# Pulse Detector for specific frequencies in Verilog

Aristotle University of Thessaloniki (AUTH)

*Course: Digital Systems*

Mastoreka Maria

19/06/2025

ΤΜΗΜΑ ΦΥΣΙΚΗΣ Α.Π.Θ.
Πρόγραμμα Μεταπτυχιακών Σπουδών
**Ηλεκτρονικής Φυσικής**
*(Ραδιοηλεκτρολογίας)*

## Project goal

- ➢ Detect the frequency of an incoming digital signal
- ➢ Valid frequency values: *10 MHz, or 20 MHz, 50 MHz.*

**Challenges**

- Digital circuits cannot directly detect frequency — they react to pulses
- The input signal is asynchronous to the system clock. → Without synchronization, metastability may occur

We must:

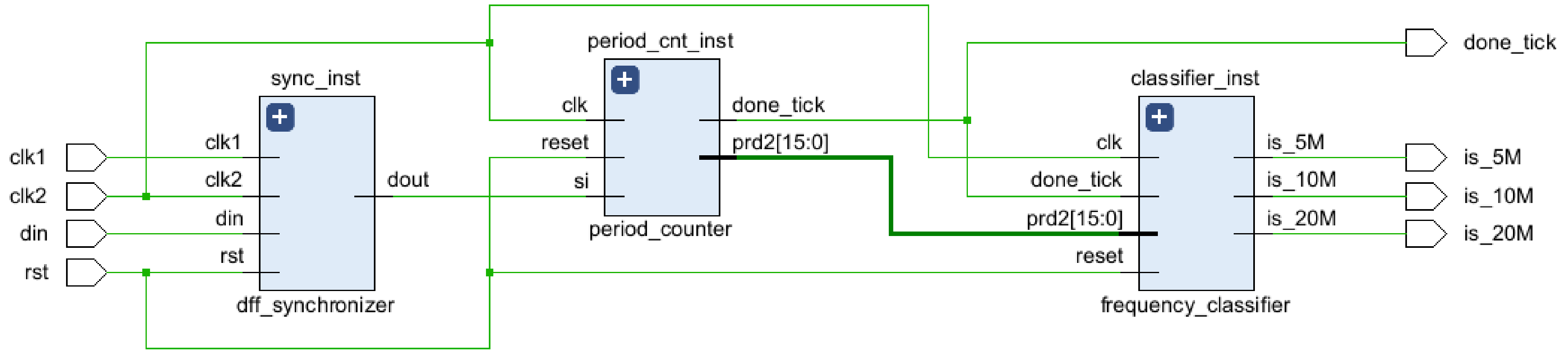
1. Detect the time between rising edges (period)
2. Classify the signal based on this period

# System Architecture & Approach

- ✓ Synchronize the input and avoid metastability
- ✓ Detect rising edges (0 → 1 transitions)
- ✓ Measure the time between edges (signal period)
- ✓ Classify the period into a frequency range (50 MHz, 10 MHz, 20 MHz)
- ✓ Activate the corresponding output signal

**Modules**

1. Dual Flip Flop Synchronizer
2. Period counter
3. Frequency Classifier



# Dff Synchronizer

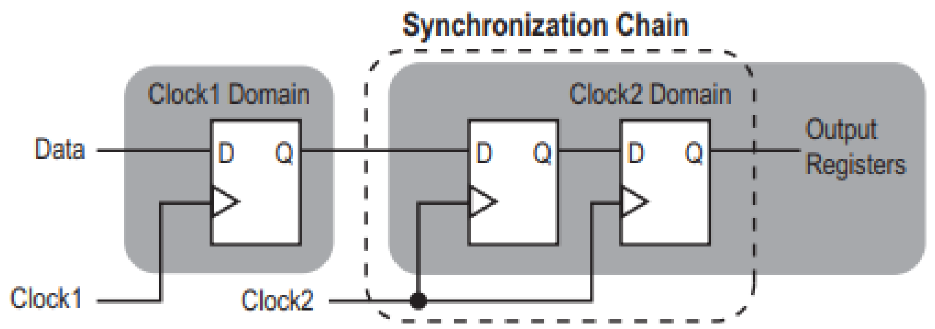
*Flip-flops require proper timing: setup time (tsu) and hold time (th). If these constraints are violated, metastability may occur.*

