PWM Timer Core

with Wishbone Interface

Sep 5th, 2025

Authors

Name	Kareem Ashraf Mostafa
E-Mail	kareem.ash05@gmail.com
Level	EECE Junior @ Cairo University

Name	Yasmeen Haitham Ismail
E-Mail	yasmeenhaitham77@gmail.com
Level	EECE Senior-1 @ Cairo University

Name	Shahd Ismail Elmasry
E-Mail	shahdelmasry019@gmail.com
Level	EECE Senior-1 @ Helwan University

Name	Mariam Ehab
E-Mail	ehabmariam481@gmail.com
Level	EECE Senior-1 @ Helwan University

Table of Contents

Introduction

Project Overview

This project involved the design, verification, and implementation of a highly configurable Pulse Width Modulation (PWM) and Timer controller IP core. The core is compliant with the Wishbone Bus Architecture Revision B.4, making it suitable for integration into larger System-on-Chip (SoC) designs.

The primary objective was to create a versatile peripheral capable of generating analog-control equivalent signals via digital means (PWM) and providing precise timing interrupts, based on the specifications outlined in the provided user manual. The core was developed using Verilog, simulated extensively to validate its functionality using **QuestaSim**, and synthesized for a target FPGA platform using **Xilinx Vivado**.

Core Features Overview

The implemented PWM/Timer core exhibits the following key features:

- **Daul Operating Mode:** Functions as a 16-bit PWM generator or a 16-bit timer/interrupt controller.
- **Flexible Clocking:** Selects between the system Wishbone clock or an external clock input for operation.
- **Multi-Channel PWM**: Configurable number of channels using "**num_ch**" parameter. Each channel with its own period and duty cycle.
- **Runtime Reconfiguration**: Allows all critical parameters (period, duty cycle, divisor, control settings) to be modified during operation via the Wishbone interface.
- **Standard-Complaint-Interface**: Integrates seamlessly via a Wishbone B4 slave interface for register access and control.

High-Level Architecture

Top-Level Block Diagram

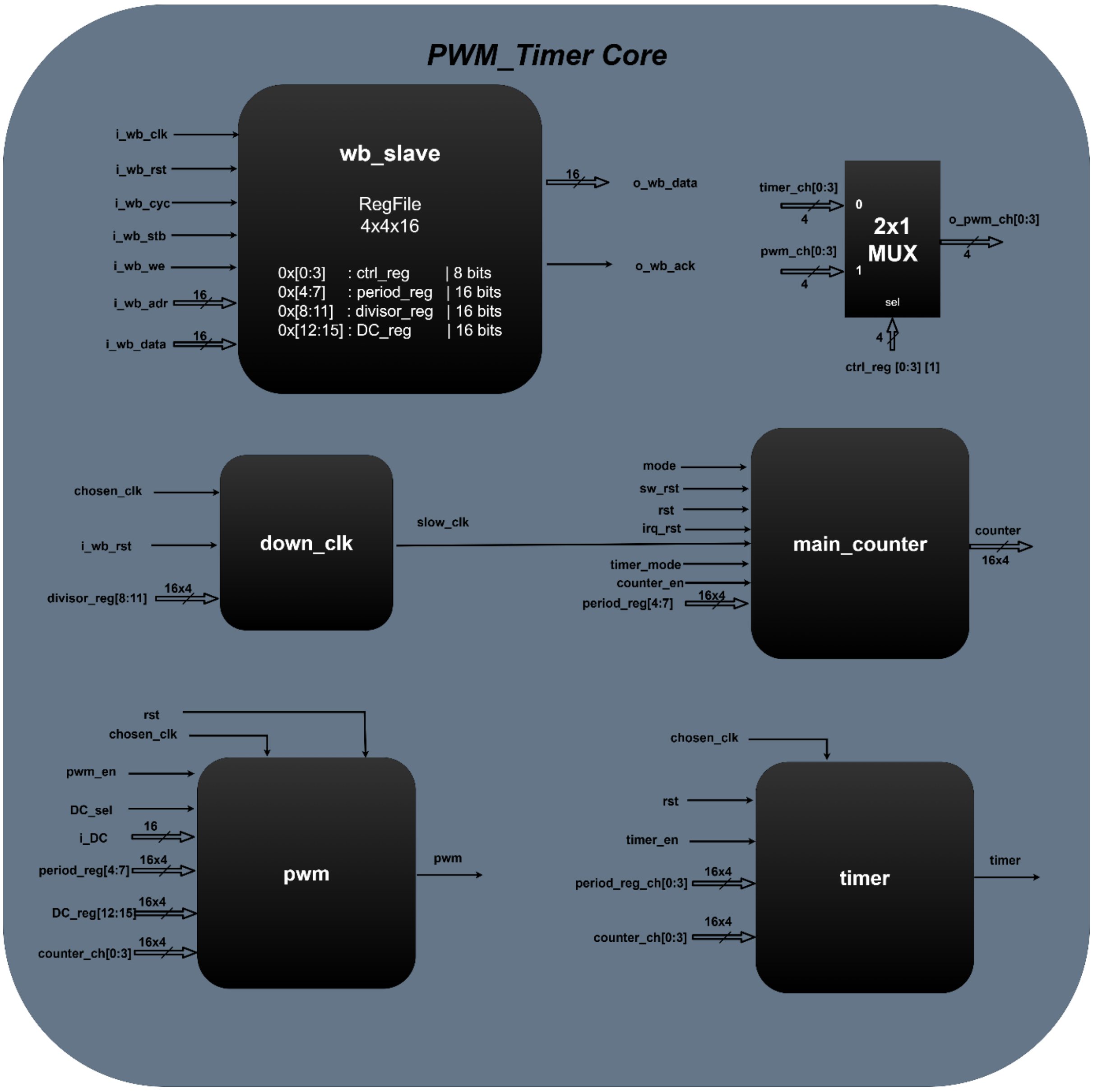


Figure 1: Block-Diagram

Interface Description

- The core is controlled by a **Wishbone B4** slave interface, provides read/write access to the internal register file.
- All internal modules (down_clk, main_counter, pwm, and timer) source their configuration values (ctrl, period, divisor, and DC) from this register file.
- The core supports four independent channels, each capable of outputting either PWM signal or an interrupt.
- The operating clock for the PWM/timer logic can be selected between the Wishbone clock (i_wb_clk) or an external clock (i_extclk).
- In PWM mode, the duty cycle value for each channel can be sourced from either the internal (DC_reg) or the dedicated external input port (i_DC), with (i_DC_valid) signaling valid data.
- A programmable divisor register (divisor_reg) allows the selected source clk to be down-clocked before driving the main counters.
- The main counter in each channel can be synchronously reset via a control bit, ensuring a known state before operation.

