

VICHARAK

Resource report :

Tcl Console	Messages	Log	Reports	Design Runs	Utilization	x
Hierarchy						
Hierarchy						
Summary						
v Slice Logic						
v Slice LUTs (<1%)						
LUT as Logic (<1%)						
v Slice Registers (<1%)						
Register as Flip						
Memory						

Name	Slice LUTs (20800)	Slice Registers (41600)	Bonded IOB (106)	BUFGCTRL (32)
i2c_master_write	17	13	4	1

Tcl Console	Messages	Log	Reports	Design Runs	Utilization	x
Summary						
Hierarchy						
Summary						
v Slice Logic						
v Slice LUTs (<1%)						
LUT as Logic (<1%)						
v Slice Registers (<1%)						
Register as Flip						
Memory						
DSP						
v IO and GT Specific						

Resource	Utilization	Available	Utilization %
LUT	17	20800	0.08
FF	13	41600	0.03
IO	4	106	3.77

utilization_1

Tcl Console	Messages	Log	Reports	Design Runs	Utilization	x
Primitives						
v Slice LUTs (<1%)						
LUT as Logic (<1%)						
v Slice Registers (<1%)						
Register as Flip						
Memory						
DSP						
v IO and GT Specific						
Bonded IOB (4%)						
v Clocking						
BUFGCTRL (3%)						
Specific Feature						
Primitives						
Black Boxes						

Ref Name	Used	Functional Category
FDRE	12	Flop & Latch
LUT6	7	LUT
LUT5	6	LUT
LUT4	3	LUT
OBUF	2	IO
LUT3	2	LUT
LUT2	2	LUT
IBUF	2	IO
FDSE	1	Flop & Latch
BUFG	1	Clock

Timing reports :

Tcl Console Messages Log Reports Design Runs **Timing** x Utilization

Design Timing Summary

General Information
Timer Settings
Design Timing Summary

> Check Timing (47)
Intra-Clock Paths
Inter-Clock Paths
Other Path Groups
User Ignored Paths
> Unconstrained Paths

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): inf	Worst Hold Slack (WHS): inf	Worst Pulse Width Slack (WPWS): NA
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): NA
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: NA
Total Number of Endpoints: 33	Total Number of Endpoints: 33	Total Number of Endpoints: NA

There are no user specified timing constraints.

Tcl Console Messages Log Reports Design Runs **Timing** x Utilization

Check Timing

General Information
Timer Settings
Design Timing Summary
Check Timing (47)
Intra-Clock Paths
Inter-Clock Paths
Other Path Groups
User Ignored Paths
> Unconstrained Paths

Timing Check	Count	Worst Severity
unconstrained_internal_endpoints	31	High
no_clock	13	High
no_output_delay	2	High
no_input_delay	1	High
constant_clock	0	
pulse_width_clock	0	
multiple_clock	0	
generated_clocks	0	
loops	0	
partial_input_delay	0	
partial_output_delay	0	
latch_loops	0	

Timing Summary - timing_1

Power report :

Tcl Console Messages Log Reports Design Runs **Power** x Timing Utilization

Summary

Settings
Summary (1.544 W, Margin)
Power Supply
Utilization Details
Hierarchical (1.47 W)
Signals (0.25 W)
Data (0.226 W)
Clock Enable (0.02 W)
Set/Reset (0 W)
Logic (0.166 W)
I/O (1.054 W)

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power: 1.544 W
Design Power Budget: Not Specified
Power Budget Margin: N/A
Junction Temperature: 32.7°C
Thermal Margin: 52.3°C (10.4 W)
Effective θ_{JA} : 5.0°C/W
Power supplied to off-chip devices: 0 W
Confidence level: Low
[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

On-Chip Power

Dynamic: 1.470 W (95%)
Device Static: 0.074 W (5%)

17% Signals: 0.250 W (17%)
11% Logic: 0.166 W (11%)
72% I/O: 1.054 W (72%)

Tcl Console

Messages

Log

Reports

Design Runs

Power

×

Timing

Utilization

Q

≡

⚙

↺

»

Power Supply

Settings

Summary (1.544 W, Margin)

Power Supply

Utilization Details

Hierarchical (1.47 W)

Signals (0.25 W)

Data (0.226 W)

Clock Enable (0.02 W)

Set/Reset (0 W)

Logic (0.166 W)

I/O (1.054 W)

Supply Source	Voltage (V)	Total (A)	Dynamic (A)	Static (A)	
Vccint	1.000	0.433	0.420	0.013	
Vccaux	1.800	0.099	0.086	0.013	
Vcco33	3.300	0.000	0.000	0.000	
Vcco25	2.500	0.000	0.000	0.000	
Vcco18	1.800	0.498	0.497	0.001	
Vcco15	1.500	0.000	0.000	0.000	
Vcco135	1.350	0.000	0.000	0.000	
Vcco12	1.200	0.000	0.000	0.000	
Vccaux_io	1.800	0.000	0.000	0.000	
Vccbram	1.000	0.000	0.000	0.000	
MGTAVcc	1.000	0.000	0.000	0.000	
MGTAVtt	1.200	0.000	0.000	0.000	
Vccadc	1.800	0.020	0.000	0.020	

Architecture overview :

This I2C master write module is a state machine that controls the clock (SCL) and data (SDA) lines to talk to an I2C device. The whole thing runs off the main input clock and a reset signal. It's set up to send a hardcoded address (7'h50) and a fixed data value (8'haa) for demonstration.

