IC TRAINING CENTER VIETNAM

IC OVERVIEW RTL DESIGN AND VERIFICATION

----------o0o----------

THE FINAL PROJECT – TIMER IP

Supervisor: Mr. Nguyen Thanh Phong

Members:

1. Bui Minh Nhut

HO CHI MINH CITY, MARCH 2025

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my supervisor**, Mr. Nguyen Thanh Phong**, for his invaluable guidance, patience, and expertise throughout this project. His profound knowledge in digital design was instrumental in the successful development of the **64-bit Timer IP Core**.

I am also sincerely thankful to the technical team at ICTC Center, especially **Mr. Duc Le and Mr. Thong Nguyen**, for their generous support, practical advice, and for sharing their hardware implementation experience that greatly contributed to this research work.

TABLE OF CONTENTS

LIST OF IMAGES

LIST OF TABLES

ABSTRACT

This project presents the design and implementation of a 64-bit Timer IP core, compatible with the AMBA APB (Advanced Peripheral Bus) protocol. The Timer IP is capable of operating in both standard and advanced modes, providing precise time tracking, interrupt generation, and flexible configuration features suitable for embedded systems.

The timer features a 64-bit up-counter, programmable through a memory-mapped register set, accessible via a 4KB APB address space ranging from 0x4000_1000 to 0x4000_1FFF. It supports both basic timing functionality and advanced operational modes including byte-wise access, wait state insertion, and error handling.

The counter operates at a system clock frequency of 200 MHz, with optional control over counting speed via the div_en and div_val fields in the Timer Control Register (TCR). Interrupts are triggered when the counter reaches a specified compare value, provided the interrupt is enabled. The timer can also be halted in debug mode via the Halt Control mechanism.

The IP is structured into modular components, including an APB interface, register block, counter logic, control unit, and interrupt generator. All modules are designed in Verilog HDL and verified through comprehensive simulation testbenches. The design

ensures compatibility with SoC architectures requiring precise timing control and efficient interrupt-driven event handling.

CHAPTER 1: DESIGN SPECIFICATION

1.1. Timer Features

The Timer IP core includes the following key features:

- **64-bit Up Counter**
  - Operates in count-up mode to support extended timing ranges.
- **Address Space**
  - Occupies a 4KB address range from 0x4000_1000 to 0x4000_1FFF.
- **APB Slave Interface**
  - The register set is configured via the AMBA APB bus; the Timer functions as an APB slave device.
- **Standard Level Operation**
  - Supports 32-bit APB data transfers.
  - No wait states.
  - No error handling.
- **Advanced Level Operation**
  - **Wait state support**: One-cycle wait state to align with APB protocol timing.
  - **Byte access support**: Enables individual byte-wise access using pstrb.
  - **Debug mode halt**: Timer can halt counting when debug mode is enabled.
- **System Clock and Reset**
  - Operates at a **200 MHz** system clock.
  - Uses **active-low asynchronous reset**.
- **Counter Operation Control**
  - The counter can operate based on the system clock or a divided clock (via div_val, up to divide-by-256).
- **Timer Interrupt**
  - Interrupt generation is supported and can be enabled or disabled via control registers.

1.2. Detail Functional Description

1.2.1. Counter Module

64-bit count-up.

Counting mode:

- **Default mode**: counter's speed is same as system clock.
- **Control mode**: when enabled by writing 1 to TCR.div_en bit, the counter's speed is determined by the divisor value set in TCR.div_val.

Counter continues counting when interrupt occurs.

Counter continues counting when overflow occurs.

- **[Standard only]**: when timer_en changes from High to Low (H → L), counter needs to be initialized by software. When timer_en is L → H again, timer can work normally.
- **[Advanced]**: supports halted mode (described in next section).
- **[Advanced]**: When timer_en changes from High to Low, the counter is cleared to its initial value. When timer_en is L → H again, timer can work normally.

Note: div_en and div_val are not related to frequency divider (clock divider). These settings only control **when** the counter increments, not the clock source itself.

1.2.2. Halted Mode (Advanced Level Only)

In advanced mode, the Timer IP core provides a **halted mode** that allows the counter to pause its operation temporarily during debugging. This feature is particularly useful for real-time system analysis and control, where it is essential to halt the timer without losing its state or timing consistency.

Conditions for Entering Halted Mode

The counter can enter the halted state only when both of the following conditions are met:

1. **Debug mode is active**, indicated by the debug_mode input signal being set to high.
2. **A halt request is issued**, by writing logic high (1) to the halt_req bit in the Timer Halt Control and Status Register (THCSR).

Once these two conditions are satisfied, the counter stops incrementing, and the halt_ack bit in THCSR is asserted (1) to acknowledge that the halt request has been accepted.

Behavior in Halted Mode

- While in halted mode, the counter retains its current value and suspends all counting activity.
- The timer will remain halted as long as halt_req remains high.
- To resume normal counting, the halt_req bit must be cleared to 0. Once cleared, the halt_ack bit will also deassert, and the counter will resume incrementing from the previous value.
- Importantly, the **timing interval between counts remains consistent**, even when the timer is halted and resumed. This ensures accurate and predictable behavior for time-dependent systems.

Timing Example

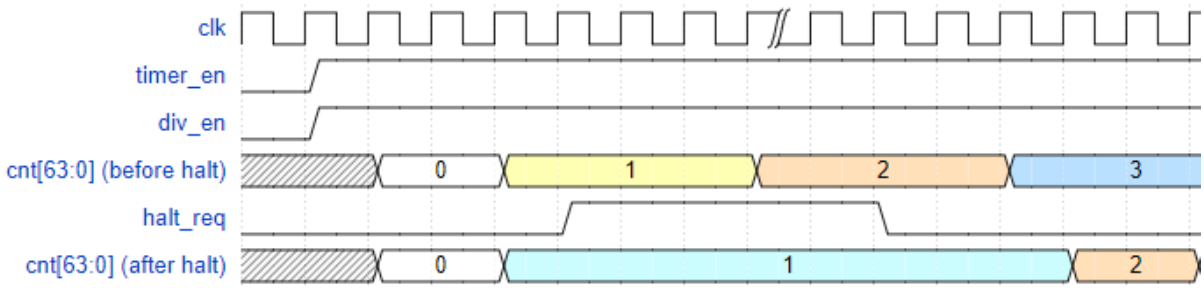


Figure 1.1 - below illustrates a waveform example with div_val = 2

Timer interrupt is generated when **both conditions are met**:

- Interrupt is **enabled** (TIER.int_en = 1).
- Counter value **matches** the compare value (TCMP1:TCMP0).

When interrupt is triggered:

- tim_int output is asserted (1).
- TISR.int_st bit is set to 1.

Interrupt signal (tim_int) remains active until one of the following occurs:

- Software writes 1 to TISR.int_st → clears interrupt.
- Interrupt is disabled (TIER.int_en = 0).

Masking behavior:

- Disabling int_en will **mask** the interrupt signal (tim_int = 0), but int_st bit remains **set** until cleared.

Counter continues to run normally during interrupt events.

Compare value is 64-bit wide, stored in:

- TCMP0: lower 32 bits.
- TCMP1: upper 32 bits.

1.2.4. Counting Mode

- In **default mode**, the counter increments at system clock rate. → This is equivalent to div_val = 0.
- In **control mode**, counting speed can be adjusted by configuring:
  - TCR.div_en = 1
  - TCR.div_val[3:0]: sets the number of cycles to wait before incrementing.
- **Restriction** when timer_en = 1:
  - **Standard level**: Testbench should avoid changing div_en and div_val.
  - **Advanced level**: Hardware blocks changes to div_en and div_val, generates an **error response** if access is attempted.
- div_val does **not divide the system clock**, it only controls when the counter is allowed to increment.