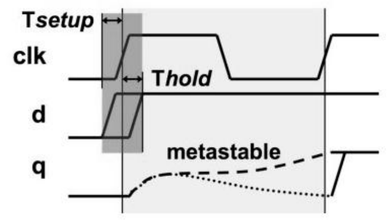
**Metastability**

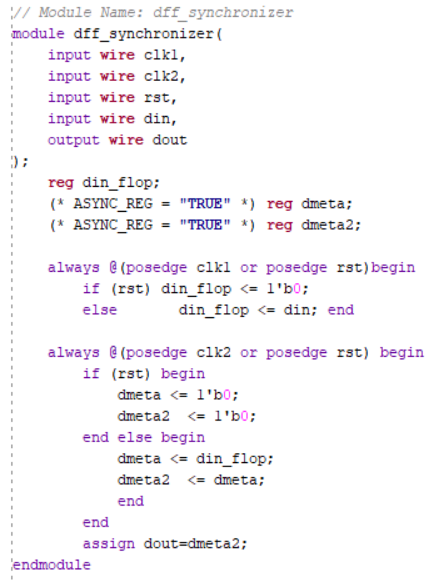
- Occurs when the input of a flip-flop changes very close to a clock edge
- The output may become unstable or undefined
- A common issue with asynchronous signals from external sources

**Solution : Dual Flip-Flop Synchronizer**

- The first flip-flop captures the asynchronous input
- The second delays the signal by one clock cycle, allowing it to stabilize. This reduces the probability of metastability errors
- The more flip-flops added, the lower the error probability, but also the longer the delay
- In this design: transition from slower (clk1) to faster clock (clk2)

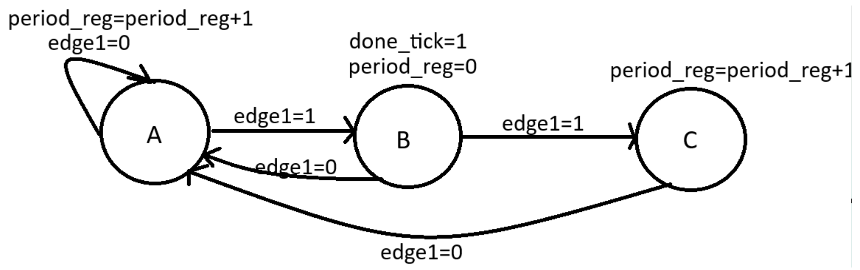




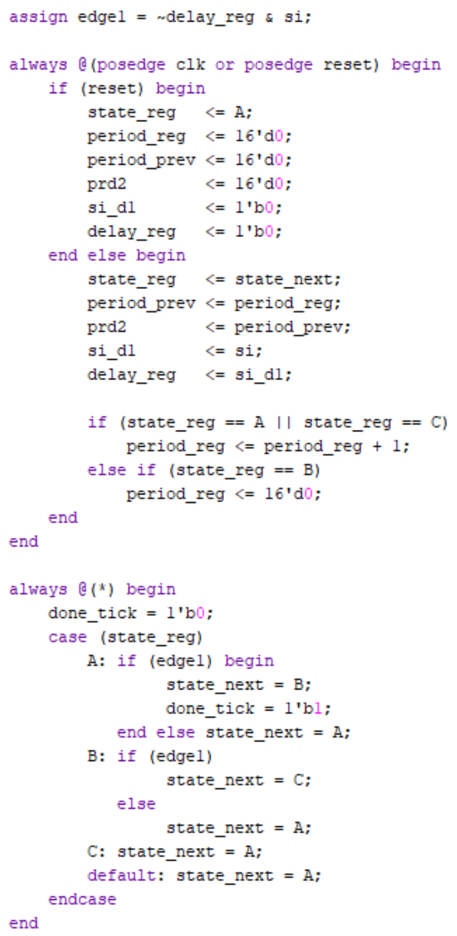


```
// Module Name: dff_synchronizer
module dff_synchronizer(
    input wire clk1,
    input wire clk2,
    input wire rst,
    input wire din,
    output wire dout
);
    reg din_flop;
    (* ASYNC_REG = "TRUE" *) reg dmeta;
    (* ASYNC_REG = "TRUE" *) reg dmeta2;

    always @(posedge clk1 or posedge rst)begin
        if (rst) din_flop <= 1'b0;
        else       din_flop <= din; end

    always @(posedge clk2 or posedge rst) begin
        if (rst) begin
            dmeta <= 1'b0;
            dmeta2  <= 1'b0;
        end else begin
            dmeta <= din_flop;
            dmeta2  <= dmeta;
            end
        end
        assign dout=dmeta2;
endmodule
```

