SPI & I2C Design

With
AMBA AXI Protocol

DATE
2025년 05월 27일

[Harman Semicon Academy] 교육생 : 정현우

목차

Design Goals

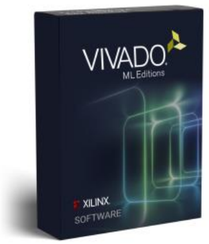
AXI4 Protocol 구조 및 동작 원리 이해

Xillinx Microblaze CPU 와 AXI, SPI,I2C 연동

Vitis IDE를 이용한 SW Application 구현

Systemverilog를 이용한 UVM 검증

Tools & Components

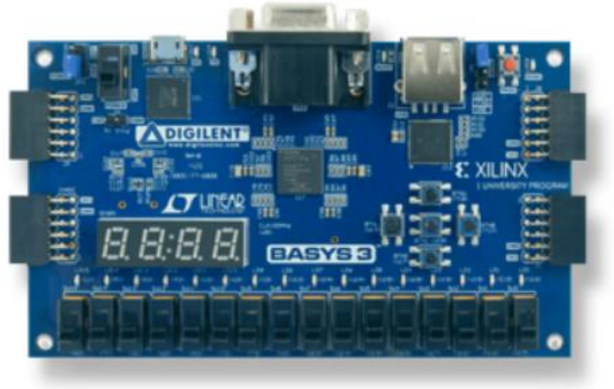


AMD Xillinx
Vivado 2020.2v

Xillinx Vitis 2020.2v

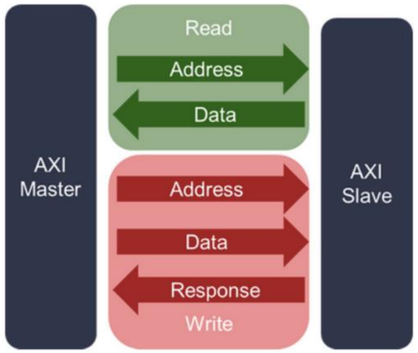
SystemVerilog

Synopsys
VCS



Basys3 FPGA Board

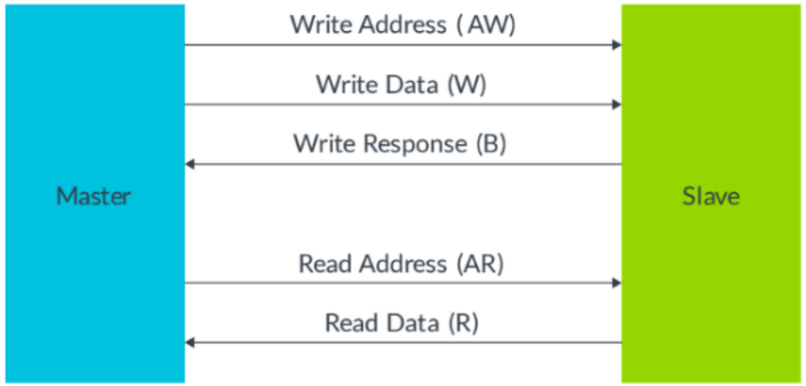
AXI4 Protocol



AXI Master & Slave

Advanced eXtensible Interface

ARM의 AMBA(Advanced Microcontroller Bus Architecture) 버스 프로토콜 중 하나로 고속 데이터 전송이 가능하며, 시스템 내 다양한 모듈 간 효율적인 통신을 위해 사용됨



< AXI CHANNELS>

Write Address Channel - AW
- 쓰기 요청의 주소 및 제어 정보를 전달

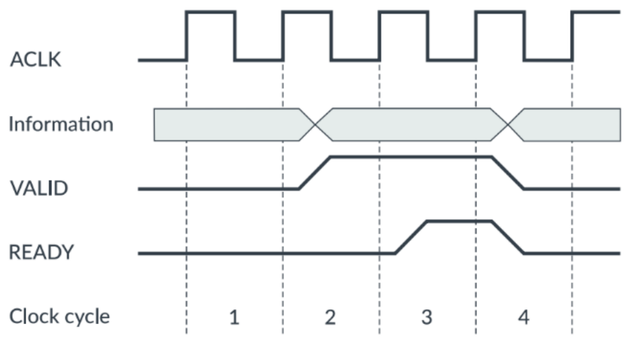
Write Data Channel - W
- 마스터가 슬레이브에게 실제 쓸 데이터를 전달

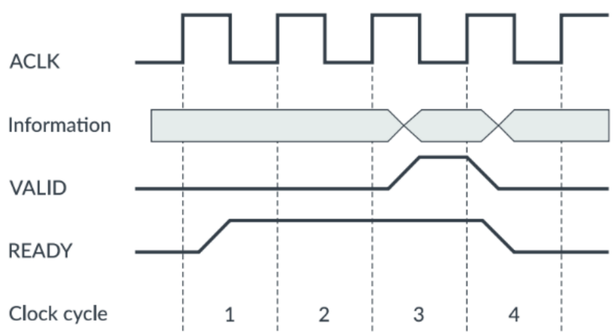
Write Response Channel - B
- 슬레이브가 마스터에게 쓰기 작업이 완료되었음을 알리고, 그 상태(성공/실패)를 응답

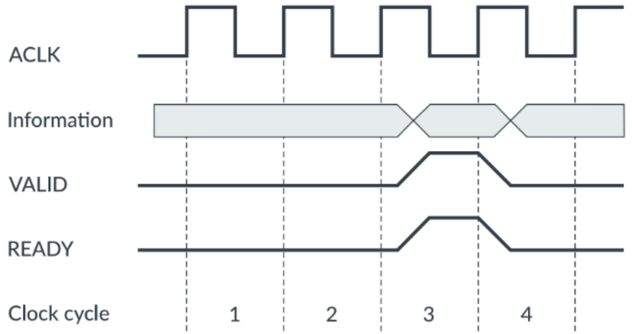
Read Address Channel - AR
- 읽기 요청의 주소 및 제어 정보를 전달

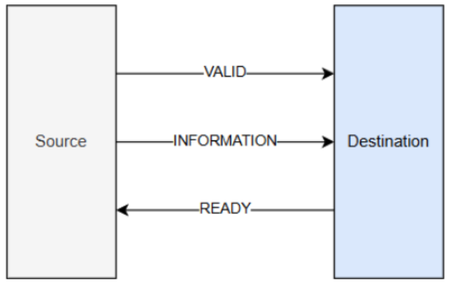
Read Data Channel - R
- 슬레이브가 마스터에게 요청받은 읽기 데이터를 전달하고, 데이터의 상태를 함께 전송

AXI4 Channel Transfer Hanshake









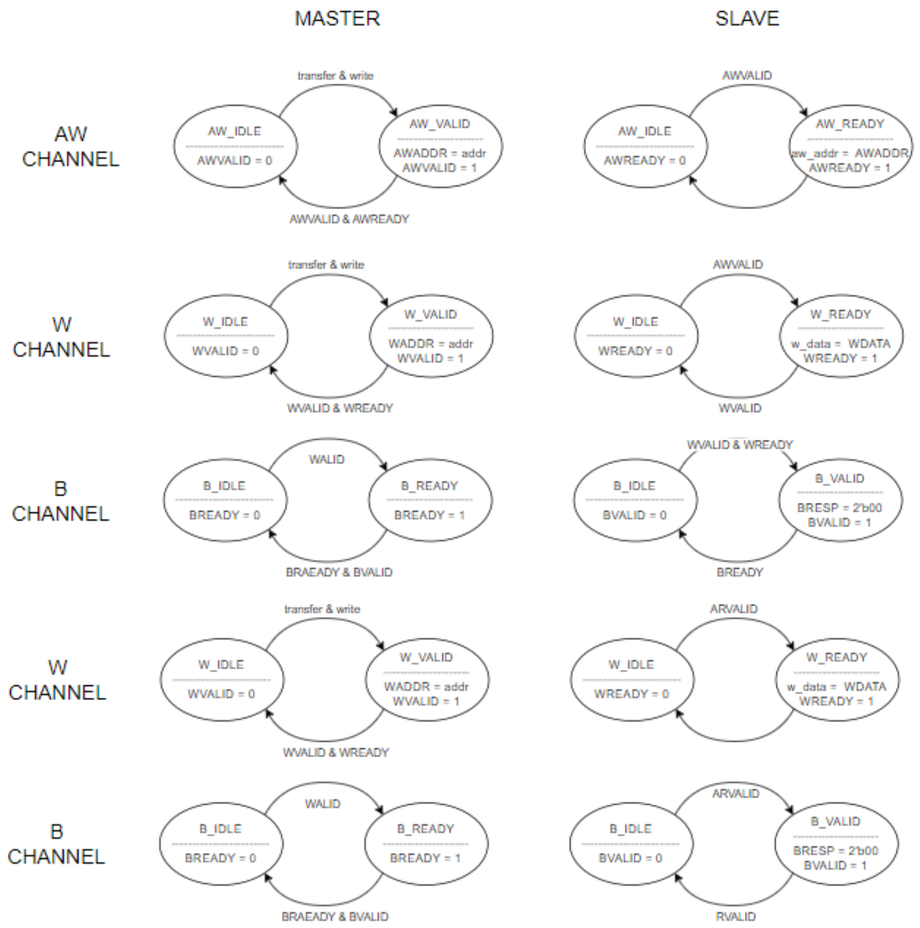
AXI에서 모든 채널의 데이터 전송은 VALID/READY Handshake 에 의해 동작
각 채널 마다 따로 모두 존재하며, 해당 채널의 데이터를 주고 받을 때 사용

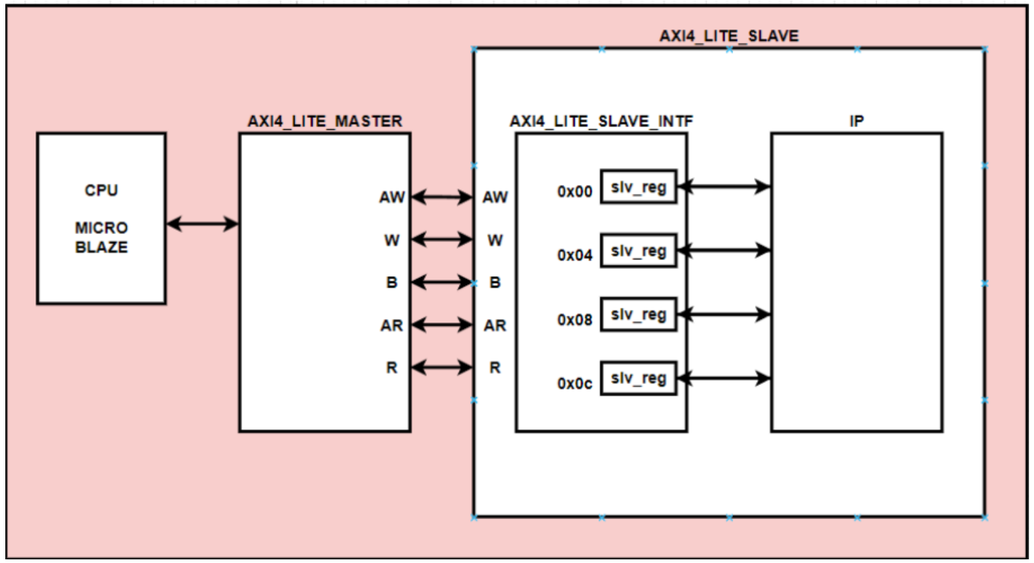
Valid : 송신 측에서 데이터가 유효함을 알리는 신호
Ready : 수신 측에서 데이터가 받을 준비가 됐음을 알리는 신호