The waveform below illustrates how div_val affects counting behavior:

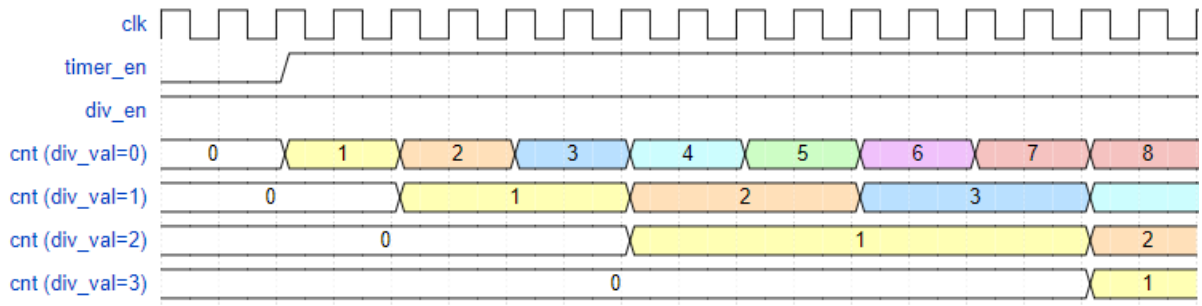


Figure 1.2 - Waveform of counter in control mode with different div_val values

1.2.5. Register/APB Slave

- **Address space**: 4KB (0x4000_1000 – 0x4000_1FFF)
- **Access to reserved area**:
  - **Read** returns zero (RAZ – Read-As-Zero)
  - **Write** is ignored (WI – Write-Ignored)

- **System clock frequency**: 200 MHz
- **Standard level**:
    - Only supports **APB 32-bit** transfer
    - No **wait states** and **no error handling**
- **Advanced level**:
    - **Support byte access**: Bus can access individual bytes in a register using pstrb
    - **Support 1-cycle wait state** to improve timing and APB response
    - **Support error handling** for prohibited access:
        - Writing invalid value to TCR.div_val
        - Changing div_en or div_val while timer_en is high (timer is running)
    - When error occurs:
        - **Data is not written** into the register bits/fields
        - An **error response** is generated (pslverr = 1)

1.3. Register Summary

The Timer IP includes a set of memory-mapped registers starting at base address 0x4000_1000, which are used to configure and monitor the timer's operation. Each register is assigned a specific offset from the base address.

These registers are accessed via the APB bus, and are organized as shown in the table below:

Offset	Abbreviation	Register Name	Description
0x00	TCR	Timer Control Register	Controls timer operation, including timer_en, div_en, and div_val.
0x04	TDR0	Timer Data Register 0	Holds lower 32 bits of the 64-bit counter value.
0x08	TDR1	Timer Data Register 1	Holds upper 32 bits of the 64-bit counter value.
0x0C	TCMP0	Timer Compare Register 0	Lower 32 bits of the interrupt compare value.
0x10	TCMP1	Timer Compare Register 1	Upper 32 bits of the interrupt compare value.
0x14	TIER	Timer Interrupt Enable Register	Enables or disables timer interrupt generation.
0x18	TISR	Timer Interrupt Status Register	Indicates whether an interrupt has been triggered.
0x1C	THCSR	Timer Halt Control Status Register	Manages halt request and acknowledge in debug mode.
Others	—	Reserved	Reserved address space. Read returns zero, write is ignored.

Table 1.1 Timer Register Summary

1.4. Register Specification

Timer Control Register -TCR

Bit	Name	Type	Default Value	Description
31:12	Reserved	–	20'h0	Reserved
11:8	div_val	RW	4'b0001	Counter control mode setting: • 0000: divide by 1 (no division) • 0001: divide by 2 (default) • 0010: divide by 4 • 0011: divide by 8 • 0100: divide by 16 • 0101: divide by 32 • 0110: divide by 64 • 0111: divide by 128 • 1000: divide by 256 • Others: **prohibited**, ignored on write. 🔷 Note: Cannot change div_val while timer_en = 1. 🔶 Advanced level: writing prohibited values triggers **error response**.
7:2	Reserved	RO	6'b0	Reserved
1	div_en	RW	1'b0	Counter control mode enable: • 0: Disabled – counter runs at system clock speed. • 1: Enabled – counting speed is controlled by div_val. 🔷 Note: Cannot change div_en while timer_en = 1. 🔶 Advanced level: changing div_en while running triggers **error response**.
0	timer_en	RW	1'b0	Timer enable: • 0: Disabled – counter stops. • 1: Enabled – counter starts counting. 🔶 Advanced level: When timer_en changes from 1 → 0, counter is reset (TDR0/1 = 0).

Table 1.2 – Timer Control Register (TCR)

Timer Data Register 0 (TDR0)

Bit	Name	Type	Default Value	Description
31:0	TDR0	RW	32'h0000_0000	Lower 32 bits of the 64-bit counter value. ◆ Advanced level: The value of this register is cleared to its initial value when timer_en transitions from **High to Low (H→L)**.

Table 1.3 – Timer Data Register 0 (TDR0)

Timer Data Register 1 –TDR1

Bit	Name	Type	Default Value	Description
31:0	TDR1	RW	32'h0000_0000	Upper 32 bits of the 64-bit counter value. ◆ Advanced level: The value of this register is cleared to its initial value when timer_en transitions from **High to Low (H→L)**.

Table 1.4 – Timer Data Register 1 (TDR1)

Timer Compare Register 0 (TCMP0)

Bit	Name	Type	Default Value	Description
31:0	TCMP0	RW	32'hFFFF_FFFF	Lower 32 bits of the 64-bit compare value. Interrupt is asserted when the counter value equals the full compare value (TCMP1:TCMP0).

Table 1.5 – Timer Compare Register 0 (TCMP0)

Timer Compare Register 1 –TCMP1

Bit	Name	Type	Default Value	Description
31:0	TCMP1	RW	32'hFFFF_FFFF	Upper 32 bits of the 64-bit compare value. Interrupt is asserted when the counter value equals the full compare value (TCMP1:TCMP0).

Table 1.6 – Timer Compare Register 1 (TCMP1)

Timer Interrupt Enable Register (TIER)

Bit	Name	Type	Default Value	Description
31:1	Reserved	RO	31'h0	Reserved
0	int_en	RW	1'b0	Timer interrupt enable: • 0: Timer interrupt is disabled. • 1: Timer interrupt is enabled. When this bit is 0, no interrupt output is generated. When set to 1, interrupt is triggered if the counter matches the compare value. If int_en is cleared (0) while an interrupt is active, the interrupt output (tim_int) is masked but the interrupt pending bit (TISR.int_st) is **not cleared**.

Table 1.7 – Timer Interrupt Enable Register (TIER)

Timer Interrupt Status Register (TISR)

Bit	Name	Type	Default Value	Description
31:1	Reserved	RO	31'h0	Reserved
0	int_st	RW1C	1'b0	Timer interrupt status bit (interrupt pending bit): • 0: No interrupt trigger condition has occurred. • 1: Interrupt trigger condition has occurred. - **Write 1** to clear the bit when it is set. - **Write 0** when bit = 1 → no effect. - **Write to this bit when it is 0** → no effect. ◆ Note: Counter continues counting normally even when an interrupt is triggered (i.e., counter reached compare value).

Table 1.8 – Timer Interrupt Status Register (TISR)

Timer Halt Control Status Register (THCSR)

Bit	Name	Type	Default Value	Description
31:2	Reserved	RO	30'h0	Reserved
1	halt_ack	RO	1'b0	**[Standard level]**: Reserved bit (no function) **[Advanced level]**: Timer halt acknowledge - 0: Timer is **not halted** - 1: Timer is **halted** Timer only accepts halt request when debug_mode = 1.
0	halt_req	RW	1'b0	**[Standard level]**: Normal read/write bit, no functional behavior **[Advanced level]**: Timer halt request - 0: No halt requested - 1: Timer is requested to halt.

Table 1.9 – Timer Halt Control Status Register (THCSR)

CHAPTER 2: RTL DESIGN

2.1 IO Port List

Signal name	Width	Direction	Description
sys_clk	1	Input	System clock signal. All internal logic is synchronized to this clock.
sys_rst_n	1	Input	Asynchronous system reset, active low.
tim_psel	1	Input	APB select signal. High when the timer is selected for a transaction.
tim_pwrite	1	Input	APB write control signal. 1 = write, 0 = read.
tim_penable	1	Input	APB enable signal. High during second phase of transfer.
tim_paddr	12	Input	APB address bus. Covers 4KB space (2^{12} = 4096), from 0x4000_1000 to 0x4000_1FFF.
tim_pwdata	32	Input	APB write data bus. Carries data from master to timer.
tim_prdata	32	Output	APB read data bus. Carries data from timer to master.
tim_pstrb	4	Input	Byte strobe signal. Each bit enables one byte of tim_pwdata (Advanced level only).
tim_pready	1	Output	Indicates that the timer is ready to complete the APB transaction.
tim_pslverr	1	Output	APB slave error signal. Asserted on illegal access or error condition (Advanced level only).
tim_int	1	Output	Timer interrupt output. Asserted when counter reaches compare value and interrupt is enabled.
dbg_mode	1	Input	Debug mode indicator. Enables halt functionality when halt_req is set (Advanced level only).

