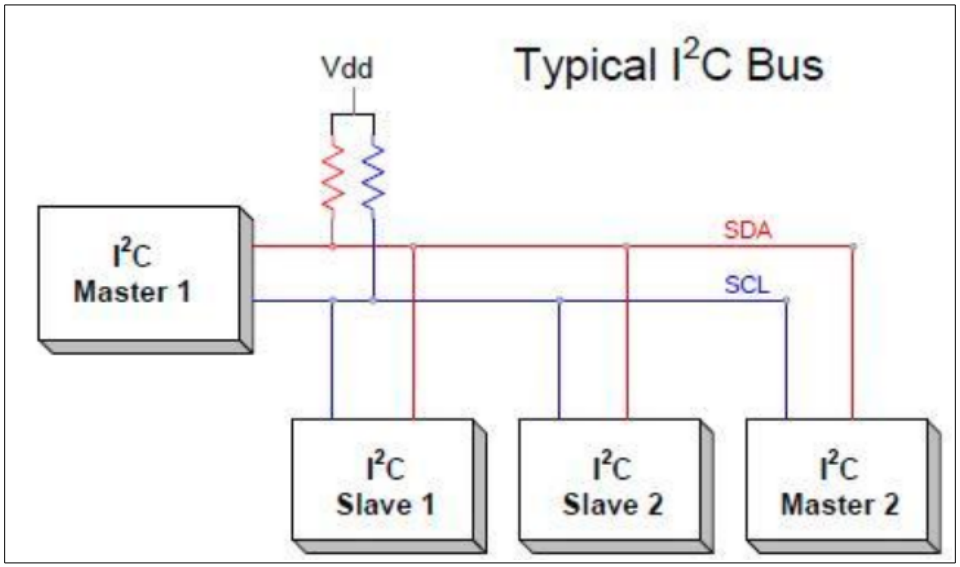
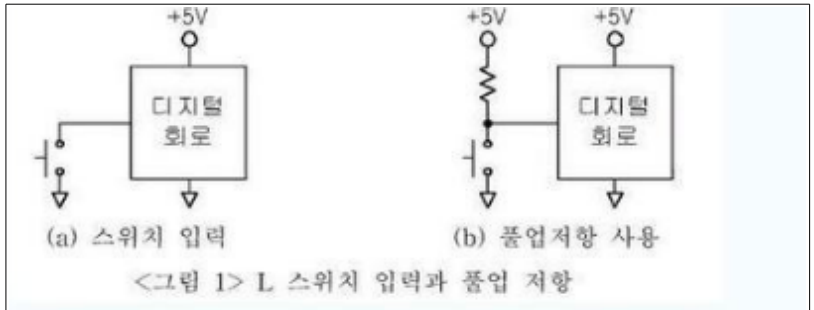
I2C 프로토콜 verilog





<그림 1> L 스위치 입력과 풀업 저항

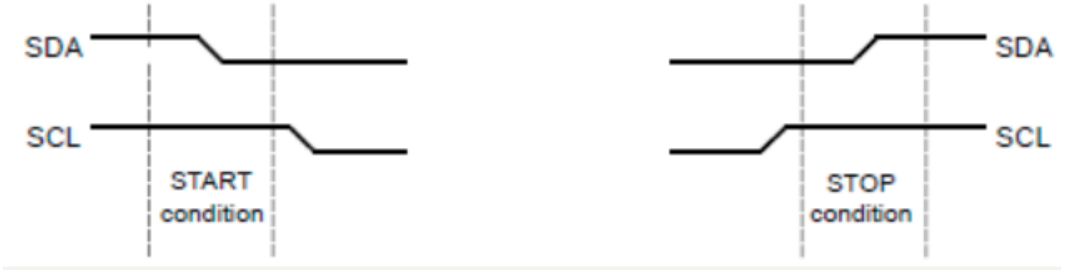
스위치	ON	OPP
(a)그림	0V(Low)	Floating
(b)그림	0V(Low)	+5V(High)

I2C 통신은 통신의 중심이 되는 Master와 통신하고자 하는 주변기기를 Slave로 설정할 수 있다. 일반적으로 mcu 1대와 여러대의 I2C통신을 지원하는 기기와 연결할 경우 Master 1대와 slave 여러대 통신이 가능하다. Master는 2대 이상도 가능하다.

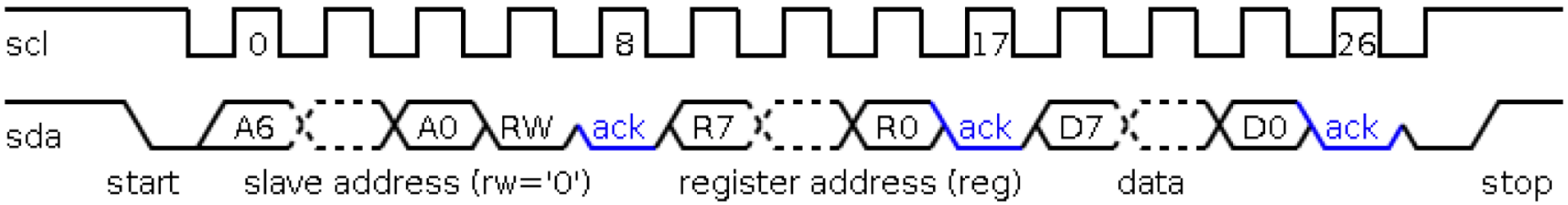
I2C에서 사용되는 두 선은 SDA와 SCL로 구성되어 있다. SDA는 데이터를 주고 받는 역할을 하고, SCL은 동기화를 위한 CLOCK 역할을 한다. 사용 방법은 마스터와 슬레이브에 각각 이 두 선을 연결해주기만 하면 된다. 그러나 이러한 회로만으로는 오픈드레인 상태가 되기 때문에 풀업저항을 달아 전압을 공급해준다.

→ 칩에 입력을 5V or 0V를 가한다. 하지만 입력을 가하지 않는 경우에 칩 자체에서는 5V를 입력하였는지 1V를 입력하였는지 모른다. 이런 문제에 의해 오작동이 발생 할 수 있다. 이경우를 플로팅 되었다라고 한다. 플로팅 상태는 잡음에 매우 취약해지므로 시스템이 불안정해진다. 이를 해결 하기 위해 풀업/풀다운 저항을 사용한다.

→ 풀업저항을 사용한다면 스위치가 열려 있을 때 칩에는 항상 5V의 전압이 가해진다. 따라서 회로에 입력을 몰라도 항상 5V의 전압을 가진 상태가 된다. 스위치를 닫는다면 그라운드 쪽으로 전류가 흘러 회로의 전압은 0V이고, 1을 입력한 것으로 인식하게 된다. 핀을 높은 저항에 매달아 둔다하여 '풀 업 저항' 이라 한다.

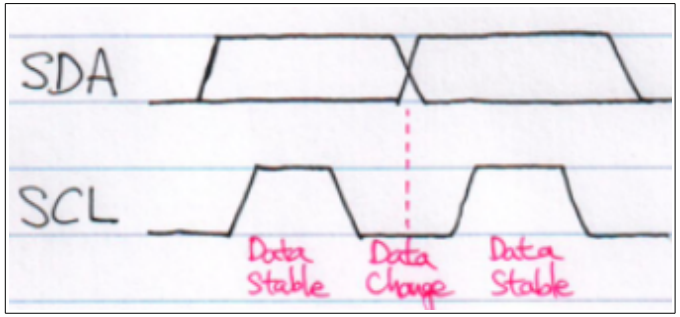


I2C 통신이 되지 않고 있는 상황일 때는 SDA와 SCL은 High 상태를 유지한다. Master가 I2C 통신을 개시하게 될 때 SDA가 Low 상태로 내려가게 되고, 이 때 SCL은 I2C통신이 시작되었음을 인지하고, 클럭 역할을 하기 시작한다.
I2C 통신이 완료 되었을 때 Master는 I2C 통신을 종료하게 된다. SCL이 High 상태로 돌아왔을 때 SDA가 Low에서 High로 변하게 되면 I2C통신이 완료된다.