공통적으로 ACLK의 상승 에지에서 Valid, Ready 신호 모두 HIGH 일 때 데이터 전송

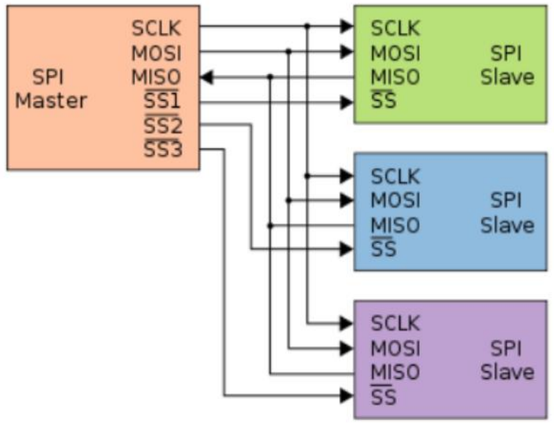
양 쪽에서 주고 받을 준비가 되었을 때 데이터를 전송함으로써 안정적인 데이터 전송이 가능

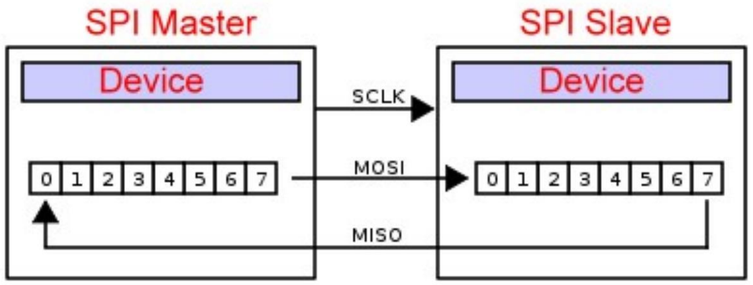
System Block Diagram





SPI (Serial Peripheral Interface)





SPI (Serial Peripheral Interface) 통신

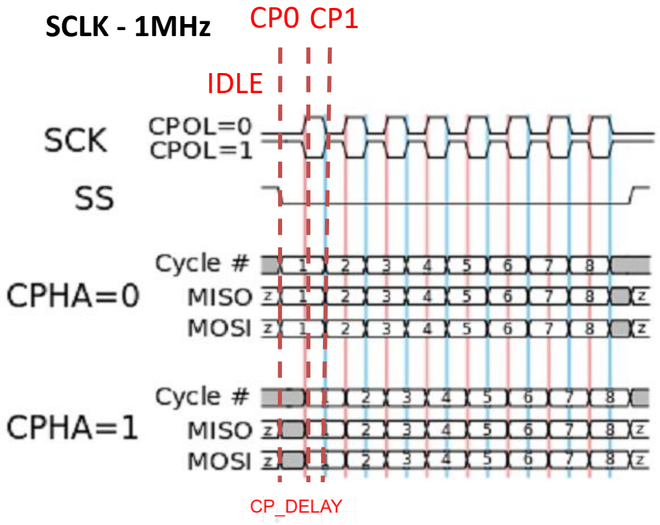
마이크로컨트롤러(MCU)나 프로세서가 주변 장치(센서, 메모리, 디스플레이 등)와 짧은 거리에서 고속으로 통신하기 위해 널리 사용되는 동기식 직렬 통신 프로토콜

- 마스터-슬레이브 (Master-Slave) 구조 , 1:N 연결 관계
- 적은 수의 신호선으로 구성되어 구현이 비교적 간단
- 전이중 (Full-duplex) 통신 방식
- 상대적으로 빠른 데이터 전송 속도를 제공

SPI 마스터와 슬레이브는 SCLK(클록)에 동기화되어 MOSI(마스터 출력)와 MISO(마스터 입력) 라인을 통해 동시에 데이터를 직렬로 주고받음

시프트 레지스터 형태로 비트를 밀어내서 전송하고 받음

SPI_Protocol



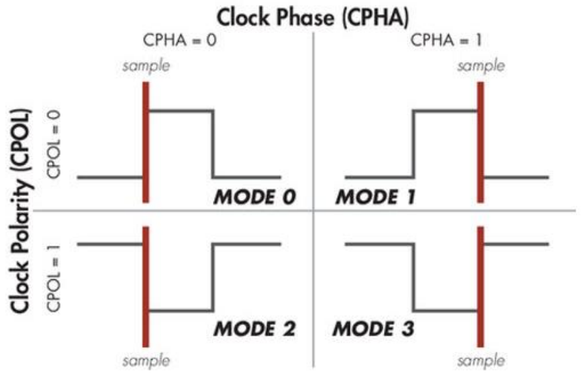


Figure 8. WRITE mode sequence

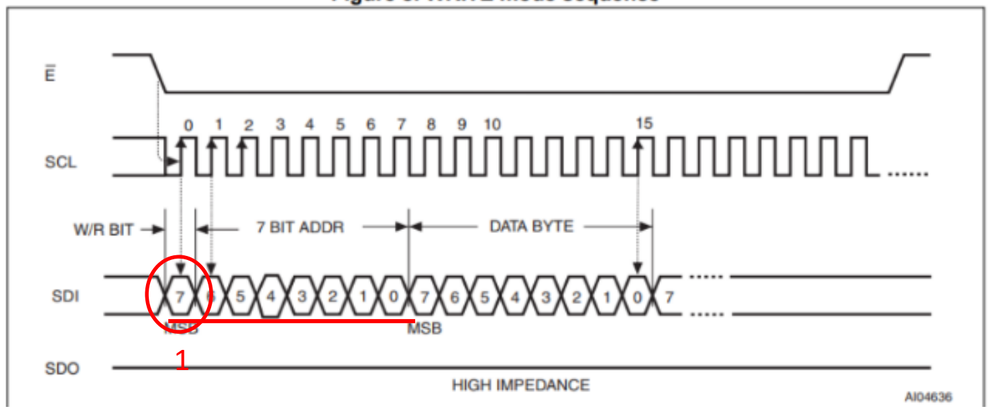
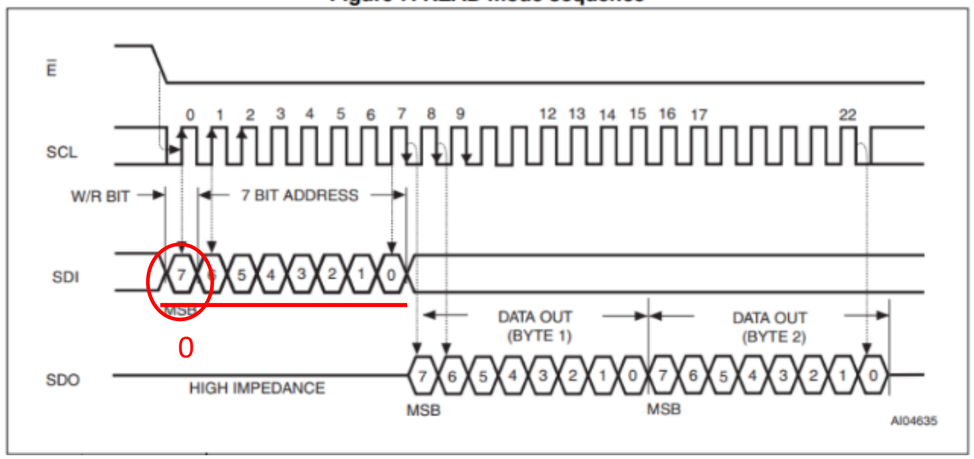