Implementation Details

Wishbone Slave Interface (wb_slave)

This module implements the Wishbone B4 slave protocol to manage all communication between a host processor and the core. It decodes the 16-bit word address (i_wb_adr) to select one of four internal registers: the 8-bit control register (ctrl_reg), the 16-bit period register (period_reg), the 16-bit clock divisor register (divisor_reg), or the 16-bit duty cycle register (DC_reg). The module generates a synchronous acknowledgment (o_wb_ack) to validate completed read and write cycles. All register writes are synchronous to the positive edge of i_wb_clk and gated by the Wishbone cycle, strobe, and write enable signals. The register file serves as the sole point of configuration for all other core modules.

Clock Divider (down_clk)

The core function of the (down_clk) module is to divide the frequency of the selected source clock (chosen_clk) and generate an output clock (slow_clk) that serves as the clock input for the (main_counter) module. It implements a programmable pre-scaler using the 16-bit value stored in the (divisor_reg). The module operates by counting a specified number of (chosen_clk) cycles to generate a single cycle of the (slow_clk) output, effectively performing frequency division. This allows the main counter to operate at a much slower rate than the system clock, enabling the generation of long periods and low-frequency PWM waveforms. If the divisor value is set to 0 or 1, the module is bypassed, and (chosen_clk) is passed directly through to (slow_clk).

Main Counter (main_counter)

This is a core 16-bit up-counter module for each channel. Its counting operation is gated by the global (counter_en) signal (from the control register). The counter increments on each enabled clock edge until it reaches the value stored in the (period_reg) for its channel, at which point it automatically resets to zero on the next cycle. It can also be synchronously reset at any time by the (sw_rst) (software reset) control bit. The current count value is continuously output for use by the PWM and timer comparator logic.

PWM Generator (pwm)

The PWM generator module produces the pulse-width modulated output for each channel. It continuously compares the current value of the channel's main counter against the duty cycle value. The duty cycle value is selected via a multiplexer controlled by the DC_sel bit from the control register; the source is either the internal DC_reg or the external i_DC input (the latter is typically latched on a valid signal). The o_pwm output for the channel is asserted high when the counter value is less than the selected duty cycle value and asserted low otherwise. The output is only enabled when the core is configured in PWM mode and the output enable bit is set.

Timer/Interrupt Generator (timer)

This module operates when the core is configured in timer mode. It compares the main counter's value with the period register. When the two values match, it triggers a timer event. This event sets an internal interrupt flag, which generates a high pulse on the channel's o_pwm output. The interrupt flag must be explicitly cleared by the host processor writing to the interrupt bit in the control register. The behavior after an event is configurable: if the "continuous run" bit is set, the counter will reset and begin counting again immediately; if not, it will halt until the interrupt is cleared and restarted.

Design Verification

A rigorous, multi-level verification strategy was employed to ensure the functional correctness and robustness of the PWM/Timer core. The process was automated using **TCL (do)** scripts for **QuestaSim**, enabling regression testing and continuous integration. **Self-checking** testbenches were developed for each major module and the top-level system, providing immediate pass/fail results.

Top-Level System Verification

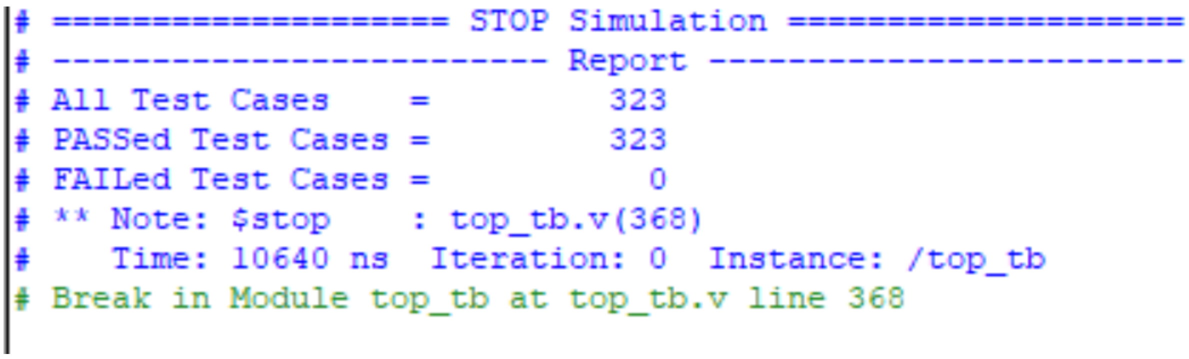
The top-level testbench instantiates the complete DUT and subjects it to a comprehensive suite of 16 verification scenarios designed to validate real-world operation and corner cases. The testbench is fully self-checking, using automatic stimulus generation and response monitoring against predicted outcomes.

Verification Scenarios

The following verification scenarios were executed automatically:

1. **Functional Correctness =>** RESET Behavior
2. **Functional Correctness =>** Direct wb write/read for all registers
3. **Functional Correctness =>** Random wb write/read for all registers
4. **Functional Correctness =>** pwm core: DC_reg = 50%, period_reg = 4
5. **Functional Correctness =>** pwm core: DC_reg = 25%, period_reg = 8
6. **Functional Correctness =>** pwm core: i_DC = 75%, period_reg = 4
7. **Functional Correctness =>** pwm core: random DC = 40%, period_reg = 5
8. **Functional Correctness =>** timer core: [one-shot mode] , period_reg = 4
9. **Functional Correctness =>** timer core: [cont mode] , period_reg = 4
10. **Functional Correctness =>** timer core: [random mode] , period_reg = 7
11. **Functional Correctness =>** timer core: [random mode] , period_reg = 4, divisor_reg = 4
12. **Functional Correctness =>** Random core: DC = 50% , period_reg = 4
13. **Corner Case =>** pwm core: DC_reg = 4 > period_reg = 3
14. **Corner Case =>** pwm core: i_DC = 5 > period_reg = 4
15. **Corner Case =>** timer core: [random mode], period_reg = 1
16. **Corner Case =>** Assign different data for each channel and check outputs in parallel