# Period Counter



*This module measures the period of the input pulse signal.*



```verilog
assign edgel = ~delay_reg & si;

always @(posedge clk or posedge reset) begin
    if (reset) begin
        state_reg   <= A;
        period_reg  <= 16'd0;
        period_prev <= 16'd0;
        prd2        <= 16'd0;
        si_dl       <= 1'b0;
        delay_reg   <= 1'b0;
    end else begin
        state_reg   <= state_next;
        period_prev <= period_reg;
        prd2        <= period_prev;
        si_dl       <= si;
        delay_reg   <= si_dl;

        if (state_reg == A || state_reg == C)
            period_reg <= period_reg + 1;
        else if (state_reg == B)
            period_reg <= 16'd0;
    end
end

always @(*) begin
    done_tick = 1'b0;
    case (state_reg)
        A: if (edgel) begin
                state_next = B;
                done_tick = 1'b1;
           end else state_next = A;
        B: if (edgel)
                state_next = C;
           else
                state_next = A;
        C: state_next = A;
        default: state_next = A;
    endcase
end
```

**Key Features**

1. Rising Edge Detection using delay registers (delay_reg, si_d1) and comparison logic
2. Implemented as a 3-state FSM, since delay_reg represents si delayed by 2 clock cycles
   - ✓ Rising edge is detected by identifying the 110 sequence in the delayed signal
3. The internal counter increments on every clock cycle, measuring the time between two rising edges
4. The measured period is stored in prd2
5. Done_tick is activated when a new measurement is available

# Frequency Classifier

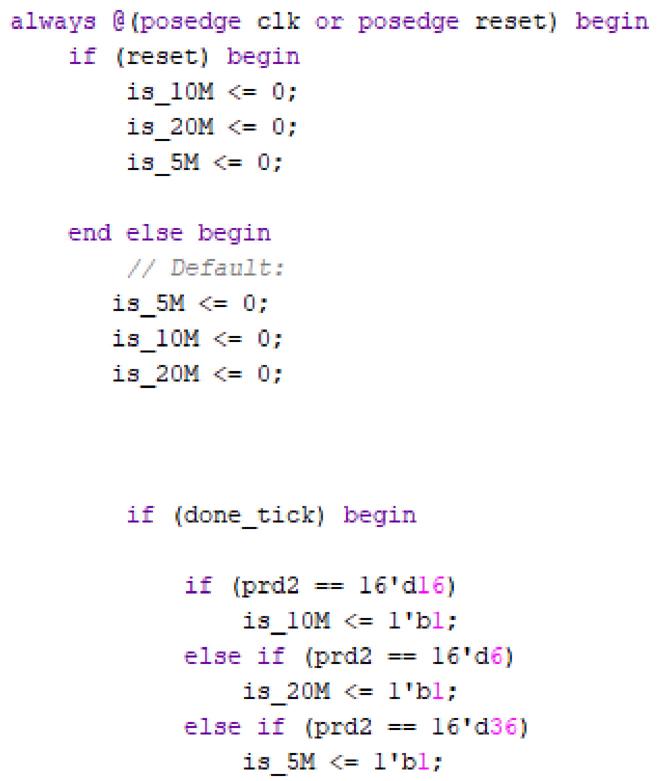
*This module classifies the signal frequency based on the period measured by the period_counter.*

**Key Features:**

- The period value (prd) is used to estimate the input frequency
- Once done_tick is activated, it checks if the measured period corresponds to one of the predefined frequency ranges
- The signal is classified into 10 MHz, or 20 MHz, 50 MHz.
- The corresponding output flag is asserted (is_20M, is_10M, is_50M)

<u>Example Calculation:</u>

$$\text{Prd} = \frac{T_{signal}}{T\_clock} \quad \text{For 50 MHz} \rightarrow \text{prd} = \frac{50ns}{5ns} = 10$$

