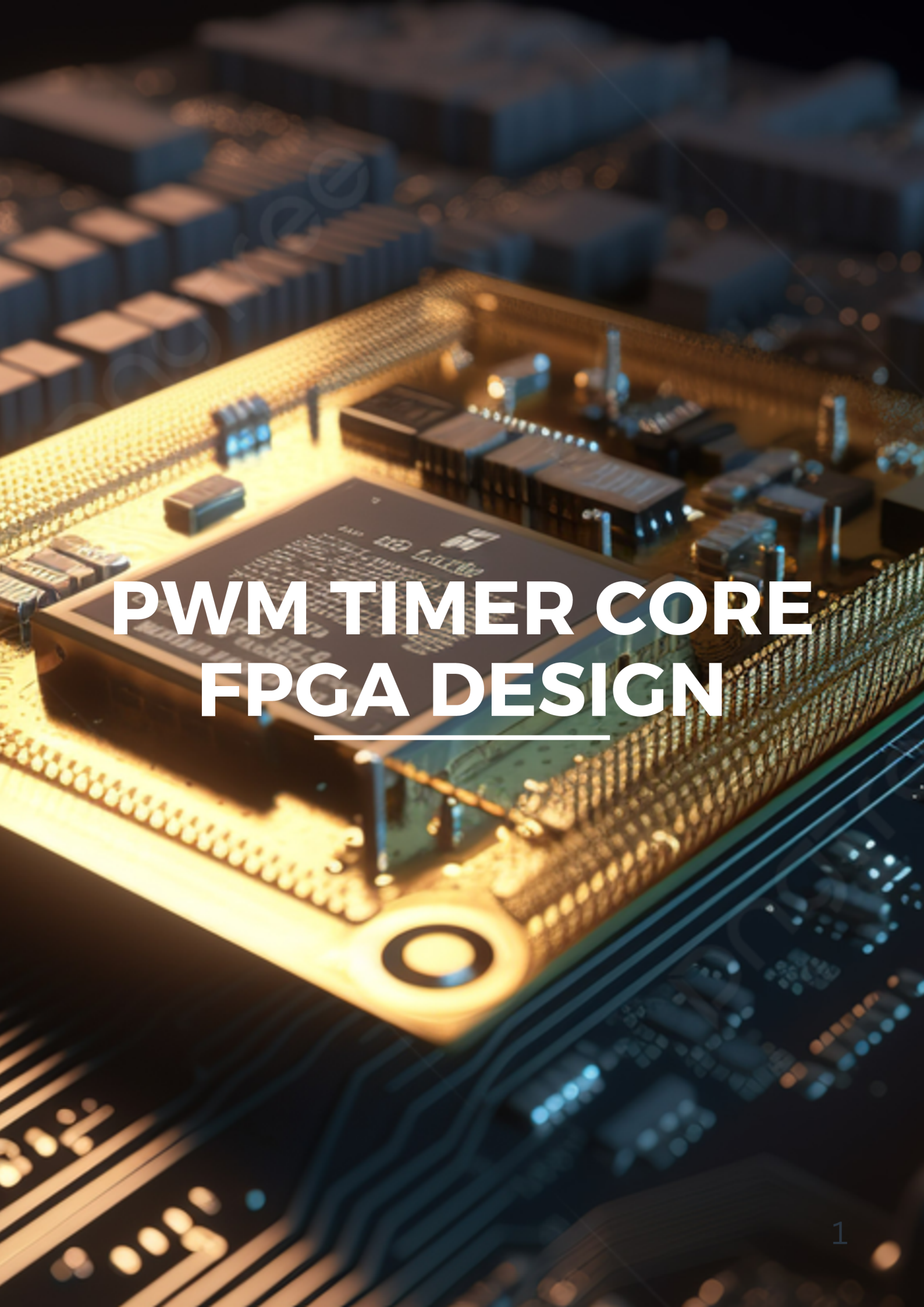
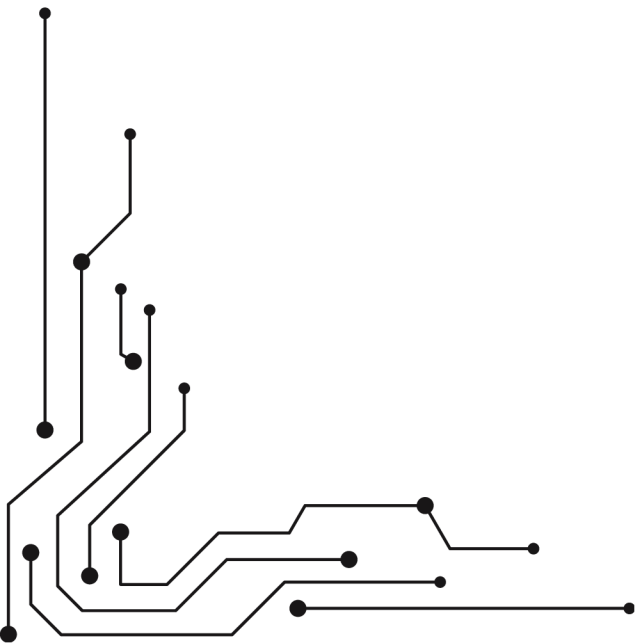


