Timer IP Design Specification

Fundamental IC Design and Verification

Version 1.1

September 16, 2025

Table of Contents

List of Figures

List of Tables

1 Overview

1.1 Introduction

The Timer IP is a digital hardware block designed to generate accurate timing intervals and control the timing of various operations within a System on Chip (SoC). It functions as a programmable counter that can trigger interrupts upon reaching a user-defined value.

This module is customized from the CLINT (Core Local Interruptor) module of the RISC-V architecture. Its primary applications include, but are not limited to:

- Generating periodic interrupts for an operating system's scheduler.
- Creating precise delays.
- Event generation and counting.
- Pulse Width Modulation (PWM) signal generation.

The timer is configured via an AMBA APB slave interface, making it easily integrable into modern SoC designs.

1.2 Feature List

The Timer IP provides a comprehensive set of features for timing and control:

- **64-bit Count-up Timer:** A free-running 64-bit counter.
- **APB Slave Interface:** Configuration is managed through a standard APB bus interface with a 12-bit address space.
- **Programmable Counter Speed:** The counter can be driven by the system clock or a divided clock (programmable divisor up to 256).
- **Interrupt Support:** A maskable, level-triggered hardware interrupt can be generated.
- **Active-Low Asynchronous Reset.**
- **APB Wait State:** Supports a single-cycle wait state to improve timing flexibility.
- **APB Byte Access:** Allows the bus to write to individual bytes within a 32-bit register using write strobes.
- **APB Error Handling:** Provides an error response for prohibited register accesses.
- **Debug Halt Mode:** The counter can be stopped (halted) during system debug mode.
- **Automatic Counter Reset:** The counter is automatically cleared to its initial value when the timer is disabled.

1.3 High-Level Architecture

The Timer IP is composed of three main sub-blocks: an APB Slave Interface, a Register File, and the Core Timer Logic.

Block Diagram Placeholder

Figure 1: High-Level Block Diagram of the Timer IP

- **APB Slave Interface:** This module decodes the APB signals to manage read/write access to the register file. It implements the logic for wait states, byte enables, and error responses.
- **Register File:** A set of registers that control the timer's operation and hold its status.
- **Core Timer Logic:** This block contains the 64-bit counter, the logic for clock division, the comparator for the interrupt trigger, and the debug halt logic.

1.4 Interface Signals (IO Port List)

The top module is named `timer_top`.

Table 1: IO Port List for timer_top

Signal Name	Width	Direction	Description
`sys_clk`	1	Input	System clock. All synchronous logic operates on its rising edge.
`sys_rst_n`	1	Input	Active-low asynchronous reset.
APB Slave Interface			
`tim_paddr`	12	Input	APB address bus. Selects a register within the Timer IP.
`tim_psel`	1	Input	APB select signal. Indicates the Timer IP is selected.
`tim_penable`	1	Input	APB enable signal. Latches address and control in the access phase.
`tim_pwrite`	1	Input	APB write control. 1 for write, 0 for read.
`tim_pwdata`	32	Input	APB write data bus.
`tim_prdata`	32	Output	APB read data bus.
`tim_pready`	1	Output	APB ready signal. Can be held low for one cycle to insert a wait state.
`tim_pslverr`	1	Output	APB slave error. Indicates an unsuccessful transfer.
`tim_pstrb`	4	Input	APB write strobes for byte-level write access.
Timer Signals			
`tim_int`	1	Output	Timer interrupt signal. Asserted when counter matches compare value.
`dbg_mode`	1	Input	Debug mode signal. Used to enable the halt functionality.

2 Register Specification

2.1 Register Summary

Table 2: Register Summary

Address Offset	Abbreviation	Register Name
0x000	TCR	Timer Control Register
0x004	TDR0	Timer Data Register 0 (Counter bits [31:0])
0x008	TDR1	Timer Data Register 1 (Counter bits [63:32])
0x00C	TCMP0	Timer Compare Register 0 (Compare bits [31:0])
0x010	TCMP1	Timer Compare Register 1 (Compare bits [63:32])
0x014	TIER	Timer Interrupt Enable Register
0x018	TISR	Timer Interrupt Status Register
0x01C	THCSR	Timer Halt Control Status Register
Others	-	Reserved (Access is RAZ/WI)

2.2 Detailed Register Descriptions

2.2.1 TCR - Timer Control Register (Offset: 0x000)

Table 3: TCR Bit Fields

Bit	Name	Type	Default	Description
31:12	Reserved	RO	20'h0	Reserved. Reads as 0.
11:8	`div_val`	RW	4'b0001	**Counter control mode setting.** See slide 15 for encoding. Writing a prohibited value generates an error response.
7:2	Reserved	RO	6'b0	Reserved. Reads as 0.
1	`div_en`	RW	1'b0	**Counter control mode enable.** 1: Enabled.
0	`timer_en`	RW	1'b0	**Timer enable.** 1: Enabled. H-¿L transition resets counter.

