上海交通大學

实验报告

课程名称：SystemVerilog 电路设计与验证

实验名称：APB_I2C 主机 SystemVerilog 验证

学院(系)：电子信息与电气工程学院

专　　业：微纳电子学系

学生姓名：周翔宇

学　　号：122039910081

2023 年 1 月 28 日

APB-I2C 主机 SystemVerilog 验证

一、 实验目的

实现以 APB 为 Interface 的 I2Cmaster 的 System Verilog testbench。

1. 驱动设计

完成平台设计，该平台至少能够通过 interface 连接 DUT，并拥有 SRC_Agent 来驱动 DUT 进行工作：

(1)完成读操作

(2)完成写操作

2. 功能验证

完成模块 MONITOR 来对信号完成所关心数据的采集，包括 APB 驱动中的数据和地址和 I2C 总线上的数据和地址等等。在此基础上设计 Scoreboard 配合 Golden Model 模块完成验证。

3. 断言设计

(1) APB 总线的时序正确性

(2) I2C 总线的时序正确性

(3) I2C 内部的状态转换

4. 覆盖率设计：

对 I2C 内部的寄存器进行功能覆盖率的统计。

5. 随机化设计：

配置随机化。

二、 实验过程

1. 整体框架

整个 testbench 平台的框架如下图。源端 SRC 端为 APB agent，负责 APB 接口的配置，需要产生输入的 APB 数据，同时采集各个特殊时间点的输入数据用于 monitor。目的端 DST 端为 I2C agent 收集 DUT 中产生的 SCL,SDA 数据，用于最终结果的比较。Golden model 的主要实现功能是将 I2C agent 所收集到的 SDA,SCL 单比特数据转化成 8bit 的数据流，从而与 APB 端输入的原始数据作比较。Comparator 比较 SRC 端和 DST 端经过 I2C 转回的 8bit 数据是否相同，由于

进行的是一次读写，因此 Comparator 还将 SRC 端最初写入的数据和 DST 端最终读出的数据进行了对比。Coverage 主要对 I2C 内部 6 个寄存器的可能改动的域进行了功能覆盖。

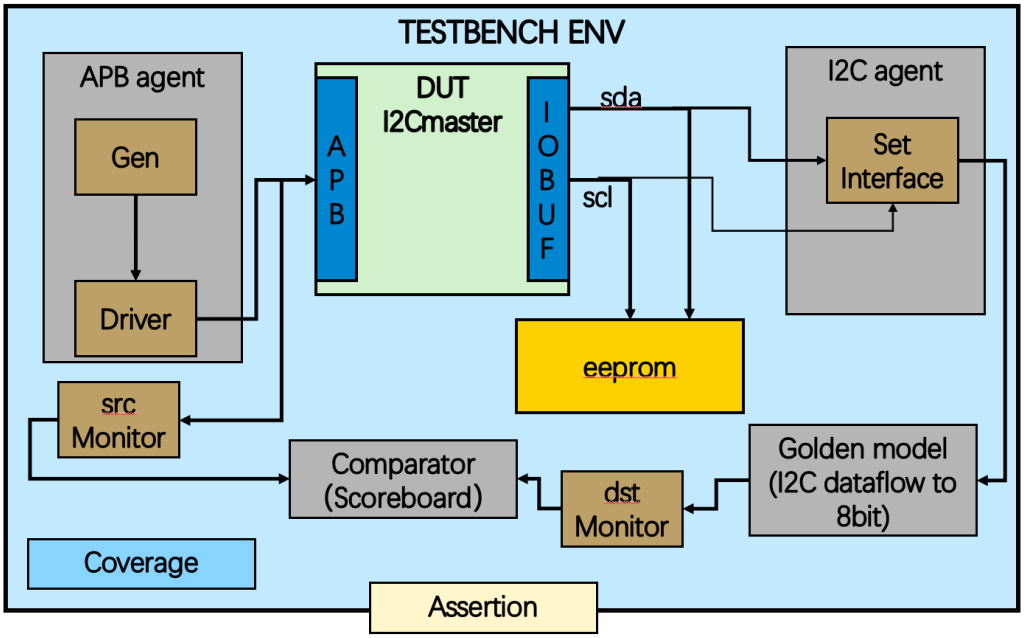


图 1. testbench 平台

2. 对原本 DUT 的修改

为了使得验证平台更好的采集到最终的 SDA,SCL 数据，将 DUT 中的这两个信号引作输出，以供 Interface 连接。

```
module i2c_eeprom_sim (
    // general signals
    input  wire        i2c_clk   ,
    input  wire        i2c_rstn  ,

    // APB signals
    input  wire [31:0] apb_addr  ,
    input  wire [31:0] apb_wdata ,
    input  wire        apb_write ,
    input  wire        apb_sel   ,
    input  wire        apb_enable,
    output wire [31:0] apb_rdata ,
    output wire        apb_ready ,
    output wire        apb_slverr,
    output wire        sda,
    output wire        scl
);
```