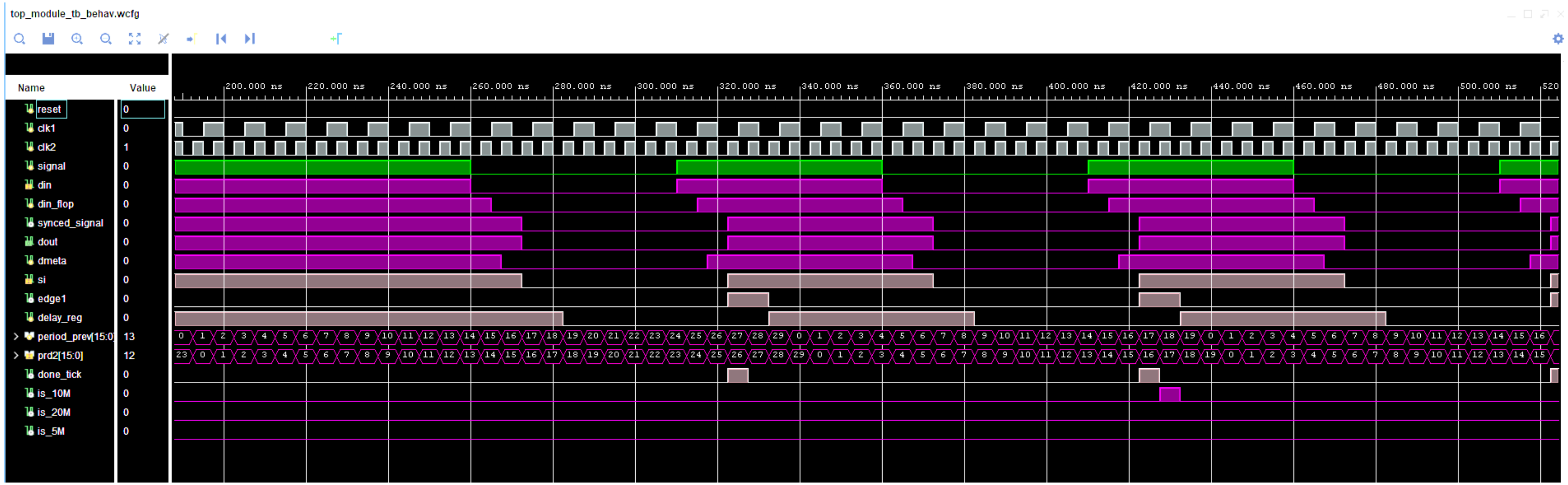


```
always @(posedge clk or posedge reset) begin
    if (reset) begin
        is_10M <= 0;
        is_20M <= 0;
        is_5M <= 0;

    end else begin
        // Default:
      is_5M <= 0;
      is_10M <= 0;
      is_20M <= 0;



        if (done_tick) begin

            if (prd2 == 16'd16)
                is_10M <= 1'b1;
            else if (prd2 == 16'd6)
                is_20M <= 1'b1;
            else if (prd2 == 16'd36)
                is_5M <= 1'b1;
```

# Simulation (1/3)

**10 MHz Input Signal**

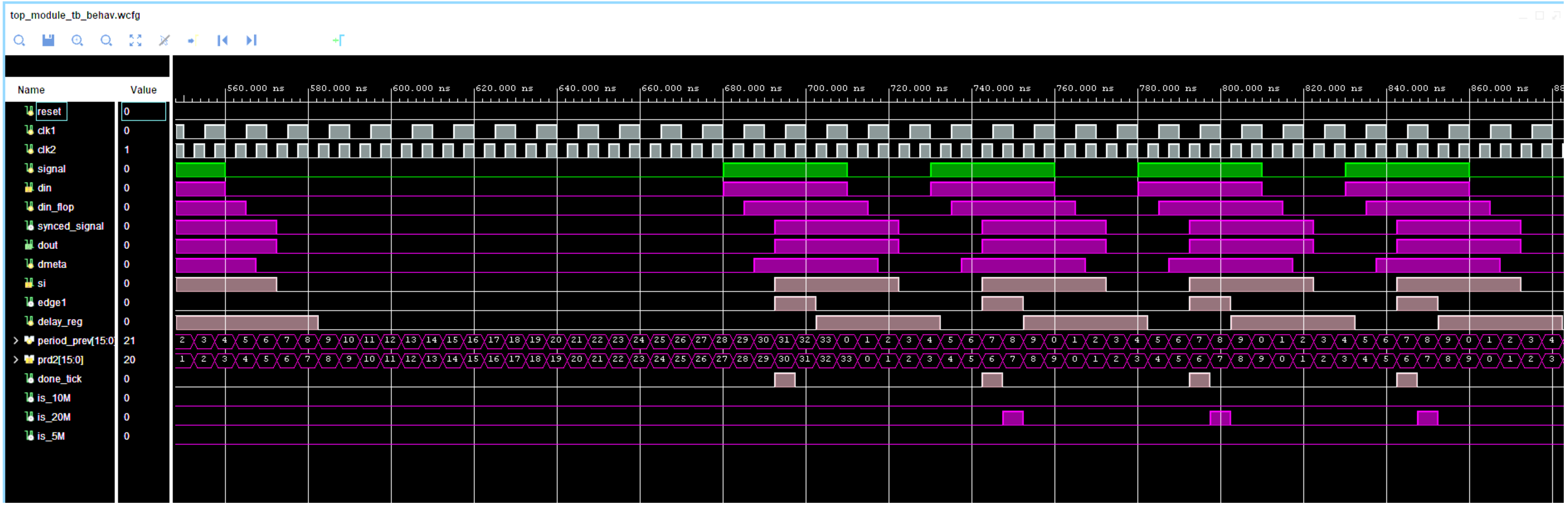
- The classifier correctly identifies the signal as 10 MHz and activates the is_10M output flag
- The result confirms correct detection and classification logic


