Gowin I3C SDR IP

Reference Design

RD509-1.1E,09/26/2018

Revision History

Date	Version	Description
01/30/2018	1.0E	Initial version published.
09/26/2018	1.1E	IP name modified.

Contents

1 Overview

This manual mainly describes the following three types of reference designs.

1. Communication between I3C SDR Master and I3C SDR Slave;
2. Communication between I3C SDR Master and GW-I2C Slave;
3. Communication between I3C SDR Master and ASIC-I2C Slave.

2 Communication Between I3C SDR Master and I3C SDR Slave

2.1 Purpose

This reference design is employed to verify whether the communication runs well between the I3C SDR Master and I3C SDR Slave.

2.2 Function

The name for this reference design is "w_r_ibi_w". The functions are as follows:

1. I3C SDR Master writes (sends slave address and write operation bit 8'h30, write data to Slave);
2. Enters IBI state (Slave sends address and read operation bit; Master shall read the mandatory data byte, and slave sends read data).

2.3 MCU and Submodules

MCU is the top.v module in the Master project.

Introduction

1. The top.v module defines a number of ports, some of which are defined but not used for timing purposes. The purpose of the module is to observe certain signals that can be pulled to these reserved ports;
2. top. V is mainly used for the clock division, jitter elimination, and data read/write.
   - counter: submodule, used for clock division. First, the 50 MHz clock frequency is divided into 10 MHz via PLL, and then the clock frequency is divided into 10 KHz via the "counter " submodule;
   - deUstb: submodule, used for button jitter elimination;
   - Write/Read data module: I3C SDR Master output port "STATE" will output the corresponding I3C SDR state.
     According to its state, complete the corresponding functions. If the STATE is S_CM_WAIT_AD, I3C SDR Master is in the state of awaiting address, and MCU can send the address; if the STATE is S_CM_WAIT_W_DA, I3C SDR Master is in a state of awaiting writing data, and MCU can write data.

Code

1. Clock Division
   After being processed by PLL and counter, the clock frequency is divided from 50 MHz into 10 KHz:

```
GW_PLL myPll(
    .clkout(midClk), //output clkout 10M
    .clkin(clk_ext) //input clkin 50M
 );
counter cnt1(lowOut,lowClk,lowOver,midClk,1'b1);
defparam cnt1.OVER = 1_000;
```

2. Instantiate I3C SDR Master:

```
I3C_REG_top master(
    LGYS, CMS, ACS, AAS, STOS, STAS,
    LGYO, CMO, ACO, AAO, SIO, STOO, STAO,
                          LGYC, CMC, ACC, AAC, SIC, STOC, STAC,
                         STA_HD_S,
                         SEND_AD_HIGH_S,
                         SEND_AD_LOW_S,
                         ACK_HIGH_S,
                         ACK_LOW_S,
                         STO_HIGH_S,
                         STO_LOW_S,
                         SEND_DA_HIGH_S,
                         SEND_DA_LOW_S,
                         RCVE_DA_HIGH_S,
                         RCVE_DA_LOW_S,
                         ADDRESS_S,
                         PARITYERROR,
                         DI, DOBUF, DO, STATE,
                         SCLH, SCLL,
                         j10_1, j10_5, j10_2, j10_6,
                         CE, RST, CLK
                        );
```