图 2. DUT 中引出 sda scl

3. Interface

该验证平台采用了两个 interface，分别是 src 端的 apb interface 和 dst 端的 i2c interface。

4. APB agent

APB agent 可以分为三个部分，第一个是 src driver 实现最简单的对地址进行读写操作，同时根据 APB 配置过程完成内部寄存器的相应读写，第二个是 src generator，生成随机的 data 和 addr 以及 device ID，第三个是 src agent 对整个 src 端通过生成的数据进行相应配置。

```
task  write(input address_t address,input data_t data_w);
    @(posedge active_channel.clk);
    this.active_channel.apb_sel = 1;
    this.active_channel.apb_wdata = data_w;
    this.active_channel.apb_write = 1;
    this.active_channel.apb_addr = address;
    @(posedge active_channel.clk)
    this.active_channel.apb_enable =1;
    @(posedge active_channel.clk);
    this.active_channel.apb_sel = 0;
    this.active_channel.apb_enable =0;
    this.active_channel.apb_wdata = 0;
    this.active_channel.apb_write = 0;
    this.active_channel.apb_addr = 0;
endtask

task read(input address_t address,output data_t data_r);
    @(posedge active_channel.clk);
    this.active_channel.apb_sel = 1;
    this.active_channel.apb_write = 0;
    this.active_channel.apb_addr = address;
    @(posedge active_channel.clk)
    this.active_channel.apb_enable =1;
    @(posedge active_channel.clk);
    data_r = this.active_channel.apb_rdata;
    this.active_channel.apb_sel = 0;
    this.active_channel.apb_enable =0;
    this.active_channel.apb_wdata = 0;
    this.active_channel.apb_write = 0;
    this.active_channel.apb_addr = 0;
endtask
```

图 3. Driver 中的基本读写

利用 I2C 协议进行传输数据时，需要对内部寄存器进行配置，在 agent 中完成进行六个内部寄存器的读写功能实现。

在 I2C 配置过程中，首先要设置时钟分频和打开 I2C 功能，然后写入数据开始传输，检测是否传输完成。在利用 cmd 寄存器发起传输指令之后要持续读取 sta 寄存器中的值以检测单个字节传输是否结束。

```
task i2c_reg_config(input logic [31:0] clk_scale,
                    input logic [31:0] reg_ctrl
);
    write(ADDR_REG_CLK,clk_scale);
    delay();
    write(ADDR_REG_CTR,reg_ctrl);
endtask

task i2c_cmd_write(input logic [31:0] cmd);
    write(ADD_REG_CMD,cmd);
endtask

task i2c_cmd_read(output logic [31:0] cmd);
    write(ADD_REG_CMD,cmd);
endtask

task i2c_status_config(output logic [31:0] status);
    read(ADD_REG_STA,status);
endtask

task i2c_data_write(input logic [31:0] data_w);
    write(ADD_REG_TX,data_w);
endtask

task i2c_data_read(output logic [31:0] data_r);
    read(ADDR_REG_RX,data_r);
endtask
```

图 4. CLK CTR CMD STA TX RX6 个寄存器的读写

```
i2c_reg_config(CLK_SCALE,32'h00000080);//clk ctr
$display("I2C write config starts!");
i2c_data_write(DEVICE_32);              //device
i2c_cmd_write(32'h00000090);
do begin
    i2c_status_config(status);
end while(status[1] == 1);
$display("sending control over");
```

图 5. 传输 8 比特 control 信号

在整个写操作结束时需要检测的是整个 I2C 传输过程是否 STOP。

```
i2c_cmd_write(32'h00000040);//stop
do begin
    i2c_status_config(status);
end while(status[6] == 1);
$display("write is over");
```

图 6. 检测 I2C 通信 stop

对于读操作需要在传输完 3 比特数据后 Restart。在传输完新的 read 指令后读取 RX 寄存器中的值。

```
i2c_data_write(MEM_ADDRL_32);       //addrL
i2c_cmd_write(32'h00000010);
do begin
    i2c_status_config(status);
end while(status[1] == 1);
$display("sending addrl over");

i2c_data_write(DEVICE_32+1);       //read control a1
i2c_cmd_write(32'h00000090);
do begin
    i2c_status_config(status);
end while(status[1] == 1);
$display("sending cmd over");

i2c_cmd_write(32'h00000060);
$display("try to get the data_r");
```

图 7. 读过程中的 restart 与读取最终数据

generator 中利用 rand 生成随机的分频系数、device ID、读写地址、写入的数据。再在 agent 中将这些数据写入 src driver 的驱动函数中。

```
task src_config(output read_data);
    apb_random_datapkg random_data;
    logic [7:0] device_id_8bit;
    src_generator.data_gen();
    mailbox_gen2drv.get(random_data);
    device_id_8bit = {4'ha,random_data.device_id,1'b0};
    $display("the random_data is %p",random_data);
    src_driver.src_config(random_data.clk,device_id_8bit,random_data.addr,random_data
endtask
```

图 8.src agent 驱动写入随机数

5. I2C agent

该部分是对端口的连接。

```
class i2c_agent ;
    virtual duttb_intf_dst.TBconnect dch;
    function void set_interface(
        virtual duttb_intf_dst.TBconnect dch
    );
        this.dch = dch;
    endfunction
endclass
```

图 9.I2C agent 连接 interface

6. Golden model

整个 monitor、scoreboard、golden model 部分一起组成 pakage。将采集到的

dst 端 sda 数据流转换成 8 比特数据的 golden model，采集源端、经处理后的目的端数据的 src monitor、dst monitor，和对 src 数据与经过 golden model 的 dst 端数据对比后进行评价的 comparator(即 scoreboard)。

golden model 是将 dst 端的 sda 和 scl 的单比特数据流转换成 8 比特的对应数据，其重点在于检测到 I2C 传输的开始与结束。

```
task pre_sda (input logic sda,output logic sda_pre);
    sda_pre = dst.sda;
    @(posedge dst.clk);
endtask

task detect_i2c_start(output logic i2c_start);
    logic sda_pre;
    pre_sda(dst.sda,sda_pre);
    i2c_start = sda_pre & (!dst.sda) & dst.scl;
endtask

task detect_i2c_end(output logic i2c_end);
    logic sda_pre;
    pre_sda(dst.sda,sda_pre);
    i2c_end = !sda_pre & dst.sda & dst.scl;
endtask
```