Summary Results



```
# ==================== STOP Simulation ====================
# ------------------------- Report -----------------------
# All Test Cases     =         323
# PASSed Test Cases =         323
# FAILed Test Cases =           0
# ** Note: $stop    : top_tb.v(368)
#    Time: 10640 ns  Iteration: 0  Instance: /top_tb
# Break in Module top_tb at top_tb.v line 368
```

Figure 2: Top-TB-Report

Waveform for top_tb.v

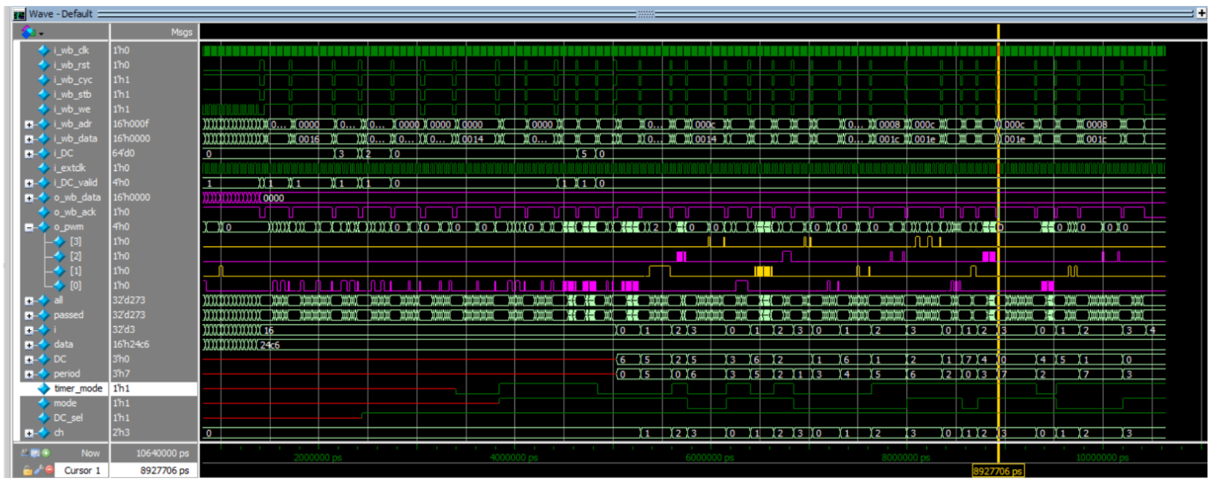


Figure 3: top-tb-waveform

Module Level Verification

Module (wb_slave)

The following verification scenarios were executed automatically:

1. **Functional Correctness =>** RESET Behavior
2. **Functional Correctness =>** Write random data to all registers
3. **Functional Correctness =>** Read from all registers
4. **Corner Case =>** i_wb_adr >= mem_depth

Module (down_clk)

The following verification scenarios were executed automatically:

1. **Functional Correctness =>** RESET Behavior
2. **Functional Correctness =>** Even divisor_reg = 4
3. **Functional Correctness =>** Odd divisor_reg = 5
4. **Corner Case =>** Error Handling: divisor_reg = 0
5. **Corner Case =>** Error Handling: divisor_reg = 1
6. **Corner Case =>** Even divisor_reg = 100
7. **Corner Case =>** Odd divisor_reg = 101

Module (main_counter)

The following verification scenarios were executed automatically:

1. **Functional Correctness =>** RESET Behavior
2. **Functional Correctness =>** Run it in pwm mode to assert that it counts up to period_reg-1
3. **Functional Correctness =>** Run it in timer mode and set ctrl[3] (cont mode) then assert that it counts up again if ctrl[2] is enabled
4. **Functional Correctness =>** Run it in timer mode and reset ctrl[3] (one-shot mode) then assert that it hangs on 0 value
5. **Corner Case =>** RESET during counting
6. **Corner Case =>** Disable main_counter during counting. Assert that counter holds the past value

Module (pwm)

The following verification scenarios were executed automatically:

1. **Functional Correctness =>** RESET Behavior
2. **Functional Correctness =>** DC_reg = 50%
3. **Functional Correctness =>** DC_reg = 25%
4. **Functional Correctness =>** DC_reg = 75%
5. **Corner Case =>** (DC_reg = 4) > (period = 3)
6. **Corner Case =>** (i_DC = 3) > (period = 2)

Module (timer)

The following verification scenarios were executed automatically:

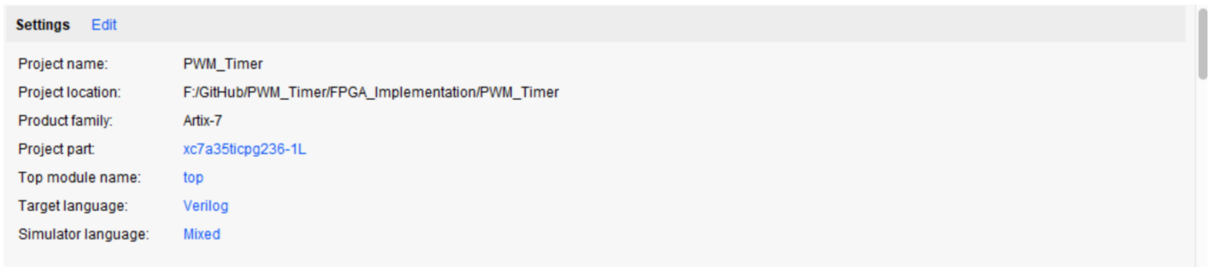
1. **Functional Correctness =>** RESET Behavior
2. **Functional Correctness =>** cont. mode, period = 4
3. **Functional Correctness =>** one-shot mode, period = 4
4. **Corner Case =>** cont. mode, period = 1
5. **Corner Case =>** one-shot mode, period = 1

Synthesis & Implementation

The PWM/Timer core was successfully processed through the AMD Xilinx Vivado design flow for a target FPGA device. The following steps confirm the design was synthesized, implemented, and timed correctly, resulting in a valid bitstream for hardware deployment.