3. Key jitter elimination:

```
                       deUstb deKey1(key1,~key_1,CLK);
```

4. Write/Read data, etc.:

```
           always @(posedge CLK)begin
               if(RST)begin
                   dataTobeSend <=8'h10;
                   DI <=0;
                   SIC <= 0;
                   ........
               end
```

```
        else begin
            if(SIO)begin
                case(STATE)
                    S_CM_ADRS_ABTR_OK:begin
                        SIC <=1;
                        ACS <=1;

                        DI<=8'h00;
                        STOS<=0;
                        STAS<=0;
                    end
                    S_CM_WAIT_AD:begin
                        DI<=8'h30;
                        SIC<=1;

                        STOS<=0;
                        STAS<=0;
                        ACS<=0;
                    end
                    S_CM_WAIT_NACK:begin
                        STOS <=1;
                        SIC<=1;

                        DI<=8'h00;
                        STAS<=0;
                        ACS<=0;
                    end
                    S_CM_WAIT_W_DA:begin
                        if(dataCnt_w==3)begin
                            STOS<=1;
                            ACS<=0;
                            SIC<=1;
                            dataCnt_w <= 0;

                        end
                        else begin
                            if(w_cnt_flag)begin
                                DI <= dataTobeSend;
                                dataTobeSend <= dataTobeSend+1;
                                dataCnt_w <= dataCnt_w+1;
                                w_cnt_flag<=0;

                                STOS<=0;
                                ACS<=0;
                            end
                        end
                        SIC<=1;


                        STAS<=0;

                    end
                    S_CM_READ_OK_SI:begin
                        if(!readOver) begin
                            if(read_cnt_flag)begin
                                dataCnt_r <= dataCnt_r+1;
                                read_cnt_flag<=0;
                                end
                            end
                            SIC<=1;

                            DI<=8'h00;
```

```
                        STOS<=0;
                        STAS<=0;
                        ACS<=0;
                    end



                S_IDLE:begin
                    pass <=1;
                    SIC<=1;



                    DI<=8'h00;
                    STOS<=0;
                    STAS<=0;
                    ACS<=0;
                end

                default: begin
                    DI<=8'h00;
                    SIC<=0;
                    STAS<=0;
                    STOS<=0;
                    ACS<=0;
                end
            endcase
        end
        else begin
            case(STATE)
                S_CM_W_STO:begin
                    AAS<=1;
                    ACS<=1;
                    DI<=8'h00;
                    SIC<=0;
                    STAS<=0;
                    STOS<=0;

                end
                S_CM_READ_SL_OK:begin
                    if(dataCnt_r == 3 )begin
                        if(!readOver)begin
                            STOS<=1;
                        end
                        readOver <=1;
                        ACS <= 0;
                    end
                    else begin
                        read_cnt_flag<=1;
                        ACS<=1;
                    end

                    DI<=8'h00;
                    SIC<=0;
                    STAS<=0;

                end
                S_CM_ADRS_ABTR_OK:begin
                    SIC <=1;
                    ACS <=1;

                    DI<=8'h00;
```

```
                STAS<=0;
                STOS<=0;

            end
            default:begin
                DI<=8'h00;
                SIC<=0;
                STOS<=0;
                STAS<=0;
                ACS<=0;
                w_cnt_flag<=1;
            end
         endcase
     end
  end
  if(key3)begin
     STAS<=1;
   end
end
```

5. Pull out the observed signals:

```
assign j9_4 = sda_line;//a1:0
assign j9_5 = scl_line;
assign j9_6 = STAS;
........
```

2.4 Board Level Connection

This reference design implements communication between the I3C SDR Master and its Slave using double boards; i.e., set I3C SDR Master on a GW1N-9K board and set I3C SDR Slave on the other GW1N-9K board. As such, three lines are required to connect the boards: one for SCL, one for SDA, and one for GND. For the physical constraints of the other observed signals, please refer to dk_start_ge1n4.cst.

2.5 Result Observation

This reference design uses an oscilloscope to observe whether SCL and SDA are correct. In addition, if you want to observe the Bus signals, such as STATE, DI, and DOBUF, the logic analyzer is recommended.

3 Communication Between I3C SDR Master and GW-I2C Slave

3.1 Purpose

This reference design predominantly verifies whether the communication runs well between the I3C SDR Master and GW-I2C Slave.

3.2 Function

The name for this reference design is "write_read". The functions are as follows:

1. I3C SDR Master writes (sends slave address and write operation bit 8'hA0, sends register address and data to Slave);
2. I3C SDR Master reads (sends slave address and read operation bit 8'hA0, sends slave address and read operation bit 8'hA1, and then reads data from Slave).

