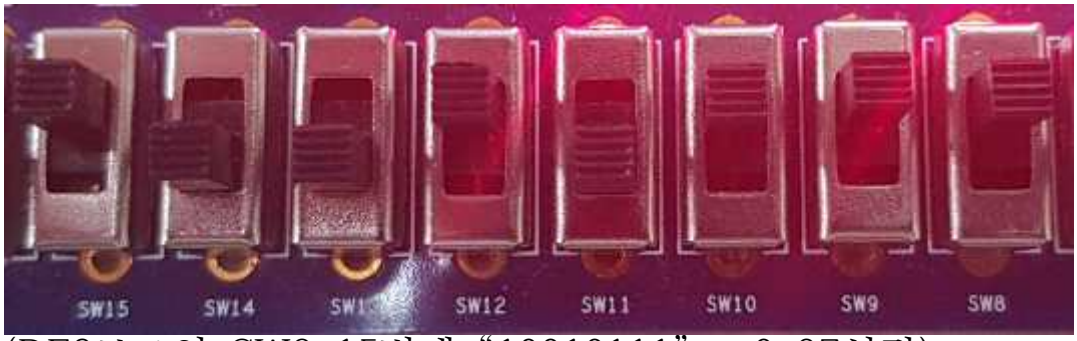
(DE2보드의 SW0~7번에 “00100100” = 0x24설정)



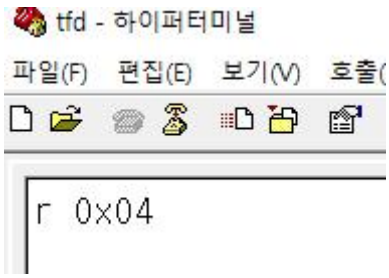


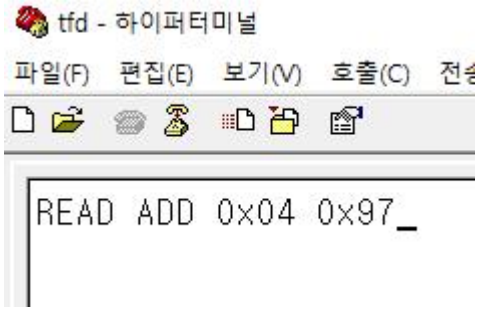
(0x03 번지에 있는 데이터 값을 읽어옴. 즉 SW0~7에 설정한 0x24가 읽혀옴)

5)Reg4 = 0x04 번지 test



(DE2보드의 SW8~15번에 “10010111” = 0x97설정)





(0x04 번지에 있는 데이터 값을 읽어옴. 즉 SW8~15에 설정한 0x97이 읽혀옴)

5. 토의 및 실습소감

교수님의 과제 평가를 보고 많은 것을 느꼈습니다. 메카트로닉스공학과에서 vhdl에 흥미를 느껴 전자과의 수업을 듣고 있습니다. 하지만 저희 학과에서 다루지 않는 프로그램과 설계방식에 어려움이 많아 점점 손을 놓게 되고 있습니다. 그런데 교수님의 과제 평가를 보니 제가 작성한 리포트들이 떠오르게 되었습니다. 정말로 제대로 실험하고 제출하였는가. 노력하였는가. 많은 생각을 하였고 그렇기에 이번 리포트는 최대한 노력을 하였습니다. 변명입니다만 이번학기에 실습과목이 6개나 되어 리포트에 소홀하였던 점. 부끄럽습니다. 이번 실습을 하면서 정말 어려운 과목이라는 것을 다시 느끼게 되었습니다. 이번실습 이후에 하나의 실습. 하나의 시험이 있지만 성적이 어떻게 나오든 아마 4학년때 재수강을 하지 않을 까 합니다. 실력은 없지만 모든 과목들 중에 가장 재미를 느끼는 과목이기 때문입니다. 학과 전공을 마무리 지어 놓고 이쪽 길을 걷고자 노력하지 않을까 싶습니다. 긴 글. 허접한 리포트 읽어주셔서 감사합니다.