Table 1.10 – Top Module Port List (timer_top)

2.2 Block Diagram

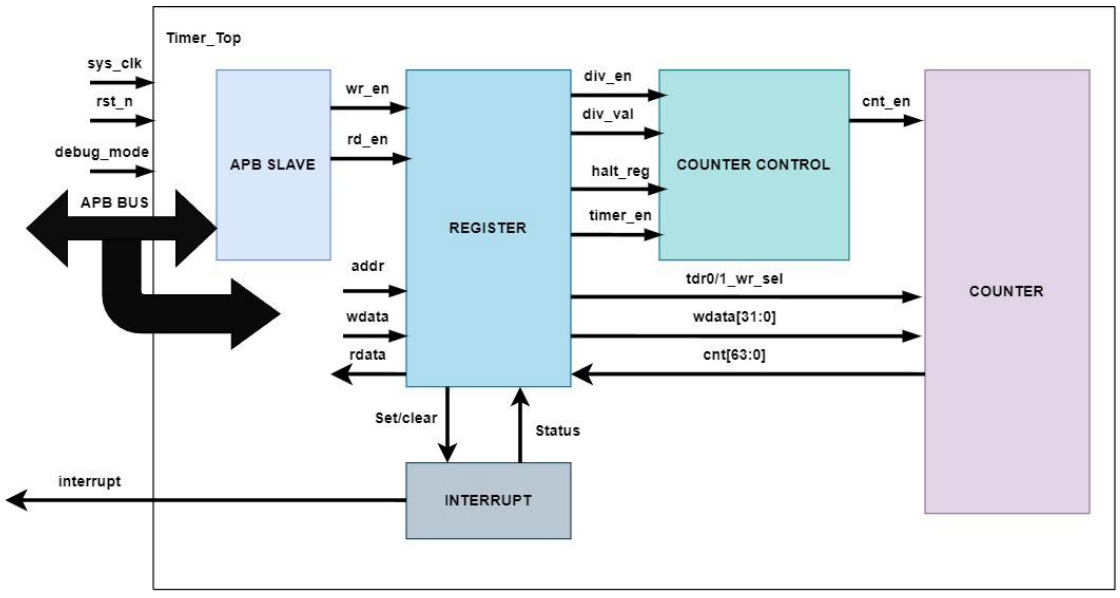


Figure 1.3 - Timer IP Block Diagram

The Timer IP block diagram comprises the following components:

1. APB Slave

Handles APB bus communication, receiving system signals (sys_clk, rst_n, debug_mode) and APB interface signals (address, data, control).

2. Register

Stores and manages control/status registers (TCR, TDR0/1, TCMP0/1, TIER, TISR, THCSR). It exchanges data/control signals with APB Slave and provides configuration parameters to Counter Control.

3. Counter Control

Controls the counter's operation (cnt_en) based on configuration signals (div_en, div_val, halt_reg, timer_en) from Register.

4. Counter

64-bit counter incremented based on the cnt_en signal. It communicates data with Register and Interrupt blocks.

5. Interrupt

Generates and manages interrupt signals, compares counter value against threshold values (TCMP registers), and communicates status/set-clear signals with the Register.

This diagram clearly illustrates internal signal flows and interactions within the Timer IP.

2.3 Timing Diagram

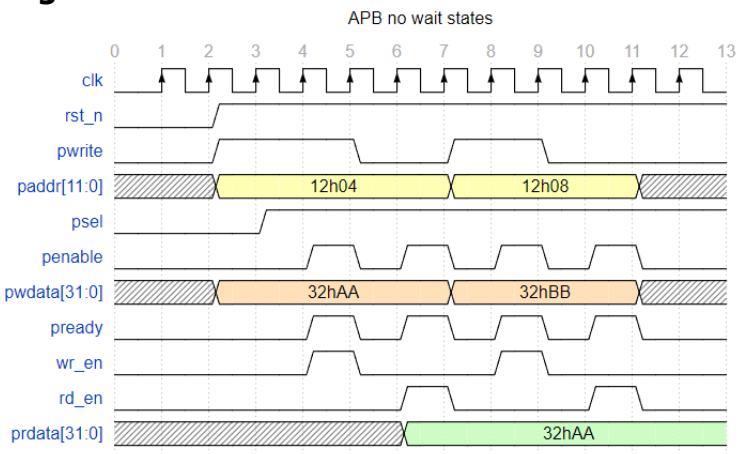


Figure 1.4 – APB no wait states

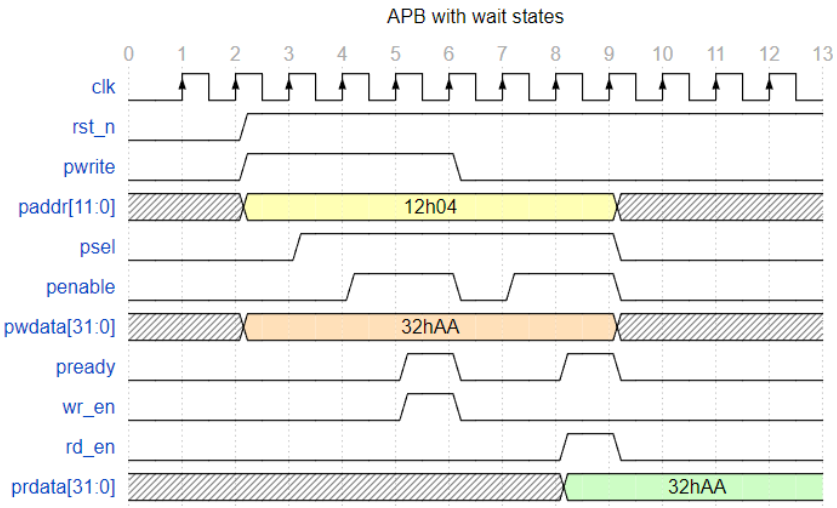


Figure 1.5 – APB with wait states

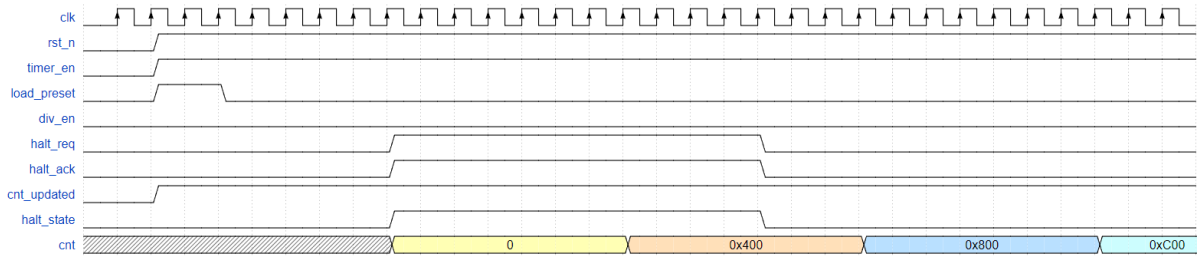


Figure 1.6 – APB no wait states

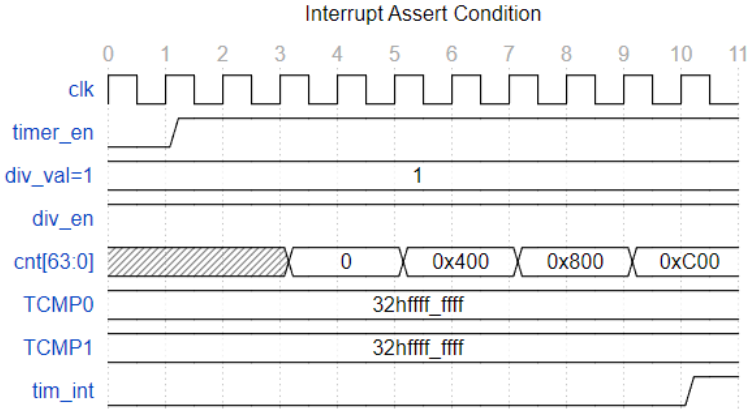


Figure 1.7 – Interrupt Assert Condition

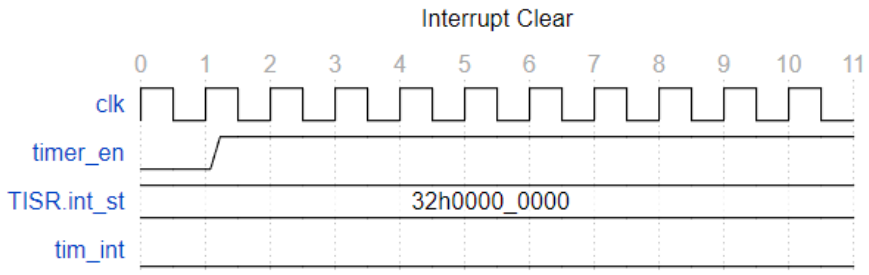


Figure 1.8 – Interrupt Clear

2.4 Logic Diagram

Logic Diagram for APB Slave module:

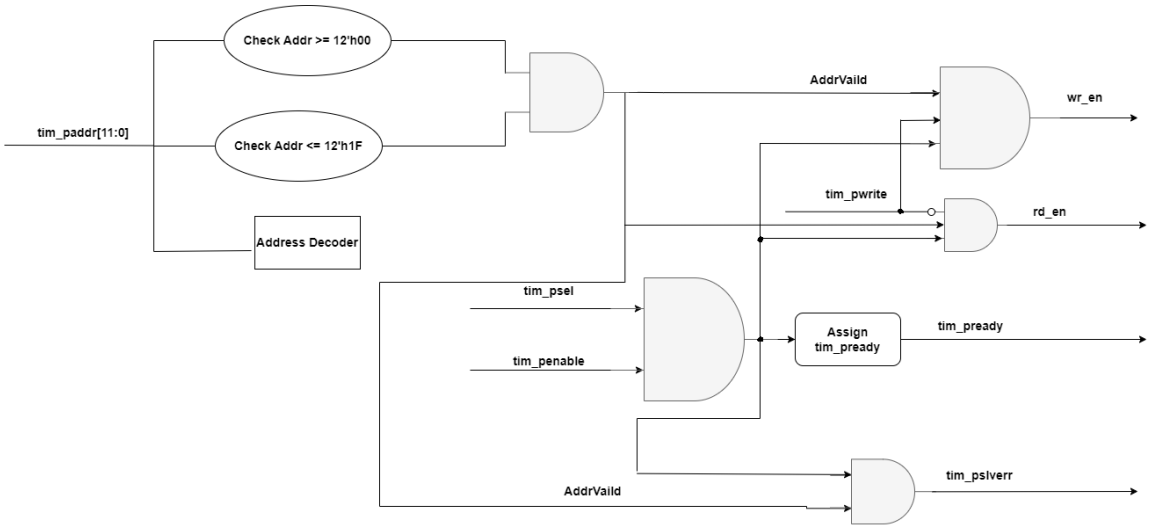


Figure 1.9 – Logic Diagram for APB Slave module

Clock Divider logic diagram

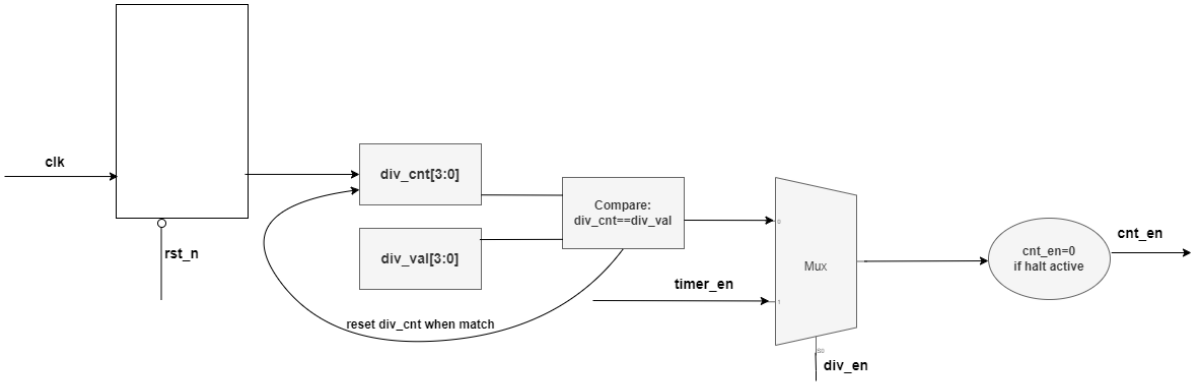


Figure 1.10 – Clock Divider logic diagram

TCR Register's write logic diagram

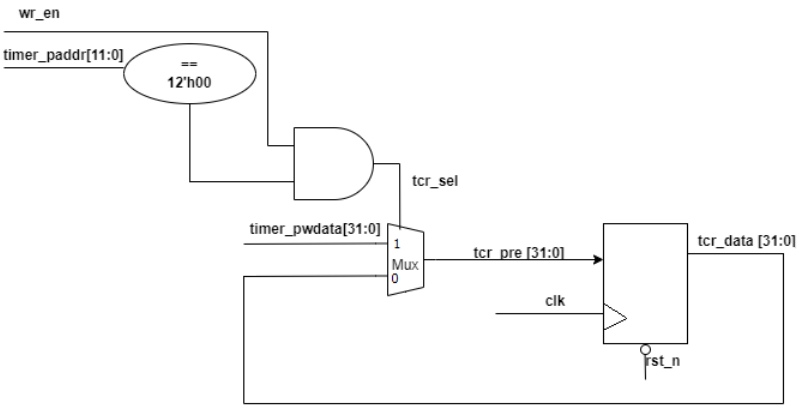


Figure 1.11 – TCR Register's write logic diagram

Timer Data Register 0:

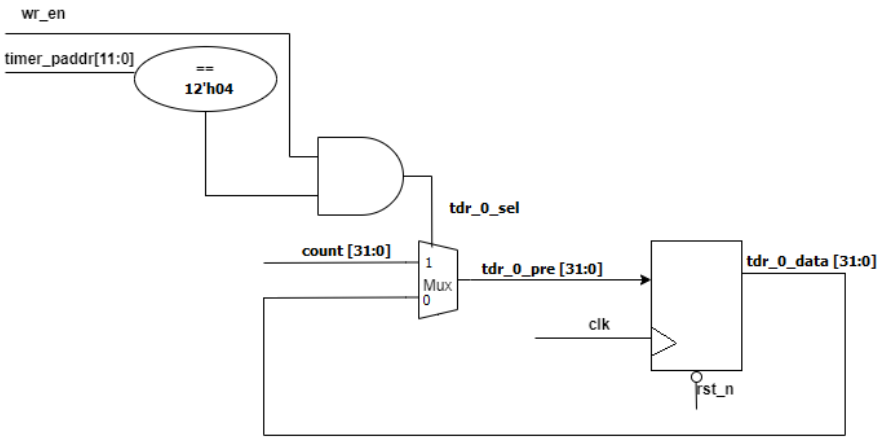


Figure 1.12 – Timer Data Register 0 logic diagram

Timer Data Register 1:

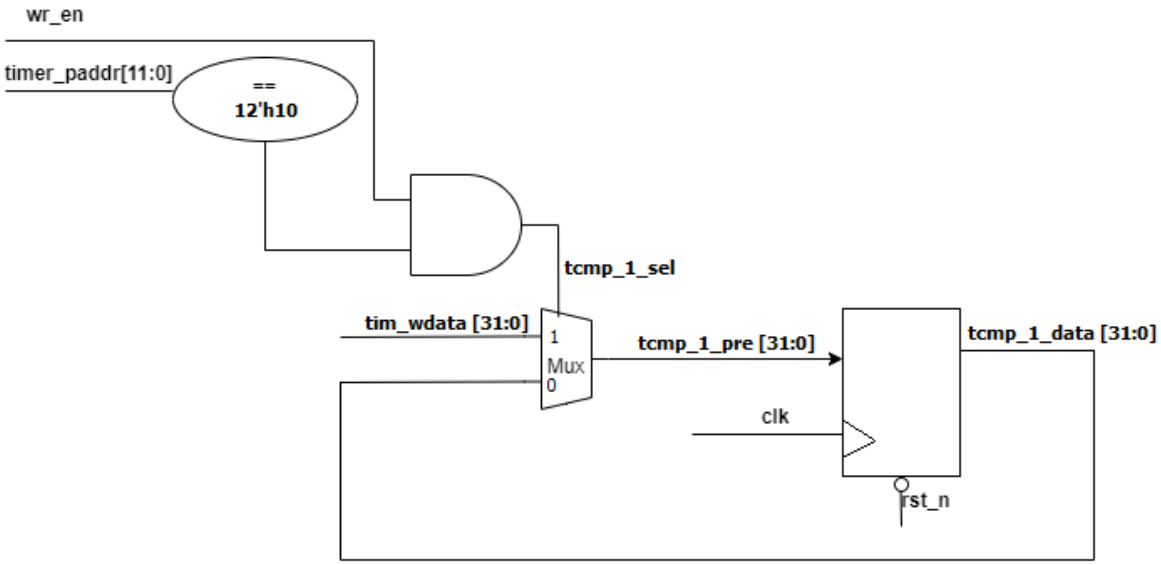


Figure 1.13 – Timer Data Register 1 logic diagram

TIER Register's write logic diagram

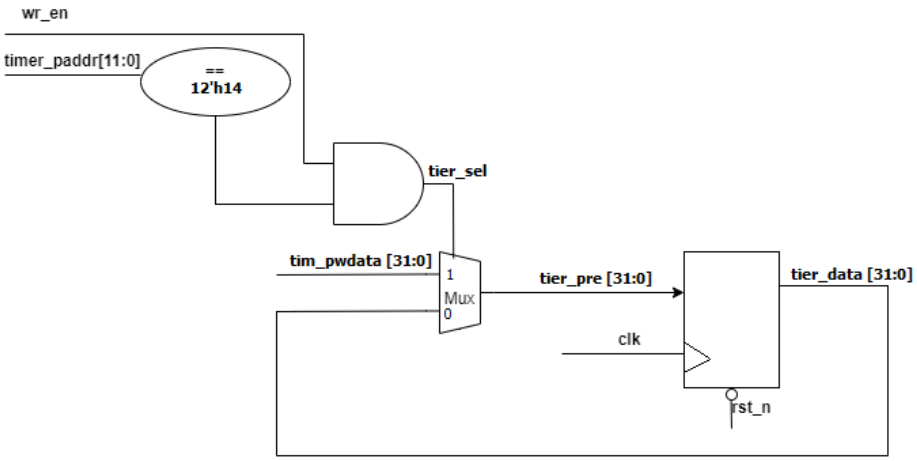


Figure 1.14 – TIER Register's write logic diagram

Timer Interrput Status Register:

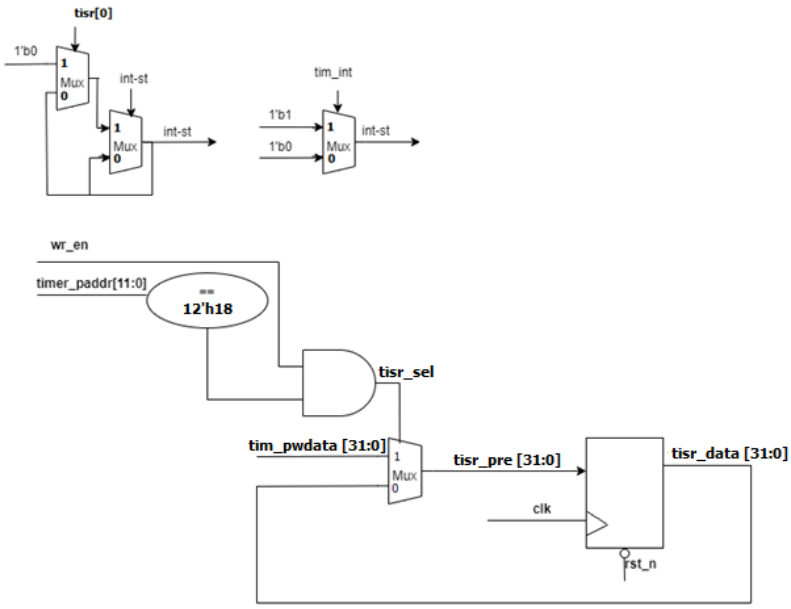


Figure 1.15 – Timer Interrput Status Register logic diagram

THCSR Register's logic diagram

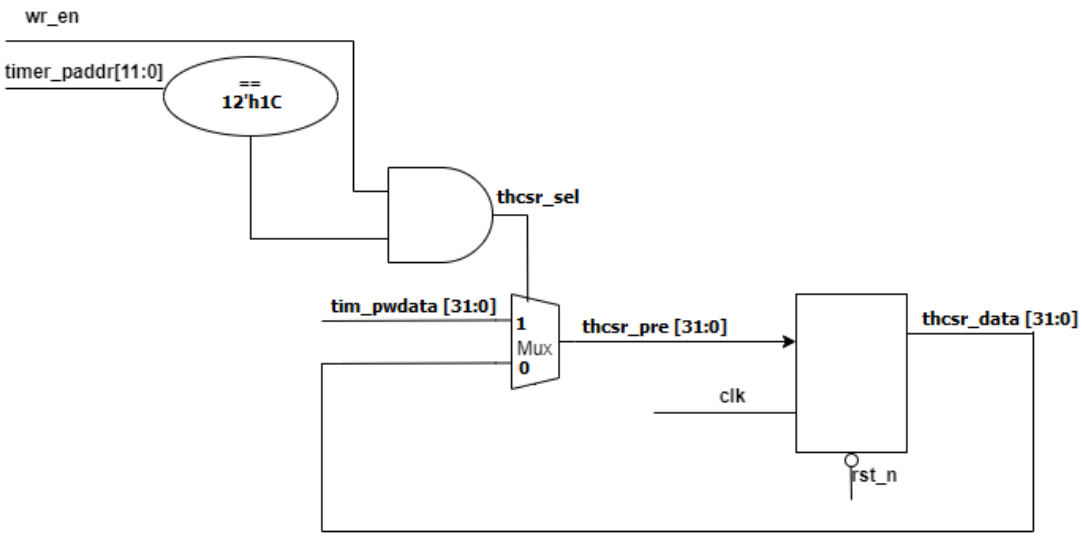


Figure 1.16 – THCSR Register's logic diagram

Register Read Logic

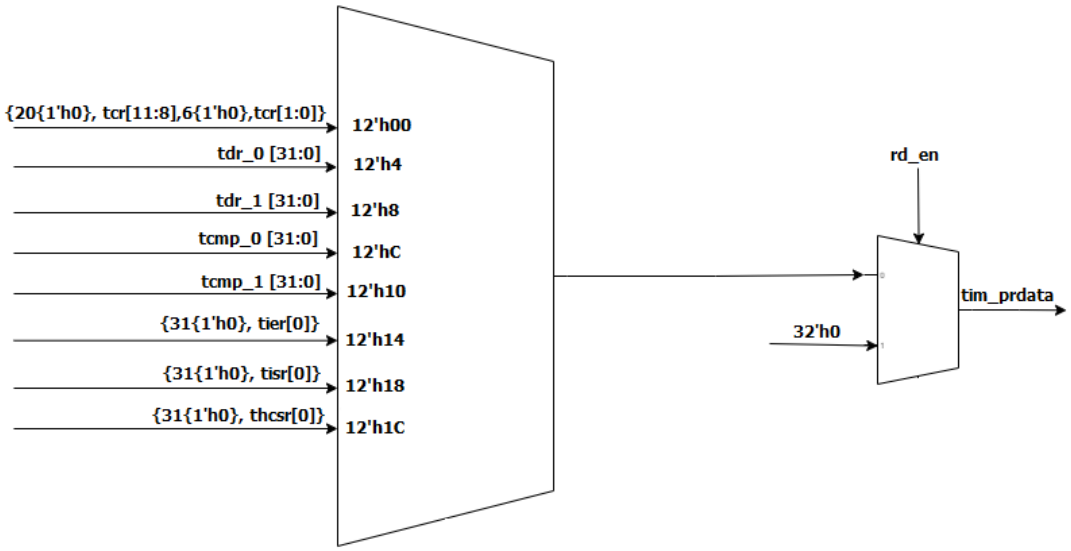


Figure 1.17 – Register Read Logic

Counter Enable – Clear mode and Halt mode logic diagram

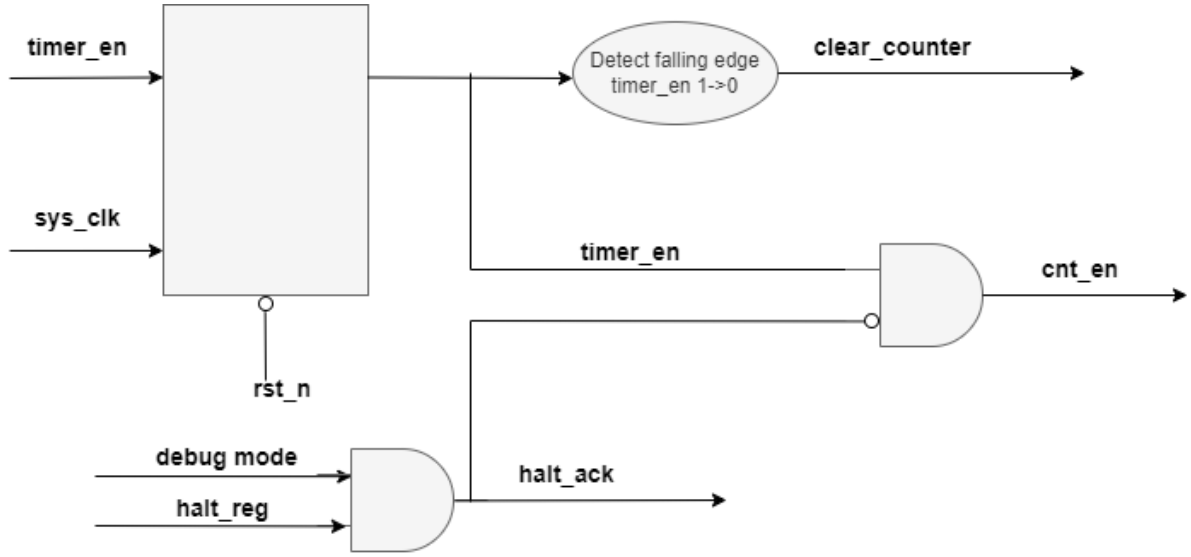


Figure 1.18 – Counter Enable – Clear mode and Halt mode logic diagram

Logic diagram for Counter Module

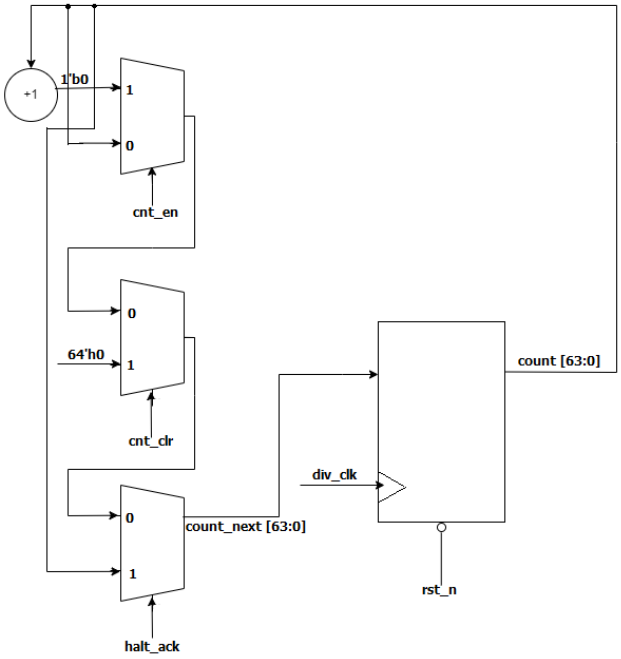


Figure 1.19 – Logic diagram for Counter Module

Logic diagram for Interrupt module :

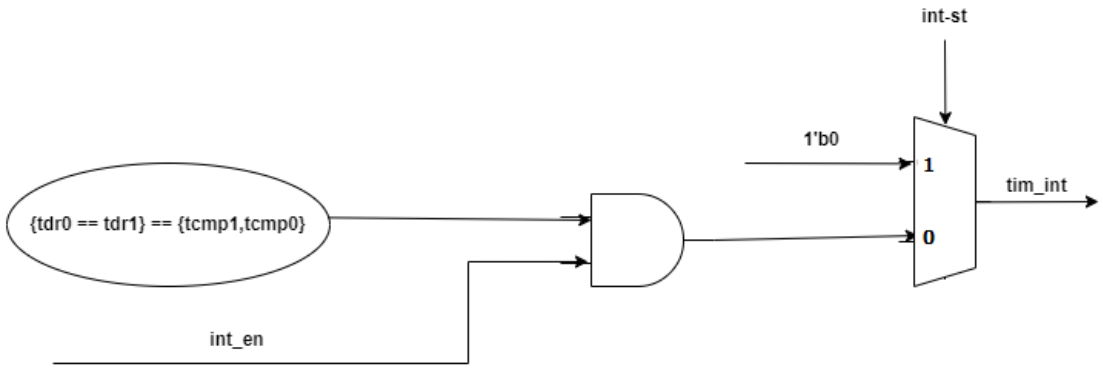


Figure 1.20 – Logic diagram for Interrupt module

CHAPTER 3: VERIFICATION PLAN

3.1 Verification Plan

ID	Item	Sub item 1	Sub item 2	Method	Class	Test sequence
1	**APB Protocol Check**	Write operation (w_en)	Read operation (r_en)	Directed	Functional	**Call** apb_protocol_check(...) from **apb_check.v** with different APB signal conditions. • *Fail cases*: use (psel=0, penable=0) or mismatch expected tim_pready => check [FAIL] messages. • *Pass cases*: use (psel=1, penable=1, pwrite=1 or 0) => expect [PASS] and correct w_en / r_en. Examine the log for each transaction.
2	**Register Initial Value**	TDR / TCMP / TIER / TISR	Various default offsets	Directed	Functional	**Use** register_initial_check(...) from **register_val_check.v** across addresses 0x00, 0x04, 0x08, 0x0C, ... • After reset, each address is read in a **pwrite=0** scenario (internally the task toggles APB signals). • Confirm [PASS] if read values match the design's expected defaults (e.g., 0x00000100 for TDR0, 0xFFFFFFFF for TCMP).
3	**Register R/W Check**	TCR, TDRx, etc.	w_en / r_en toggling	Directed	Functional	**Invoke** register_rw_check(...) from **register_mode.v** to write and then read back various registers (e.g., TCR at 0x00, TDR0 at 0x04). • Verify the testbench log for [PASS] if w_en=1 during write, r_en=1 during read, and tim_rdata matches tim_wdata. • *Fail tests* use invalid APB signals (psel=0, penable=0) leading to [FAIL].
4	**Counter Check**	Basic increment	timer_en=1, div_en=0	Directed	Functional	**Use** counter_check(...) from **counter_check.v** with TCR configured so that timer_en=1, div_en=0, count_clr=0 (e.g. 0x00000003). • Let the simulation run for multiple clock cycles. Confirm uut.c_inst.count is increasing each clock. Expect [PASS] if the final count > initial count in the logs.