Figure 7. READ mode sequence



1. SS 가 LOW

2. r/w 모드와 주소가 포함된 8비트 신호 전송

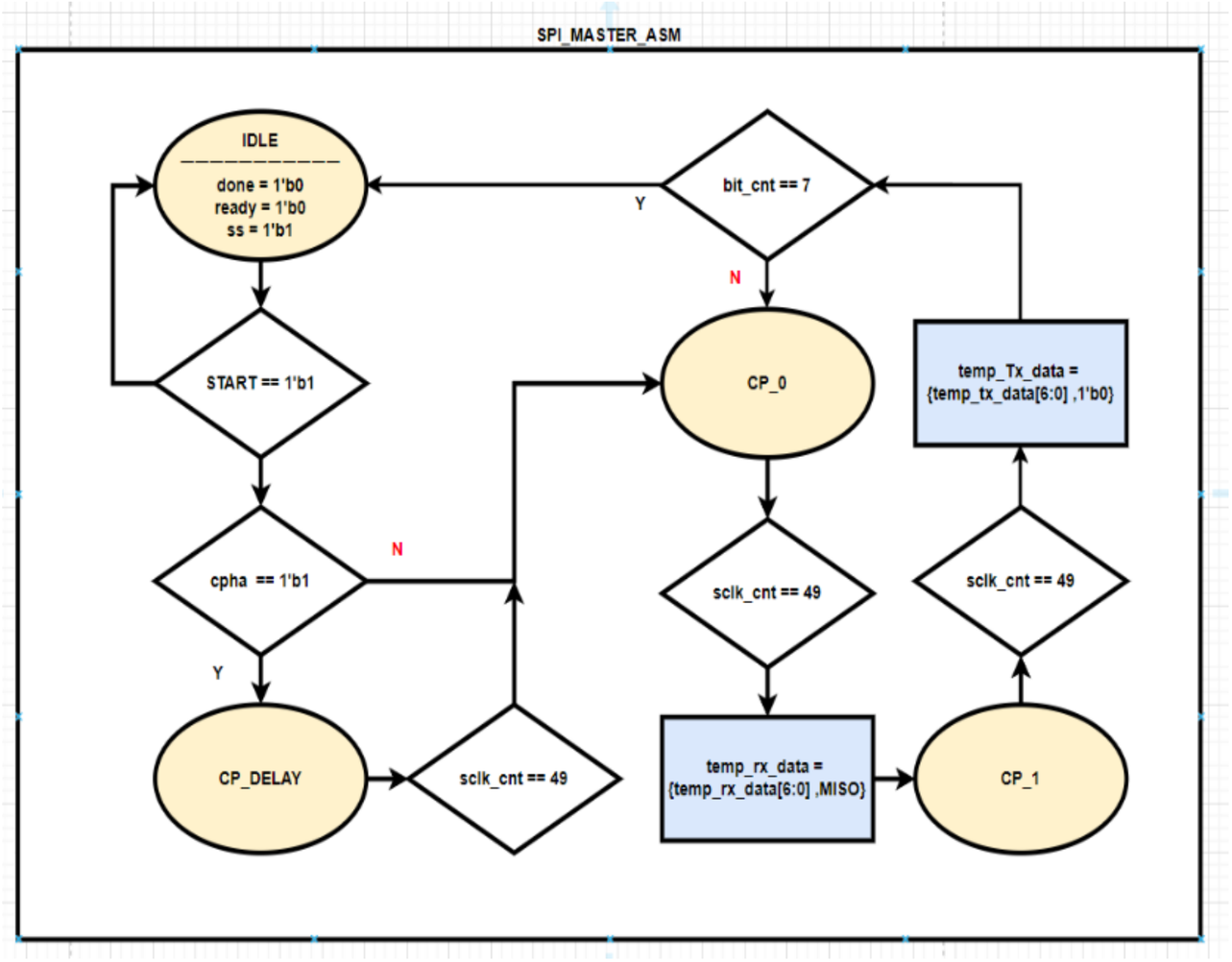
3. MSB가 1일 경우 WRITE
 0인 경우 READ

4. WRITE 의 경우

MOSI를 통해 마스터에서 슬레이브로 해당 주소에 쓸 데이터 바이트 전송

READ 경우
슬레이브에서 해당 주소 데이터를 MISO를 통해 출력

SPI_MASTER



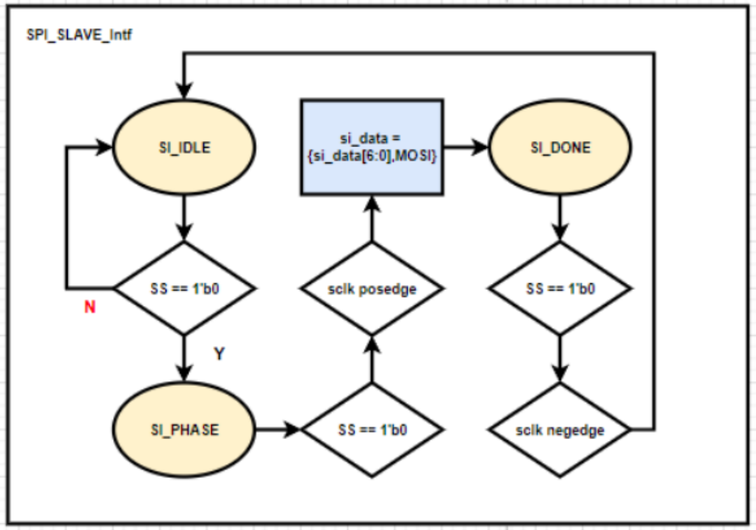
CPHA 값에 따라 IDLE에서 CP_DELAY 또는 CP0 상태로

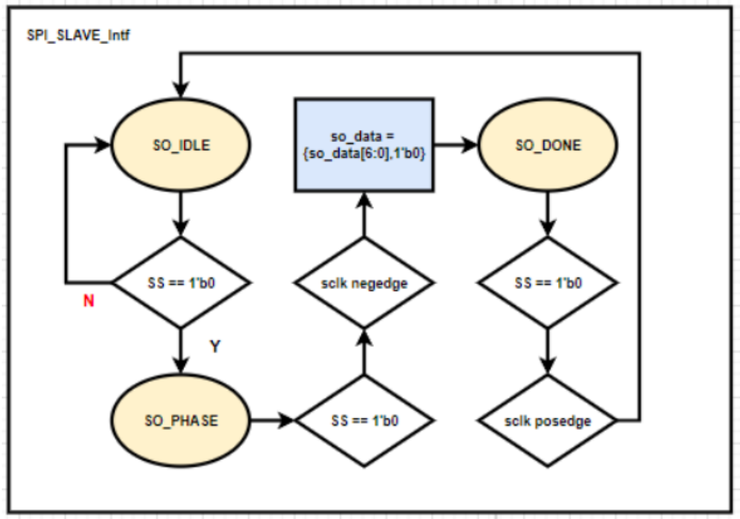
SCLK가 상승 엣지에서 MISO 라인 위의 데이터를 읽어(샘플링) 내부 시프트 레지스터에 저장

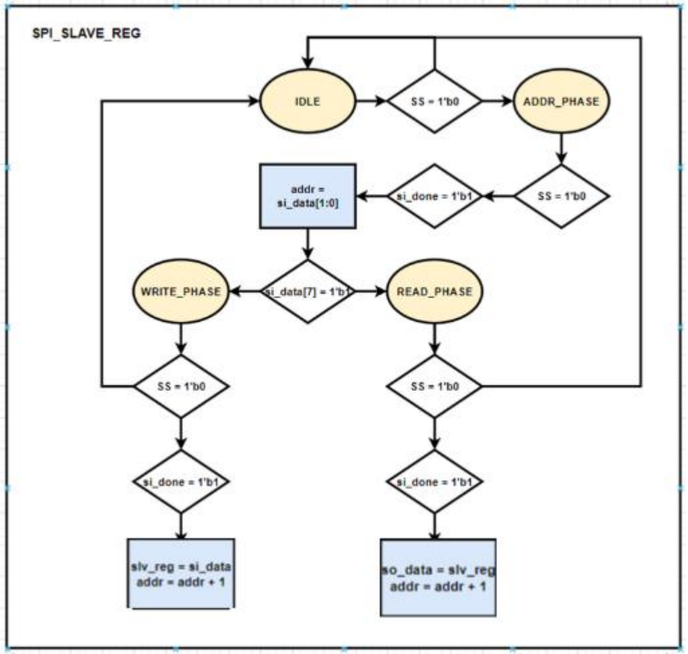
SCLK가 하강 엣지에서 내부 Shift-out 레지스터를 한 비트씩 → MOSI