top_module_tb_behav.wcfg

| Name | Value |
|---|---|
| reset | 0 |
| clk1 | 0 |
| clk2 | 1 |
| signal | 0 |
| din | 0 |
| din_flop | 0 |
| synced_signal | 0 |
| dout | 0 |
| dmeta | 0 |
| si | 0 |
| edge1 | 0 |
| delay_reg | 0 |
| period_prev[15:0] | 13 |
| prd2[15:0] | 12 |
| done_tick | 0 |
| is_10M | 0 |
| is_20M | 0 |
| is_5M | 0 |

# Simulation (2/3)

**20 MHz Input Signal**

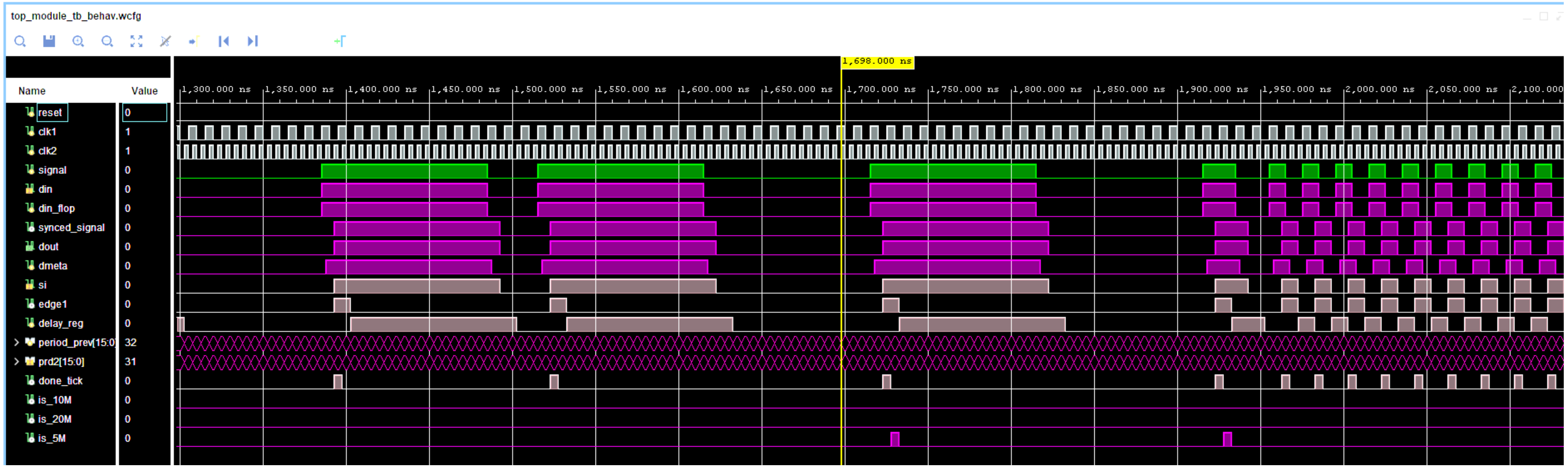


top_module_tb_behav.wcfg

| Name | Value |
|---|---|
| reset | 0 |
| clk1 | 0 |
| clk2 | 1 |
| signal | 0 |
| din | 0 |
| din_flop | 0 |
| synced_signal | 0 |
| dout | 0 |
| dmeta | 0 |
| si | 0 |
| edge1 | 0 |
| delay_reg | 0 |
| period_prev[15:0] | 21 |
| prd2[15:0] | 20 |
| done_tick | 0 |
| is_10M | 0 |
| is_20M | 0 |
| is_5M | 0 |