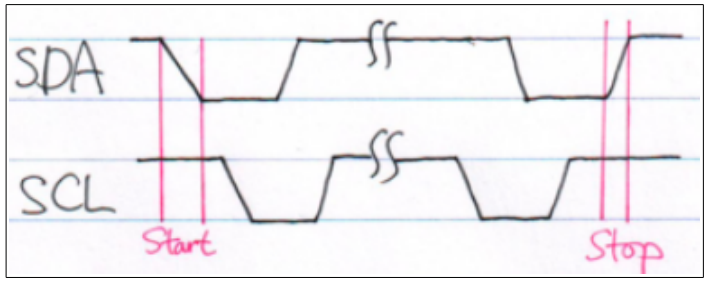
위 그림은 I2C통신을 가지는 센서 디바이스를 슬레이브로서 사용하고, fpga보드를 마스터로 사용했을 때의 데이터 타이밍도이다. 이를 FSM으로 STATE를 IDLE, START, ADDR, ACK, DATA, STOP으로 나눠 설계한다.

I2C 조건



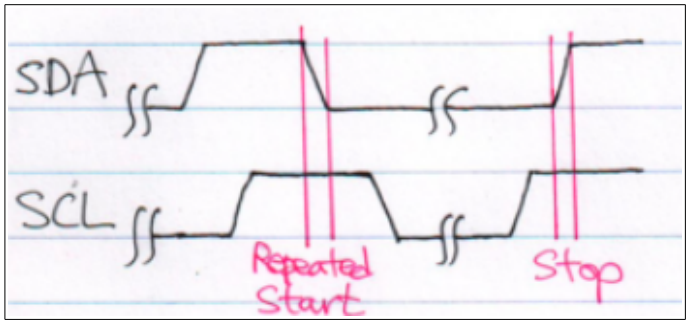
(1) I2C bit format

SDA선으로 전달되는 비트신호는 SCL선으로 전달되는 클럭신호와 동기화 되어 있다.
그림 I2C bit format을 보면 SCL이 Low일 때만, SDA신호가 변할 수 있다. SCL이 High일 때는 SDA가 일정 해야한다는 것을 볼 수 있다.



(2) Start and Stop Conditions

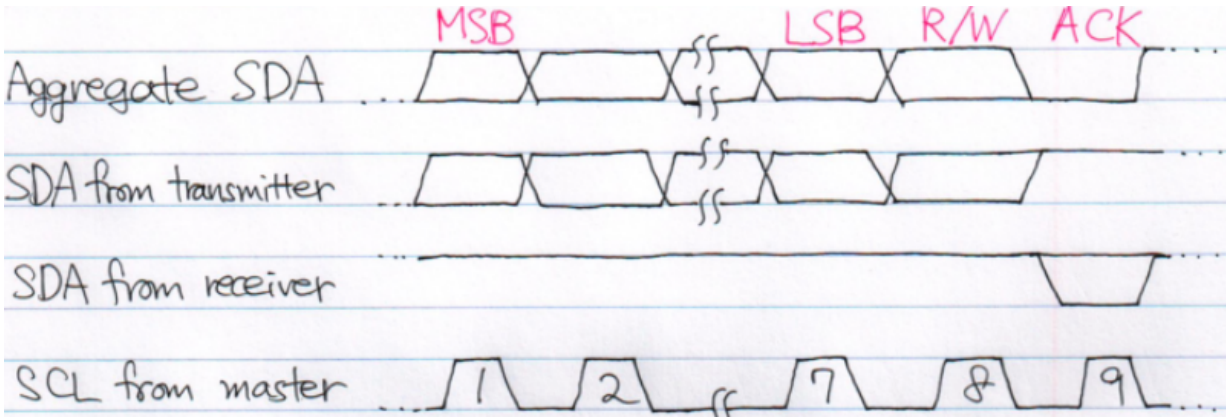
start 와 stop 신호는 (1) bit format 에서 예외이다. 연속된 비트 신호를 보내는 경우,
SCL이 High일 때 SDA의 하강엣지가 나오면 이 신호의 시작을 나타낸다.
반대로 SCL 이 High일 때, SDA의 상승 엣지가 나오면 이 신호의 끝을 나타낸다.



(3) Repeated Start Condition

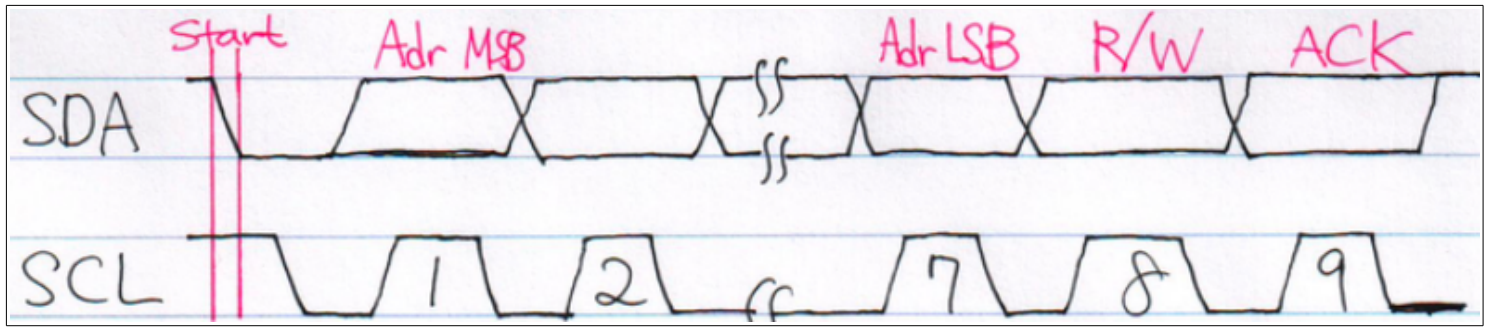
연속된 비트 신호의 끝을 나타내는 상승엣지가 나온 뒤 곧바로 하강에지가 나오면,
다음 신호가 바로 시작하는 것을 나타낸다. Start, Stop신호 사이에는 I2C버스가 사용중임을 알 수 있고,
하나의 마스터 장치만 i2c 버스를 사용 할 수 있다.

(4) Packet format in I2C



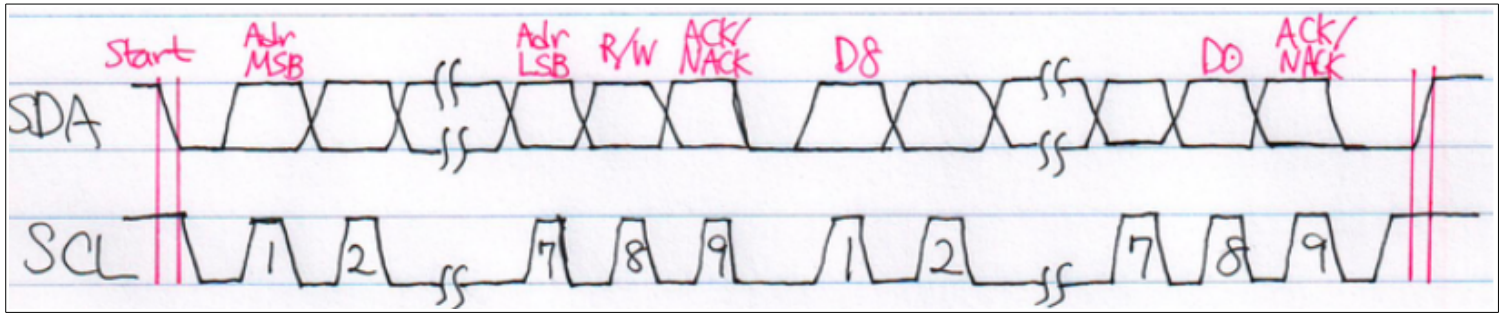
I2C 통신에서 주소 또는 데이터는 패킷의 형태로 전송되어야 한다. 각 패킷은 9개 비트로 구성되는데, 처음 8비트는 transmitter에 의해 보내진다. 마지막 9번째 비트는 receiver가 잘 받았다는 의미로 low신호, ACK(acknowledge)신호를 보낸다. 만약 receiver가 마지막에 low신호를 보내지 않으면 SDA선이 high상태가 되고, transmitter는 이를 NACK(not acknowledge)으로 인식한다. Transmitter가 NACK신호를 인식하면 방금 보낸 패킷을 다시 보내거나 패킷의 전송을 중단하는 등의 조치를 취한다.