8비트 데이터 수신 시 done 신호를 high로

SPI_SLAVE







SS(CHIP SELECT) 가 LOW가 되며 통신 시작

SCLK 상승 에지 마다 데이터(MOSI)를 수신, 8개 비트 수집

8비트 데이터 수신 시 데이터의 MSB 비트를 판별

1인 경우 WRITE , 0 인 경우 READ

8비트 데이터 수신 시 데이터의 MSB 비트를 판별

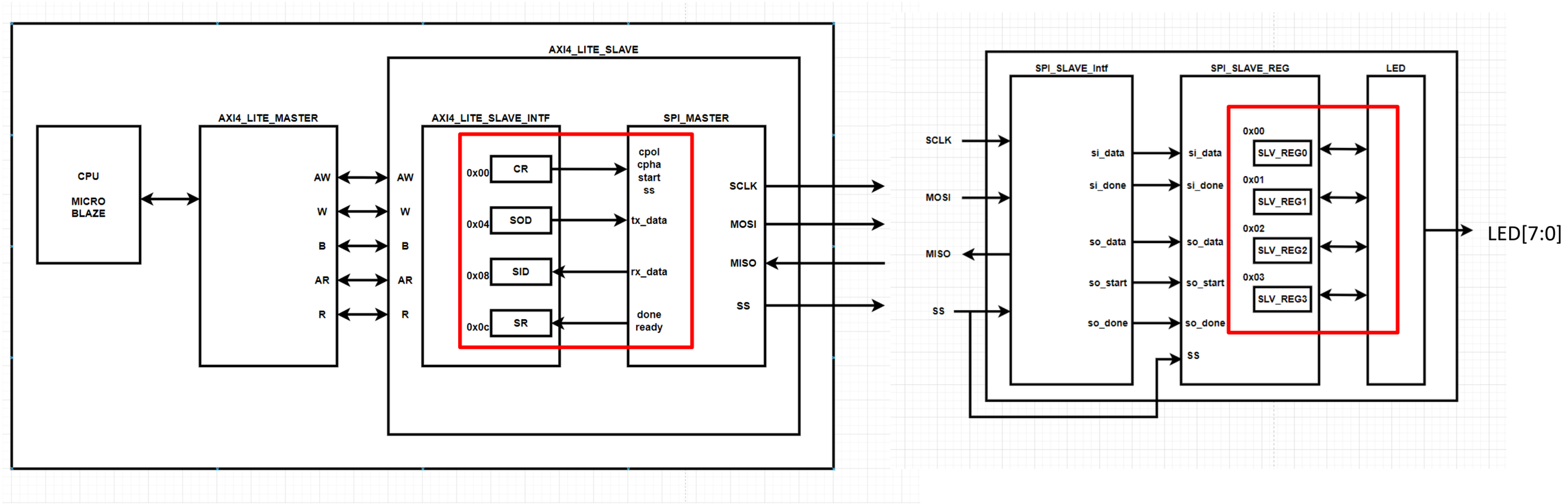
1인 경우 WRITE , 0 인 경우 READ

데이터 하위 두 비트로 레지스터 주소를 선택

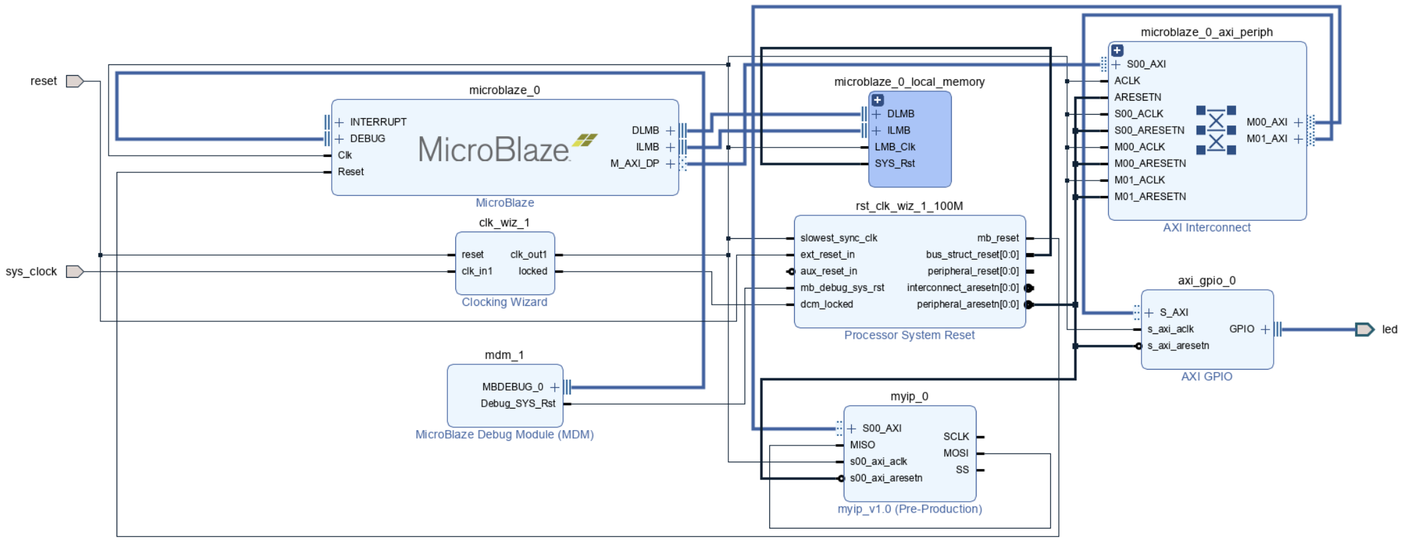
WRITE 인 경우 MOSI를 통해 들어오는 데이터 내부 레지스터 저장

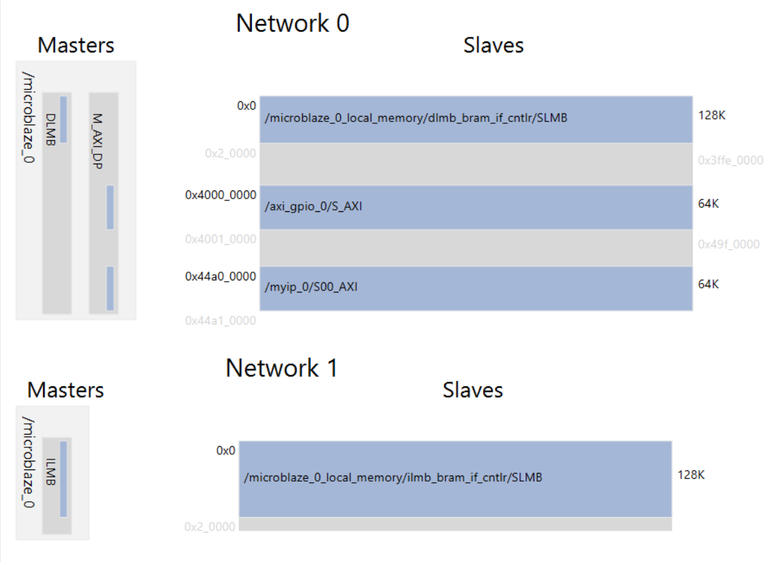
READ 인 경우 MISO를 통해 내부 레지스터에 있는 값 내보냄

AXI_SPI_BlockDiagram



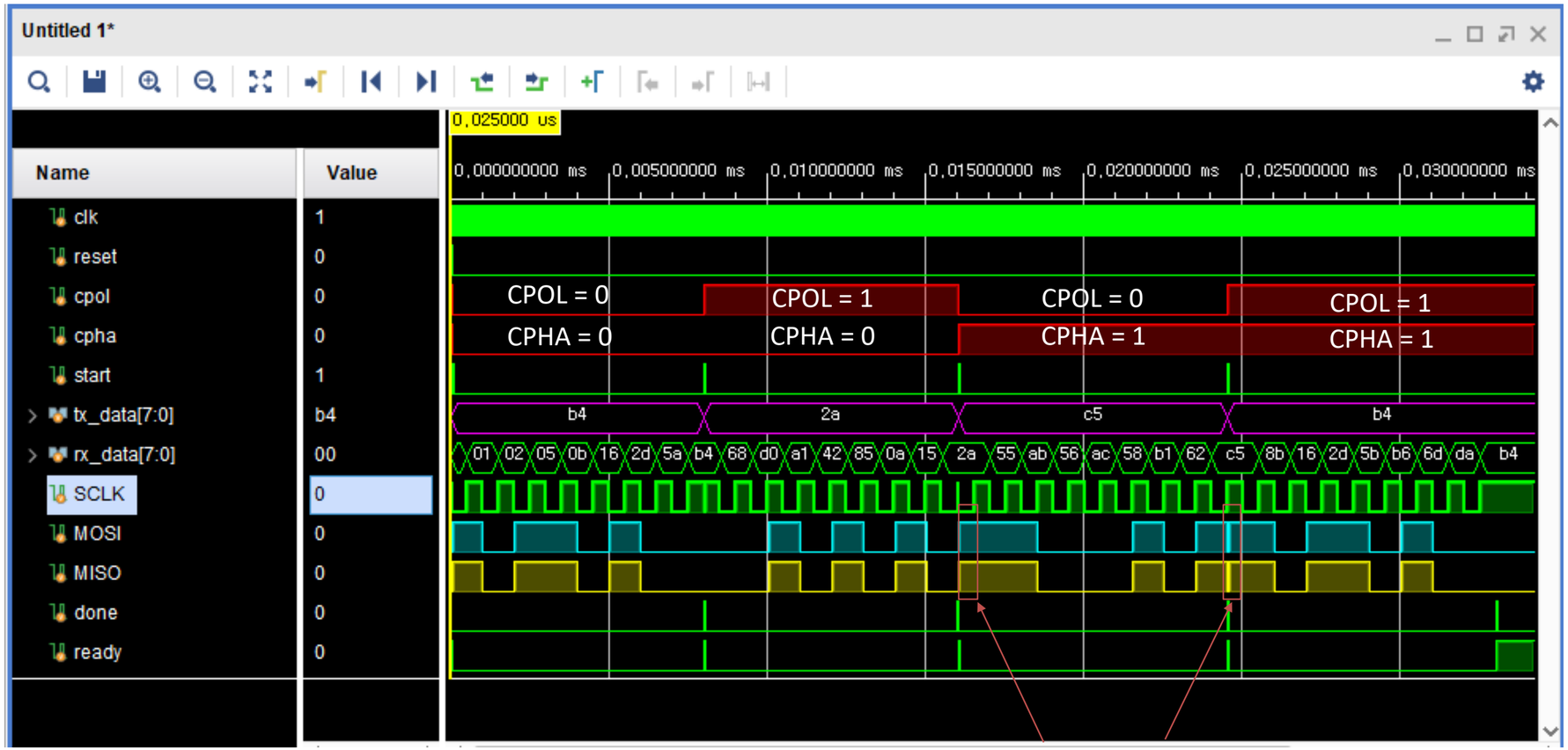
AXI4- SPI Schemetic & Memory Map





SPI_Verification

< SPI_MASTER >

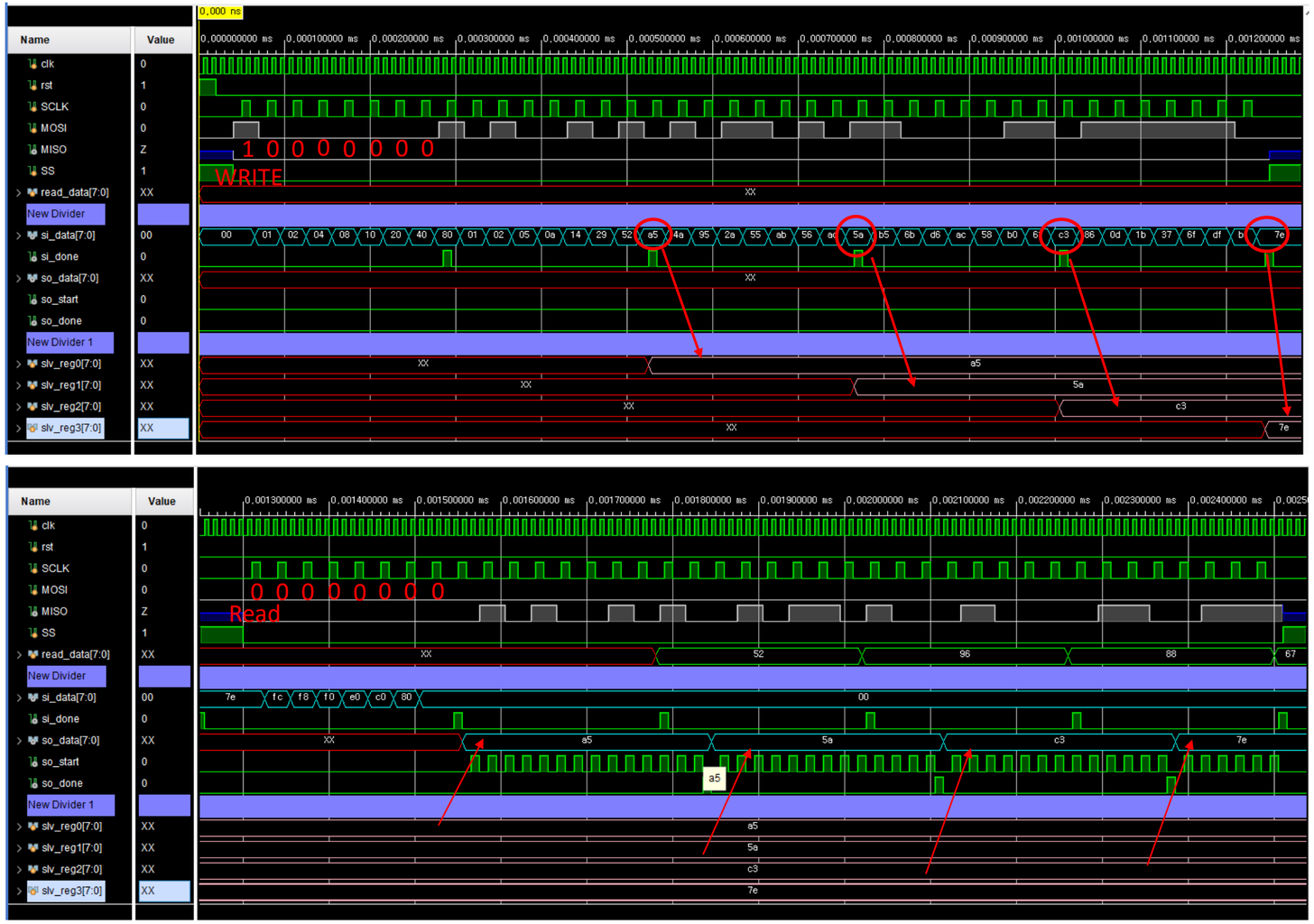


CPHA = 1 일때 반 클럭 DELAY

SPI_Verification

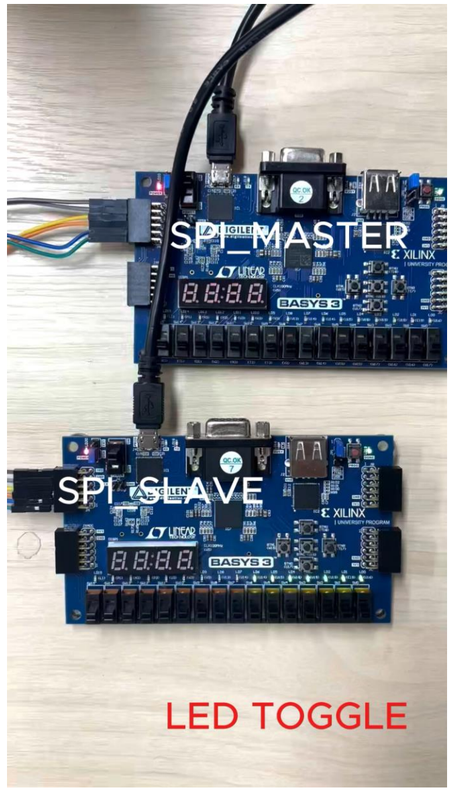
7`b0

< SPI_SLAVE >



SPI Application Video

```
typedef struct {
__IO uint32_t CR;      // 0x00: {CPOL, CPHA, START, SS}
__IO uint32_t SOD;     // 0x04: Master → Slave
__IO uint32_t SID;     // 0x08: Slave → Master
__IO uint32_t SR;      // 0x0C: 상태 레지스터 (미사용 가능)
} SPI_TypeDef;
#define SPI_BASEADDR   0x44A00000
#define SPI            ((SPI_TypeDef *) SPI_BASEADDR)
#define CR_SS     (1U << 0)
#define CR_START  (1U << 1)
#define CR_CPHA   (1U << 2)
#define CR_CPOL   (1U << 3)
void SPI_MODE(SPI_TypeDef *spi, uint32_t mode);
void SPI_SS_Enable(SPI_TypeDef *spi);
void SPI_SS_Disable(SPI_TypeDef *spi);
void SPI_WRITE(SPI_TypeDef *spi, uint32_t tx_data);
uint32_t SPI_READ(SPI_TypeDef *spi);
// 메인 함수
int main(void) {
uint32_t led = 0xFF;
SPI_MODE(SPI, 0);
while (1) {
    SPI_SS_Enable(SPI);
    SPI_WRITE(SPI, 0x80);
    SPI_WRITE(SPI, led);
    SPI_SS_Disable(SPI);

    led ^= 0xFF;
    usleep(500000);
}

return 0;
Use code with caution.
}
void SPI_MODE(SPI_TypeDef *spi, uint32_t mode) {
uint32_t cr = 0;
if (mode & 0x02) cr |= CR_CPHA;
if (mode & 0x03) cr |= CR_CPOL;
spi->CR = cr;
}
void SPI_SS_Enable(SPI_TypeDef *spi) {
spi->CR &= ~CR_SS;
```