Target Device: [Xilinux Artix-7 xc7a35ticpg236-1L]

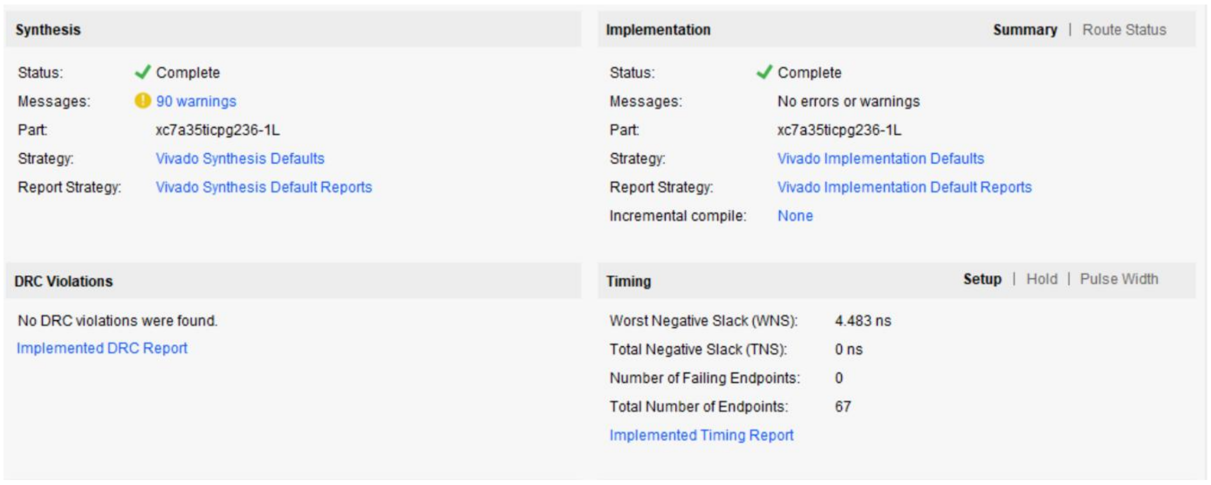
Project report



Settings Edit

Project name:	PWM_Timer
Project location:	F:/GitHub/PWM_Timer/FPGA_Implementation/PWM_Timer
Product family:	Artix-7
Project part:	xc7a35ticpg236-1L
Top module name:	top
Target language:	Verilog
Simulator language:	Mixed

Figure 4: project-report-1



Synthesis

Status:	✓ Complete
Messages:	90 warnings
Part:	xc7a35ticpg236-1L
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports

Implementation Summary | Route Status

Status:	✓ Complete
Messages:	No errors or warnings
Part:	xc7a35ticpg236-1L
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Reports
Incremental compile:	None

DRC Violations

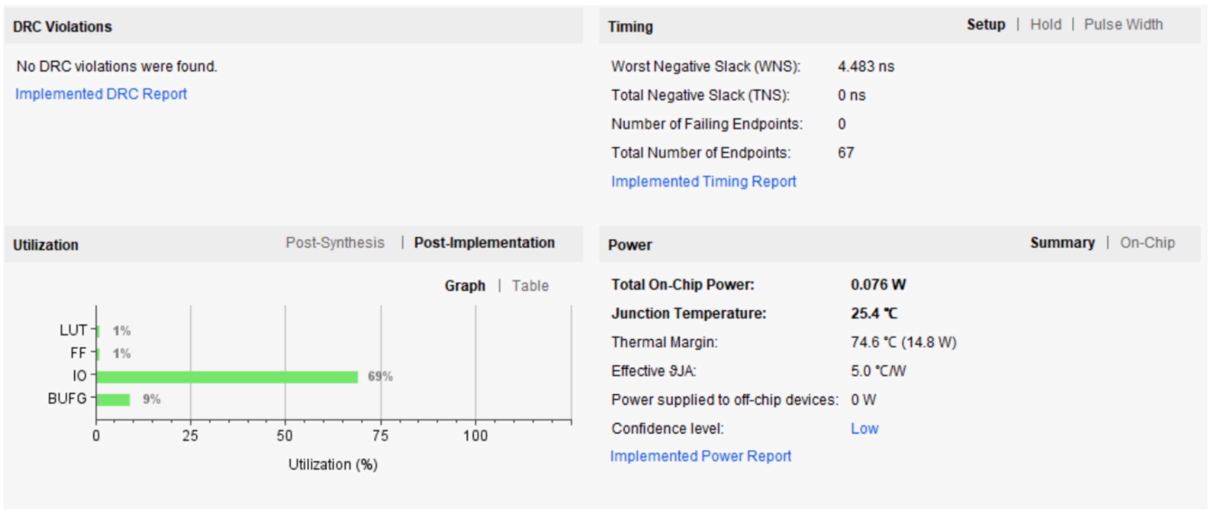
No DRC violations were found.
Implemented DRC Report

Timing Setup | Hold | Pulse Width

Worst Negative Slack (WNS):	4.483 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	67

Implemented Timing Report

Figure 5: project-report-2



DRC Violations

No DRC violations were found.
Implemented DRC Report

Timing Setup | Hold | Pulse Width

Worst Negative Slack (WNS):	4.483 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	67

Implemented Timing Report

Utilization Post-Synthesis | Post-Implementation

Graph | Table

Resource	Utilization (%)
LUT	1%
FF	1%
IO	69%
BUFG	9%

Power Summary | On-Chip

Total On-Chip Power:	**0.076 W**
Junction Temperature:	**25.4 ℃**
Thermal Margin:	74.6 ℃ (14.8 W)
Effective ϑJA:	5.0 ℃/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