3.3 MCU and Submodules

Introduction

MCU is the top.v in the Master project. This top.v and the top.v in the Master project of communication between I3C SDR Master and I2C Slave are all modified based on the top.v in the Master project of **Communication Between I3C SDR Master and I3C SDR Slave**, so the top.v modules for the reference designs are similar.

top. V is mainly used for clock division, jitter elimination, and data read/write.

1. counter: submodule, used for clock division. First, the 50 MHz clock frequency is divided into 10 MHz via PLL. Then the clock frequency is

divided into 10 KHz via the “counter” submodule;
2. deUstb: submodule, used for button jitter elimination;
3. Write/Read data module: I3C SDR Master output port STATE will output the corresponding I3C SDR state. According to its state, complete the corresponding functions. If the STATE is S_CM_WAIT_AD, I3C SDR master is in a state of awaiting address, and MCU can send the address. If the STATE is S_CM_WAIT_W_DA, I3C SDR Master is in a state of awaiting writing data, and MCU can write the data.

Code

1. Clock division. After being processed by PLL and counter, the clock frequency is divided from 50 MHz into 10 KHz:

```
GW_PLL myPll(
    .clkout(midClk), //output clkout 10M
    .clkin(clk_ext) //input clkin 50M
 );
counter cnt1(lowOut,lowClk,lowOver,midClk,1'b1);
defparam cnt1.OVER = 1_000;
```

2. Instantiate 3C Master:

```
I3C_REG_top master(
    LGYS, CMS, ACS, AAS, STOS, STAS,
    LGYO, CMO, ACO, AAO, SIO, STOO, STAO,
     LGYC, CMC, ACC, AAC, SIC, STOC, STAC,
    STA_HD_S,
    SEND_AD_HIGH_S,
    SEND_AD_LOW_S,
    ACK_HIGH_S,
    ACK_LOW_S,
    STO_HIGH_S,
    STO_LOW_S,
    SEND_DA_HIGH_S,
    SEND_DA_LOW_S,
                    RCVE_DA_HIGH_S,
                    RCVE_DA_LOW_S,
                    ADDRESS_S,
                    PARITYERROR,
                    DI, DOBUF, DO, STATE,
                    SCLH, SCLL,
                    SDA,SCL,SDA_PULL,SCL_PULL,
                    CE, RST, CLK
                   `ifdef DEBUG_REG
                   ,dbg_ABTR
                   `endif
                   );
```