(5) Address Packet Format



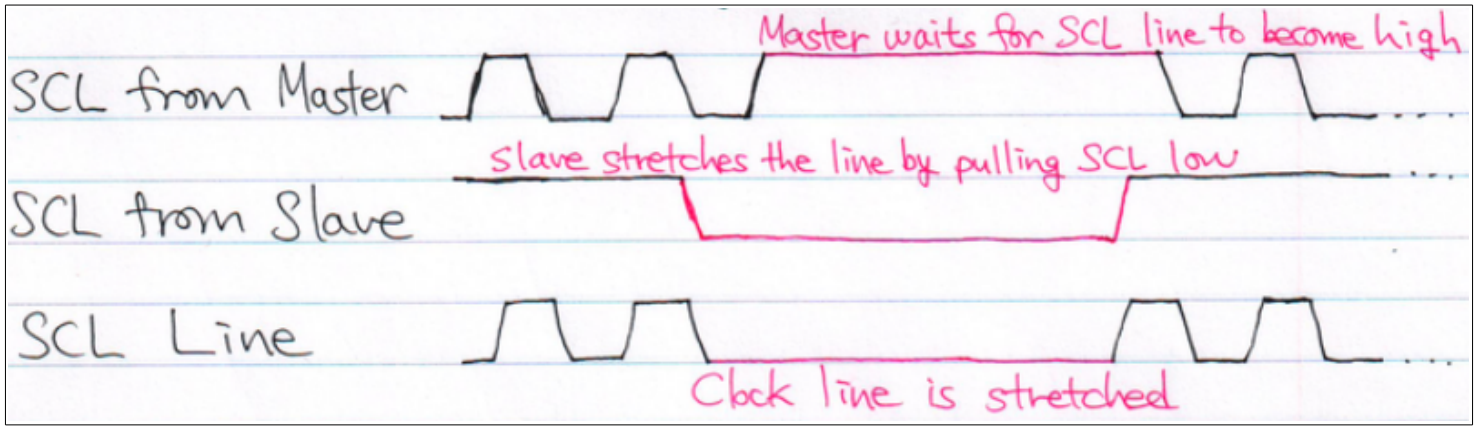
I2C라인을 통해 주소 패킷을 전달하는 모습이다. Transmitter는 SCL이 high일 때 SDA라인을 내려서 전송의 시작을 알리고, 주소의 MSB부터 LSB까지 7개 비트를 보낸다. 전송하는 주소가 7비트여서Noto Sans CJK JP Regular 128개의 주소가 가능하지만, 16개가 예약(reserverd)되었거나 다른 용도로 사용하게 되어있다. 그래서 7비트 주소를 사용하면 I2C 버스에 112개의 장치까지 연결할 수 있다. 그 다음에 R/W(read/write) 여부를 알려주는 신호를 마지막으로 보내면, receiver가 ACK신호를 transmitter로 보낸다.

(6) Typical Data Transmission



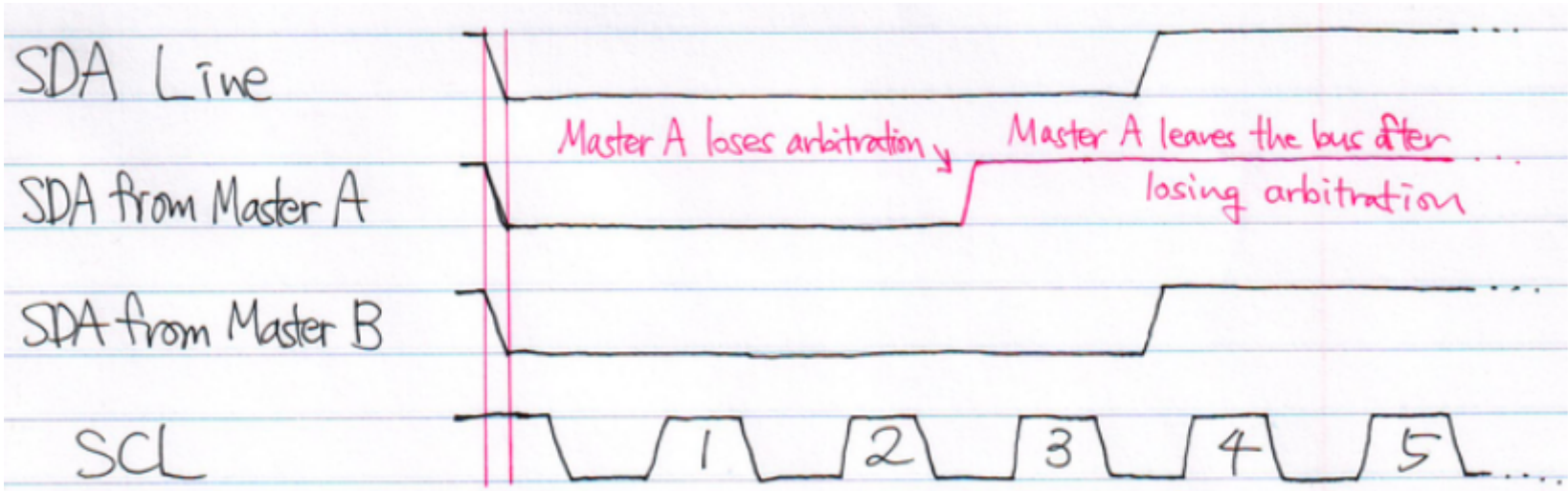
주소패킷과 데이터패킷을 이어서 전송하는 것을 보여준다. 주소 패킷과 비슷한 방식으로 데이터 패킷을 보내는데, 주소패킷과 달리 데이터패킷은 8비트를 보낸다.