Figure 6: project-report-3

Elaborated Design

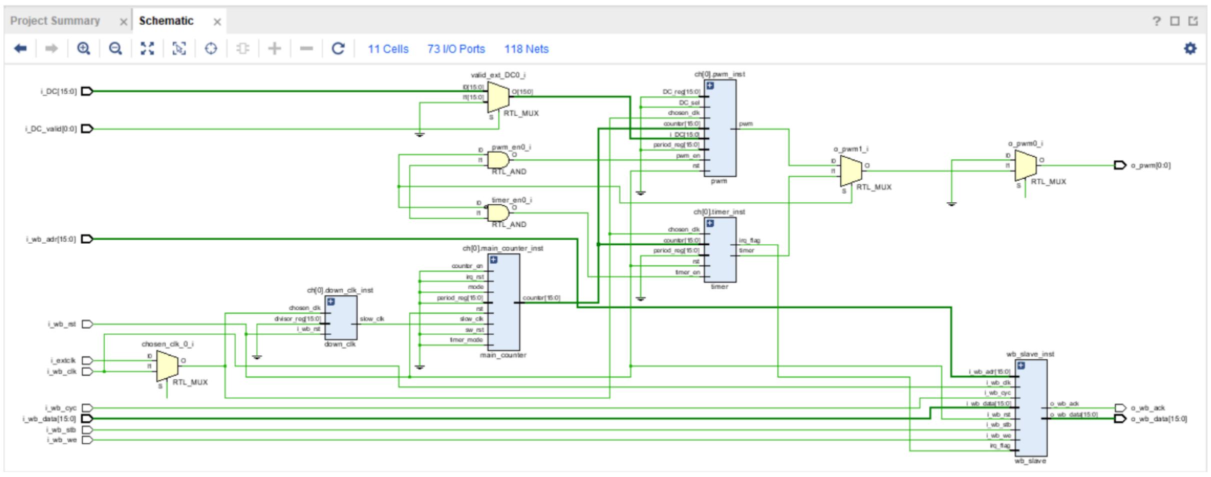


Figure 7: elaborated-design

Synthesis

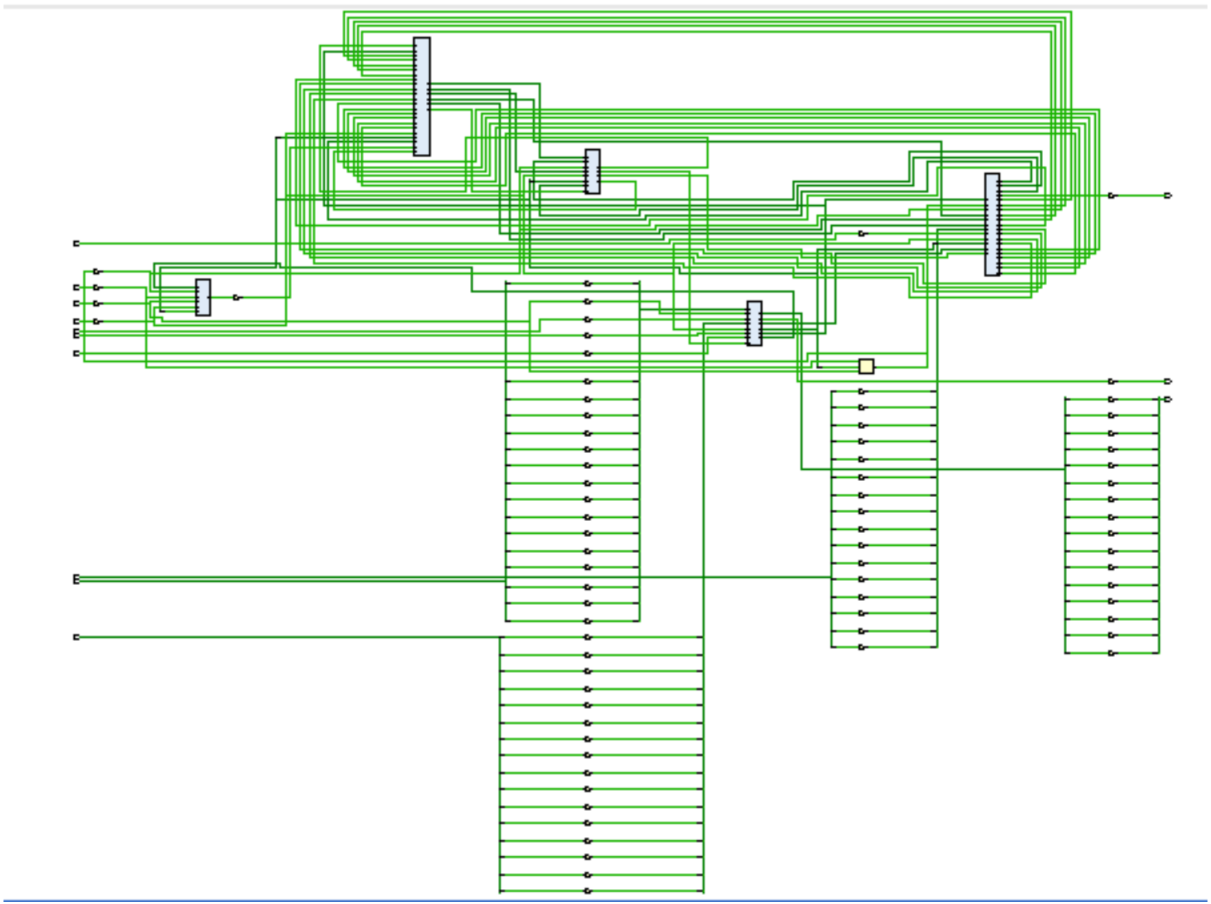
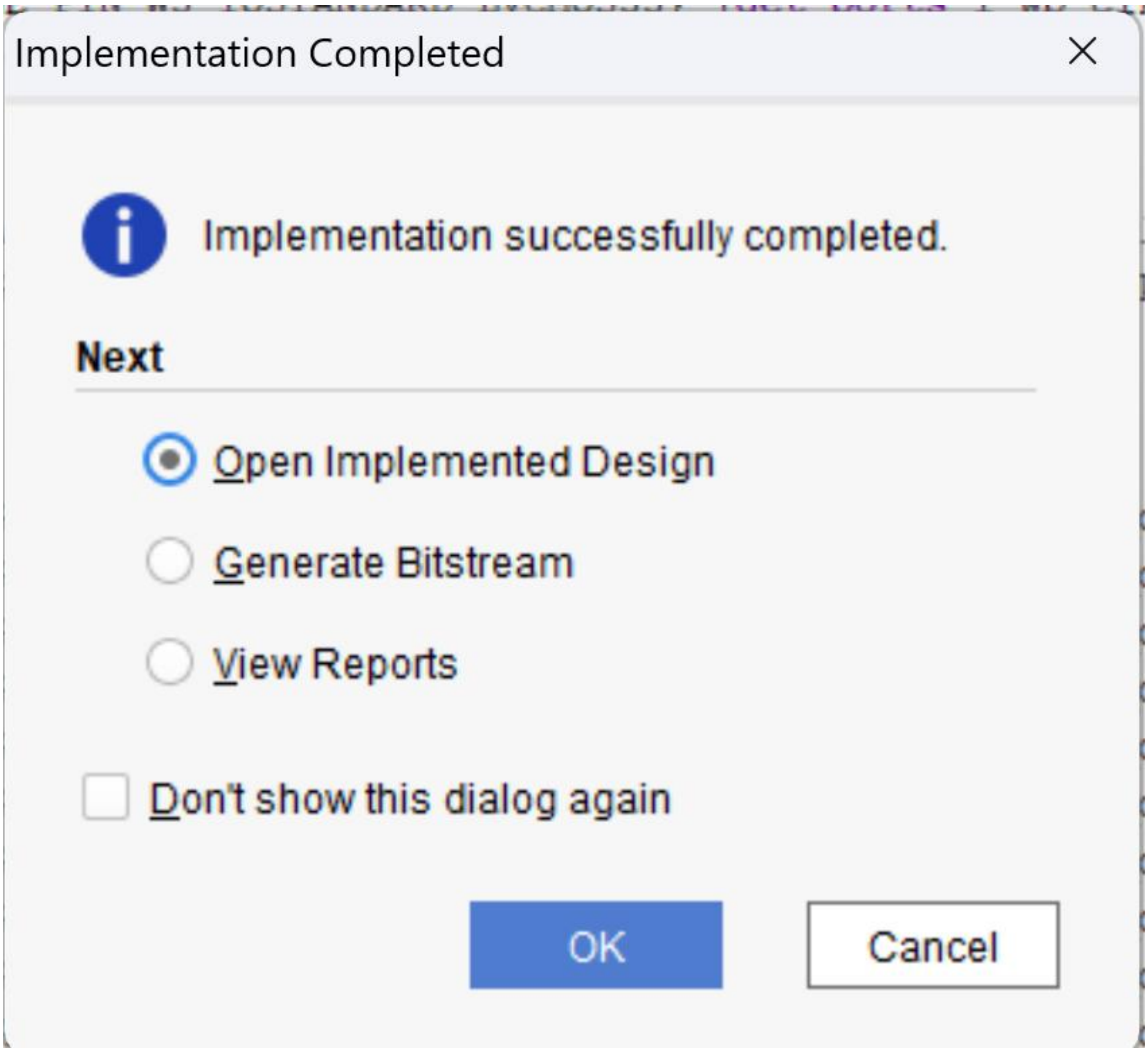