PWM TIMER CORE FPGA DESIGN

Introduction & Project Overview

- Developed a 16-bit Multi-Channel PWM/Timer Core for FPGA.
- Supports up to 4 PWM/Timer channels with independent Period and Duty Cycle.
- Implements Wishbone B4 Slave Interface for runtime control.
- Features down clocking up to 1/65535 of the original frequency.
- Resolved Clock Domain Crossing (CDC) challenges with dual clocks and divided clock using synchronizers.
- Target platform: Artix-7 FPGA with successful Synthesis.



Design Architecture

- Implemented in Verilog with a modular architecture for PWM/Timer core.

```
module PWM_TMR_CORE (
    // Wishbone Slave Interface
    input  wire        CLK_I,
    input  wire        RST_I,
    input  wire [15:0] ADR_I,
    input  wire [15:0] DAT_I,
    output reg  [15:0] DAT_O,
    input  wire        WE_I,
    input  wire        STB_I,
    input  wire        CYC_I,
    output reg         ACK_O,
    // Additional Inputs
    input  wire        EXT_CLK,
    input  wire [15:0] I_DC,
    // Outputs
    output reg  [3:0]  o_pwm   // 4 channels PWM & Interrupt Output
);
```

- Clock Domain Crossing (CDC) using synchronization logic for dual clocks.

```
reg [7:0] ctrl_sync [1:0]; // Synchronize ctrl_reg bits to selected_clk
// dual synchronizer for Ctrl Register
always @(posedge selected_clk or negedge RST_I) begin
    if (!RST_I) begin
        ctrl_sync[0] <= 0;
        ctrl_sync[1] <= 0;
    end
    else begin
        ctrl_sync[0] <= ctrl_reg;
        ctrl_sync[1] <= ctrl_sync[0];
    end
end

// ctrl sync output
wire [7:0] ctrl_sync_out = clk_sel ? ctrl_sync[1] : ctrl_reg;
```

- Integrates Wishbone B4 Slave Interface for register access (Ctrl, Divisor, Period, DC).

```
// Wishbone Write & Read Logic in all registers
always @(posedge CLK_I, negedge RST_I) begin
    if (!RST_I) begin
// reset all registers
        ctrl_reg[7:6]    <= 2'b00;
        ctrl_reg[4:0]    <= 5'b00000;
        divisor_reg      <= 16'h0
        period_reg1      <= 16'h0000;
        period_reg2      <= 16'h0000;
        period_reg3      <= 16'h0000;
        period_reg4      <= 16'h0
        dc_reg1          <= 16'h0000;
        dc_reg2          <= 16'h0000;
        dc_reg3          <= 16'h0000;
        dc_reg4          <= 16'h0
        DAT_O            <= 16'h0000;
        ACK_O            <= 1'b0;
    end
    else if (wb_valid) begin
        if (WE_I) begin     //write operation
            ACK_O <= 1;
            case (addr_low)
                0:  ctrl_reg     <= DAT_I[7:0];
                2:  divisor_reg  <= DAT_I;
                4:  period_reg1  <= DAT_I;
                6:  dc_reg1      <= DAT_I;
                8:  dc_reg2      <= DAT_I;
                10: dc_reg3      <= DAT_I;
                12: dc_reg4      <= DAT_I;
                14: period_reg2  <= DAT_I;
                16: period_reg3  <= DAT_I;
                18: period_reg4  <= DAT_I;
            endcase
        end
        else if (!WE_I) begin  //read operation
            ACK_O <= 1;
            case (addr_low)
                0:  DAT_O       <= {8'h0, ctrl_reg};
                2:  DAT_O       <= divisor_reg;
                4:  DAT_O       <= period_reg1;
                6:  DAT_O       <= dc_used_syn;
                8:  DAT_O       <= dc_reg2;
                10: DAT_O       <= dc_reg3;
                12: DAT_O       <= dc_reg4;
                14: DAT_O       <= period_reg2;
                16: DAT_O       <= period_reg3;
                18: DAT_O       <= period_reg4;
                default: DAT_O  <= 16'h0000;
            endcase
        end
    end
    else if (counter_rst) begin //
        ctrl_reg[5]   <= 0;
    end
    else if ((main_counter1 == (period_sync_out1-1)) && !mode_pwm) begin //tmr mode
        ctrl_reg[5] <= 1;
    end

    else ACK_O <= 0;
end
```