(7) Clock Stretching



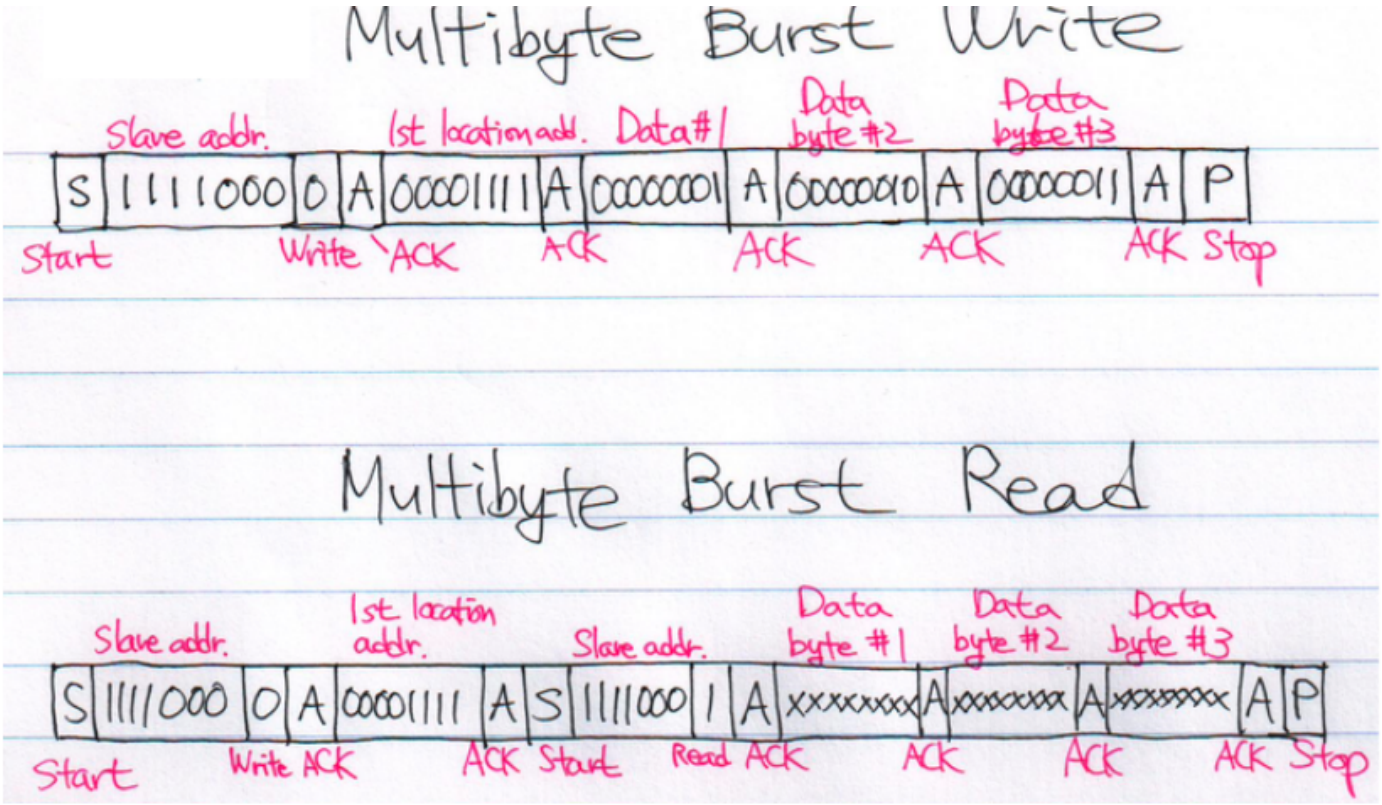
Slave 장치가 데이터를 받을 준비가 안되었을 경우, slave 장치는 SCL선의 상태를 low로 내려서 master 장치에게 데이터 전송을 미뤄달라는 요청을 할 수 있다. 이렇게 되면 master 장치는 slave 장치가 SCL선의 상태를 high로 올리기 전까지 SCL선으로 클럭신호를 발생시키지 않는다. Slave장치가 SCL선을 low로 내린 뒤에 master장치는 SCL선으로 클럭신호를 발생시키는데, 이것을 clock stretching이라고 부른다.
I2C 버스에 여러개의 master 장치를 연결할 수 있지만, 한 순간에는 단 하나의 마스터 장치만 I2C 버스를 사용할 수 있다. 만약 여러개의 master장치가 동시에 버스를 사용하려고 할 경우, arbitration이라는 일이 일어난다. 각각의 master(transmitter)는 SDA라인으로 출력한 뒤 SDA라인의 상태가 실제로 의도한 상태 인지를 체크하는데, 만약 의도한 상태가 아니면 I2C 버스를 사용하는 것을 포기한다.

(8) Arbitration



Master A는 0010 000을 보내고, master B는 0001 111을 동시에 보내려고 한다.
Master A가 세번째 클럭에서 high출력을 했지만 SDA선의 신호는 올라가지 않았다. 이 순간 Master A는 의도대로 출력이 되지 않음을 감지하고, I2C 버스의 사용을 포기한다. 그럼 결국 Master B가 경쟁에서 이겨서 I2C버스를 사용하게 되는 것이다.

Burst 동작 - 연속된 데이터 주소에 효율적으로 값을 쓰기 위해 I2C 통신에는 burst mode가 있다.

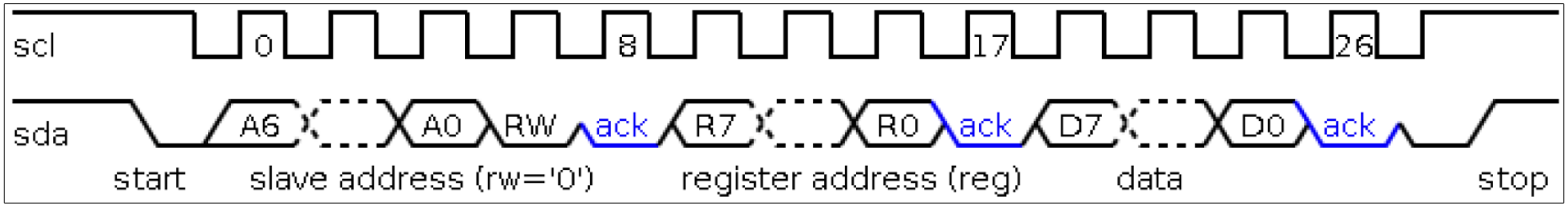


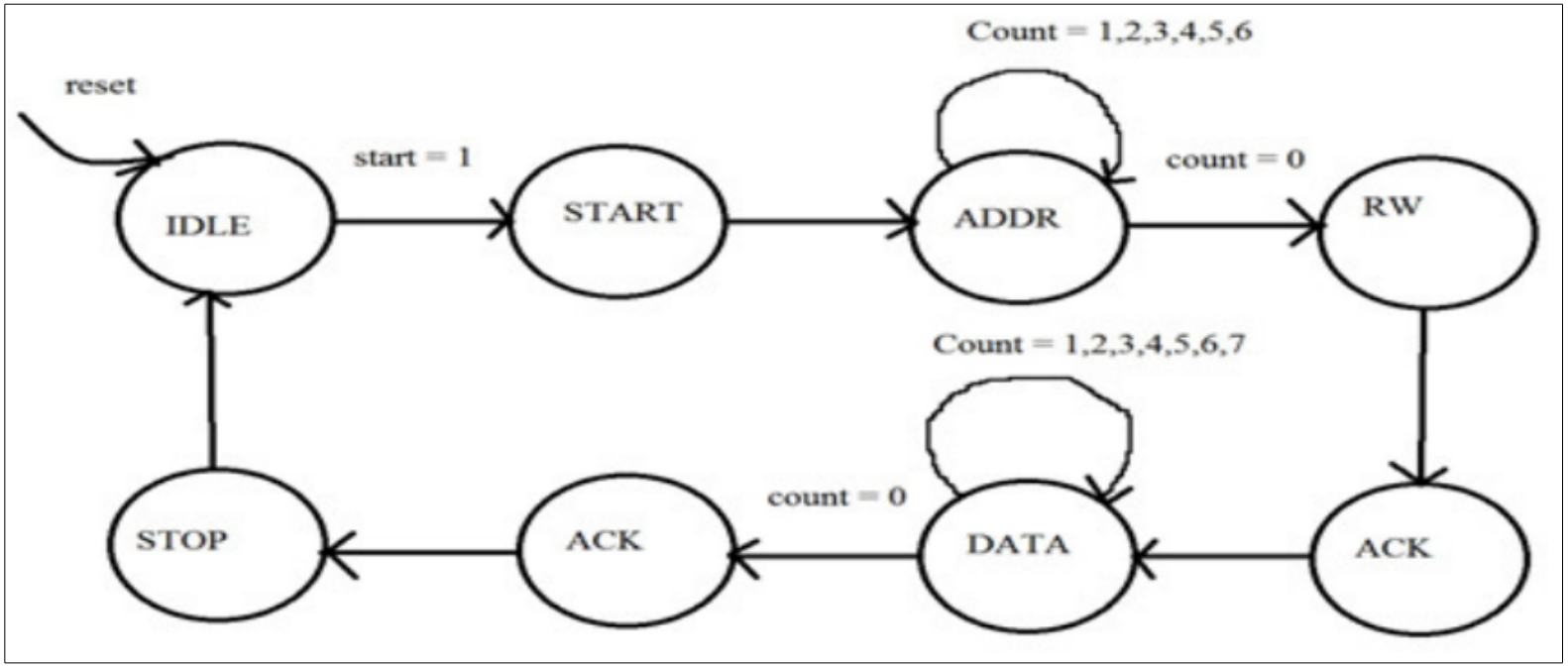
Burst mode에서는 slave 장치의 주소, 데이터의 첫번째 주소를 보낸 뒤, 데이터를 바이트별로 보낸다. 이 때 I2C장치는 stop신호가 나타날 때까지 데이터의 주소를 1씩 증가시킨다. 연속된 데이터 주소의 값을 읽어올 때도 비슷한 방식으로 동작한다.