3. Key jitter elimination:

```
                   deUstb deKey1(key1,~key_1,CLK);
```

4. Write/Read data, etc.:

```
always @(posedge CLK)begin
    if(RST)begin
        dataTobeSend <=8'h20;
        DI <=0;
        SIC <= 0;
        STOS<=0;
        STAS<=0;
        pass<=0;
        w_cnt_flag<=0;
        r_cnt_flag<=0;
        read_flag <=0;
        readOver <=0;
        send_ad_flag<=0;
        led_4_flag<=0;
        led_5_flag<=0;
        led_6_flag<=0;

    end
    else begin
        if(SIO)begin
            case(STATE)
                S_CM_WAIT_AD:begin
                    if(send_ad_flag && !reg_addr)begin
                        DI<=8'hA1;
                    end
                    else begin
                        DI<=8'hA0;
                        reg_addr<=1;
                        if(read_flag)begin
                            send_ad_flag<=1;
                        end
                    end
                    SIC<=1;
                end

                S_CM_WAIT_W_DA:begin//9
                    if(reg_addr)begin
                        DI<=8'h10;
                        //reg_addr<=0;
                        led_5_flag<=1;
                        if(read_flag)begin
                            STAS <=1;
                        end
                    end
                    else begin
                        if(!read_flag)begin
                            if(&dataCnt_w)begin
                                STAS <= 1;
                                dataCnt_w <= 0;
                                read_flag <= 1;
                                led_4_flag <= 1;
                            end
                            else begin
                                if(w_cnt_flag && !reg_addr)begin
                                    DI <= dataTobeSend;
                                    dataTobeSend <= dataTobeSend+1;
                                    dataCnt_w <= dataCnt_w+1;
                                    w_cnt_flag<=0;
                                end
                            end
                        end
                    end
                    SIC<=1;
                end

                S_CI2CM_SEND_ACK_OK:begin
                    if(dataCnt_r == 6)begin
                        STOS <=1;
                        dataCnt_r <= 0;
                        led_6_flag <= 1;

                    end
                    else begin
                        if(r_cnt_flag)begin
                            dataCnt_r <= dataCnt_r +1;
```

```
                                    r_cnt_flag <=0;
                               end

                          end
                          AAS <= 0;
                          SIC <= 1;
                     end

                     S_CM_SL_END:begin
                          SIC <=1;
                     end

                     S_IDLE:begin
                          pass <=1;
                          SIC<=1;
                     end

                     default:   begin
                          DI<=8'h00;
                          SIC<=0;
                     end
                endcase
           end
           else begin
                case(STATE)
                     S_CI2CM_SEND_ACK:begin//17-----state current I2C master read slave
                          if(dataCnt_r == 5)begin
                               AAS<=0;
                               readOver <= 1;
                          end
                          else begin
```

```
                    if(readOver)begin
                         AAS <= 0;
                    end
                    else begin
                         AAS <= 1;
                    end
               end
          end

          S_CI2CM_WAIT_ACK_HF:begin
               reg_addr <= 0;
          end

          default:begin
               DI<=8'h00;
               SIC<=0;
               STOS<=0;
               STAS<=0;
               AAS <=0;
               w_cnt_flag <=1;
               r_cnt_flag <=1;

          end
     endcase
     if(key4) begin
          STAS<=1;
          pass <=0;
          send_ad_flag <=0;
          led_4_flag <=0;
          led_5_flag <=0;
          led_6_flag <=0;
```

```
            end
        end
    end
end
```

5. Pull out the observed signal:

```
assign j9_4 = sda_line;//a1:0
assign j9_5 = scl_line;
assign j9_6 = STAS;
........
```

3.4 Board Level Connection

This reference design implements communication between the I3C SDR Master and GW-I2C Slave using double boards; i.e., set the I3C SDR Master on a GW1N-9K board and set the GW-I2C Slave on the other GW1N-9K board. As such, three lines are required to connect the boards: one for SCL, one for SDA, and one for GND.

3.5 Result Observation

This reference design uses an oscilloscope to observe whether SCL and SDA are correct. In addition, if you want to observe the Bus signals, such as STATE, DI, and DOBUF, the logic analyzer is recommended.

4 Communication Between I3C SDR Master and ASIC-I2C Slave

4.1 Purpose

This reference design is mainly employed to verify whether the communication runs well between the I3C SDR Master and ASIC-I2C Slave.

4.2 Function

The name for this reference design is "write". Its main function is that the I3C SDR Master writes (sends slave address and write operation bit 8'hA0, sends register address and data to Slave).

4.3 MCU and Submodules

Introduction

MCU is top.v in the Master project. top. V is mainly used for the clock division, jitter elimination, and data read/write operations.

1. counter: submodule, used for clock division. First, the 50 MHz clock frequency is divided into 10 MHz via PLL. Then, the clock frequency is divided into 10 KHz via the "counter" submodule;
2. deUstb: submodule, used for button jitter elimination;
3. Write/Read data module: I3C SDR Master output port STATE, will output the corresponding I3C SDR state. According to its state, complete the corresponding functions. If the STATE is S_CM_WAIT_AD, the I3C SDR Master is in a state of awaiting address, and MCU can send the address. If the STATE is S_CM_WAIT_W_DA, I3C SDR Master is in a state of awaiting writing data, and MCU can

write data.