I2C (Inter - Integrated Circuit)

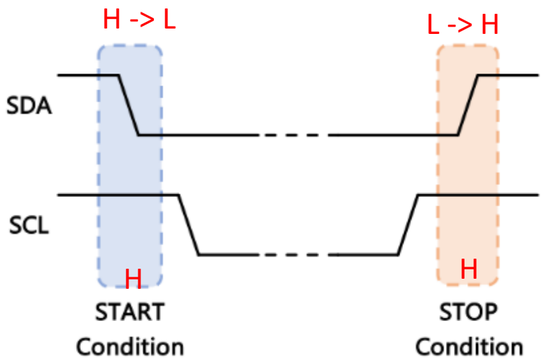
Mode[11]	Maximum speed
Standard mode (Sm)	100 kbit/s
Fast mode (Fm)	400 kbit/s
Fast mode plus (Fm+)	1 Mbit/s
High-speed mode (Hs)	1.7 Mbit/s
High-speed mode (Hs)	3.4 Mbit/s
Ultra-fast mode (UFm)	5 Mbit/s

I2C (Inter-Integrated Circuit) 통신

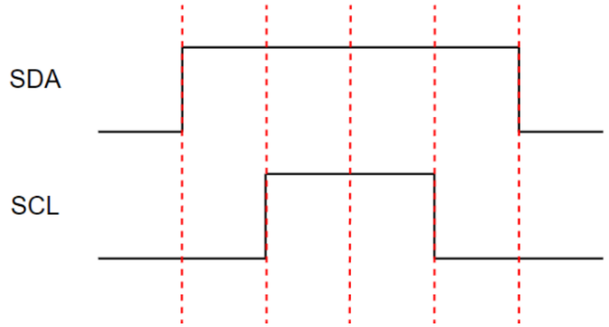
필립스(현재 NXP 반도체)에서 개발한 동기식 직렬 통신 프로토콜

- SDA, SCL 로 구성된 2-wire bus 구조
- SPI, UART에 비해선 느린 속도
- 1:N구조의 멀티 슬레이브 지원
- 짧은 거리 통신에 주로 사용
- SDA와 SCL 라인은 오픈 드레인(Open-drain) 또는 오픈 컬렉터 (Open-collector) 방식으로 구동 , 풀업 저항 필요
- 이번 설계에서는 STANDARD MODE (100KBps) 기준으로 설계

I2C_Protocol

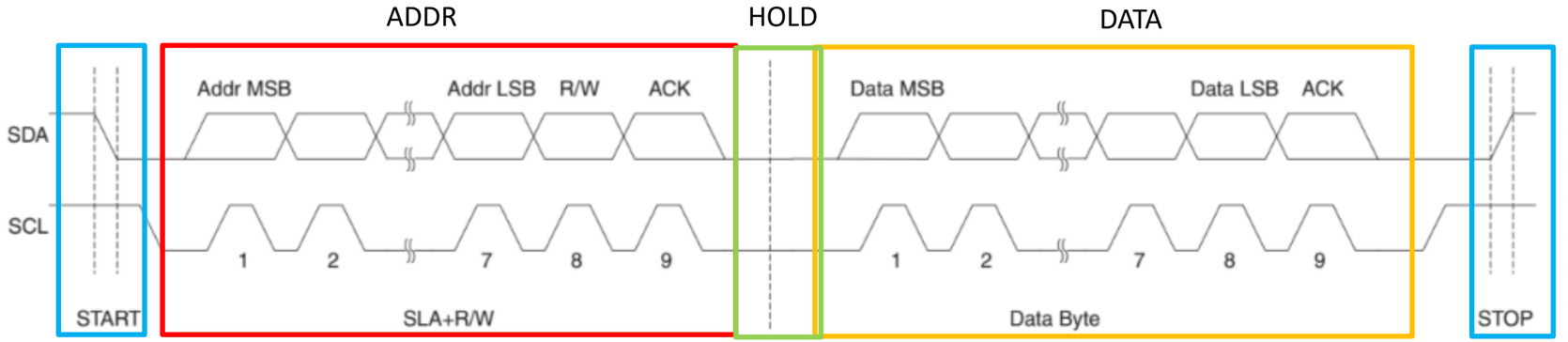


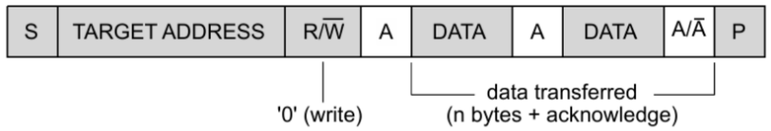
SCL이 HIGH 상태일 때,
SDA 이 HIGH에서 LOW 또는 LOW에서 HIGH 상태로 변하는 지에 따라 START, STOP 판단

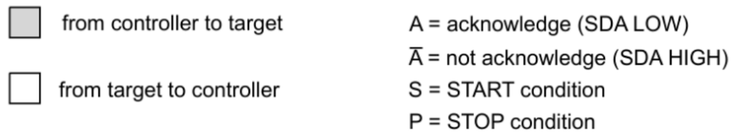


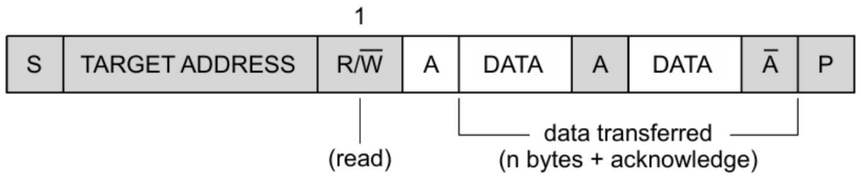
SCL 라인이 HIGH 상태일 때, SDA 라인이 HIGH 유지하면 데이터 '1'로 ,LOW 유지 시 데이터 '0' 판단

I2C_Protocol









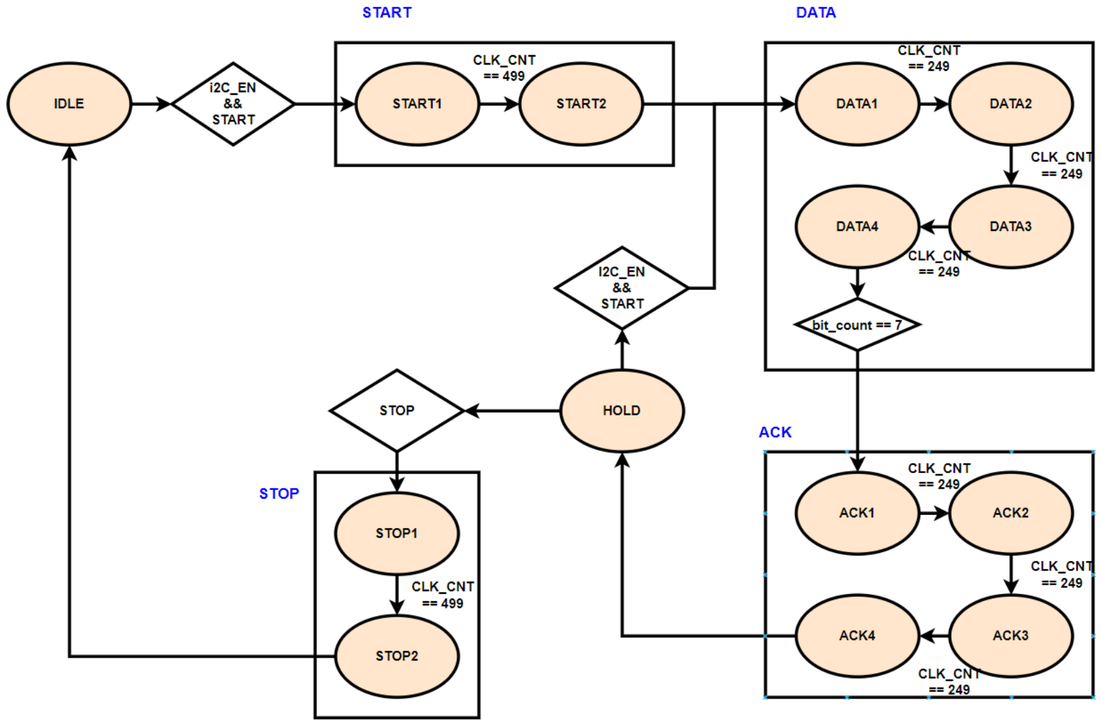
첫 타겟 주소 바이트 전송을 통해 주소와 쓰기를 할지 읽기를 할 지 명령(LSB)을 전달하고 데이터를 전송하고, 슬레이브는 각 데이터 수신 후 ACK로 응답