- 4-Main counters (16-bit) resets via system or manual control.

```
// Main Counters
always @(posedge DIV_CLK) begin
    if (counter_rst) begin
        main_counter1 <= 0;
        main_counter2 <= 0;
        main_counter3 <= 0;
        main_counter4 <= 0;
        // ctrl_reg[5]   <= 0;    //interrupt Flag

    end
    else if (counter_en && cont_count) begin
        main_counter1 <= main_counter1 + 1'b1;
        if ((main_counter1 == (period_sync_out1-1)) && !mode_pwm) begin //tmr mode
            main_counter1 <= 0;
        end
        else if (mode_pwm) begin //pwm mode
            main_counter2 <= main_counter2 + 1'b1;
            main_counter3 <= main_counter3 + 1'b1;
            main_counter4 <= main_counter4 + 1'b1;
            if (main_counter1 == (period_sync_out1-1))  main_counter1 <= 0;
            if (main_counter2 == (period_sync_out2-1))  main_counter2 <= 0;
            if (main_counter3 == (period_sync_out3-1))  main_counter3 <= 0;
            if (main_counter4 == (period_sync_out4-1))  main_counter4 <= 0;
            end
        end
    end
```

- Supports two modes: PWM with 4 independent channels and Timer with interrupt.

```
// PWM/timer Outputs Logic
always @(posedge selected_clk or negedge RST_I) begin
    if (!RST_I || counter_rst) begin
        o_pwm[0] <= 1'b0;
        o_pwm[1] <= 1'b0;
        o_pwm[2] <= 1'b0;
        o_pwm[3] <= 1'b0;
    end
    else if (mode_pwm && pwm_en && counter_en) begin
        o_pwm[0] <= (main_counter1 < dc_used_syn);      // PWM channel 1 output in PWM Mode
        o_pwm[1] <= (main_counter2 < dc_sync_out2);     // PWM channel 2 output in PWM Mode
        o_pwm[2] <= (main_counter3 < dc_sync_out3);     // PWM channel 3 output in PWM Mode
        o_pwm[3] <= (main_counter4 < dc_sync_out4);     // PWM channel 4 output in PWM Mode
    end
    else if (!mode_pwm && interrupt_flag) begin
        o_pwm[0] <= 1'b1; // Interrupt output in Timer Mode ch1 only
    end
end
```

- Includes down clocking logic with a 16-bit divisor register using integer clock divider module that instantiated in the top.

```
module CLK_Division(
input  wire         ref_clk ,
input  wire         rst ,
input  wire         clk_En ,
input  wire  [15:0] Div_rat ,
output wire         Div_Clk
);
//////////////////////////////////////////////////////////////////////////////////
wire [6:0] half_period ;
wire [6:0] full_period ;
wire       clk_En_in ;
wire       odd ;
reg        flag ;
reg  [6:0] Counter ;
reg        Div_Clk_O ;


//////////////////////////////////////////////////////////////////////////////////
assign half_period = ((Div_rat >> 1) - 1 );
assign full_period = (Div_rat >> 1) ;
assign odd = Div_rat[0] ;
assign Div_Clk = !clk_En ? ref_clk : Div_Clk_O ; //output logic
//////////////////////////////////////////////////////////////////////////////////

always@(posedge ref_clk or posedge rst) begin
    if (rst)begin
        Div_Clk_O <= 1'b0 ;
        Counter <= 7'b0 ;
        flag <= 1'b0 ;
    end
    else if (clk_En) begin
        if (!odd && (Counter == half_period)) begin
            Counter <= 0 ;
            Div_Clk_O <= ~Div_Clk_O ;
        end
        else if ((odd && (Counter == half_period) && flag) || (odd && (Counter == full_period) && !flag)) begin
            Counter <= 0 ;
            Div_Clk_O <= ~Div_Clk_O ;
            flag <= ~flag ;
        end
        else begin
            Counter <= Counter + 1'b1 ;
        end
    end
end

endmodule
```