图 10. 检测 I2C 的开始结束在于 sda 的上升沿和下降沿

EEPROM 的写过程会利用 I2C 传输 4 字节数据，而读过程是 5 字节数据，且含有 Restart 过程。

7. Monitor

monitor 中分为 src 端的 monitor 用于收集 TX 寄存器中写入的值和最终 RX 寄存器中读出的值，而 dst 端的 monitor 用于收集 golden model 中得到的 8 比特数据。因此对于采集数据的时间点有一定要求，只有当写入/读出的地址为 TX/RX 寄存器时才会读取 WDATA/RDATA.

由一写一读的 APB 配置过程中可知写入的第 3 个数据为写入 EEPROM 中的数据值，因此将该数据保留下来。dst 端采集的数据为 golden model 中得到的 9 个 8 比特数据，同时最后一个得到的数据为 EEPROM 中读取的数据值，将其保留。

```
task i2c_trans_read(
        output logic [7:0] i2c_8bit [5]
                    );
    logic [8:0] i2c_9bit [5];
    logic i2c_start,i2c_end;
    int i ,j ;
    do begin
        detect_i2c_start(i2c_start);
    end while(!i2c_start);
    assert(i2c_start)$display("COMUNICATION START(READ),the time is %t",$time);
    //start i2c communication
    for(i=0;i<3;i++)
    begin
        for(j=8;j>=0;j--)
        begin
            @(posedge dst.scl);
            i2c_9bit [i][j]=dst.sda;
            $display("[%d]the bit is %b,the time is %t",j,i2c_9bit [i][j],$time);
        end
        i2c_8bit[i] = i2c_9bit[i][8:1];
        $display("i2c_trans_read the data is 8'h%h,the acl is %b,the time is %t",i2c_8b
    end
    do begin
        detect_i2c_start(i2c_start);
    end while(!i2c_start);
    $display("COMUNICATION RESTART(READ),the time is %t",$time);
    for(i=3;i<5;i++)
    begin
        for(j=8;j>=0;j--)
        begin
            @(posedge dst.scl);
            i2c_9bit [i][j]=dst.sda;
            $display("[%d]the bit is %b,the time is %t",j,i2c_9bit [3][j],$time);
        end
        i2c_8bit[i] = i2c_9bit[i][8:1];
        $display("i2c_trans_read the data is 8'h%h,the acl is %b,the time is %t",i2c_8b
    end
    do begin
        detect_i2c_end(i2c_end);
    end while(!i2c_end);
    assert(i2c_end)$display("COMUNICATION END(READ),the time is %t",$time);
endtask
```

图 11. EEPROM 读过程中 sda scl 数据转换的过程

```
task src_get_signal(output logic event_src_valid);
    @(posedge src.clk);
        event_src_valid = ((src.apb_addr == ADD_REG_TX)&&src.apb_write&&src.apb_enable);
endtask

task src_read_signal(output logic event_src_read_valid);
    @(posedge src.clk);
        event_src_read_valid = ((src.apb_addr == ADDR_REG_RX)&&!src.apb_write&&src.apb_enable);
endtask
```

图 12. 采集的标志信号

```
task get_src_write_data(
    );
    //mailbox #(logic [7:0]) mailbox_temp;
    int get_cnt = 0;
    logic get_signal;
    logic [7:0] wdata;
    logic [7:0] data2check;
    do begin
    src_get_signal(get_signal);
    if(get_signal)
    begin

        $display("[APB src write monitor] get a data is 8'h%h,the time is %t",src.apb_wdata[7:0],$time);
        mailbox_data_src.put(src.apb_wdata[7:0]);
        $display("[mailbox_src] put : %h",src.apb_wdata[7:0]);
        if(get_cnt == 3) wdata = src.apb_wdata[7:0];
        get_cnt++;
    end
    end while(get_cnt<9);
    mailbox_temp.put(wdata);

    $display("-----------src write data collect over--------------");
```

图 13. 保留写入存储器的值

```
task get_src_read_data();
    logic [7:0] wdata;
    logic read_signal;
    do begin
    src_read_signal(read_signal);
    if(read_signal)
    begin
        $display("[APB src read monitor] get a data is 8'h%h,the time is %t",src.apb_rdata[7:0],$time);
        mailbox_data_src.put(src.apb_rdata[7:0]);
        $display("[mailbox_src] put : %h",src.apb_rdata[7:0]);
    end
    end while(!read_signal);

    $display("-----------src read data collect get--------------");
    mailbox_temp.get(wdata);
    $display("the APB_I2C src read the written data is 8'h%h",wdata);
    mailbox_data_src.put(wdata);
```