Figure 8: synthesis-schematic

Implementation

Successfully Implemented


Implementation Completed

Implementation successfully completed.

Next

(●) Open Implemented Design
( ) Generate Bitstream
( ) View Reports

☐ Don't show this dialog again

Figure 9: successful-implementation

Device

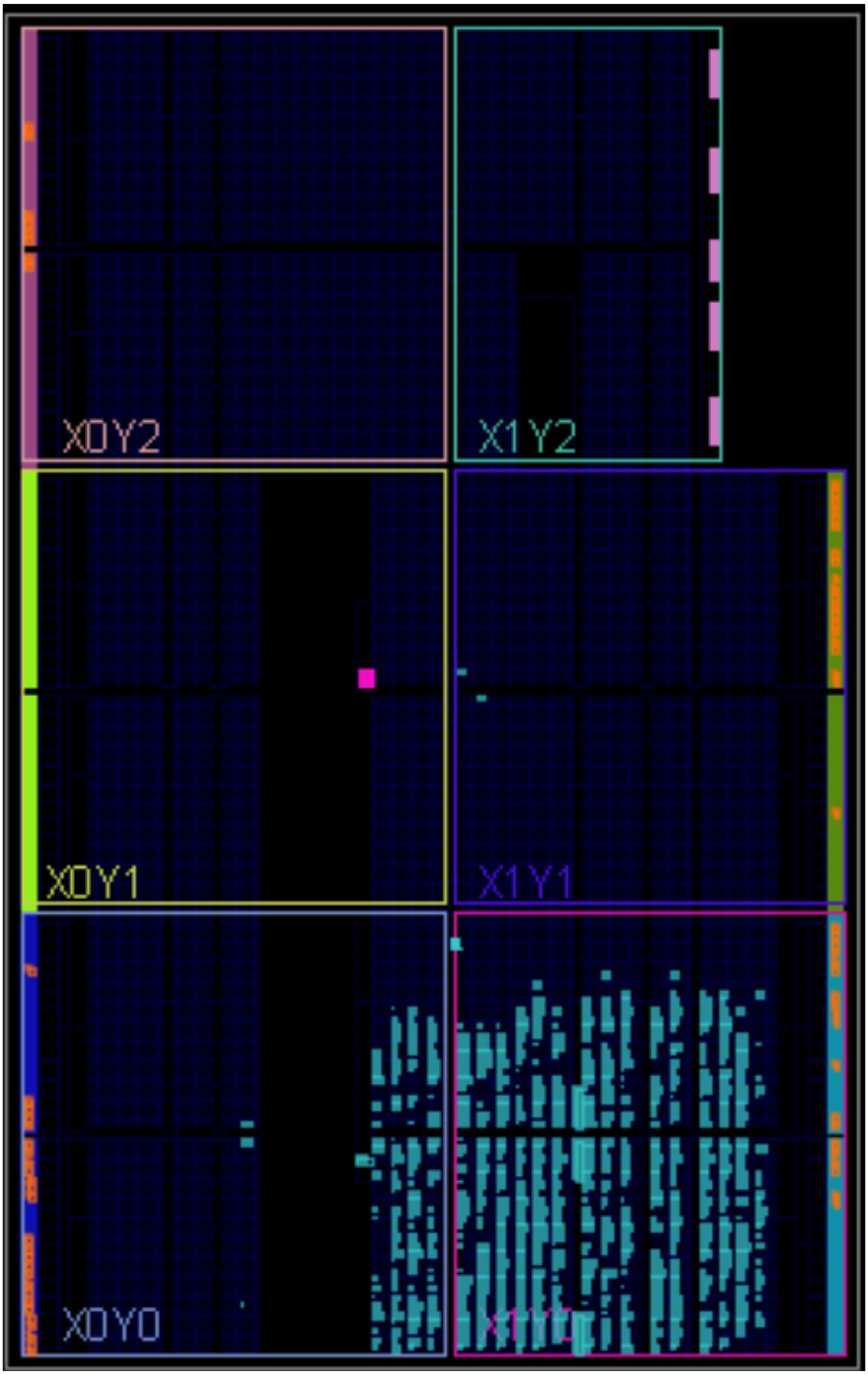


Figure 10: device

Device Zoomed

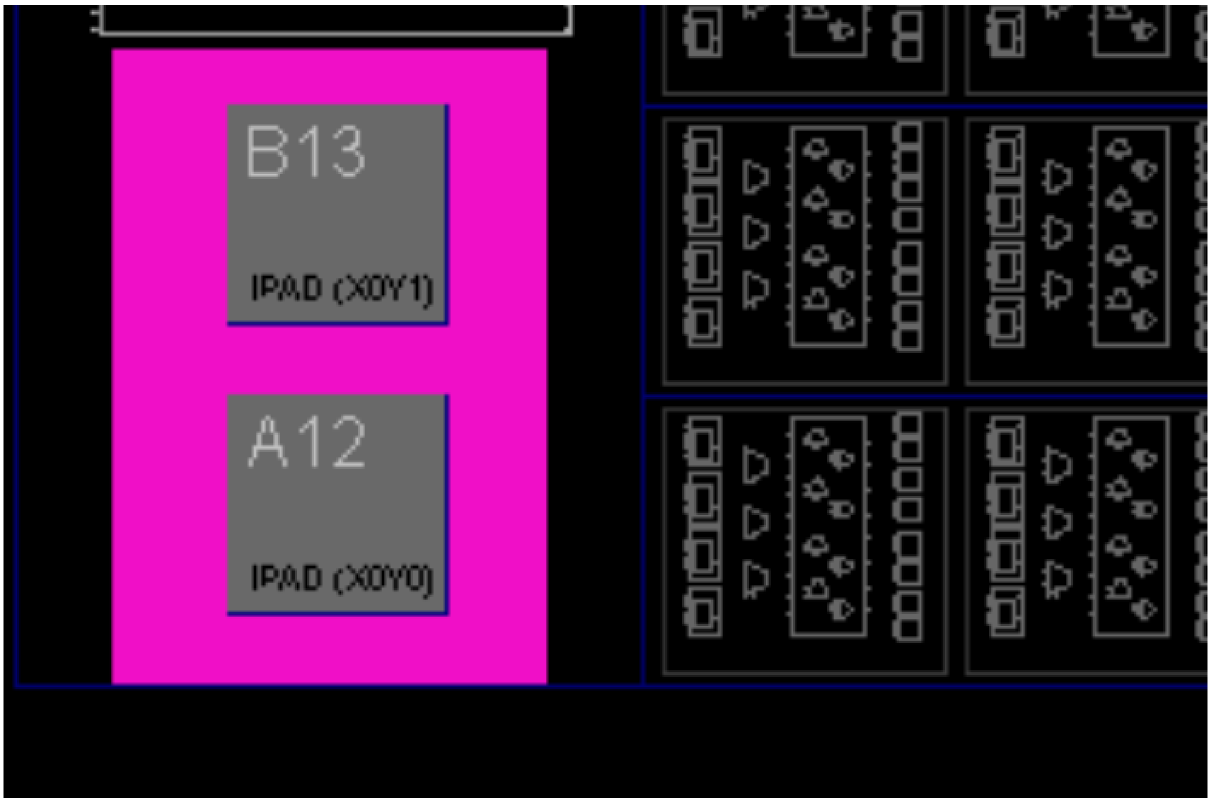
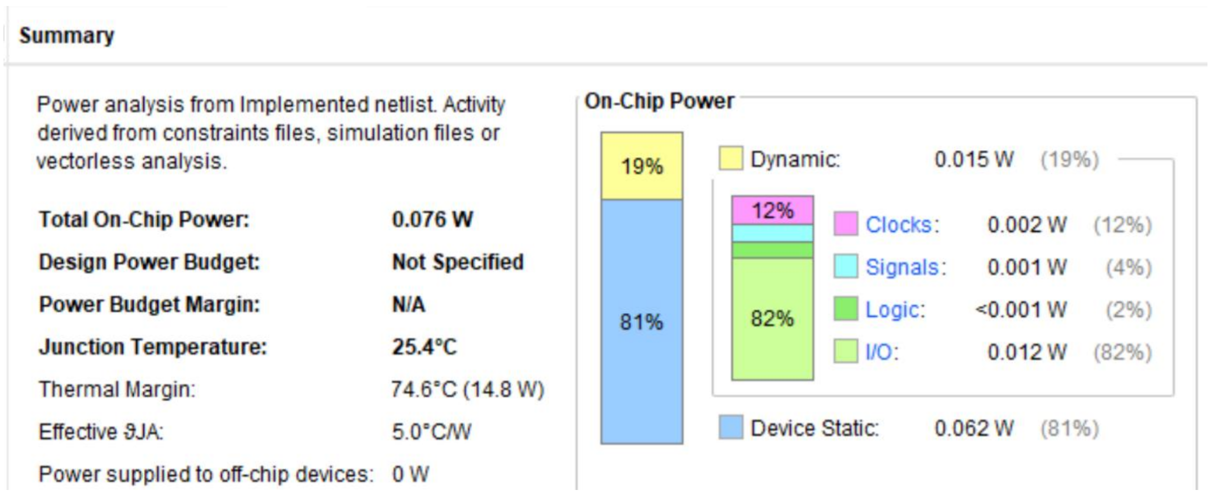


Figure 11: device-zoomed