Testbench & Test Cases

- Code Snippet for important Testbench Tasks

```
// write task by value and the address of the register
task WRITE(input [4:0] address, input [15:0] value);
begin
    @(negedge CLK_I);
    ADR_I[4:0] = address;
    DAT_I      = value;
    CYC_I      = 1;
    STB_I      = 1;
    WE_I       = 1;
    @(negedge CLK_I);
    CYC_I      = 0;
    STB_I      = 0;
end
endtask

// read task by the address of the register
task READ(input [4:0] address);
begin
    ADR_I[4:0] = address;
    CYC_I      = 1;
    STB_I      = 1;
    WE_I       = 0;
    @(negedge CLK_I);
    CYC_I      = 0;
    STB_I      = 0;
end
endtask
```

Timer mode task

```
//timer mode task
task TIMER_MODE_CONFG(input [15:0] PERIOD, input [7:0] TIMER_CONFIG_TYPE, input [15:0] Divider, input Interrupt_Clear);
begin
    //write on the control registers
    WRITE(2, Divider);
    /* should be the first one to start the main counter with the correct clk & time ignoring the glitch
    that caused by changing the period at the same edge */
    WRITE(4, PERIOD);
    WRITE(0, TIMER_CONFIG_TYPE); // should be the last one to start the main counter at the suitable time

    //more counting to compensate for the clk divider that changes the main counter period
    period_i = (Divider == 0 || Divider == 1)? PERIOD: (PERIOD*Divider);  //clk divider problem solution
    for(i = 0; i < (period_i); i = i + 1) begin
        @(negedge CLK_I);
        // request to clear the interrupt
        if(i == (period_i - 1)) begin
            if(Interrupt_Clear)  begin
                if(TIMER_CONFIG_TYPE[0]) begin // if ext clk (cdc) need more time for synchronizers latency
                    #100; //more waiting until the interrupt flag rises before the reading operation
                    @(negedge CLK_I);
                    READ(0);
                    end
                    else begin
                        @(negedge CLK_I);
                        READ(0);
                    end
                    WRITE(0, (DAT_O_DUT & ~(1 << 5))); //Interrupt clear
                    //delay for sync at two clock domains condition
                    @(negedge CLK_I);
                    @(negedge CLK_I);
                    @(negedge CLK_I);
                end
            end
        end
    end
endtask
```

4-channel Pwm mode task

```
//pwm mode task
task PWM_MODE_CONFG(input [63:0] PERIOD, input [63:0] DC, input [7:0] PWM_CONFIG_TYPE, input [15:0] Divider);
begin
    //write on the control registers for 4 channels
    WRITE(4,  PERIOD[15:0]);    // address of period reg 1
    WRITE(14, PERIOD[31:16]);   // address of period reg 2
    WRITE(16, PERIOD[47:32]);   // address of period reg 3
    WRITE(18, PERIOD[63:48]);   // address of period reg 4
    WRITE(2,  Divider);         // address of divisor reg
    WRITE(6,  DC[15:0]);        // address of dc reg 1
    WRITE(8,  DC[31:16]);       // address of dc reg 2
    WRITE(10, DC[47:32]);       // address of dc reg 3
    WRITE(12, DC[63:48]);       // address of dc reg 4
    WRITE(0, PWM_CONFIG_TYPE);  // should be the last one to start the main counter at the suitable time
    @(negedge CLK_I);           // writing delay
end
endtask
```