Note: Attempting to change `div_val` or `div_en` while `timer_en` is High will generate an APB error response, and the write will be ignored.

2.2.2 TDR0/TDR1 - Timer Data Registers (Offsets: 0x004, 0x008)

These two registers form the 64-bit counter value. The value is cleared to its initial default when `TCR.timer_en` transitions from High to Low.

- **TDR0 (0x004):** Lower 32 bits of the counter. Default: 32'h0000_0000.
- **TDR1 (0x008):** Upper 32 bits of the counter. Default: 32'h0000_0000.

2.2.3 TCMP0/TCMP1 - Timer Compare Registers (Offsets: 0x00C, 0x010)

These two registers hold the 64-bit value that is compared against the counter to generate an interrupt.

- **TCMP0 (0x00C):** Lower 32 bits of the compare value. Default: 32'hFFFF_FFFF.
- **TCMP1 (0x010):** Upper 32 bits of the compare value. Default: 32'hFFFF_FFFF.

2.2.4 TIER - Timer Interrupt Enable Register (Offset: 0x014)

Table 4: TIER Bit Fields

Bit	Name	Type	Default	Description
31:1	Reserved	RO	31'h0	Reserved.
0	`int_en`	R/W	1'b0	**Timer interrupt enable:** 0: Disabled, 1: Enabled.

2.2.5 TISR - Timer Interrupt Status Register (Offset: 0x018)

Table 5: TISR Bit Fields

Bit	Name	Type	Default	Description
31:1	Reserved	RO	31'h0	Reserved.
0	`int_st`	RW1C	1'b0	**Interrupt pending bit.** Write 1 to clear.

2.2.6 THCSR - Timer Halt Control Status Register (Offset: 0x01C)

Table 6: THCSR Bit Fields

Bit	Name	Type	Default	Description
31:2	Reserved	RO	30'h0	Reserved.
1	`halt_ack`	RO	1'b0	**Timer halt acknowledge:** 1 indicates timer is halted.
0	`halt_req`	RW	1'b0	**Timer halt request:** 1 requests the timer to halt.

3 Functional Description

3.1 Counter Module

The core is a 64-bit synchronous count-up counter that runs when `TCR.timer_en` is 1.

3.1.1 Counting Modes

- **Default Mode (`div_en` = 0):** The counter increments on every rising edge of `sys_clk`.
- **Control Mode (`div_en` = 1):** The counter increments at a rate determined by `TCR.div_val`.

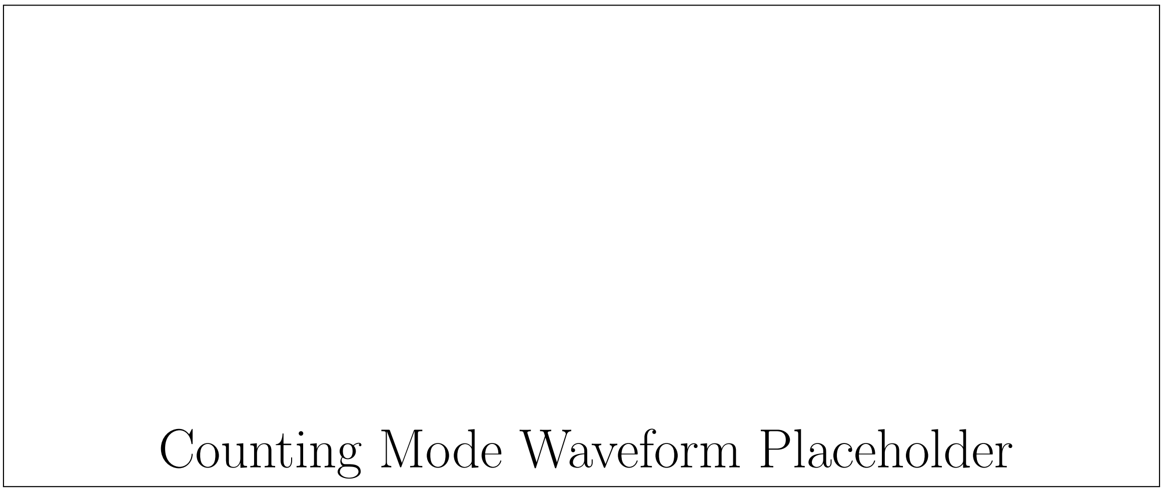

Figure 2: Example Waveform for Control Counting Mode

3.1.2 Halted (Debug) Mode

The counter can be halted when:

1. The input signal `dbg_mode` is high.
2. The `THCSR.halt_req` bit is set to 1.

The IP asserts `THCSR.halt_ack` to confirm the halt. The counter stops but retains its value. To resume, software must clear the `halt_req` bit.

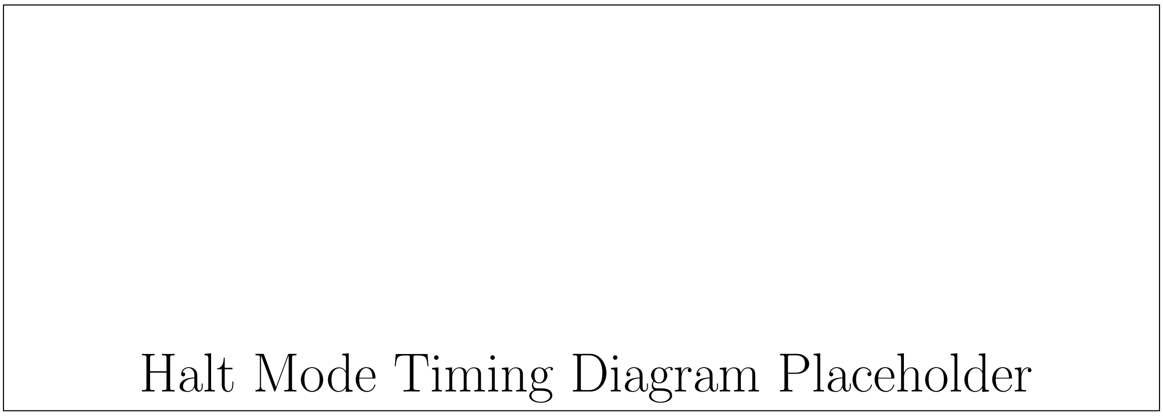

Figure 3: Halted Mode Timing Diagram

3.2 Interrupt Module

- **Trigger:** The interrupt condition occurs when the counter value equals the compare value.
- **Status:** On trigger, `TISR.int_st` is set to 1. It is cleared by a write-1-to-clear operation.
- **Output:** `tim_int` = `TISR.int_st` AND `TIER.int_en`.

3.3 APB Slave Interface

The timer acts as an APB slave, responding to transactions from an APB master.

- **Wait State:** `pready` can be held low for one clock cycle to extend a transaction.
- **Byte Access:** The 4-bit `pstrb` signal enables writing to specific byte lanes.

- **Error Handling:** `pslverr` is asserted for one cycle on prohibited accesses. When an error occurs, the write is ignored. Prohibited accesses include:
  - Writing a reserved value to `TCR.div_val`.
  - Changing `TCR.div_en` or `TCR.div_val` while `TCR.timer_en` is high.