The logic is driven by a finite state machine (FSM) that walks through each step of the I2C write protocol. It starts idle, then kicks off the transaction with a start condition by pulling SDA low. Next, it shifts out the 7-bit address, one bit per clock cycle, followed by the write bit (which is set to 1 in this case). After that, it sends out each bit of the data byte. The FSM handles the required acknowledgments and finally ends the transaction with a stop condition where SDA goes high again. The counter variable 'count' keeps track of which bit is being sent during the address and data phases.

For address translation, the 7-bit device address is stored in the 'addr' register, and the FSM just shifts out each bit one after another, starting with the most significant bit. Since we're doing a write operation, the read/write bit (which is the least significant bit of the address byte in I2C) is set to 1'b0 to indicate a write. The module doesn't do any complex address mapping - it just sends the address as-is.

State 0 (Idle): The module is at rest. The data line (SDA) is held high, which is the bus's default inactive state.

State 1 (Start): Initiates the transaction. It pulls the SDA line low while the clock (SCL) is high to generate the unique start condition.

State 2 (Addr): Shifts out the 7-bit device address, one bit per clock cycle, starting with the most significant bit (MSB). A counter keeps track of the bit position.

State 3 (RW): Sends the Read/Write bit. Setting this to '0' tells the slave device that this will be a write operation.

State 4 (Wack): Waits for an acknowledgment bit from the slave device. The master releases SDA, and the slave is expected to pull it low.

State 5 (Data): Shifts out the 8-bit data byte, one bit per clock cycle, again starting from the MSB. A counter is used to track the shifting.

State 7 (Wack2): Waits for a final acknowledgment from the slave, confirming it received the data byte successfully.

State 6 (Stop): Ends the transaction. The master ensures SCL is high and then pulls SDA high to create the stop condition, freeing the bus.

Challenges :

First of all, I did not have any idea about I2C. I am in my 5th semester right now and have done only one course in this field "Digital Systems". That's why I could not complete it. Whatever I have done, I just learned it in last 2 weeks from online sources. I am trying to increase my knowledge and experience by doing such internships.

There were a few design challenges here. Getting the SCL clock generation right was tricky - it's gated based on the state machine so the clock only runs during actual data transmission phases and stays high during start, stop, and idle states. Another challenge was properly sequencing the SDA changes to avoid glitches, making sure the data line is stable when the clock is high. The state machine also had to be carefully designed to handle the bit-by-bit shifting of both address and data while maintaining proper I2C timing requirements.

Design code:

```
`timescale 1ns/1ps
module i2c_master_write (
    input clk,
    input reset,
    output i2c_scl,
    output reg i2c_sda);

    reg [7:0] state, count, data;
    reg [6:0] addr;
    reg i2c_scl_enable = 0;
```

```

assign i2c_scl = (i2c_scl_enable == 0) ? 1 : ~clk;

always @(negedge clk) begin
    if (reset == 1) begin
        i2c_scl_enable <= 0;
    end else begin
        if ((state == 0) || (state == 1) || (state == 6)) begin
            i2c_scl_enable <= 0;
        end else begin
            i2c_scl_enable <= 1;
        end
    end
end

always @(posedge clk) begin
    if (reset == 1) begin
        state <= 0;
        i2c_sda <= 1;
        addr <= 7'h50;
        count <= 8'd0;
        data <= 8'haa;
    end

    else begin
        case (state)
            0: begin // idle state
                i2c_sda <= 1;
                state <= 1;
            end

            1: begin // start state
                i2c_sda <= 0;
                state <= 2;
                count <= 6;
            end

            2: begin // addr state
                i2c_sda <= addr[count];
                if (count == 0) state <= 3;
                else count <= count - 1;
            end

            3: begin // RW
                i2c_sda <= 1;

```

```

        state <= 4;
    end

    4: begin                                // wack
        state <= 5;
        count <= 7;
    end

    5: begin                                //data
        i2c_sda <= data[count];
        if (count == 0) state <= 7;
        else count <= count - 1;
    end

    7: state <= 6;                            // wack2
    6: begin                                // stop
        i2c_sda <= 1;
        state <=0;
    end
endcase
end
end
endmodule

```

Test bench code:

```

`timescale 1ns/1ps
module tb_i2c_master_write;
    reg clk, reset;
    wire i2c_sda, i2c_scl;

    i2c_master_write uut(
        .clk(clk),
        .reset(reset),
        .i2c_scl(i2c_scl),
        .i2c_sda(i2c_sda));

    initial begin
        $dumpfile ("dump.vcd");
        $dumpvars (0, tb_i2c_master_write);
    end

    initial begin
        $dumpfile ("dump.vcd");
        $dumpvars (0, tb_i2c_master_write);
    end

```

```

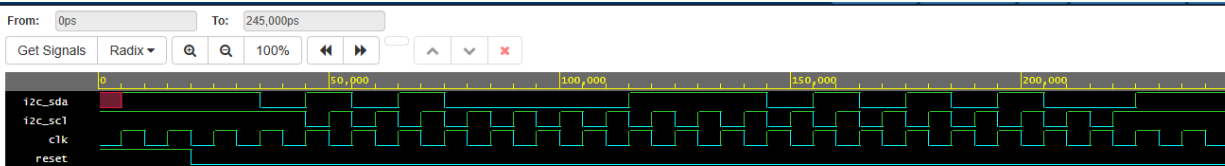
        $dumpvars (1, uut);
    end

    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    initial begin
        reset = 1; #20 reset = 0; #220;
        #5 $finish;
    end
end
endmodule

```

Simulations :



Note: To revert to EPWave opening in a new browser window, set that option on your profile page.