Power Report

- Total On-Chip Power: 0.076 W



Summary

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

Total On-Chip Power:	**0.076 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.4°C**
Thermal Margin:	74.6°C (14.8 W)
Effective ϑJA:	5.0°C/W
Power supplied to off-chip devices:	0 W

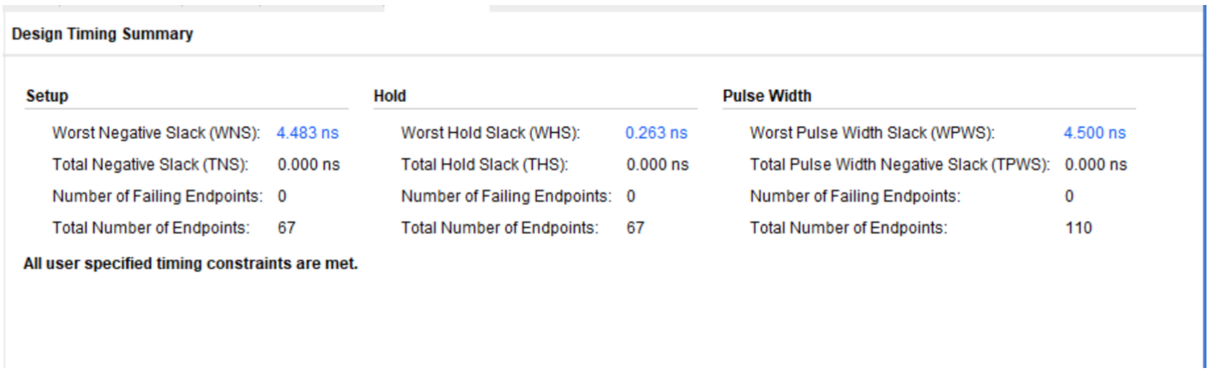
On-Chip Power

Dynamic:	0.015 W	(19%)
Clocks:	0.002 W	(12%)
Signals:	0.001 W	(4%)
Logic:	<0.001 W	(2%)
I/O:	0.012 W	(82%)
Device Static:	0.062 W	(81%)