쓰기의 경우 슬레이브로부터 ACK/NACK를 받은 후 쓸 데이터를 전송 후 다시 슬레이브로부터 ACK/NACK를 받는 방식

읽기의 경우 슬레이브로부터 데이터를 받고 마스터가 데이터를 받았는지 ACK/NACK의 응답 신호를 전송

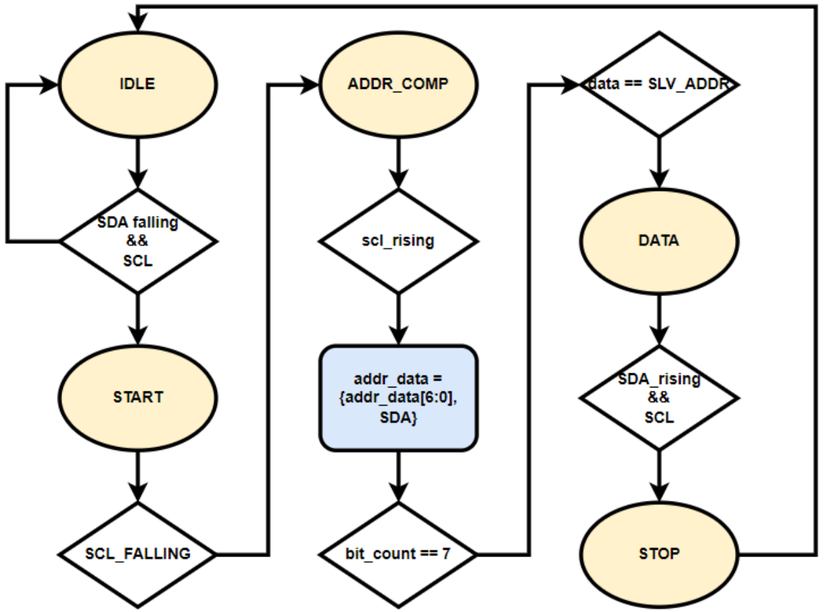
I2C ASM Chart

< I2C MASTER >



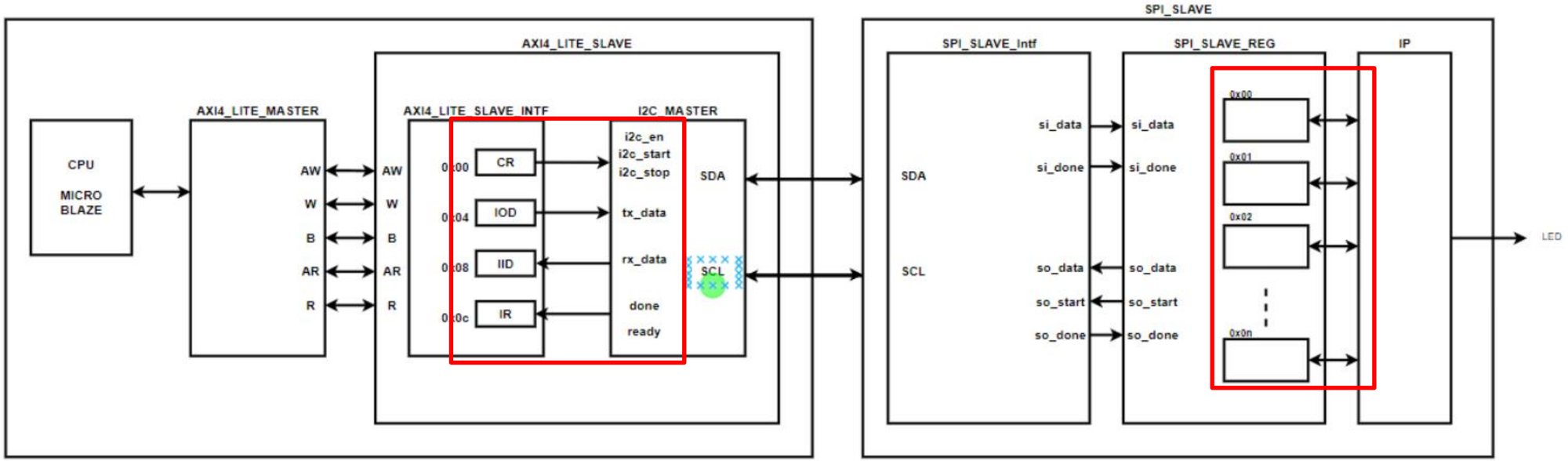
START CONDITION -> ADDR DATA -> ACK ->
HOLD 상태에서 들어오는 신호에 따라 판별

< I2C SLAVE >

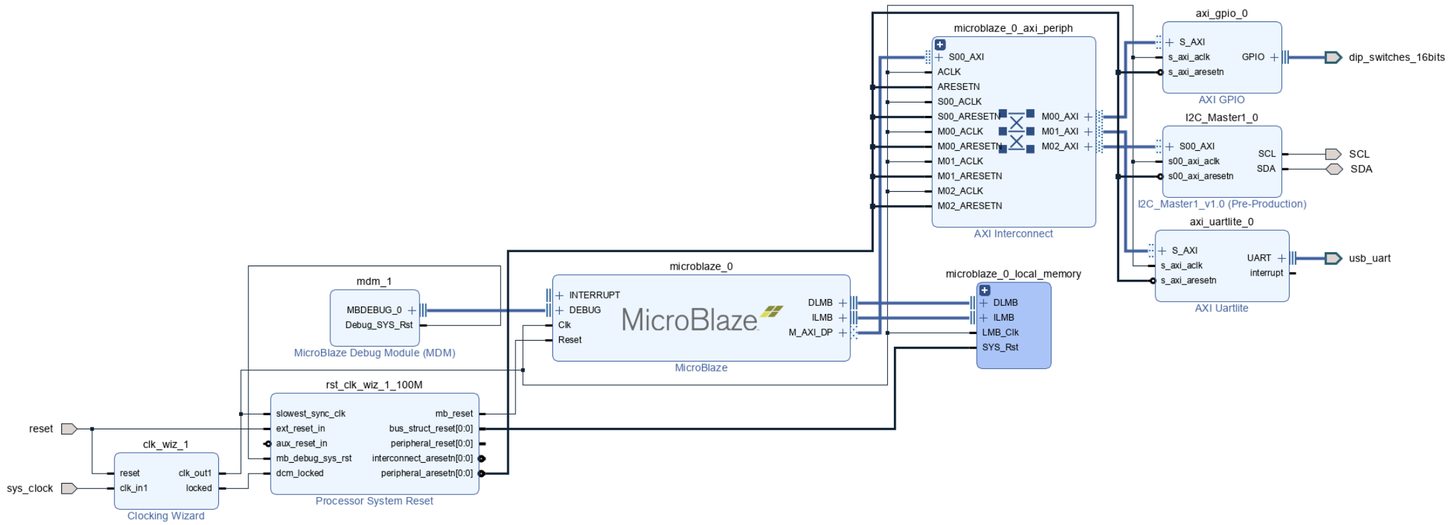


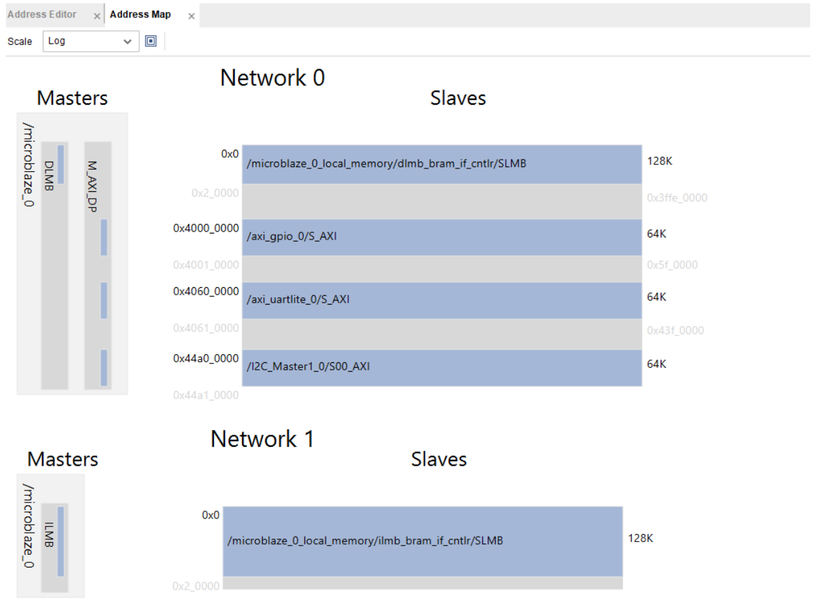
START -> ADDR 데이터 수집 후 슬레이브 주소와
비교 -> DATA 수집 -> STOP