# Simulation (3/3)

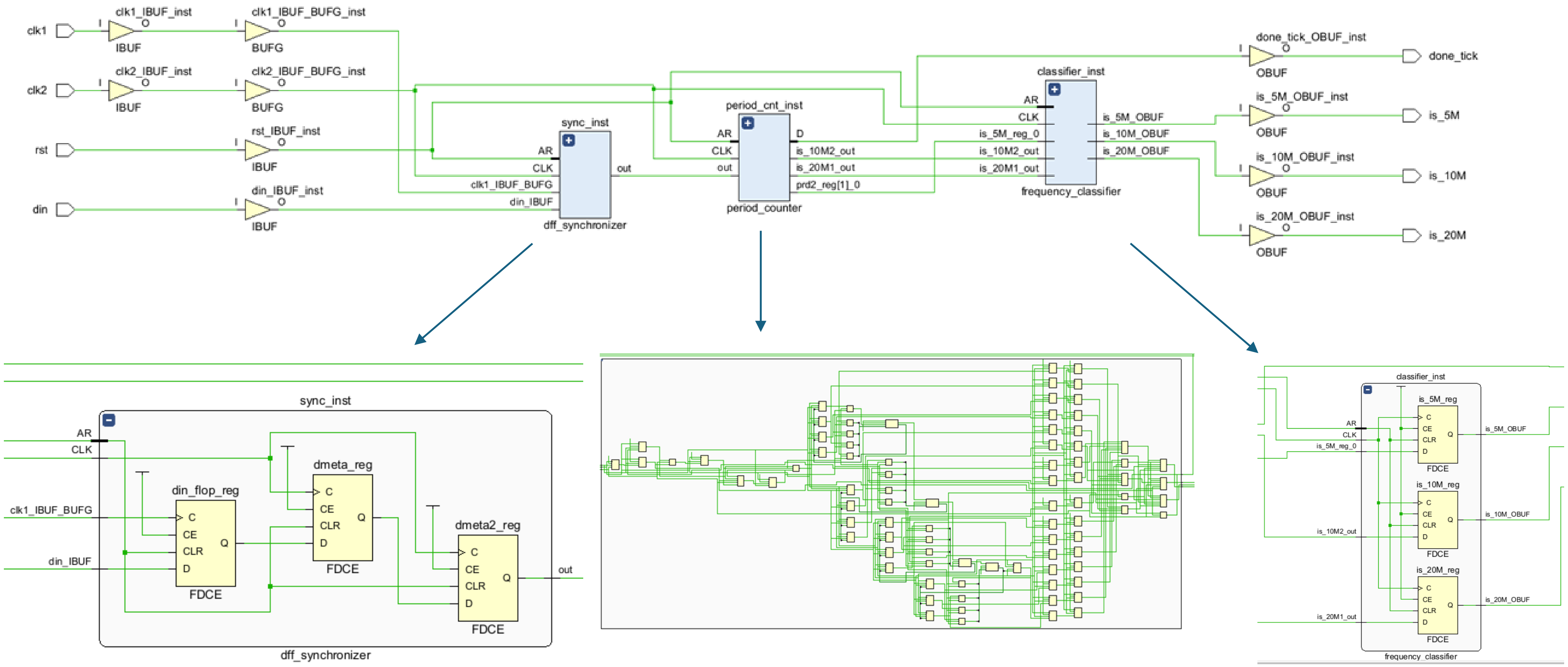
**50 MHz Input Signal**



| Name | Value |
|---|---|
| reset | 0 |
| clk1 | 1 |
| clk2 | 1 |
| signal | 0 |
| din | 0 |
| din_flop | 0 |
| synced_signal | 0 |
| dout | 0 |
| dmeta | 0 |
| si | 0 |
| edge1 | 0 |
| delay_reg | 0 |
| period_prev[15:0] | 32 |
| prd2[15:0] | 31 |
| done_tick | 0 |
| is_10M | 0 |
| is_20M | 0 |
| is_5M | 0 |