5	**Counter Check**	Divided increment	timer_en=1, div_en=1	Directed	Functional	Still **in** counter_check.txt, test lines calling counter_check(...) with TCR enabling div_en (and a valid div_val in the lower nibble). • Watch the log to confirm slower increments (the internal int_count must reach int_count_max before toggling cnt_en). • [PASS] if the counter increments only after the correct divide cycle.
6	**Counter Clear/Disable**	count_clr=1	timer_en=0	Directed	Functional	**Within** counter_check(...), pass a TCR that sets count_clr=1 or timer_en=0 mid-test. E.g. 0x00000103 => running, then 0x00000101 => timer_en=0 or count_clr=1. • Confirm that the 64-bit count stops incrementing when timer_en=0, and that it resets to 0 when count_clr=1.
7	**Div Value Control**	Valid div_val (0..8)	Prohibit settings	Directed	Functional	**Call** counter_control_div_val_check(...) from **div_counter_ctrl.v** with TCR values 0x00000003, 0x00000103, ... 0x00000803.

						• Observe the code's log output for each div_val (0..8). The module checks if int_count matches the correct maximum (e.g. 7 for div_val=3). • For an out-of-range div_val, watch for "prohibit settings" or ignoring the new value.
8	**Timer & Div Control**	timer_en & div_en combos	Counting modes	Directed	Functional	**Use** counter_control_timer_div_check(...) from **counter_ctrl_mode.v** with TCR variations: • (timer_en=0, div_en=0): no increment. • (timer_en=1, div_en=0): full-speed count. • (timer_en=0, div_en=1): no increment (timer disabled). • (timer_en=1, div_en=1): slow increment. Verify [PASS] if these modes behave as coded in the logs.
9	**Interrupt Check**	TIER (Enable/Disable)	TISR (RW1C) / tim_int	Directed	Functional	**Call** interrupt_check(...) from **interrupt_check.v** with various addresses for TIER (0x14), TCMP0 (0x0C), TCMP1 (0x10), TCR (0x00), and TISR (0x18). • *Case 1*: TIER=1 => enable interrupts. Force or wait for uut.i_inst.match=1 => check tim_int=1, TISR=1. • *Case 2*: TIER=0 => no interrupt triggered on match. • Write 1 to TISR => verify it clears (tim_int=0, TISR=0).
10	**Interrupt Multi-Match**	TCMP0 & TCMP1	Re-trigger interrupt	Directed	Functional	**In** interrupt_check(...), configure TIER=1, set TCMP0=5, TCMP1=10. Let the counter run (timer_en=1, div_en=1 or 0). • The code forcibly or naturally increments count until match=5 => tim_int=1. Clear TISR => continue counting => match=10 => another interrupt triggers. Check [PASS] if both matches fire separate interrupts and TISR can be cleared each time.