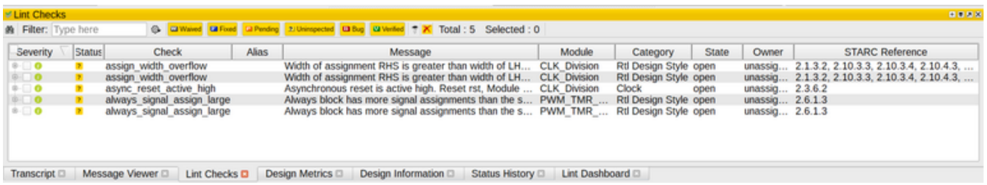
- Code Snippet for test cases

```
initial begin
    //INITIALIZATION
    INITIALIZATION;
    //TEST CASE 0
    RST_ASSERTION;
    //TEST CASE 1
    WRITE(4, 16'h0013);
    //TEST CASE 2
    READ(4);
    //TEST CASE 3
    TIMER_MODE_CONFG(10, 8'b0001_0100, 16'h0000, 1'b0); //Stop timer runs continuously (without Clear the interrupt flag or enable cont bit) Watch the main counter
    #50
    //TEST CASE 4
    TIMER_MODE_CONFG(15, 8'b0001_1100, 16'h0001, 1'b0); //Cont timer runs continuously (without Clear the interrupt flag but enable cont bit) Watch the main counter
    #50
    //TEST CASE 5
    WRITE(0, (8'b0001_1100 | (1 << 7))); //set counter reset
    TIMER_MODE_CONFG(20, 8'b0001_0100, 16'h0000, 1'b1); //Cont timer runs continuously (without enable cont bit but Clear the interrupt flag) Watch the main counter
    #100
    //TEST CASE 6 (DC = 25%)
    WRITE(0, (8'b0001_0110 | (1 << 7))); //set counter reset when we change between the modes
    PWM_MODE_CONFG(64'h0008_000c_0010_0014, 64'h0002_0003_0004_0005, 8'b0001_0110, 16'h0000);
    #300
    //TEST CASE 7 (DC = 50%)
    PWM_MODE_CONFG(64'h0008_000c_0010_0014, 64'h0004_0006_0008_000a, 8'b0001_0110, 16'h0001); //DC = 50%
    #300
    //TEST CASE 8 (DC = 75%) With Using External DC REG for channel 1
    I_DC = 15; // will ignore the dc reg value for channel 1
    PWM_MODE_CONFG(64'h0008_000c_0010_0014, 64'h0006_0009_000c_0000, 8'b0101_0110, 16'h0001);
    #300
```

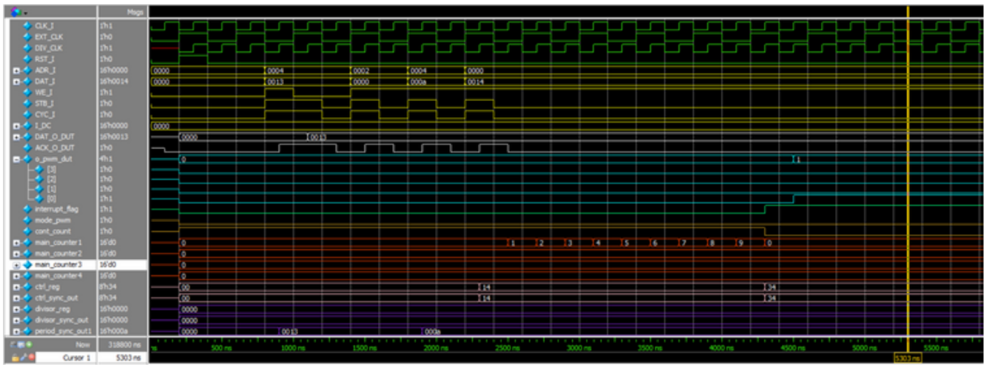
```
    //TEST CASE 9 (DC = 90%) With Using External DC REG for channel 1 && divider by 2
    WRITE(0, (8'b0101_0110 | (1 << 7))); //set counter reset
    I_DC = 16'h0009; // will ignore the dc reg value for channel 1
    PWM_MODE_CONFG(64'h0014_000a_001e_000a, 64'h0012_0009_001b_0000, 8'b0101_0110, 16'h0002);
    #300
    //TEST CASE 10 (DC = 40%) With Using Internal DC REG for channel 1 && divider by 10
    WRITE(0, (8'b0001_0110 | (1 << 7))); //set counter reset (16'h0095)
    I_DC = 16'h002d; // will ignore the I_DC reg value for channel 1
    PWM_MODE_CONFG(64'h001e_0014_000a_0005, 64'h000c_0008_0004_0002, 8'b0001_0110, 16'h000a);
    #300
    /************************Using Two Clock Domains*****************************/
    //TEST CASE 11 (DC = 50%) With Using Internal DC REG for channel 1 && divider by 8
    WRITE(0, (8'b0001_0111 | (1 << 7))); //set counter reset
    I_DC = 16'h0008; // will ignore the I_DC value for channel 1
    PWM_MODE_CONFG(64'h0008_000c_0010_0014, 64'h0004_0006_0008_000a, 8'b0001_0111, 16'h0008);
    #500
    //TEST CASE 12
    WRITE(0, (8'b0001_0101 | (1 << 7))); //set counter reset
    TIMER_MODE_CONFG(10, 8'b0001_0101, 16'h0001, 1'b1); //Cont timer runs continuously (without enable cont bit but Clear the interrupt flag) Watch the main counter
    #200
    //TEST CASE 13
    WRITE(0, (8'b0001_0101 | (1 << 7))); //set counter reset
    TIMER_MODE_CONFG(10, 8'b0001_1101, 16'h000, 1'b0); //Cont timer runs continuously (without Clear the interrupt flag but enable cont bit) Watch the main counter
    #200
    $stop ;
end
```