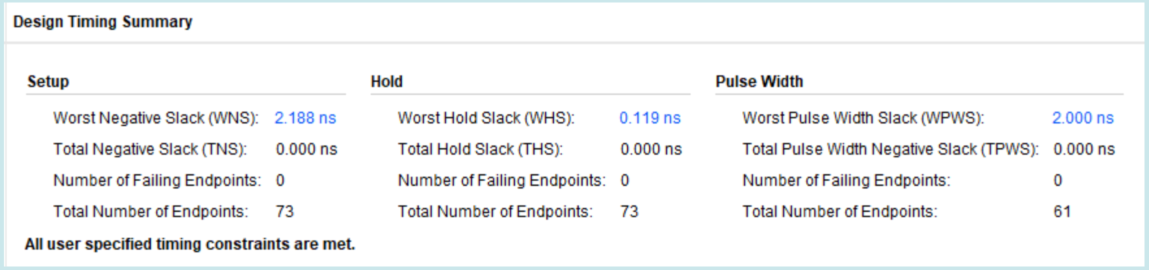
# Synthesis and Implementation

## Schematic



# Results

## Timing Results

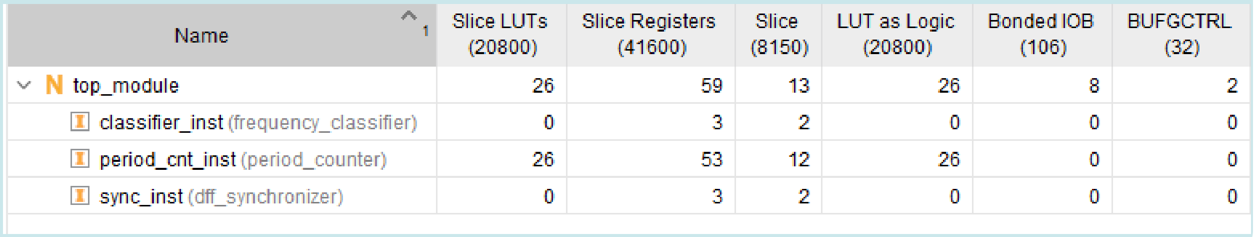


**Design Timing Summary**

| **Setup** | | **Hold** | | **Pulse Width** | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 2.188 ns | Worst Hold Slack (WHS): | 0.119 ns | Worst Pulse Width Slack (WPWS): | 2.000 ns |
| Total Negative Slack (TNS): | 0.000 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 73 | Total Number of Endpoints: | 73 | Total Number of Endpoints: | 61 |

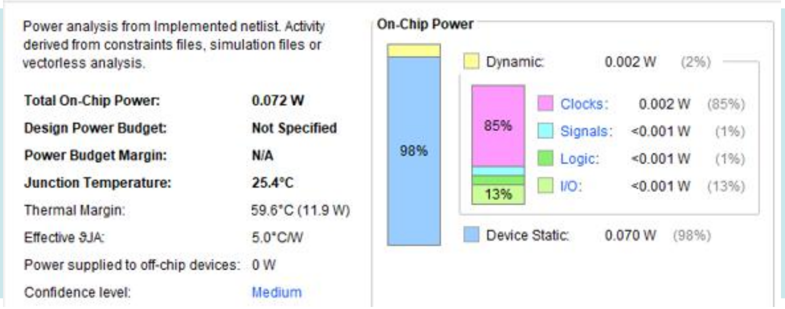
**All user specified timing constraints are met.**

## Resource Utilization



| Name | Slice LUTs (20800) | Slice Registers (41600) | Slice (8150) | LUT as Logic (20800) | Bonded IOB (106) | BUFGCTRL (32) |
|---|---|---|---|---|---|---|
| top_module | 26 | 59 | 13 | 26 | 8 | 2 |
| classifier_inst (frequency_classifier) | 0 | 3 | 2 | 0 | 0 | 0 |
| period_cnt_inst (period_counter) | 26 | 53 | 12 | 26 | 0 | 0 |
| sync_inst (dff_synchronizer) | 0 | 3 | 2 | 0 | 0 | 0 |

## Power Consumption



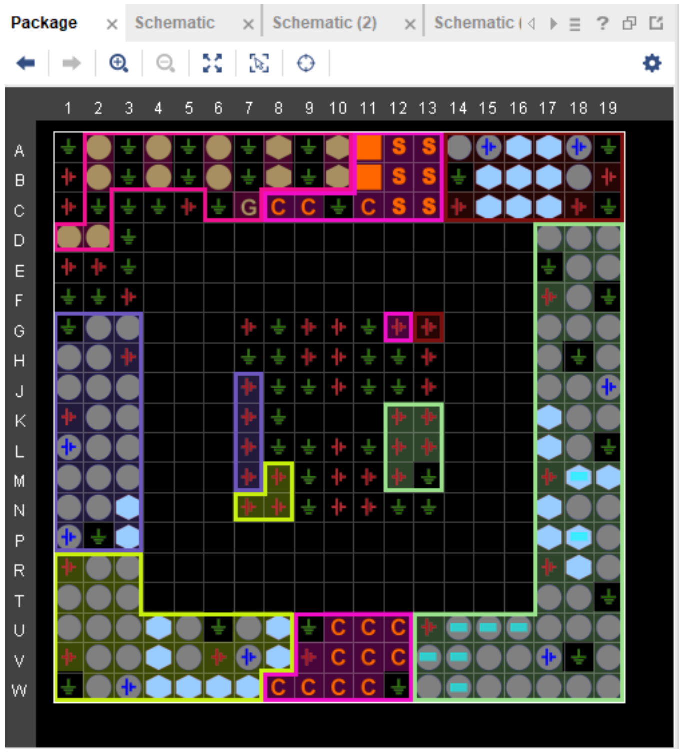
Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