Table 1.11 – Table of Verification Plan

3.2 Verification Environment

Makefile

```
###############################################################################
####################
#This file created by Huy Nguyen
#Updated date: 6/30/2019
#Example run string: make TESTNAME={name_of_testcase} {optional}
#         make TESTNAME=counter_test all
###############################################################################
####################
#Define variables
TESTNAME   ?= run_test
TB_NAME    ?= test_bench
```

```
RADIX    ?= decimal
REGRESS_LIST  ?= pat.list

#==============================
all: build run

all_wave: build run wave

all_cov: build_cov run_cov

build:
  mkdir -p log
  cp -rf ../testcases/$(TESTNAME).v run_test.v
  vlib work
  vmap work work
  vlog  -sv -f compile.f | tee compile.log
build_cov:
  mkdir -p log
  cp -rf ../testcases/$(TESTNAME).v run_test.v
  vlib work
  vmap work work
  vlog -sv +cover=bcesft -f compile.f
run:
  vsim -debugDB -l $(TESTNAME).log -voptargs=+access -assertdebug -c
$(TB_NAME) -do "log -r /*;run -all;"
  cp $(TESTNAME).log ./log
  mv $(TESTNAME).log sim.log
run_cov:
  vsim -coverage -l $(TESTNAME).log -c $(TB_NAME) -voptargs="+cover=bcesft" -
assertdebug -do "coverage save -onexit $(TESTNAME).ucdb; log -r /*;run -all"
  cp $(TESTNAME).log ./log
  mv $(TESTNAME).log sim.log
wave:
  vsim -i -view vsim.wlf -do "add wave vsim:/$(TB_NAME)/*; radix -$(RADIX)" &
gen_cov:
  mkdir -p coverage
  vcover merge IP.ucdb *.ucdb
  vcover report IP.ucdb -output coverage/summary_report.txt
  vcover report -zeros -details -code bcesft -annotate -All -codeAll IP.ucdb -
output coverage/detail_report.txt
gen_html:
  mkdir -p coverage
  vcover merge IP.ucdb *.ucdb
  vcover report -zeros -details -code bcesft -annotate -testhitdataAll -html
IP.ucdb
clean:
  rm -rf work
  rm -rf vsim.dbg
```

```
  rm -rf *.ini
  rm -rf *.log
  rm -rf *.wlf
  rm -rf transcript
  rm -rf *.ucdb
  rm -rf run_test.v
help:
  @echo ""
  @echo "*************************************"
  @echo "** make clean    : clean all compiled data"
  @echo "** make build    : build the design"
  @echo "** make build_cov: build the design in coverage mode"
  @echo "** make run      : run simulation"
  @echo "** make run_cov  : run simulation in coverage mode"
  @echo "** make all      : build and run simulation"
  @echo "** make all_cov  : build and run simulation in coverage mode"
  @echo "** make wave     : open waveform"
  @echo "** make gen_cov  : merge, generate coverage data base and text
coverage report"
  @echo "** make gen_cov  : merge, gemerate coverage data base and html
coverage report"
  @echo "*************************************"
  @echo ""
```

CHƯƠNG 4: NHỮNG THÁCH THỨC CỦA NOMA

4.1 RTL Design

APB Slave Module:

```
module APB_slave (
  input   wire         sys_clk,
  input   wire         sys_rst_n,
  input   wire         tim_psel,
  input   wire         tim_pwrite,
  input   wire         tim_penable,
  input   wire [11:0]  tim_paddr,

  output wire          tim_pready,
  output wire          r_en,
  output wire          w_en
);

  parameter TCR   = 12'h00;
  parameter TDR0  = 12'h04;
  parameter TDR1  = 12'h08;
  parameter TCMP0 = 12'h10;
  parameter TCMP1 = 12'h1C;
  parameter TIER  = 12'h14;
  parameter TISR  = 12'h18;

  wire pready_in;
  reg  pready_out;
  reg  pready_temp;

  assign pready_in = tim_psel && tim_penable;

  always @(posedge sys_clk or negedge sys_rst_n) begin
    if(!sys_rst_n)
      pready_out <= 1'b0;
    else
      pready_out <= pready_in;
  end

  wire pready_up;

  assign pready_up = pready_in && (~pready_out);

  always @(posedge sys_clk or negedge sys_rst_n) begin
  if(!sys_rst_n)
     pready_temp <= 1'b0;
  else
```

```
    pready_temp <= pready_up;

  end

  //pready signal
  assign tim_pready = pready_up;

  //address valid check
  wire AddrValid;
  assign AddrValid = (tim_paddr[11:0] <= 12'h18) && (tim_paddr[11:0] >=
12'h0);

  //write enable signal
  assign w_en = pready_up && tim_pwrite && AddrValid;

  //read enable signal
  assign r_en = pready_up && (~tim_pwrite) && AddrValid;

endmodule
```

Couter-64bit Module

```
module Counter (
  input   wire          sys_clk,
  input   wire          sys_rst_n,
  input   wire          timer_en,
  input   wire          cnt_en,
  input   wire          count_clr,

  output  wire  [63:0]  count

);

reg [63:0] count_temp;

  always @(posedge sys_clk or negedge sys_rst_n) begin
    if(!sys_rst_n)
      count_temp  <= 64'b0;
    else if(count_clr)
       count_temp <= 64'b0;
     else if(cnt_en) begin
       if(timer_en)
           count_temp <= count_temp + 1'b1;
         else
          count_temp <= count_temp;
```

```
          end
      else
        count_temp <= count_temp;
  end

  assign count = count_temp;

endmodule
```

Clock Divider Module

```
module Clock_Divider (
  input   wire         sys_clk,
  input   wire         sys_rst_n,
  input   wire         div_en,
  input   wire [3:0]   div_val,
  input   wire         timer_en,

  output  wire         cnt_en
);

reg   [7:0] int_count;
reg   [7:0] int_count_temp;
reg   [7:0] int_count_max;
wire  [7:0] int_count_next;


  always @* begin
    case (div_val)
     4'b0000: int_count_max = 8'b0000_0000;
     4'b0001: int_count_max = 8'b0000_0001;
     4'b0010: int_count_max = 8'b0000_0011;
     4'b0011: int_count_max = 8'b0000_0111;
     4'b0100: int_count_max = 8'b0000_1111;
     4'b0101: int_count_max = 8'b0001_1111;
     4'b0110: int_count_max = 8'b0011_1111;
     4'b0111: int_count_max = 8'b0111_1111;
     4'b1000: int_count_max = 8'b1111_1111;
     default: int_count_max = 8'b0000_0001;
    endcase
  end

  always @(posedge sys_clk or negedge sys_rst_n) begin
    if(!sys_rst_n) begin
     int_count      <= 8'b0000_0000;
           int_count_temp <= 8'bx;
    end else if(cnt_en)
      int_count <= 8'b0000_0000;
    else if(timer_en) begin
```

```
      int_count <= int_count_next + 1'b1;
                  int_count_temp <= int_count_next;
    end else
      int_count <= int_count_next;
  end

  assign int_count_next = int_count;

  assign cnt_en = (div_en && div_val != 4'b0000)  ? (int_count ==
int_count_max) : sys_clk;

endmodule
```

Interrupt Module

```
module Interrupt (
  input   wire   sys_clk,
  input   wire   sys_rst_n,
  input   wire   int_en,
  input   wire   match,
  input   wire   clear,

  output  wire   int_st,
  output  wire   tim_int
);

reg int_temp;

  always @(posedge sys_clk) begin
    if(~sys_rst_n)
      int_temp <= 1'b0;
    else if(~int_en)
      int_temp <= 1'b0;
    else if(clear)
      int_temp <= 1'b0;
    else if(int_en && match)
      int_temp <= 1'b1;
    else
      int_temp <= int_temp;
  end
assign int_st  = int_temp;
assign tim_int = int_temp;


endmodule
```