图 14. 将写入 EEPROM 的数据放在 src mailbox 的最后

```
task get_dst_data();
    logic [7:0] i2c_write[4] ;
    logic [7:0] i2c_read [5];
    logic [7:0] rdata;

    golden_model.i2c_trans_write(i2c_write);
    golden_model.i2c_trans_read(i2c_read);
    for(int i =0 ;i <4 ;i++)
    begin
        mailbox_data_dst.put(i2c_write[i]);
        $display("[mailbox_dst] put : %h",i2c_write[i]);
    end
    for(int i =0 ;i <5 ;i++)
    begin
        mailbox_data_dst.put(i2c_read[i]);
        $display("[mailbox_dst] put : %h",i2c_read[i]);
        if(i == 4) rdata = i2c_read[i];
    end
    $display("the APB_I2C dst read the written data is 8'h%h",rdata);
    mailbox_data_dst.put(rdata);
endtask
```

图 15. 将从 EEPROM 中读取的数据放在 dst mailbox 的最后

8. Comparator（scoreboard）

将 src 端采集到的 mailbox 中的数据和 dst 端采集到的数据进行对比。在比较前面 src 端发送的数据和 dst 端接收到的数据是否一致可以得到 DUT 的功能正确性，而 mailbox 中的最后一个数据分别是 src 端写入的数据和 dst 读出的数据，若相同则说明 DUT 功能正确。即一共比较 9 个过程中 src 产生和 dst 接收的数据以及最后写入的值与读出的值，共 10 个数据。

```
do begin
    mailbox_src.get(this.data_src);
    mailbox_dst.get(this.data_dst);
    $display("the src num is 8'h%h,the dst num is 8'h%h",data_src,data_dst);
    if(data_src == data_dst)begin
        cor_cnt ++;
        flag = 1;
        $display("PASS");
    end
    else begin
        err_cnt ++;
        flag = 0;
        $display("FAIL");
    end
    $display("the mailbox_src num is %d",mailbox_src.num());
end while(mailbox_src.num()!=1);
mailbox_src.get(this.data_src);
mailbox_dst.get(this.data_dst);
$display("the src written num is 8'h%h,the dst read num is 8'h%h",data_src,data_dst);
if(data_src == data_dst)begin
    cor_cnt ++;
    flag = 1;
    $display("PASS");
end
else begin
    err_cnt ++;
    flag = 0;
    $display("FAIL");
end
```

图 16. 比较 src 端和经过了 golden model 的 dst 端输出的差异

9. Coverage

对 I2C 内部寄存器的功能覆盖率分为 6 个寄存器分别进行。

对于时钟分频寄存器，本次实验采取的是 8'h1F 的分频。对于 CTR 寄存器，在不考虑中断的情况下只有 EN，即最高位会变化，因此只需检测最高位的覆盖情况。对于 STA 寄存器有正在传输数据、数据传输完成、未有应答、仲裁丢失 4 种功能状态。对于 CMD 有开始读或者写、写、停止写、读、停止读 5 种功能状态。

```
covergroup I2C_REG ;
    COVER_CLK_SCALE:coverpoint clk_reg{
        bins clk_16 = {8'h1f};
    }
    COVER_CTR:coverpoint ctr_reg{
        wildcard bins no_EN = {8'h0?};
        wildcard bins EN = {8'h8?};
    }

    COVER_STA:coverpoint sta_reg{
        bins TIP = {8'b01000010,8'b00000010};
        bins FIN = {8'b01000001};
        wildcard bins NACK = {8'b11000000};
        wildcard bins AL = {8'b??1000??};
    }

    COVER_CMD:coverpoint cmd_reg{
        bins start = {8'h90,8'ha0};
        bins write = {8'h10,8'h50};
        bins stop_write = {8'h50};
        bins read = {8'h64,8'h60};
        bins stop_read = {8'h60,8'h64};
    }

    COVER_RX:coverpoint rx_reg{
        wildcard bins data_send={8'h??};
    }

    COVER_TX:coverpoint tx_reg{
        wildcard bins data_rcv={8'h??};
    }
```

图 17. 内部寄存器功能覆盖率

10. Assertion

断言分为对 APB 协议成功读写数据的断言、对 I2C 成功读取数据的断言、对 I2C 内部状态转化的断言。

```
sequence SETUP;
    $rose(apb_sel)&&!apb_enable;
endsequence
sequence ACCESS;
    $rose(apb_enable)&&apb_sel&&$stable(apb_write)&&$stable(apb_wdata);
endsequence
sequence RECEIVE;
    apb_sel&&apb_enable&&apb_ready&&$stable(apb_write)&&$stable(apb_wdata);
endsequence
```

```
property apb_handshake_check;
    @(posedge i2c_clk)
    SETUP  |-> ##1 ACCESS ##[0:$]RECEIVE;
endproperty
```

图 18. APB 传输断言

```
sequence PRECLK;
    (apb_addr==32'h10025000)&&(apb_wdata==32'h0000001f);
endsequence

sequence CTR;
    (apb_addr==32'h10025004)&&(apb_wdata==32'h00000080);
endsequence

sequence DEVICE_ID;
    (apb_addr==32'h10025010) ##[1:$](apb_addr==32'h10025014)&&(apb_wdata==32'h00000090);
endsequence

sequence SEND_START;
    (apb_addr==32'h10025010)&&(apb_sel&&apb_enable&&apb_ready) ##1(apb_addr==32'h10025014)&&(ap
endsequence

sequence SEND_FINISH;
    ((apb_addr==32'h1002500C)&&(apb_rdata[1]==1)&&!apb_write)[->0:$] ##1 ((apb_addr==32'h100250
endsequence

sequence STOP;
    ((apb_addr==32'h1002500C)&&(apb_rdata[6]==1))[=0:$] ##1 ((apb_addr==32'h1002500C)&&(apb_rda
endsequence
```

```
property PRECLK2CTR;
    @(posedge i2c_clk)
    PRECLK and RECEIVE |-> ##[0:$] CTR and RECEIVE;
endproperty

property DEVICE_ID_SEND;
    @(posedge i2c_clk)
    CTR and RECEIVE |-> ##[0:$] DEVICE_ID and RECEIVE;
endproperty

property DATA_ADDRESS_SEND;
    @(negedge apb_enable)
    SEND_START  |->##[0:$] SEND_FINISH;
endproperty

property WRITE_STOP;
    @(negedge apb_enable)
    (apb_addr==32'h10025014)&&(apb_wdata[6]==1) and (apb_write&&apb_sel&&apb_enable&&apb_ready
endproperty
```