Simulation Results

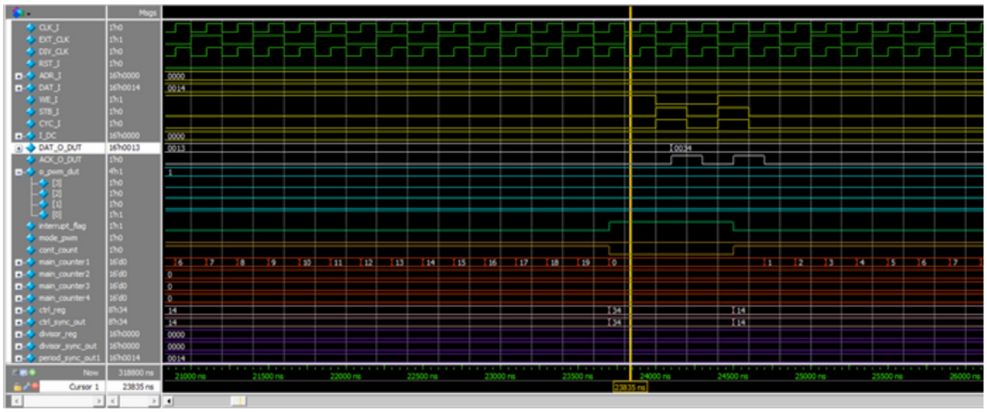
Questa Lint & Questa Sim results



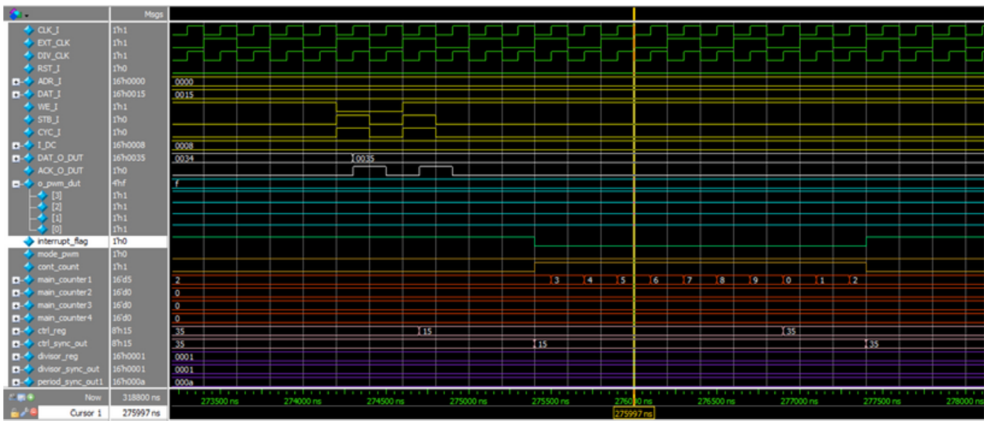
Timer Mode without clear the flag using wishbone clock for the output