Multibyte Burst Write 예제는 slave 장치(주소: 1111000)의 0x0F부터 0x11까지 세 주소에 0x01, 0x02, 0x03의 데이터를 쓰게 한다.

Multibyte Burst Read 예제는 slave 장치(주소: 1111000)의 0x0F부터 0x11까지 세 주소에 있는 값을 읽어오게 한다.

FSM(Finite state machine)





* STATE의 개수 만큼 선언

```
parameter IDLE = 4'd0;
parameter START1 = 4'd1;
parameter ADD1 = 4'd2;
parameter ACK1 = 4'd3;
parameter ADD2 = 4'd4;
parameter ACK2 = 4'd5;
parameter START2 = 4'd6;
parameter ADD3 =4'd7;
parameter ACK3 = 4'd8;
parameter DATA = 4'd9;
parameter ACK4 = 4'd10;
parameter STOP1 = 4'd11;
parameter STOP2 = 4'd12;
parameter ADD_EXT = 4'd13;
parameter ACK_EXT = 4'd14;
```

* Case (state)로 각 state 일 때, 실행할 동작과 다음 state로 넘어가기 위한 조건을 작성한다.

```
case(state)
IDLE: begin
            times <= times+1'b1;
            sda_link <= 1'b1;
            sda_r <= 1'b1;
            db_r <= `DEVICE_WRITE;
            state = START1;
        end
START1:begin   //IIC start
            if(`SCL_HIG)    // SCL high
            begin
                sda_link <= 1'b1; //sda 출력
                sda_r <= 1'b0;    // 시작 신호를 생성하기 위해 sda를
                state <= ADD1;
                num <= 4'd0;
            end
            else
                state <= START1;
        end
```