图 19. I2C 内部状态转换断言

```
do begin
    detect_i2c_start(i2c_start);
end while(!i2c_start);
assert(i2c_start)$display("COMUNICATION START(WRITE),the time is %t",$time);
```

```
do begin
    detect_i2c_end(i2c_end);
end while(!i2c_end);
assert(i2c_end)$display("COMUNICATION END(WRITE),the time is %t",$time);
```

图 20. I2C 通信断言

三、 仿真结果

仿真结果如下：

scoreboard 显示一共 10 组数据均相同，证明 DUT 功能正确。

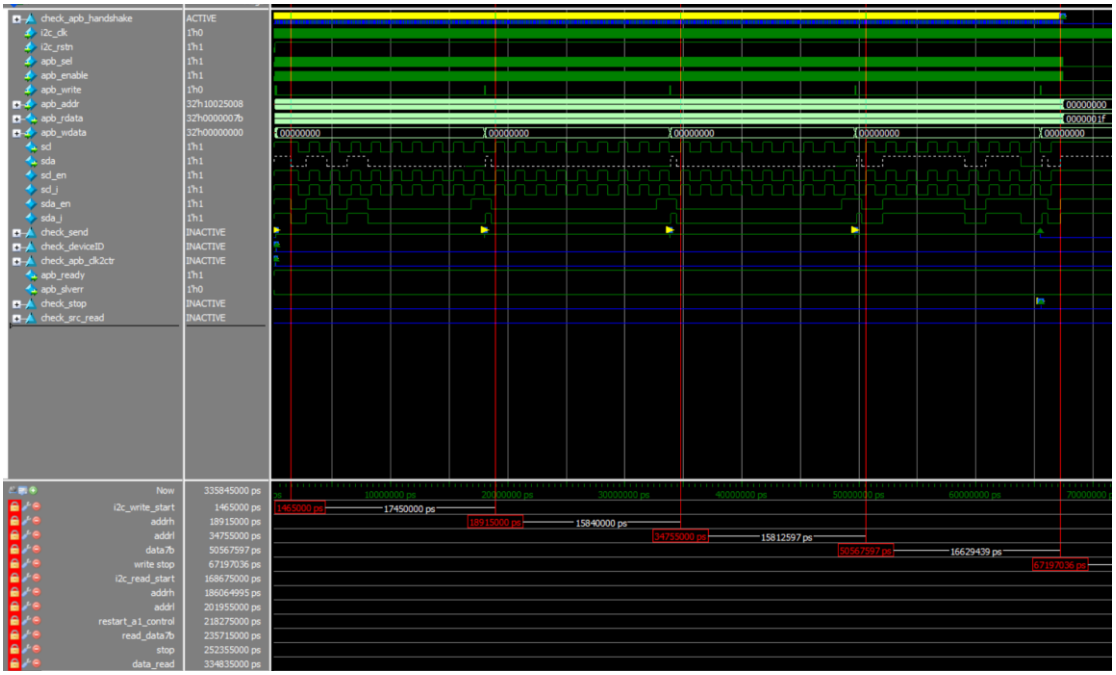


图 21. 与 demo 中写入相同值的写过程的时序图

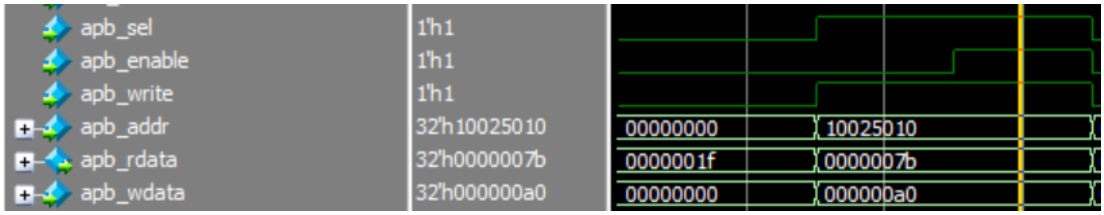


图 22. 写入数据 7b

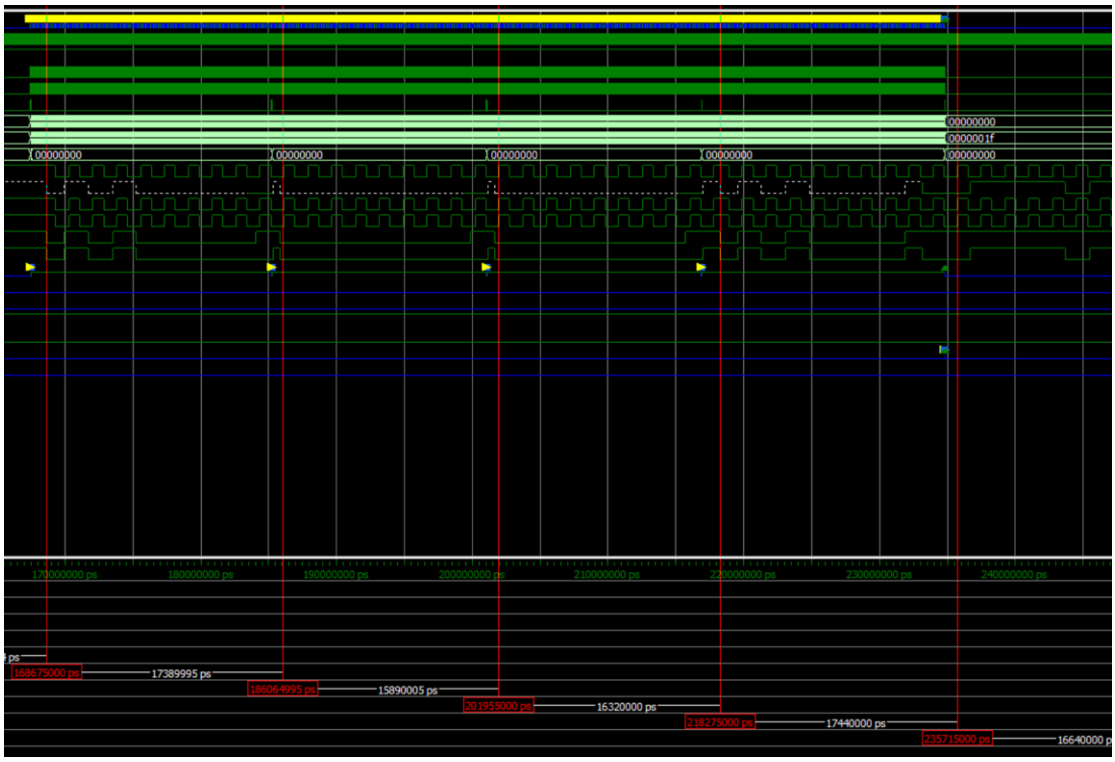


图 23. 与 demo 中读出相同值读过程的时序图

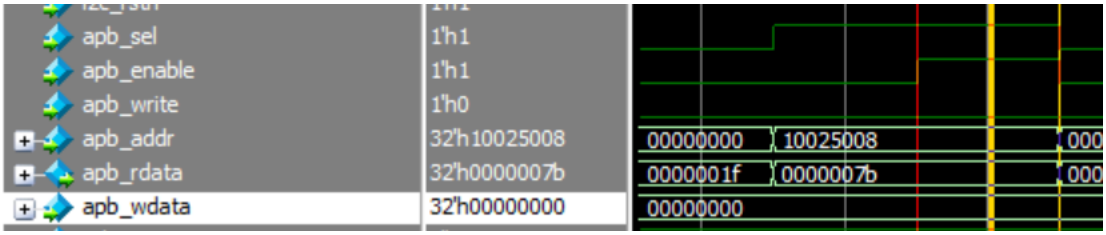


图 24. 读出数据 7b

```
# -------starting scoreboard---
# [ENV] start work : Comparator !The time is           335835000
# the src num is 8'ha0,the dst num is 8'ha0
# PASS
# the src num is 8'h00,the dst num is 8'h00
# PASS
# the src num is 8'h00,the dst num is 8'h00
# PASS
# the src num is 8'h7b,the dst num is 8'h7b
# PASS
# the src num is 8'ha0,the dst num is 8'ha0
# PASS
# the src num is 8'h00,the dst num is 8'h00
# PASS
# the src num is 8'h00,the dst num is 8'h00
# PASS
# the src num is 8'ha1,the dst num is 8'ha1
# PASS
# the src num is 8'h7b,the dst num is 8'h7b
# PASS
# the src written num is 8'h7b,the dst read num is 8'h7b
# PASS
# [COMPARETOR]compare is over
# [COMPARETOR]------------------Congratulations ! all num is right!< total         10 PASS,          0 FAIL>------------------
```