Code

1. Clock Division. After being processed by the PLL and counter, the clock frequency is divided from 50 MHz into 10 KHz:

```
GW_PLL myPll(
    .clkout(midClk), //output clkout 10M
    .clkin(clk_ext) //input clkin 50M
 );
 counter cnt1(lowOut,lowClk,lowOver,midClk,1'b1);
 defparam cnt1.OVER = 1_000;
```

2. Instantiate I3C SDR Master:

```
I3C_REG_top master(
    LGYS, CMS, ACS, AAS, STOS, STAS,
    LGYO, CMO, ACO, AAO, SIO, STOO, STAO,
     LGYC, CMC, ACC, AAC, SIC, STOC, STAC,
    STA_HD_S,
    SEND_AD_HIGH_S,
    SEND_AD_LOW_S,
    ACK_HIGH_S,
    ACK_LOW_S,
    STO_HIGH_S,
    STO_LOW_S,
    SEND_DA_HIGH_S,
    SEND_DA_LOW_S,
    RCVE_DA_HIGH_S,
    RCVE_DA_LOW_S,
    ADDRESS_S,
    PARITYERROR,
    DI, DOBUF, DO, STATE,
    SCLH, SCLL,
                        SDA,SCL,SDA_PULL,SCL_PULL,
                        CE, RST, CLK
);
```

3. Key jitter elimination:

```
deUstb deKey1(key1,~key_1,CLK);
```

4. Write/Read data, etc.:

```
 always @(posedge CLK)begin
     if(RST)begin
          dataTobeSend <=8'h10;
          DI <=0;
          SIC <= 0;
          STOS<=0;
          STAS<=0;
          pass<=0;
          w_cnt_flag<=0;
          dataCnt_w <=0;
          led_flag4<=0;
          led_flag5<=0;
          led_flag6<=0;
     end
     else begin
          if(SIO)begin
               case(STATE)
                    S_CM_WAIT_AD:begin
                         DI<=8'hA0;
                         SIC<=1;
                    End

                    S_CM_WAIT_W_DA:begin//9
```

```
            if(&dataCnt_w)begin
                led_flag4 <=1;
                STOS<=1;
                DI <= dataTobeSend;
                STOS <=1;
                dataCnt_w <= 0;
            end
            else begin
                if(w_cnt_flag)begin
                    DI <= dataTobeSend;
                    dataTobeSend <= dataTobeSend+1;
                    dataCnt_w <= dataCnt_w+1;
                    w_cnt_flag<=0;
                end
            end
            SIC<=1;
        end
        S_IDLE:begin
            pass <=1;
            SIC<=1;
        end
        default:  begin
            DI<=8'h00;
            SIC<=0;
            STAS<=0;
        end
    endcase
end
else begin
    case(STATE)
        S_CI2CM_WAIT_ACK_HF:begin
```

```
                  w_cnt_flag<=1;
              end
              default:begin
                  DI<=8'h00;
                  SIC<=0;
                  STOS<=0;
                  STAS<=0;
              end
           endcase
          if(key4)begin
              STAS<=1;
          end
      end
  end
end
```

5. Pull out the observed signal.

```
    assign j9_4 = sda_line;//a1:0
assign j9_5 = scl_line;
assign j9_6 = STAS;
........
```

4.4 Board Level Connection

This reference design implements communication between the I3C SDR Master and ASIC-I2C Slave using double boards; i.e., set I3C SDR Master on a GW1N-9K board and set ASIC-I2C Slave on the other GW1N-9K board. As such, three lines are required to connect the boards: one for SCL, one for SDA, and one for GND.

4.5 Result Observation

This reference design uses an oscilloscope to observe whether the SCL and the SDA are correct. In addition, if you want to observe the Bus signals, such as STATE, DI, and DOBUF, the logic analyzer is recommended.

PROGRAMMING FOR FUTURE