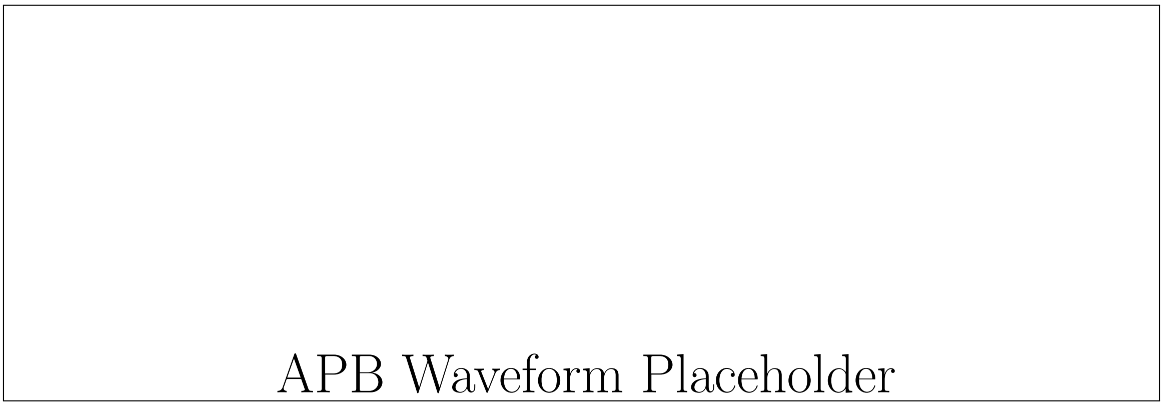

Figure 4: APB Read/Write Transaction

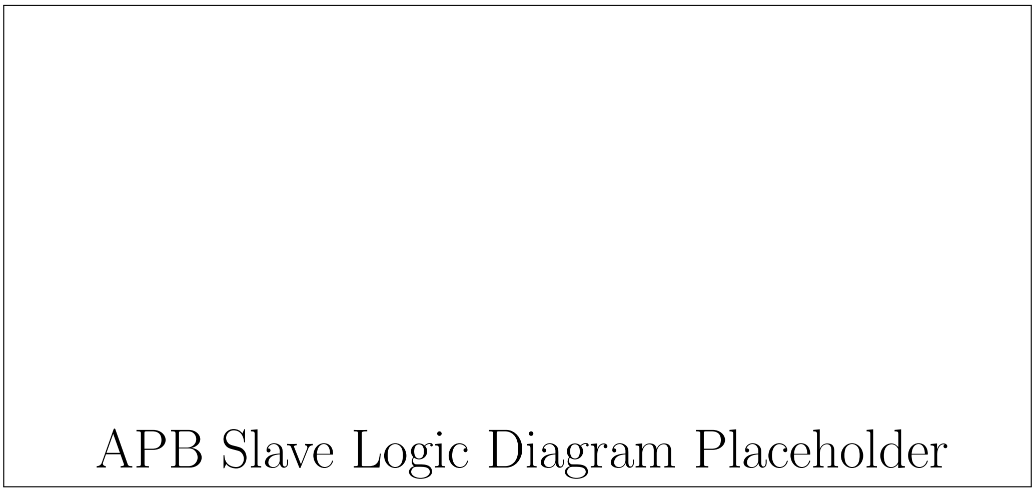

Figure 5: APB Slave Logic Diagram

4 Implementation Considerations

4.1 Reset Strategy

The IP uses a single active-low asynchronous reset, `sys_rst_n`. Upon reset, all registers and internal states must return to their specified default values.

4.2 Clocking

The entire IP operates in a single clock domain, `sys_clk`. All synchronous logic is triggered on the rising edge of this clock.

5 Verification Plan

5.1 Testbench Architecture

The verification environment should consist of a DUT, an APB BFM (Bus Functional Model), a Monitor, a Scoreboard/Checker, and a Test Sequencer.

5.2 Test Cases

1. **Register Access:** Verify all registers are accessible. Verify RAZ/WI for reserved addresses.
2. **Timer Control:** Verify timer enable, disable, and auto-reset functionality.
3. **Counting Modes:** Test default and all valid control counting modes.
4. **Counter Overflow:** Ensure correct 64-bit wraparound.
5. **Interrupt Generation:** Verify interrupt trigger, status, enable/mask, and clear mechanisms.
6. **Halt Mode:** Verify entry, exit, and behavior of the halt mode.
7. **APB Functionality:** Test byte strobes, wait states, and error responses under all specified conditions.
8. **Concurrent Operations:** Test scenarios with simultaneous APB access, counter events, and halt requests.

A Example Verilog Code Snippets

A.1 Conceptual Counter Logic

```verilog
// Internal 64-bit counter register
reg [63:0] counter_reg;

// Halt acknowledge signal from THCSR
wire halt_ack;

// Signal to enable counter increment for the current cycle
wire increment_en;

// Determine if an increment should occur
assign increment_en = (timer_en == 1'b1) && !halt_ack && (/* logic for div_val */);

// Counter logic
always @(posedge sys_clk or negedge sys_rst_n) begin
    if (!sys_rst_n) begin
        counter_reg <= 64'd0;
    // Handle automatic counter reset when timer_en goes from 1 -> 0
    end else if (timer_en_falling_edge) begin
        counter_reg <= 64'd0;
    end else if (apb_write_to_tdr) begin
        // Logic to handle direct writes to TDR0/TDR1 with byte strobes
        counter_reg <= new_tdr_value;
    end else if (increment_en) begin
        counter_reg <= counter_reg + 1;
    end
end
```

Listing 1: Conceptual 64-bit Counter Increment Logic