图 25. 输出的比较结果，全部正确

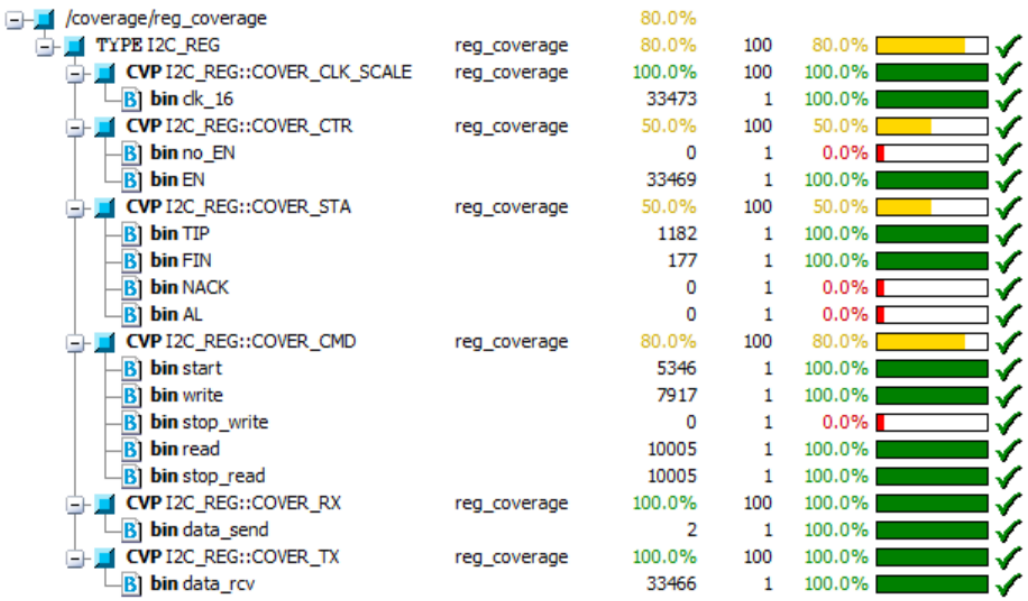
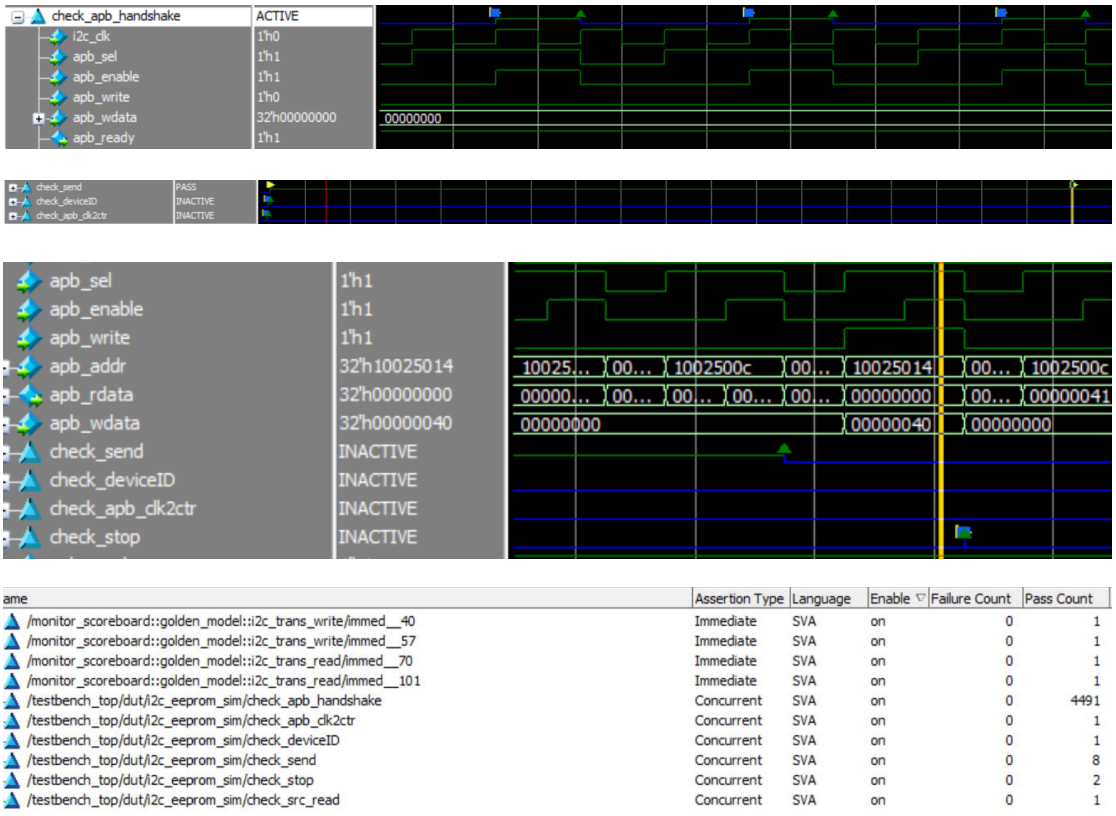


图 26. coverage 结果



ame	Assertion Type	Language	Enable	Failure Count	Pass Count
/monitor_scoreboard::golden_model::i2c_trans_write/immed__40	Immediate	SVA	on	0	1
/monitor_scoreboard::golden_model::i2c_trans_write/immed__57	Immediate	SVA	on	0	1
/monitor_scoreboard::golden_model::i2c_trans_read/immed__70	Immediate	SVA	on	0	1
/monitor_scoreboard::golden_model::i2c_trans_read/immed__101	Immediate	SVA	on	0	1
/testbench_top/dut/i2c_eeprom_sim/check_apb_handshake	Concurrent	SVA	on	0	4491
/testbench_top/dut/i2c_eeprom_sim/check_apb_clk2ctr	Concurrent	SVA	on	0	1
/testbench_top/dut/i2c_eeprom_sim/check_deviceID	Concurrent	SVA	on	0	1
/testbench_top/dut/i2c_eeprom_sim/check_send	Concurrent	SVA	on	0	8
/testbench_top/dut/i2c_eeprom_sim/check_stop	Concurrent	SVA	on	0	2
/testbench_top/dut/i2c_eeprom_sim/check_src_read	Concurrent	SVA	on	0	1