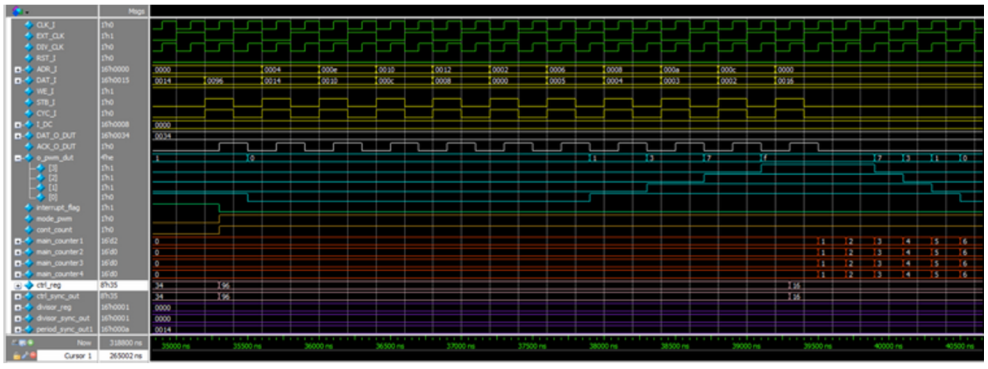
Timer Mode with clear the flag using wishbone clock for the output



Timer Mode with clear the flag using external clock for the output that let us used the synchronisers to solve cdc problem that made latency to clear the interrupt