- mpu6050 verilog code

```
`timescale 1ns / 1ps

module mpu6050(
   clk,
   scl,
   sda,
   rst_n
  // data
   );
input clk,rst_n;
output scl;
//output [7:0]data;
inout sda;

reg [2:0]cnt;     //cnt=0，scl 상승；cnt=1，scl 높은중간；cnt=2，scl 하강；cnt=3，scl 중간 낮은
reg [8:0]cnt_sum; //I2C에 필요한 클럭 생성
reg scl_r;        //생성된 클럭 펄스

reg [19:0]cnt_10ms;  // 1,048,576 =  약 100M

always@(posedge clk or negedge rst_n)
if(!rst_n)
   cnt_10ms <= 20'd0;
else
   cnt_10ms <= cnt_10ms+1'b1;

always@(posedge clk or negedge rst_n)
begin
   if(!rst_n)
```

```
    cnt_sum <= 0;
  else if(cnt_sum ==9'd499)   //200khz
    cnt_sum <= 0;
  else
    cnt_sum <= cnt_sum+1'b1;
end

always@(posedge clk or negedge rst_n)
begin
  if(!rst_n)
    cnt <= 3'd5;
  else
    begin
      case(cnt_sum)
      9'd124: cnt<=3'd1;// 높은 레벨
      9'd249: cnt<=3'd2;// 하강 에지
      9'd374: cnt<=3'd3;// 낮은 레벨
      9'd499: cnt<=3'd0;// 상승 엣지
      default: cnt<=3'd5;
      endcase
    end
end

`define SCL_POS (cnt==3'd0)
`define SCL_HIG (cnt==3'd1)
`define SCL_NEG (cnt==3'd2)
`define SCL_LOW (cnt==3'd3)

always@(posedge clk or negedge rst_n)
begin
  if(!rst_n)
```

```
      scl_r <= 1'b0;
   else if(cnt==3'd0)
      scl_r <= 1'b1;
   else if(cnt==3'd2)
      scl_r <= 1'b0;
end

assign scl = scl_r;

`define DEVICE_READ 8'hD1   // 주소장치, 읽기 연산
`define DEVICE_WRITE 8'hD0 //주소장치, 쓰기 작업
`define ACC_XH 8'h3B // 가속도 x축 상위 주소
`define ACC_XL 8'h3C // 가속도 x축 하위 주소
`define ACC_YH 8'h3D // 가속도 y축 하위 주소
`define ACC_YL 8'h3E // 가속도 y축 하위 주소
`define ACC_ZH 8'h3F // 가속도 z축 하위 주소
`define ACC_ZL 8'h40 // 가속도 z축 하위 주소
`define GYRO_XH 8'h43 // 자이로 x축 상위 주소
`define GYRO_XL 8'h44 // 자이로 x축 하위 주소
`define GYRO_YH 8'h45 // 자이로 y축 상위 주소
`define GYRO_YL 8'h46 // 자이로 y축 하위 주소
`define GYRO_ZH 8'h47 // 자이로 z축 상위 주소
`define GYRO_ZL 8'h48 // 자이로 z축 하위 주소

// 자이로 초기화 레지스터
`define PWR_MGMT_1 8'h6B
`define SMPLRT_DIV 8'h19
`define CONFIG1 8'h1A
`define GYRO_CONFIG 8'h1B
`define ACC_CONFIG 8'h1C
```

```
 // 자이로 (Gyro) 해당 레지스터 값 설정을 초기화
`define PWR_MGMT_1_VAL 8'h00
`define SMPLRT_DIV_VAL 8'h07
`define CONFIG1_VAL 8'h06
`define GYRO_CONFIG_VAL 8'h18
`define ACC_CONFIG_VAL 8'h01

parameter IDLE = 4'd0;
parameter START1 = 4'd1;
parameter ADD1 = 4'd2;
parameter ACK1 = 4'd3;
parameter ADD2 = 4'd4;
parameter ACK2 = 4'd5;
parameter START2 = 4'd6;
parameter ADD3 =4'd7;
parameter ACK3 = 4'd8;
parameter DATA = 4'd9;
parameter ACK4 = 4'd10;
parameter STOP1 = 4'd11;
parameter STOP2 = 4'd12;
parameter ADD_EXT = 4'd13;
parameter ACK_EXT = 4'd14;

reg [3:0]state;   // 상태 레지스터
reg sda_r;        // 출력
reg sda_link;     // sda_link = 1, sda 출력; sda_link = 0, sda 하이 임피던스 상태
reg [3:0]num;
reg [7:0]db_r;
// 가속도,자이로스코프 x,y,z축 16비트 reg
reg [7:0]ACC_XH_READ;
reg [7:0]ACC_XL_READ;
reg [7:0]ACC_YH_READ;
```

```
reg [7:0]ACC_YL_READ;
reg [7:0]ACC_ZH_READ;
reg [7:0]ACC_ZL_READ;
reg [7:0]GYRO_XH_READ;
reg [7:0]GYRO_XL_READ;
reg [7:0]GYRO_YH_READ;
reg [7:0]GYRO_YL_READ;
reg [7:0]GYRO_ZH_READ;
reg [7:0]GYRO_ZL_READ;
reg [4:0]times;

always@(posedge clk or negedge rst_n)
begin
  if(!rst_n)
  begin
     state <= IDLE;
     sda_r <= 1'b1;            // 데이터 라인을 하이로 당긴다.
     sda_link <= 1'b0;         // 하이 임피던스 상태
     num <= 4'd0;

 // 레지스터 초기화
     ACC_XH_READ <= 8'h00;
     ACC_XL_READ <= 8'h00;
     ACC_YH_READ <= 8'h00;
     ACC_YL_READ <= 8'h00;
     ACC_ZH_READ <= 8'h00;
     ACC_ZL_READ <= 8'h00;
     GYRO_XH_READ <= 8'h00;
     GYRO_XL_READ <= 8'h00;
     GYRO_YH_READ <= 8'h00;
     GYRO_YL_READ <= 8'h00;
```

```
    GYRO_ZH_READ <= 8'h00;
    GYRO_ZL_READ <= 8'h00;
    times <= 5'b0;
  end
  else
    case(state)
    IDLE: begin
            times <= times+1'b1;
            sda_link <= 1'b1;
            sda_r <= 1'b1;
            db_r <= `DEVICE_WRITE;
            state = START1;
          end
    START1:begin   //IIC start
            if(`SCL_HIG)    // SCL high
            begin
              sda_link <= 1'b1; //sda 출력
              sda_r <= 1'b0;    // 시작 신호를 생성하기 위해 sda를 풀다.
              state <= ADD1;
              num <= 4'd0;
            end
            else
              state <= START1;
          end
    ADD1: begin
            if(`SCL_LOW)   // SCL  LOW
            begin
              if(num == 4'd8)  // 8때, 모두 출력
              begin
                num <= 4'd0;
```

```
            sda_r <= 1'b1;
            sda_link <= 1'b0;  //Z, 하이 임피던스 상태
            state <= ACK1;
          end
          else
          begin
            state <= ADD1;
            num <= num+1'b1;
            sda_r <= db_r[4'd7-num]; //비트 출력
          end
        end
        else
          state <= ADD1;
      end
ACK1: begin
        if(`SCL_NEG)
        begin
          state <= ADD2;
          case(times)  // 다음 쓰기 레지스터 주소를 선택하십시오
            5'd1: db_r <= `PWR_MGMT_1;
            5'd2: db_r <= `SMPLRT_DIV;
            5'd3: db_r <= `CONFIG1;
            5'd4: db_r <= `GYRO_CONFIG;
            5'd5: db_r <= `ACC_CONFIG;
            5'd6: db_r <= `ACC_XH;
            5'd7: db_r <= `ACC_XL;
            5'd8: db_r <= `ACC_YH;
            5'd9: db_r <= `ACC_YL;
            5'd10: db_r <= `ACC_ZH;
            5'd11: db_r <= `ACC_ZL;
```

```
            5'd12: db_r <= `GYRO_XH;
            5'd13: db_r <= `GYRO_XL;
            5'd14: db_r <= `GYRO_YH;
            5'd15: db_r <= `GYRO_YL;
            5'd16: db_r <= `GYRO_ZH;
            5'd17: db_r <= `GYRO_ZL;
            default: begin
              db_r <= `PWR_MGMT_1;
              times <= 5'd1;
            end
          endcase
        end
        else
          state <= ACK1; //응답 대기
      end
ADD2: begin
        if(`SCL_LOW)
        begin
          if(num == 4'd8)
          begin
            num <= 4'd0;
            sda_r <= 1'b1;
            sda_link <= 1'b0;
            state <= ACK2;
          end
          else
          begin
            sda_link <= 1'b1;
            state <= ADD2;
            num <= num+1'b1;
```

```
            sda_r <= db_r[4'd7-num];  // 레지스터 주소 비트
          end
        end
        else
          state <= ADD2;
      end
ACK2: begin
        if(`SCL_NEG)
        begin
          case(times)  // 해당 레지스터 설정
            3'd1: db_r <= `PWR_MGMT_1_VAL;
            3'd2: db_r <= `SMPLRT_DIV_VAL;
            3'd3: db_r <= `CONFIG1_VAL;
            3'd4: db_r <= `GYRO_CONFIG_VAL;
            3'd5: db_r <= `ACC_CONFIG_VAL;
            3'd6: db_r <= `DEVICE_READ;
            default: db_r <= `DEVICE_READ;
          endcase
          if(times >= 5'd6)
            state <= START2;
          else
            state <= ADD_EXT;
        end
        else
          state <= ACK2; // 응답 대기
      end
ADD_EXT:begin   // 일부 설정 레지스터 초기화
        if(`SCL_LOW)
        begin
          if(num == 4'd8)
```

```
            begin
              num <= 4'd0;
              sda_r <= 1'b1;
              sda_link <= 1'b0; // sda 하이임피던스 상태
              state <= ACK_EXT;
            end
            else
            begin
              sda_link <= 1'b1;
              state <= ADD_EXT;
              num <= num+1'b1;
              sda_r <= db_r[4'd7-num]; // 레지스터의 작업 모드를 비트 단위로 설정합니다 .
            end
          end
          else
            state <= ADD_EXT;
        end
    ACK_EXT:begin
          if(`SCL_NEG)
          begin
            sda_r <= 1'b1;
            state <= STOP1;
          end
          else
            state <= ACK_EXT;// 응답 대기
        end
    START2:begin
          if(`SCL_LOW)
          begin
            sda_link <= 1'b1; // sda는 출력입니다.
```

```
            sda_r <= 1'b1; //sda raise
            state <= START2;
          end
          else if(`SCL_HIG)    // scl이 높다면
          begin
            sda_r <= 1'b0;  // 시작 신호를 생성하기 위해 sda를 풀다
            state <= ADD3;
          end
          else
            state <= START2;
        end
    ADD3: begin
          if(`SCL_LOW)   // 비트 낮은 SCL
          begin
            if(num == 4'd8)
            begin
              num <= 4'd0;
              sda_r <= 1'b1;   // Raise sda
              sda_link <= 1'b0; // scl 하이 임피던스 상태
              state <= ACK3;
            end
            else
            begin
              num <= num+1'b1;
              sda_r <= db_r[4'd7-num]; // 레지스터 주소를 비트 단위로 읽습니다.
              state <= ADD3;
            end
          end
          else state <= ADD3;
        end
```

```
ACK3: begin
        if(`SCL_NEG)
        begin
          state <= DATA;
          sda_link <= 1'b0; // sda 고 저항 상태
        end
        else
          state <= ACK3; // 응답을 기다리는 중
      end
DATA: begin
        if(num <= 4'd7)
        begin
          state <= DATA;
          if(`SCL_HIG)
          begin
            num <= num+1'b1;
            case(times)
              5'd6: ACC_XH_READ[4'd7-num] <= sda;
              5'd7: ACC_XL_READ[4'd7-num] <= sda;
              5'd8: ACC_YH_READ[4'd7-num] <= sda;
              5'd9: ACC_YL_READ[4'd7-num] <= sda;
              5'd10: ACC_ZH_READ[4'd7-num] <= sda;
              5'd11: ACC_ZL_READ[4'd7-num] <= sda;
              5'd12: GYRO_XH_READ[4'd7-num] <= sda;
              5'd13: GYRO_XL_READ[4'd7-num] <= sda;
              5'd14: GYRO_YH_READ[4'd7-num] <= sda;
              5'd15: GYRO_YL_READ[4'd7-num] <= sda;
              5'd16: GYRO_ZH_READ[4'd7-num] <= sda;
              5'd17: GYRO_ZL_READ[4'd7-num] <= sda;
              default: ;  // 일시적으로 고려되지 않는, 당신은 시스템 안정성을 향상시키기 위해 코드를 추가 할 수 있습니다
```

```
            endcase
          end
        end
        else if((`SCL_LOW)&&(num == 4'd8))
        begin
          sda_link <= 1'b1;  // sda는 출력입니다.
          num <= 4'd0;
          state <= ACK4;
        end
        else
          state <= DATA;
      end
ACK4: begin
        if(times == 5'd17)
          times <= 5'd0;
        if(`SCL_NEG)
        begin
          sda_r <= 1'b1;   // Raise sda
          state <= STOP1;
        end
        else
          state <= ACK4; // 응답을 기다리는 중
      end
STOP1:begin
        if(`SCL_LOW)   // 낮은 SCL
        begin
          sda_link <= 1'b1;  // sda 출력
          sda_r <= 1'b0;  // 낮은 sda
          state <= STOP1;
        end
```

```
            else if(`SCL_HIG) // sda가 높음
            begin
              sda_r <= 1'b1; // 정지 신호를 발생시키기 위해 sda를 올린다.
              state <= STOP2;
            end
            else
              state <= STOP1;
          end
      STOP2:begin
            if(`SCL_LOW)
            sda_r <= 1'b1;
            else if(cnt_10ms == 20'hffff0)  // 약 10ms 동안 데이터를 가져옵니다.
              state <= IDLE;
            else
              state <= STOP2;
          end
      default:state <= IDLE;
      endcase
end

assign sda = sda_link?sda_r:1'bz;

endmodule
```