图 27. 断言结果

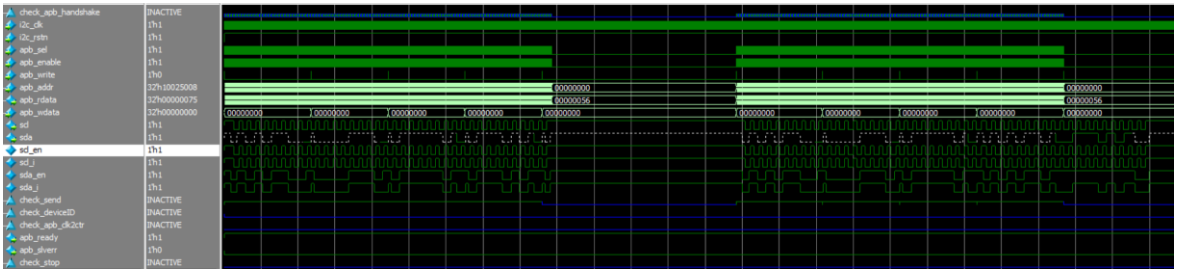


图 28. 采用随机化配置的一次写读过程

Name	Class Type	Coverage	Goal	% of Goal	Status	Included
/coverage/reg_coverage		80.0%				
TYPE I2C_REG	reg_coverage	80.0%	100	80.0%		
CVP I2C_REG::COVER_CLK_SCALE	reg_coverage	100.0%	100	100.0%		
illegal_bin clk_0		0	-	-		
bin clk		55357	1	100.0%		
CVP I2C_REG::COVER_CTR	reg_coverage	50.0%	100	50.0%		
bin no_EN		0	1	0.0%		
bin EN		55353	1	100.0%		
CVP I2C_REG::COVER_STA	reg_coverage	50.0%	100	50.0%		
bin TIP		3120	1	100.0%		
bin FIN		20372	1	100.0%		
bin NACK		0	1	0.0%		
bin AL		0	1	0.0%		
CVP I2C_REG::COVER_CMD	reg_coverage	80.0%	100	80.0%		
bin start		14046	1	100.0%		
bin write		20787	1	100.0%		
bin stop_write		0	1	0.0%		
bin read		10004	1	100.0%		
bin stop_read		10004	1	100.0%		
CVP I2C_REG::COVER_RX	reg_coverage	100.0%	100	100.0%		
bin data_send		1	1	100.0%		
CVP I2C_REG::COVER_TX	reg_coverage	100.0%	100	100.0%		
bin data_rcv		55350	1	100.0%		

```
# -------starting scoreboard---
# [ENV] start work : Comparator !The time is          554685000
# the src num is 8'hac,the dst num is 8'hac
# PASS
# the src num is 8'h0e,the dst num is 8'h0e
# PASS
# the src num is 8'h7d,the dst num is 8'h7d
# PASS
# the src num is 8'h75,the dst num is 8'h75
# PASS
# the src num is 8'hac,the dst num is 8'hac
# PASS
# the src num is 8'h0e,the dst num is 8'h0e
# PASS
# the src num is 8'h7d,the dst num is 8'h7d
# PASS
# the src num is 8'had,the dst num is 8'had
# PASS
# the src num is 8'h75,the dst num is 8'h75
# PASS
# the src written num is 8'h75,the dst read num is 8'h75
# PASS
# [COMPARETOR]compare is over
# [COMPARETOR]-------------------Congratulations ! all num is right!< total         10 PASS,          0 FAIL>-------------------
# [TB-SYS] testbench system has done all the work, exit !
```

图 29. coverage 和 scoreboard 报告

四、 总结

1. 驱动设计

完成平台设计，该平台通过 interface 连接 DUT，并拥有 SRC_Agent 来驱动

DUT 进行工作，完成了一次写和一次读。

2. 功能验证设计

利用了 golden model、monitor、comparator 将 DUT 的输入和输出进行了功能正确与否的比较，实现思路是将 sda、scl 转换为原来的 8bit 数据与输入进行比较。

3. 断言设计

设计了 APB 协议中传输成功的断言、I2C 内部寄存器间状态转换的断言、I2C 通信开始和停止的立即断言。

4. 覆盖率设计

对 I2C 内部的 6 个寄存器在 DUT 配置中可能作用到的域进行了功能覆盖。

5. 随机化设计

完成了配置的随机化（时钟分频系数、读写的从机设备、写入的数据）。

五、 发现的问题

在随机化测试中发现当读写的设备为 6 号时读写数据会出现错误，如图 30

```
# the src num is 8'hac,the dst num is 8'hac
# PASS
# the src num is 8'h0e,the dst num is 8'h0e
# PASS
# the src num is 8'h7d,the dst num is 8'h7d
# PASS
# the src num is 8'h75,the dst num is 8'h75
# PASS
# the src num is 8'hac,the dst num is 8'hac
# PASS
# the src num is 8'h0e,the dst num is 8'h0e
# PASS
# the src num is 8'h7d,the dst num is 8'h7d
# PASS
# the src num is 8'had,the dst num is 8'had
# PASS
# the src num is 8'hff,the dst num is 8'hff
# PASS
# the src written num is 8'h75,the dst read num is 8'hff
# FAIL
# [COMPARETOR]compare is over
# [COMPARETOR]------------------THERE ARE          9 PASS,          1 FAIL----------
```

图 30. 随机化验证时出错

在查看 eeprom 的代码时，发现在读 eeprom 时，ctrl_byte 本应为 r6 的地方变成了 w6，修改后 DUT 功能正确。

```
task read_from_eeprom;
    begin
        shift_in(ctrl_byte);
        if(   ctrl_byte == r7  || ctrl_byte == w6
           || ctrl_byte == r5  || ctrl_byte == r4
           || ctrl_byte == r3  || ctrl_byte == r2
           || ctrl_byte == r1  || ctrl_byte == r0
        ) begin
            address = {addr_byte_h[4:0], addr_byte_l};
```

图 31. EEPROM 中的错误