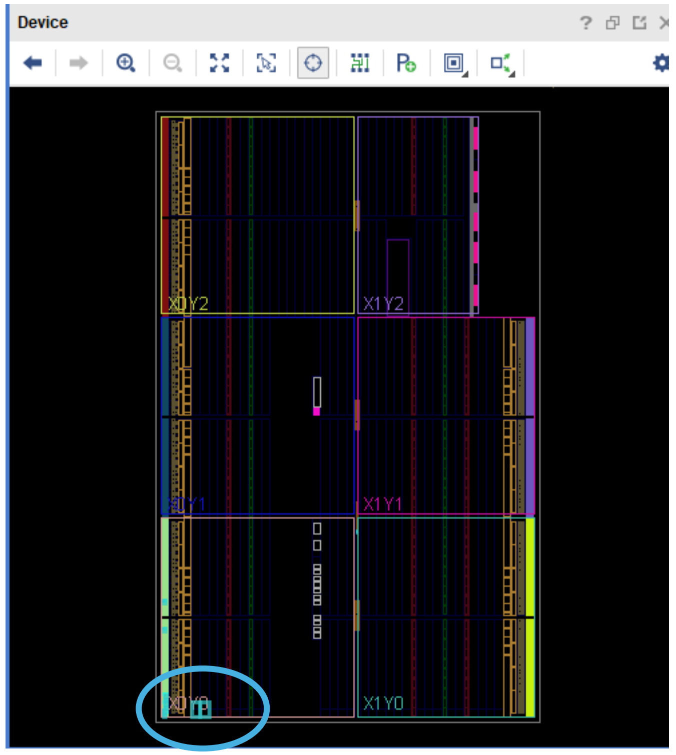
| | |
|---|---|
| **Total On-Chip Power:** | **0.072 W** |
| **Design Power Budget:** | **Not Specified** |
| **Power Budget Margin:** | **N/A** |
| **Junction Temperature:** | **25.4°C** |
| Thermal Margin: | 59.6°C (11.9 W) |
| Effective ϑJA: | 5.0°C/W |
| Power supplied to off-chip devices: | 0 W |
| Confidence level: | Medium |

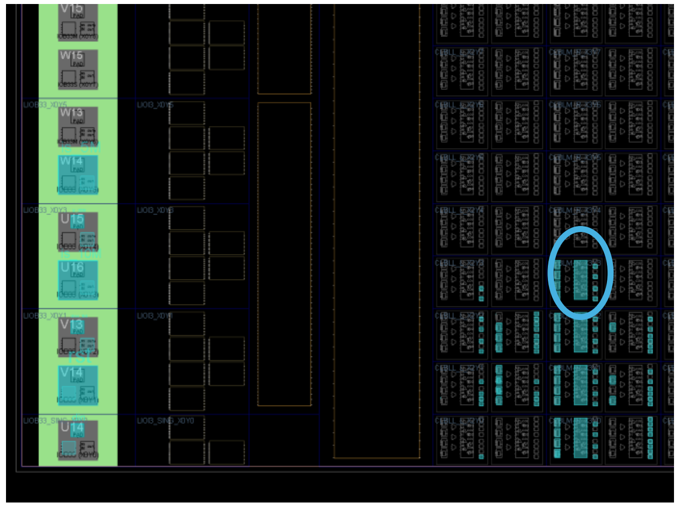
**On-Chip Power**

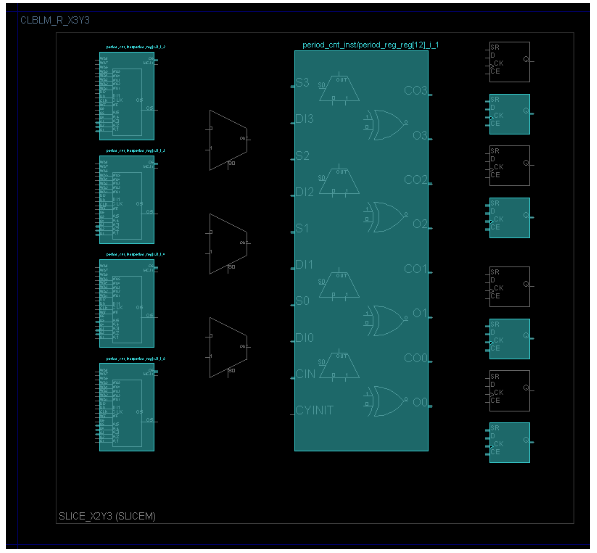
| | | | |
|---|---|---|---|
| 🟨 | Dynamic: | 0.002 W | (2%) |
| 🟪 | Clocks: | 0.002 W | (85%) |
| 🟦 | Signals: | <0.001 W | (1%) |
| 🟩 | Logic: | <0.001 W | (1%) |
| 🟩 | I/O: | <0.001 W | (13%) |
| 🟦 | Device Static: | 0.070 W | (98%) |

# Package and Device View









# Thank you!