- SDK

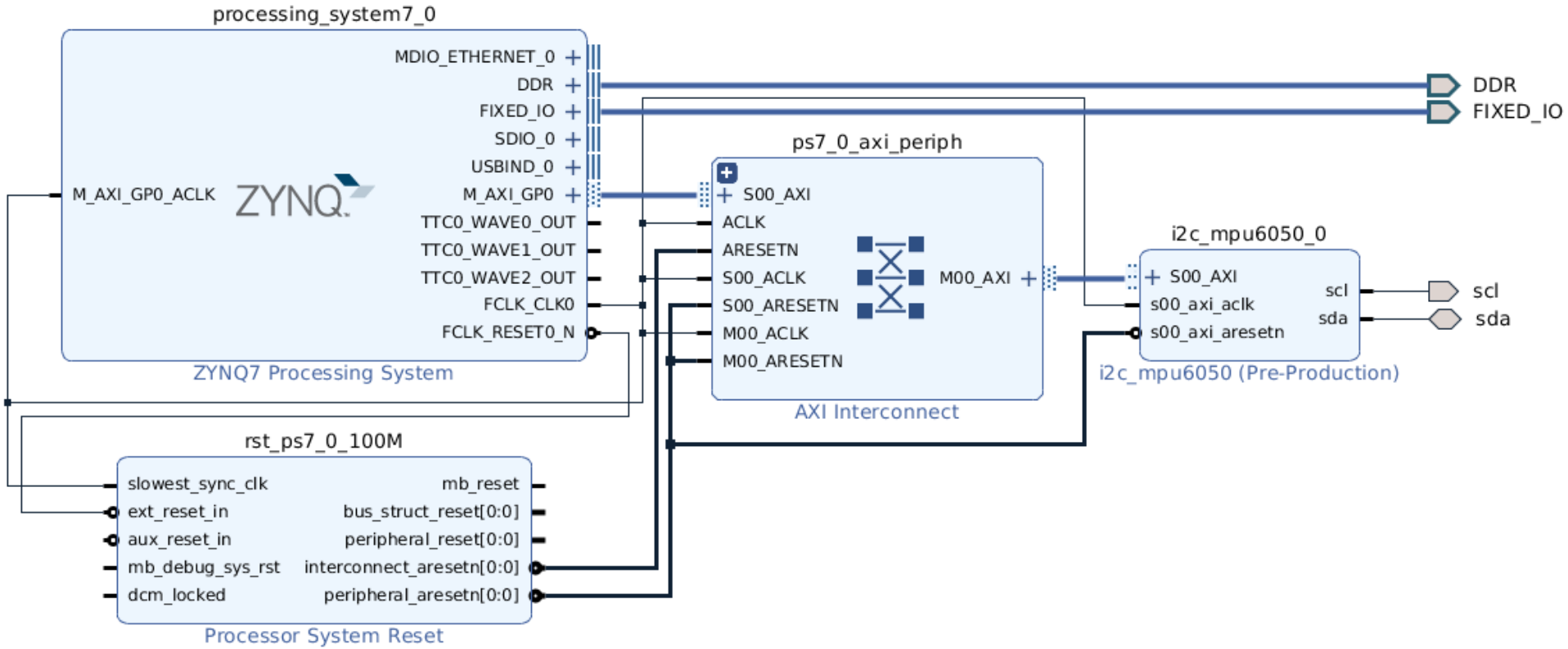
```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"
#include "xbasic_types.h"

Xuint32 *addr = (Xuint32 *)XPAR_I2C_MPU6050_0_S00_AXI_BASEADDR;

int main()
{
 init_platform();

 while(1){
      for(int i=0; i<10000; i++){
      }
 xil_printf("\n\r mpu6050_TEST \n\r");
 xil_printf("ACCEL_XOUT_H = %d\n\r",*(addr+0));
 xil_printf("ACCEL_YOUT_H = %d\n\r",*(addr+1));
 xil_printf("ACCEL_ZOUT_H = %d\n\r",*(addr+2));
 }
 return 0;
```