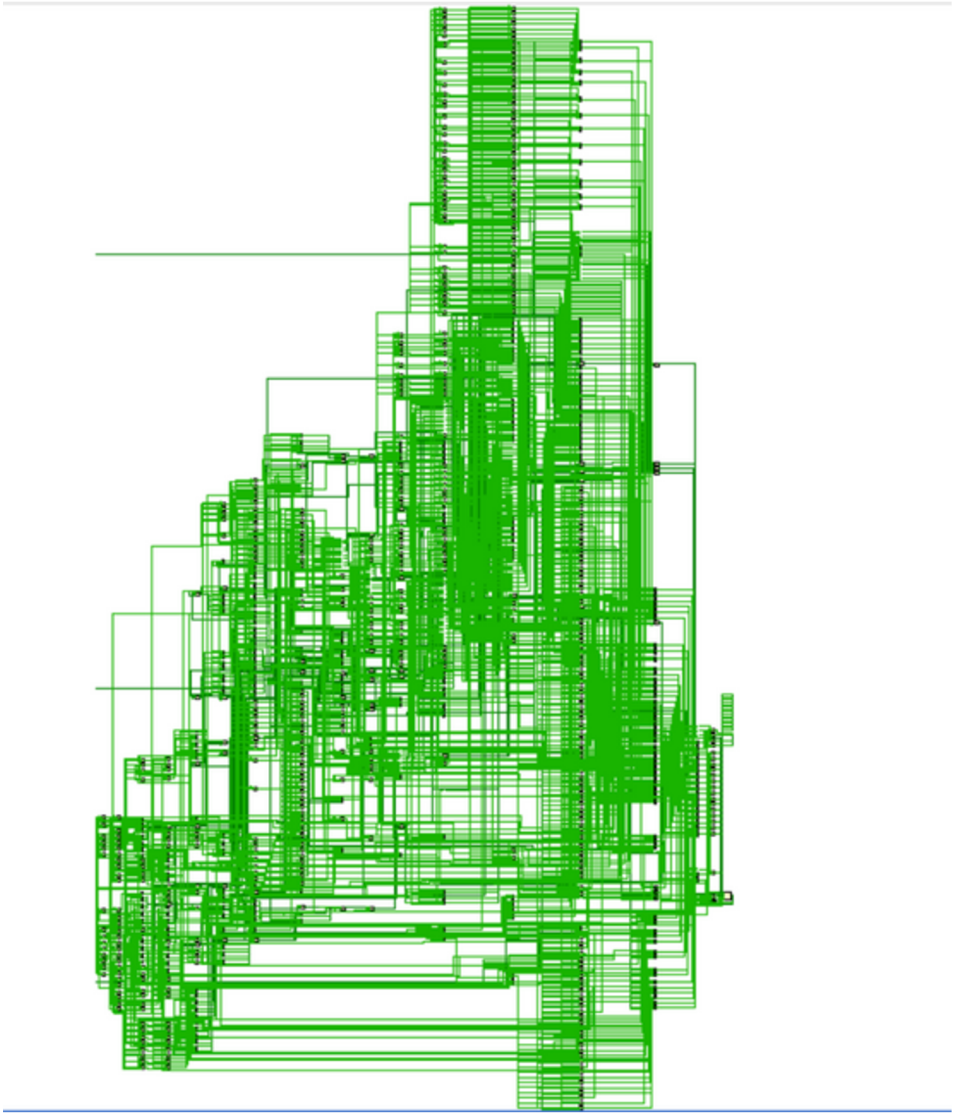


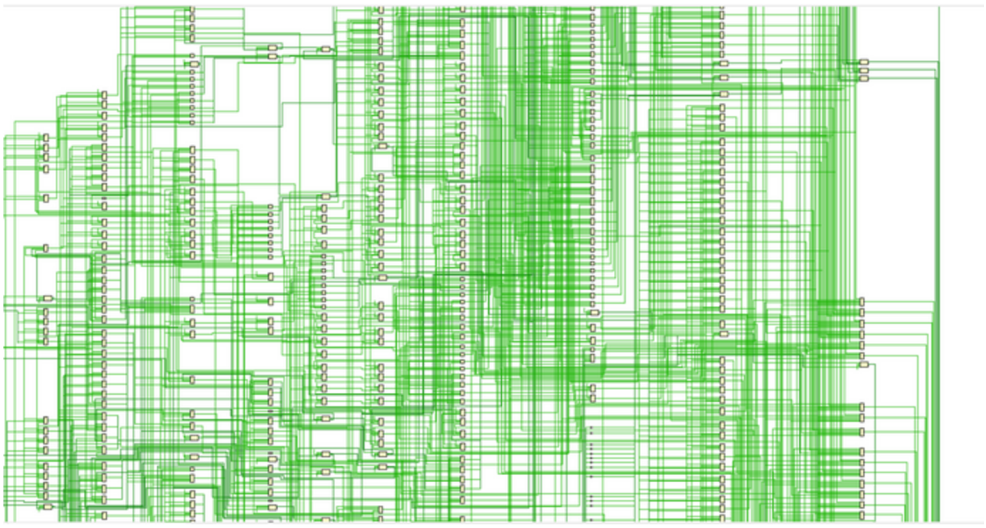
Configure Pwm mode using the 4-channels by writing on the registers with different duty cycles for each one



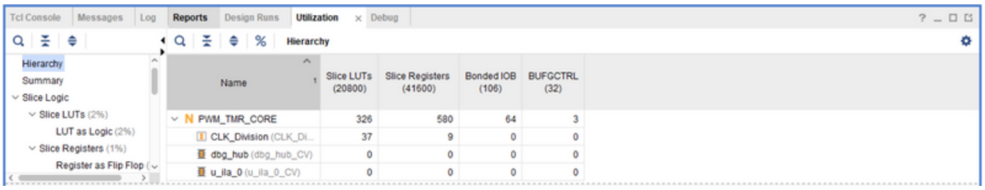
Synthesis Results

- **Schematic**

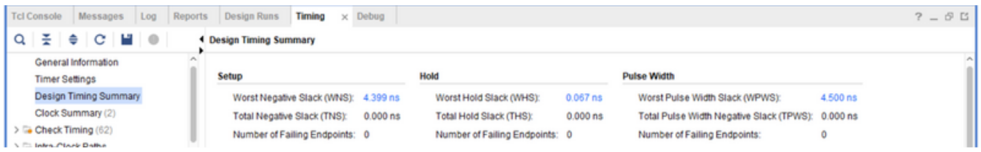




Utilization Report



Timing Report



Challenges & Lessons Learned

- Handled multiple clocks in the design (Wishbone clock, external clock, and clock divider output), leading to complex timing issues.
- Artix-7 FPGA has limited clock pins (only one dedicated clock pin available), which prevented full Implementation due to IO constraints.
- Clock Domain Crossing (CDC) required careful management to avoid metastability between Wishbone clock and external clock.

- Implemented synchronizers to resolve CDC between dual clocks, but this introduced an impact on Hold Time.
- Future designs should target FPGAs with more clock resources for seamless Implementation.