AXI_I2C_BlockDiagram

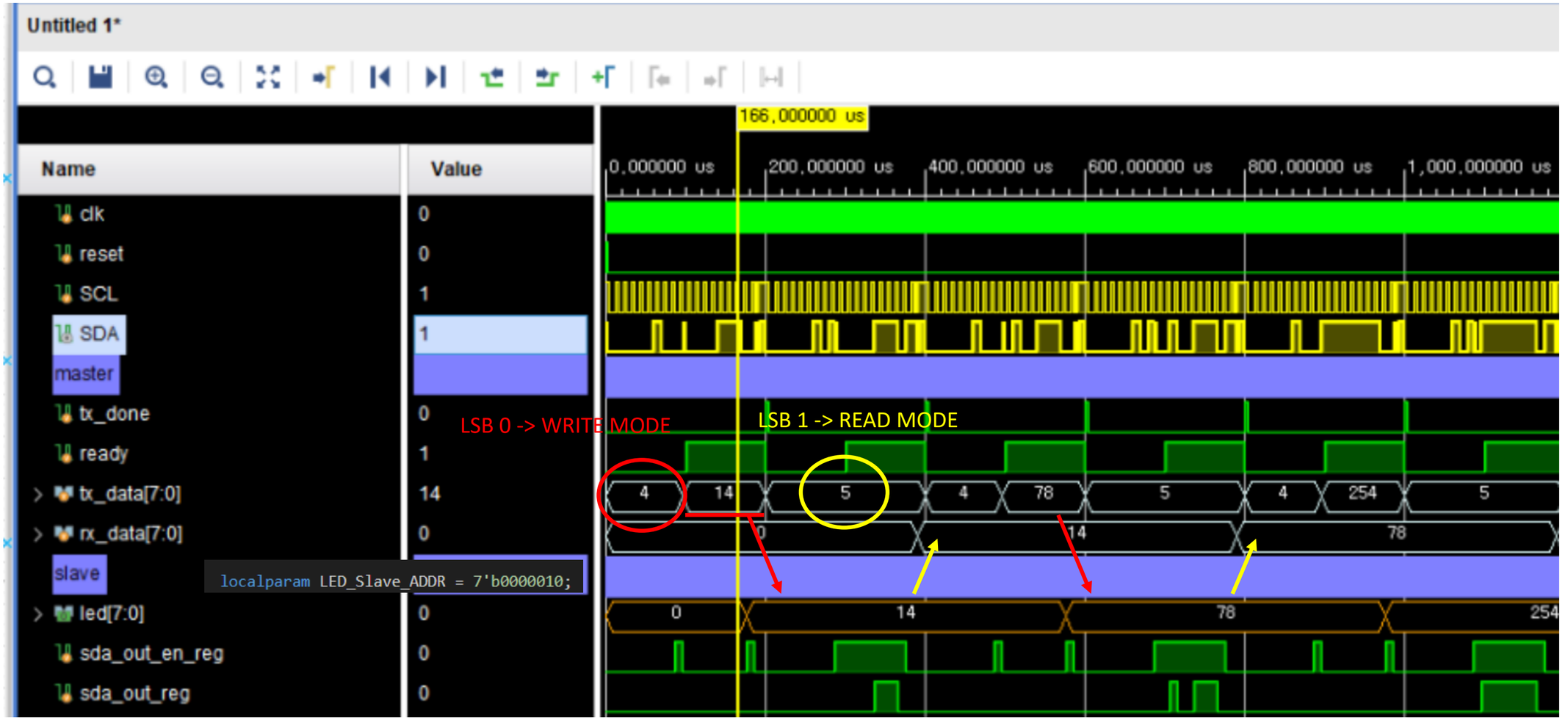


AXI-I2C Schemetic & Memory Map



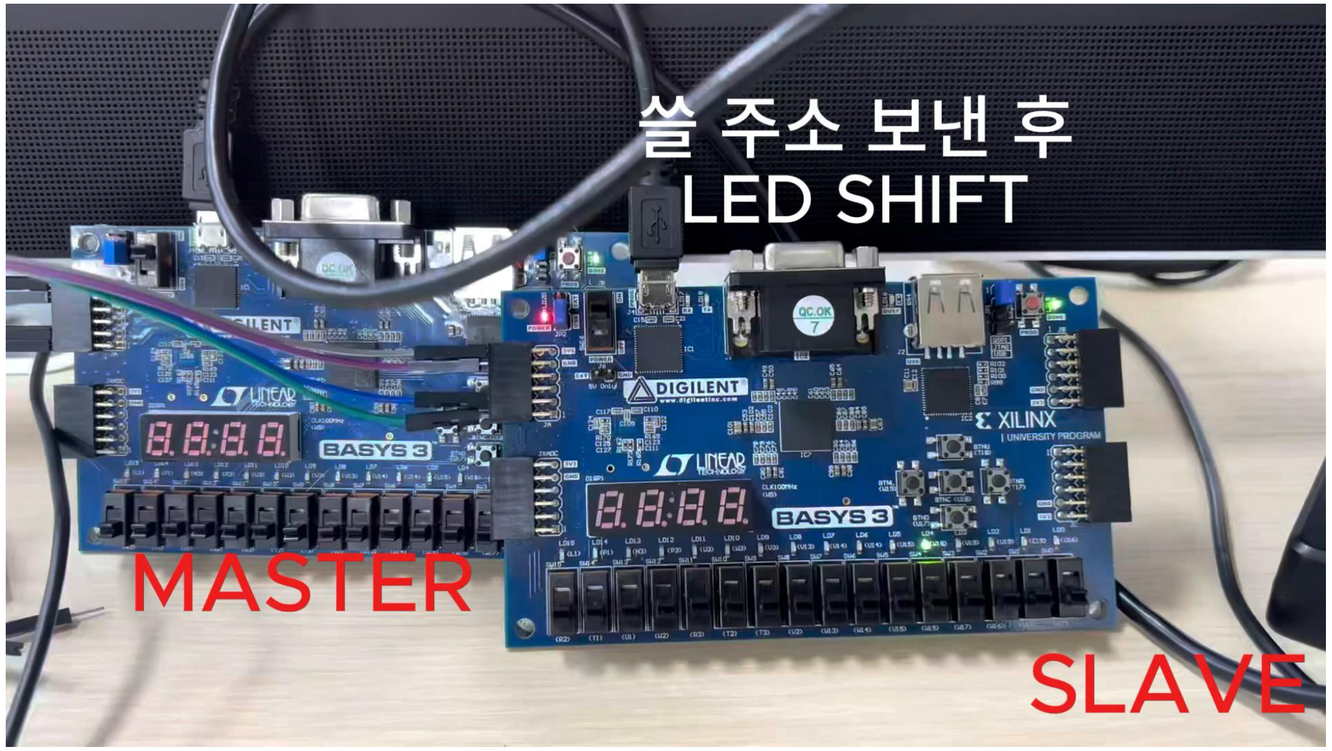


I2C_Verification

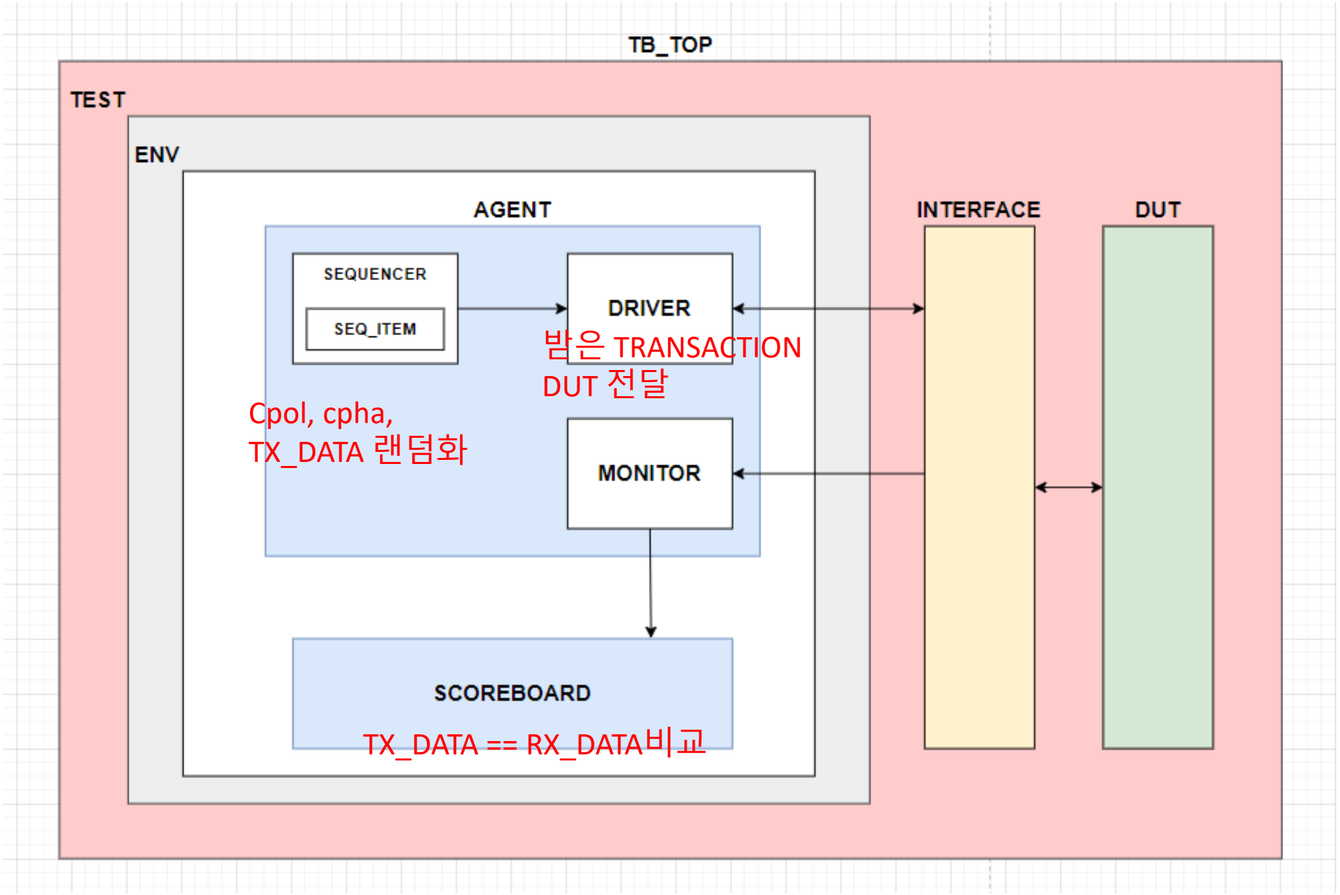


I2C Application Video

```c
typedef struct {
    volatile uint32_t CR;
    volatile uint32_t SOD;
    volatile uint32_t SID;
    volatile uint32_t SR;
} I2C_TypeDef;

#define I2C_BASEADDR (0x44A00000U)
#define I2C     ((I2C_TypeDef *)(I2C_BASEADDR))

uint32_t I2C_ON;

int main(){
    init_platform();
    while(1){

        //write to slave
        I2C -> CR = 0x03; // protocol start signal
        usleep(100000);
        I2C -> ODR = 0x34;  // 52(ADDR) + 0(WRITE)
        usleep(100000);
        while ( !((I2C->SR) & 0x02) ) { }//wait read signal
        I2C -> SOD = 2;
        usleep(100000);
        I2C -> CR = 0;
        usleep(100000);
        while ( !((I2C->SR) & 0x01) ) { }//wait tx_done signal
        usleep(100000);

        //write to slave
        I2C -> CR = 0x03;
        usleep(100000);
        I2C -> SOD = 0x34;
        usleep(100000);
        while ( !((I2C->SR) & 2) ) { }//ready 신호 대기
        I2C -> SOD = 0x04;
        usleep(100000);
        I2C -> CR = 0;
        usleep(100000);
        while ( !((I2C->SR) & 1) ) { }//tx_done 신호 대기
        usleep(100000);
```



UVM Verification




```systemverilog
// 2) Sequence Item
class spi_seq_item extends uvm_sequence_item;
    rand bit cpol;
    rand bit cpha;
    rand bit [7:0] tx_data;
    bit [7:0] rx_data;

    function new(string name = "SPI_ITEM");
        super.new(name);
    endfunction

    `uvm_object_utils_begin(spi_seq_item)
        `uvm_field_int(cpol, UVM_DEFAULT)
        `uvm_field_int(cpha, UVM_DEFAULT)
        `uvm_field_int(tx_data, UVM_DEFAULT)
        `uvm_field_int(rx_data, UVM_DEFAULT)
    `uvm_object_utils_end
endclass : spi_seq_item
```

CPOL, CPHA
TX_DATA
랜덤 값 생성

```systemverilog
virtual function void write(spi_seq_item item);
    //spi_item = item;
    total_transactions++;
    `uvm_info("SCO", $sformatf("Checking: TX=0x%0h RX=0x%0h", item.tx_data, item.rx_data), UVM_LOW);
    if (item.rx_data == item.tx_data) begin
        `uvm_info("SCO", "*** PASS ***", UVM_NONE);
        pass_count++;
    end
    else begin
        `uvm_error("SCO", $sformatf("*** FAIL: expected 0x%0h got 0x%0h ***", item.tx_data, item.rx_dat
        fail_count++;
    end
endfunction
```

SCOREBOARD

DUT를 거쳐 출력된 RX_DATA와 TX DATA 비교

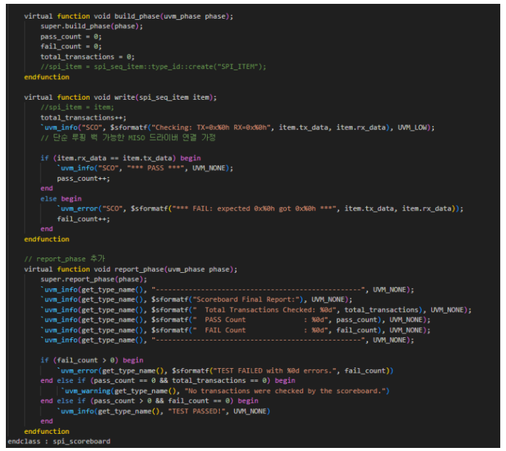
```systemverilog
virtual function void report_phase(uvm_phase phase);
    super.report_phase(phase);
    `uvm_info(get_type_name(), "------------------------------------------------", UVM_NONE);
    `uvm_info(get_type_name(), $sformatf("Scoreboard Final Report:"), UVM_NONE);
    `uvm_info(get_type_name(), $sformatf("  Total Transactions Checked: %0d", total_transactions), UVM_NONE);
    `uvm_info(get_type_name(), $sformatf("  PASS Count                 : %0d", pass_count), UVM_NONE);
    `uvm_info(get_type_name(), $sformatf("  FAIL Count                 : %0d", fail_count), UVM_NONE);
    `uvm_info(get_type_name(), "------------------------------------------------", UVM_NONE);

    if (fail_count > 0) begin
        `uvm_error(get_type_name(), $sformatf("TEST FAILED with %0d errors.", fail_count))
    end else if (pass_count == 0 && total_transactions == 0) begin
        `uvm_warning(get_type_name(), "No transactions were checked by the scoreboard.")
    end else if (pass_count > 0 && fail_count == 0) begin
        `uvm_info(get_type_name(), "TEST PASSED!", UVM_NONE)
    end
endfunction
```