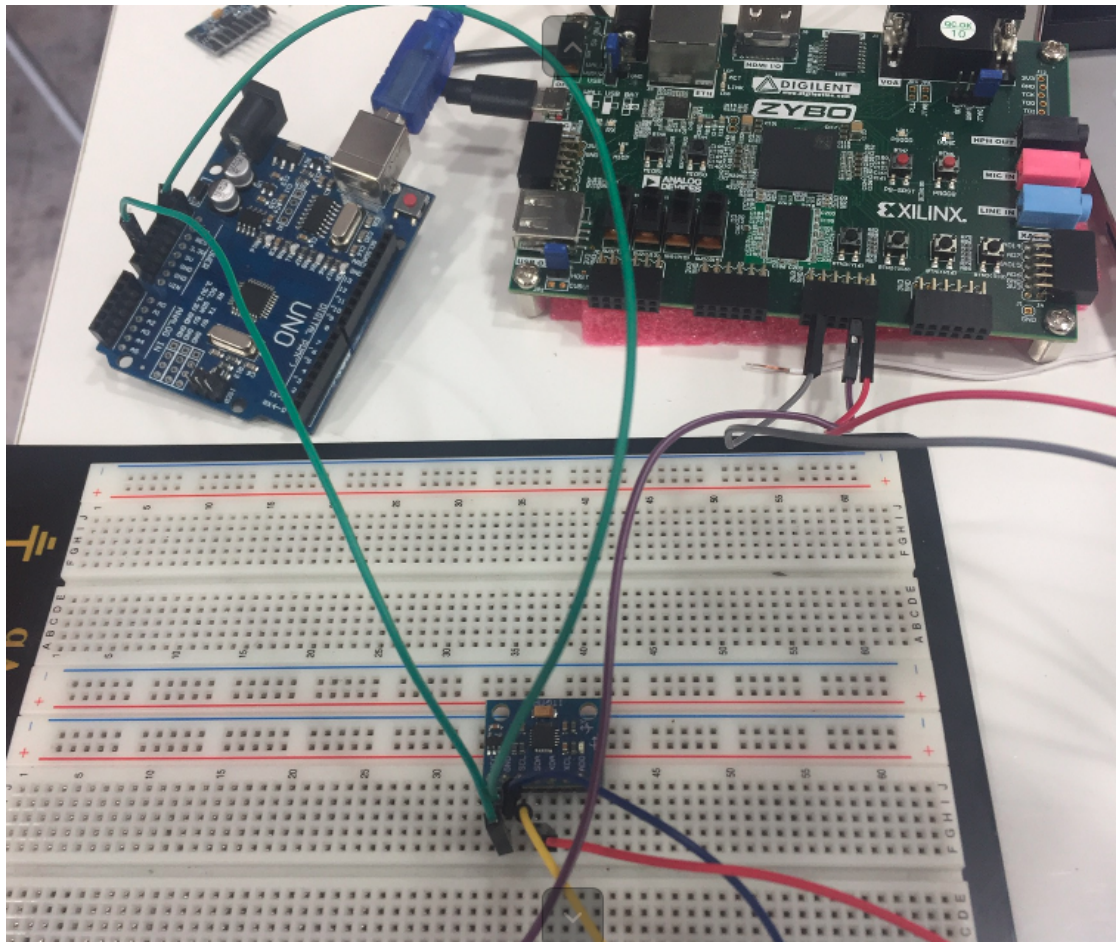
- Hardware Design



- 회로구성

아두이노 vcc 3.3V, zybo 3.3V sw, miso, mosi 핀 mpu6050 에 miso, mosi핀에 연결

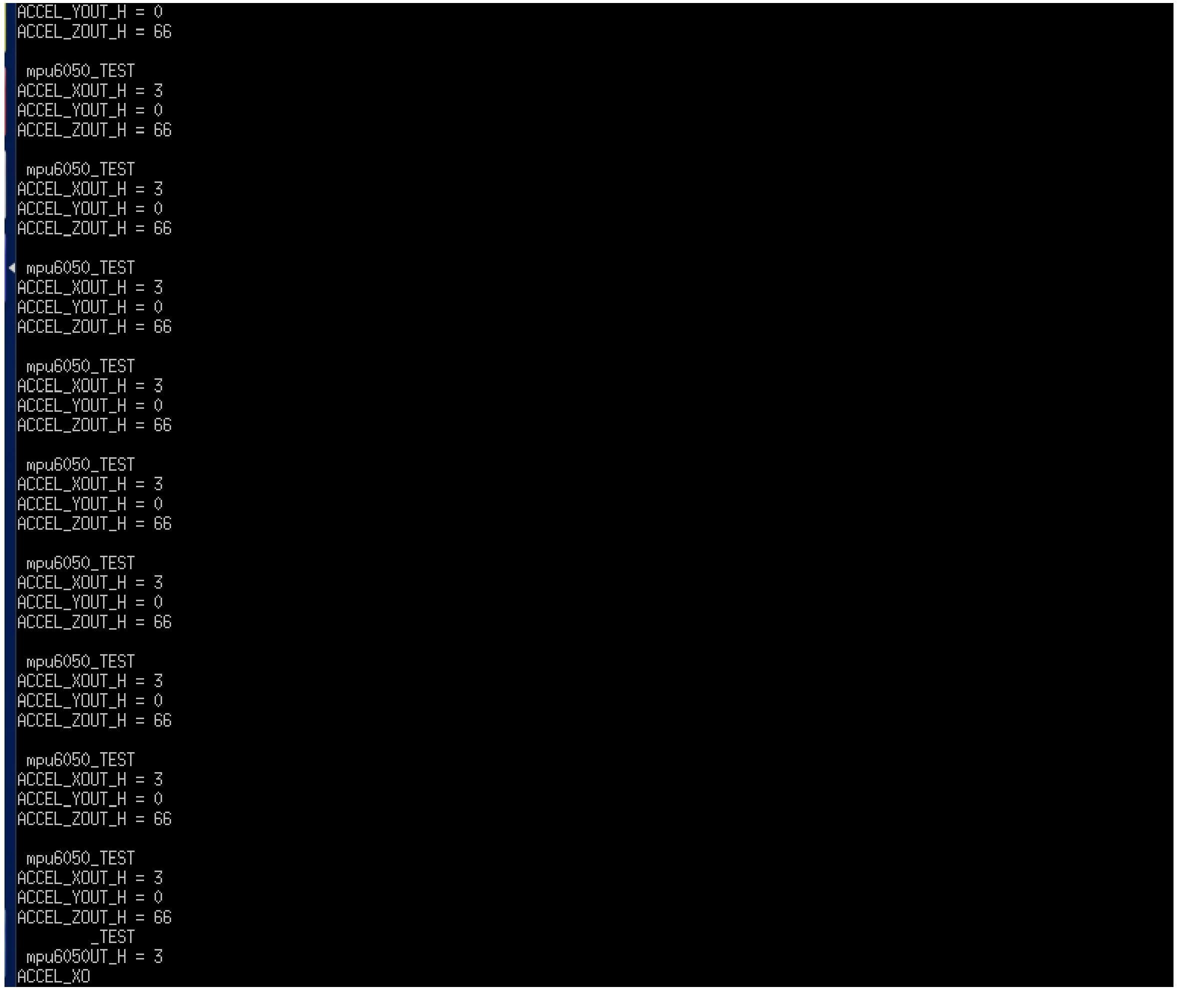
scl	OUT				V15	✓	34	LVCMOS33*
sda	INOUT				W15	✓	34	LVCMOS33*



Pmod JC (Hi-Speed)
JC1: V15
JC2: W15
JC3: T11
JC4: T10
JC7: W14
JC8: Y14
JC9: T12
JC10: U12

-Result

-mpu6050 센서 가속도값 받아오기 실험



```
ACCEL_YOUT_H = 0
ACCEL_ZOUT_H = 66

 mpu6050_TEST
ACCEL_XOUT_H = 3
ACCEL_YOUT_H = 0
ACCEL_ZOUT_H = 66

 mpu6050_TEST
ACCEL_XOUT_H = 3
ACCEL_YOUT_H = 0
ACCEL_ZOUT_H = 66

 mpu6050_TEST
ACCEL_XOUT_H = 3
ACCEL_YOUT_H = 0
ACCEL_ZOUT_H = 66

 mpu6050_TEST
ACCEL_XOUT_H = 3
ACCEL_YOUT_H = 0
ACCEL_ZOUT_H = 66

 mpu6050_TEST
ACCEL_XOUT_H = 3
ACCEL_YOUT_H = 0
ACCEL_ZOUT_H = 66

 mpu6050_TEST
ACCEL_XOUT_H = 3
ACCEL_YOUT_H = 0
ACCEL_ZOUT_H = 66

 mpu6050_TEST
ACCEL_XOUT_H = 3
ACCEL_YOUT_H = 0
ACCEL_ZOUT_H = 66

 mpu6050_TEST
ACCEL_XOUT_H = 3
ACCEL_YOUT_H = 0
ACCEL_ZOUT_H = 66

 mpu6050_TEST
ACCEL_XOUT_H = 3
ACCEL_YOUT_H = 0
ACCEL_ZOUT_H = 66
        _TEST
 mpu6050UT_H = 3
ACCEL_XO
```

→ 가만히 있을 때에도 가속도가 나오고 있다. 제대로 데이터를 받아오는지 모르겠다.

-vivado 시뮬레이터를 통해 i2c 확인