Figure 12: power-report

Timing Report

- Worst Negative Slack : **4.483** ns
- Worst Hold Slack : **0.263** ns
- Worst Pulse Width Slack: **4.5** ns



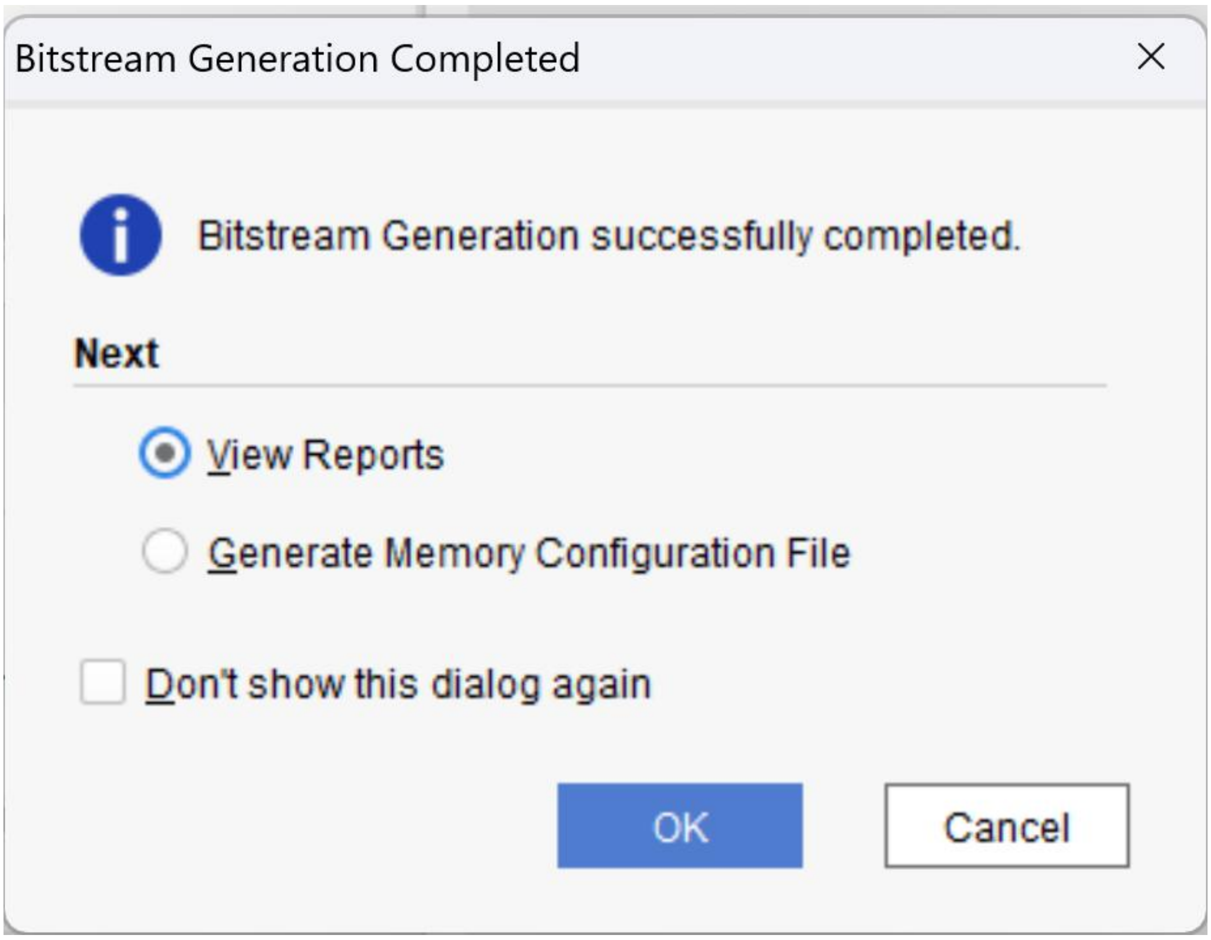
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.483 ns	Worst Hold Slack (WHS):	0.263 ns	Worst Pulse Width Slack (WPWS):	4.500 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:	67	Total Number of Endpoints:	67	Total Number of Endpoints:	110

All user specified timing constraints are met.

Figure 13: timing-report

Bitstream Generation



Bitstream Generation Completed

Bitstream Generation successfully completed.

Next
◉ View Reports
○ Generate Memory Configuration File
☐ Don't show this dialog again
OK
Cancel

Figure 14: successful-bitstream-generation

Future Work

- ASIC Flow
- 32-bit Counters/Registers: For higher resolution and longer periods
- Synchronization Input: To phase-align multiple cores or channels
- Counter Readout Registers: To aid software debugging by reading the current counter value

Conclusion

This project successfully accomplished the complete design, verification, and implementation of a multi-channel PWM/Timer IP core from specification to silicon-ready bitstream. The core delivers on all promised features, including dual operating modes, four independent channels, and flexible clocking options.

The design's robustness was proven through a rigorous, automated verification strategy, culminating in a full pass of 16 comprehensive test scenarios. Finally, the synthesis and implementation process on AMD Xilinx Vivado confirmed the design is physically realizable, meeting all timing constraints with minimal resource utilization.

This core provides a reliable, high-quality peripheral for integration into larger FPGA-based SoCs, serving as a solid foundation for applications requiring precise waveform generation or timing control. The project demonstrates a full mastery of the digital design flow, from RTL conception to final implementation.

Appendix

Here is all code and docs sources on **GitHub**:

https://github.com/kareem05-ash/PWM_Timer

Contact Us on Linkedin

Figure 15: team-linkedin-accounts