UVM Verification (SPI)

< UVM Verification with Synopsys VCS >

```
JVM_INFO ./tb/tb_SPI_MASTER.sv(58) @ 745435: uvm_test_top.ENV.AGENT.SQR@@SEQ [SEQ] spi item to driver : CPOL=0, CPHA=1, TX=49
JVM_INFO ./tb/tb_SPI_MASTER.sv(104) @ 753965: uvm_test_top.ENV.AGENT.DRV [DRV] Driving : CPOL=0, CPHA=1, TX=49
JVM_INFO ./tb/tb_SPI_MASTER.sv(170) @ 762465: uvm_test_top.ENV.AGENT.MON [spi_monitor] 모니터링된 트랜잭션 : TX=0x49, RX=0x49
JVM_INFO ./tb/tb_SPI_MASTER.sv(206) @ 762465: uvm_test_top.ENV.SCO [SCO] Checking: TX=0x49 RX=0x49
JVM_INFO ./tb/tb_SPI_MASTER.sv(210) @ 762465: uvm_test_top.ENV.SCO [SCO] *** PASS ***
```

```
JVM_INFO ./tb/tb_SPI_MASTER.sv(58) @ 771025: uvm_test_top.ENV.AGENT.SQR@@SEQ [SEQ] spi item to driver : CPOL=1, CPHA=1, TX=b6
JVM_INFO ./tb/tb_SPI_MASTER.sv(104) @ 779555: uvm_test_top.ENV.AGENT.DRV [DRV] Driving : CPOL=1, CPHA=1, TX=b6
JVM_INFO ./tb/tb_SPI_MASTER.sv(170) @ 788055: uvm_test_top.ENV.AGENT.MON [spi_monitor] 모니터링된 트랜잭션 : TX=0xb6, RX=0xb6
JVM_INFO ./tb/tb_SPI_MASTER.sv(206) @ 788055: uvm_test_top.ENV.SCO [SCO] Checking: TX=0xb6 RX=0xb6
JVM_INFO ./tb/tb_SPI_MASTER.sv(210) @ 788055: uvm_test_top.ENV.SCO [SCO] *** PASS ***
```



```
virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    pass_count = 0;
    fail_count = 0;
    total_transactions = 0;
    //spi_item = spi_seq_item::type_id::create("SPI_ITEM");
endfunction

virtual function void write(spi_seq_item item);
    //spi_item = item;
    total_transactions++;
    `uvm_info("SCO", $sformatf("Checking: TX=0x%0h RX=0x%0h", item.tx_data, item.rx_data), UVM_LOW);
    // 단순 루틴 룩 가정한 MISO 드라이버 연결 가정

    if (item.rx_data == item.tx_data) begin
        `uvm_info("SCO", "*** PASS ***", UVM_NONE);
        pass_count++;
    end
    else begin
        `uvm_error("SCO", $sformatf("*** FAIL: expected 0x%0h got 0x%0h ***", item.tx_data, item.rx_data));
        fail_count++;
    end
endfunction

// report_phase 추가
virtual function void report_phase(uvm_phase phase);
    super.report_phase(phase);
    `uvm_info(get_type_name(), "------------------------------------------------", UVM_NONE);
    `uvm_info(get_type_name(), $sformatf("Scoreboard Final Report:"), UVM_NONE);
    `uvm_info(get_type_name(), $sformatf("  Total Transactions Checked: %0d", total_transactions), UVM_NONE);
    `uvm_info(get_type_name(), $sformatf("  PASS Count                : %0d", pass_count), UVM_NONE);
    `uvm_info(get_type_name(), $sformatf("  FAIL Count                : %0d", fail_count), UVM_NONE);
    `uvm_info(get_type_name(), "------------------------------------------------", UVM_NONE);

    if (fail_count > 0) begin
        `uvm_error(get_type_name(), $sformatf("TEST FAILED with %0d errors.", fail_count))
    end else if (pass_count == 0 && total_transactions == 0) begin
        `uvm_warning(get_type_name(), "No transactions were checked by the scoreboard.")
    end else if (pass_count > 0 && fail_count == 0) begin
        `uvm_info(get_type_name(), "TEST PASSED!", UVM_NONE)
    end
endfunction
endclass : spi_scoreboard
```

```
** Report counts by severity
UVM_INFO :  618
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[DRV]   101
[RNTST]     1
[SCO]   200
[SEQ]   101
[TEST_DONE]     1
[UVM/RELNOTES]     1
[spi_monitor]   201
[spi_scoreboard]     7
[spi_test]     5

$finish called from file "/tools/synopsys/vcs/W-2024.
$finish at simulation time                829235
           V C S   S i m u l a t i o n   R e p o r t
```

```
Scoreboard Final Report:
  Total Transactions Checked: 100
  PASS Count                : 100
  FAIL Count                : 0
------------------------------------------
```

Trouble Shooting

원인

```
UVM_INFO ./tb/tb_SPI_MASTER.sv(61) @ 1876: uvm_test_top.ENV.AGENT.SQR@@SEQ [SEQ] spi item to driver : CPOL=0, CPHA=0, TX=59
UVM_INFO ./tb/tb_SPI_MASTER.sv(99) @ 1895: uvm_test_top.ENV.AGENT.DRV [DRV] Driving : CPOL=0, CPHA=0, TX=59

UVM_INFO ./tb/tb_SPI_MASTER.sv(61) @ 1896: uvm_test_top.ENV.AGENT.SQR@@SEQ [SEQ] spi item to driver : CPOL=0, CPHA=0, TX=3
UVM_INFO ./tb/tb_SPI_MASTER.sv(99) @ 1915: uvm_test_top.ENV.AGENT.DRV [DRV] Driving : CPOL=0, CPHA=0, TX=3

UVM_INFO ./tb/tb_SPI_MASTER.sv(61) @ 1916: uvm_test_top.ENV.AGENT.SQR@@SEQ [SEQ] spi item to driver : CPOL=0, CPHA=0, TX=f0
UVM_INFO ./tb/tb_SPI_MASTER.sv(99) @ 1935: uvm_test_top.ENV.AGENT.DRV [DRV] Driving : CPOL=0, CPHA=0, TX=f0

UVM_INFO ./tb/tb_SPI_MASTER.sv(61) @ 1936: uvm_test_top.ENV.AGENT.SQR@@SEQ [SEQ] spi item to driver : CPOL=0, CPHA=0, TX=93
UVM_INFO ./tb/tb_SPI_MASTER.sv(99) @ 1955: uvm_test_top.ENV.AGENT.DRV [DRV] Driving : CPOL=0, CPHA=0, TX=93

UVM_INFO ./tb/tb_SPI_MASTER.sv(61) @ 1956: uvm_test_top.ENV.AGENT.SQR@@SEQ [SEQ] spi item to driver : CPOL=0, CPHA=0, TX=50
UVM_INFO ./tb/tb_SPI_MASTER.sv(99) @ 1975: uvm_test_top.ENV.AGENT.DRV [DRV] Driving : CPOL=0, CPHA=0, TX=50

UVM_INFO ./tb/tb_SPI_MASTER.sv(61) @ 1976: uvm_test_top.ENV.AGENT.SQR@@SEQ [SEQ] spi item to driver : CPOL=0, CPHA=0, TX=99
UVM_INFO ./tb/tb_SPI_MASTER.sv(99) @ 1995: uvm_test_top.ENV.AGENT.DRV [DRV] Driving : CPOL=0, CPHA=0, TX=99
UVM_INFO /tools/synopsys/vcs/W-2024.09-SP1/etc/uvm-1.2/base/uvm_objection.svh(1276) @ 1996: reporter [TEST_DONE] 'run' phase i
s ready to proceed to the 'extract' phase
```

SPI VCS TESTBENCH에서 SEQ, DRV 다음 MON,SCO 에서 받은 값이 출력되지 않는 문제

테스트의 run_phase가 실제 트랜잭션 처리 및 모니터링 완료 전에 너무 일찍 종료되어, 모니터가 데이터를 스코어보드로 전달할 충분한 시뮬레이션 시간을 확보하지 못했기 때문

해결

```
virtual task run_phase(uvm_phase phase);
    phase.raise_objection(this);

    // reset
    `uvm_info(get_type_name(), "리셋 시퀀스 시작", UVM_MEDIUM);
    vif.reset <= 1'b1;
    // 초기화
    vif.cpol <= 1'b0;
    vif.cpha <= 1'b0;
    vif.start <= 1'b0;
    vif.tx_data <= 8'h00;
    repeat (20) @(posedge vif.clk);
    vif.reset <= 1'b0;
    `uvm_info(get_type_name(), "리셋 해제", UVM_MEDIUM);

    @(posedge vif.clk);

    wait_for_dut_initial_ready();

    `uvm_info(get_type_name(), "시퀀스 시작", UVM_MEDIUM);
    spi_seq.start(spi_env.spi_agt.spi_sqr);
    `uvm_info(get_type_name(), "시퀀스 완료", UVM_MEDIUM);

    `uvm_info(get_type_name(), "결과가 최종적으로 스코어보드까지 전달되어 검증될 충분한 시간을 확보", UVM_MEDIUM);
    repeat (20) @(posedge vif.clk); // 20 클록 사이클 동안 대기
    `uvm_info(get_type_name(), "충분한 시간 완료.", UVM_MEDIUM);

    phase.drop_objection(this);
endtask
```

test 클래스(spi_test)의 run_phase 내에서, 모든 seq_item이 driver를 통해 DUT로 전송된 직후 DUT가 그에 대한 응답인 RX_DATA를 생성하고 모니터가 그걸 캡쳐하여 스코어보드까지 전달될 때까지 충분한 시간 여유를 줌

Conclusion

I2C와 SPI 통신 설계 과정에서 마스터와 슬레이브 각각의 입장에서 신호를 해석하고 데이터를 구분하면서도 신호 간 타이밍을 맞춰줘야 했습니다. 특히, 이를 AXI 인터페이스와 연동하여 SoC 내에서 제어할 수 있도록 설계하면서, 프로토콜 간의 동작 방식 차이와 데이터 흐름을 조율하는 부분이 어려웠던 것 같습니다.

또한 UVM 기반 검증 과정에서 시퀀스에서 스코어보드까지 데이터가 제대로 전달이 되지 않았는데 다시 모듈과 각 클래스에서 트랜잭션들이 넘어가는 타이밍 문제를 생각해보고 여러 가지 시도를 해보면서 문제를 해결하게 되면서 단순히 기능만 구현하는 것이 아니라, 데이터 흐름과 타이밍 전체를 고려한 시스템적 사고가 중요하다는 것을 느꼈습니다.

THANK YOU