Register Module

```
module Register (
  input   wire          sys_clk,
  input   wire          sys_rst_n,
  input   wire          w_en,
  input   wire          r_en,
  input   wire [11:0]   addr,
  input   wire [31:0]   wdata,
  input   wire [63:0]   count,
  input   wire          int_st,

  output  wire [31:0]   rdata,
  output  wire          div_en,
  output  wire [3:0]    div_val,
  output  wire          timer_en,
  output  wire          int_en,
  output  wire          match,
  output  wire          clear,
  output  wire          count_clr
);

  parameter TCR   = 12'h00;
  parameter TDR0  = 12'h04;
  parameter TDR1  = 12'h08;
  parameter TCMP0 = 12'h0C;
  parameter TCMP1 = 12'h10;
  parameter TIER  = 12'h14;
  parameter TISR  = 12'h18;

//Declaration
wire tcr_w_sel;

  //Write data condition
  assign tcr_w_sel    = w_en && (addr == TCR);

wire [2:0] tcr_data_pre;
reg  [2:0] tcr_data;


  assign tcr_data_pre = tcr_w_sel ? wdata[2:0]: tcr_data;

  //Update data
  always @(posedge sys_clk or negedge sys_rst_n) begin
    if(!sys_rst_n)
          tcr_data  <= 3'b0;
    else
    tcr_data  <= tcr_data_pre;
```

```
  end

  assign timer_en  = tcr_data[0];
  assign div_en    = tcr_data[1];
  assign count_clr = tcr_data[2];

  //Divisor Value
wire [3:0] div_val_pre;
wire [3:0] div_val_next;
reg  [3:0] div_val_temp;
reg  [3:0] tcr_data1;

assign div_val_pre = tcr_w_sel ? wdata[11:8] : div_val_next;

// In prohibit settings (div_val > 9): div_val is not changed
assign div_val_next = ((4'b0000 <= wdata[11:8]) && (wdata[11:8] <= 4'b1000)) ?
div_val_pre : div_val_temp;

// Update data
always @(posedge sys_clk or negedge sys_rst_n) begin
    if (!sys_rst_n) begin
        div_val_temp <= 4'b0001;  // Default value after reset
        tcr_data1    <= 4'b0001;
    end else begin
        div_val_temp <= div_val_next;
        tcr_data1    <= div_val_next;  // Sửa: cập nhật trực tiếp, không dùng
giá trị trễ
    end
end
assign div_val   = tcr_data1;


  //Declaration
wire tdr0_w_sel;
wire [31:0] tdr0_data_pre;
reg  [31:0] tdr0_data;

  //Write data condition
  assign tdr0_w_sel = w_en && (addr == TDR0);
  assign tdr0_data_pre   = tdr0_w_sel ? count[31:0] : tdr0_data;

  //Update data
  always @(posedge sys_clk or negedge sys_rst_n) begin
    if(!sys_rst_n)
    tdr0_data <= 32'h0;
          else
          tdr0_data <= tdr0_data_pre;
  end
```

```
//Declaration
wire tdr1_w_sel;
wire [31:0] tdr1_data_pre;
reg  [31:0] tdr1_data;


  //Write data condition
  assign tdr1_w_sel = w_en && (addr == TDR1);
  assign tdr1_data_pre   = tdr1_w_sel ? count[63:32] : tdr1_data;

  //Update data
  always @(posedge sys_clk or negedge sys_rst_n) begin
    if(!sys_rst_n)
      tdr1_data <= 32'h0;
    else
      tdr1_data <= tdr1_data_pre;
  end

//Declaration
wire tcmp0_w_sel;
wire [31:0] tcmp0_data_pre;
reg  [31:0] tcmp0_data;

  //Write data condition
  assign tcmp0_w_sel = w_en && (addr == TCMP0);
  assign tcmp0_data_pre   = tcmp0_w_sel ? wdata : tcmp0_data;

  //Update data
  always @(posedge sys_clk or negedge sys_rst_n) begin
    if(!sys_rst_n)
      tcmp0_data <= 32'hFFFF_FFFF;
    else
      tcmp0_data <= tcmp0_data_pre;
  end

//Declaration
wire tcmp1_w_sel;
wire [31:0] tcmp1_data_pre;
reg  [31:0] tcmp1_data;

  //Write condition
  assign tcmp1_w_sel = w_en && (addr == TCMP1);
  assign tcmp1_data_pre   = tcmp1_w_sel ? wdata : tcmp1_data;

  //Update data
  always @(posedge sys_clk or negedge sys_rst_n) begin
    if(!sys_rst_n)
      tcmp1_data <= 32'hFFFF_FFFF;
```

```
    else
      tcmp1_data <= tcmp1_data_pre;
  end

//Declaration
wire tier_w_sel;
wire tier_data_pre;
reg  tier_data;

  //Write condition
  assign tier_w_sel = w_en && (addr == TIER);
  assign tier_data_pre  = tier_w_sel ? wdata[0] : tier_data;

  //Update data
  always @(posedge sys_clk or negedge sys_rst_n) begin
    if(!sys_rst_n)
      tier_data <= 1'b0;
    else
      tier_data <= tier_data_pre;
  end

  assign int_en = tier_data;



//Declaration
wire status;
wire tisr_w_sel;
wire tisr_data_pre;
reg  tisr_data;

  //Write enable condition
  assign tisr_w_sel     =   w_en && (addr == TISR);




  assign clear  = (tisr_w_sel && int_st && wdata[0]);



reg [31:0] rd;


  always @(posedge sys_clk or negedge sys_rst_n) begin
    if(r_en) begin
      case (addr)
        TCR     : rd = {20'b0 , div_val [3:0] , 5'b0 , count_clr , div_en ,
timer_en };
```

```
        TDR0    : rd = tdr0_data;
        TDR1    : rd = tdr1_data;
        TCMP0   : rd = tcmp0_data;
        TCMP1   : rd = tcmp1_data;
        TIER    : rd = {31'b0 , int_en};
        TISR    : rd = {31'b0 , int_st};
        default : rd = 32'h0;
      endcase
    end
  end

  assign rdata = rd;

  assign match = (count[63:0] == {tcmp1_data,tcmp0_data});

endmodule
```

Timer Top Module

```
module Timer_top (
  input   wire        sys_clk,
  input   wire        sys_rst_n,
  input   wire        tim_psel,
  input   wire        tim_pwrite,
  input   wire        tim_penable,
  input   wire [11:0] tim_paddr,
  input   wire [31:0] tim_wdata,

  output  wire [31:0] tim_rdata,
  output  wire        tim_pready,
  output  wire        tim_int
);


wire r_en;
wire w_en;

APB_slave apb_inst(
        .sys_clk(sys_clk),
  .sys_rst_n(sys_rst_n),
  .tim_psel(tim_psel),
  .tim_pwrite(tim_pwrite),
  .tim_penable(tim_penable),
  .tim_paddr(tim_paddr),
  .tim_pready(tim_pready),
  .r_en(r_en),
  .w_en(w_en)
);
```

```
wire int_st;
wire div_en;
wire timer_en;
wire int_en;
wire match;
wire clear;
wire count_clr;
reg [3:0]  div_val;
wire [63:0] count;

Register r_inst (
  .sys_clk(sys_clk),
  .sys_rst_n(sys_rst_n),
  .w_en(w_en),
  .r_en(r_en),
  .addr(tim_paddr),
  .wdata(tim_wdata),
  .count(count),
  .int_st(int_st),
  .rdata(tim_rdata),
  .div_en(div_en),
  .div_val(div_val),
  .timer_en(timer_en),
  .int_en(int_en),
  .match(match),
  .clear(clear),
  .count_clr(count_clr)
);

wire cnt_en;

Clock_Divider cd_inst (
         .sys_clk(sys_clk),
   .sys_rst_n(sys_rst_n),
   .div_en(div_en),
   .div_val(div_val),
   .timer_en(timer_en),
   .cnt_en(cnt_en)
);


Counter c_inst (
         .sys_clk(sys_clk),
   .sys_rst_n(sys_rst_n),
   .timer_en(timer_en),
   .cnt_en(cnt_en),
   .count_clr(count_clr),
   .count(count)
);
```

```
Interrupt i_inst (
         .sys_clk(sys_clk),
   .sys_rst_n(sys_rst_n),
         .int_en(int_en),
   .match(match),
   .clear(clear),
   .int_st(int_st),
   .tim_int(tim_int)
);


endmodule
```

4.2 Test Cases

apb_check.v

```
task run_test;
  begin
    #100;
    $display("\n=======================[ APB PROTOCOL CHECK
]=====================\n");

    $display("\n==========================[ FAIL CASE
]==========================\n");
    // Em se cho Test Case FAIL voi pwrite = 1, paddr = 12'h04, psel = 0,
penable = 0, wdata = 32'b0
    apb_protocol_check(1, 12'h004, 0, 0, 32'hDEADBEEF, 0);
    apb_protocol_check(1, 12'h004, 1, 0, 32'hCAFEBABE, 0);
    apb_protocol_check(1, 12'h004, 0, 1, 32'hCAFED00D, 0);
    $display("\n=========================[ SUCCESS CASE
]=========================\n");
    // Em se cho Test Case SUCCESS voi pwrite = 1, paddr = 12'h04, psel = 1,
penable = 1, wdata = 32'b0
    apb_protocol_check(1, 12'h04, 1, 1, 32'b0,1);
    apb_protocol_check(1, 12'h004, 1, 1, 32'h12345678, 1);
    apb_protocol_check(0, 12'h004, 1, 1, 32'h00000000, 1);

    $display("\n=====================[ ADDITIONAL PASS CASES
]=====================\n");
    // Em se cho Test Case ADDITIONAL 1 voi pwrite = 1, paddr = 12'h08, psel =
1, penable = 1, wdata = 32'b1
    apb_protocol_check(1, 12'h08, 1, 1, 32'b1,1);
    // Em se cho Test Case ADDITIONAL 2 voi pwrite = 1, paddr = 12'h0C, psel =
1, penable = 1, wdata = 32'b10
    apb_protocol_check(1, 12'h0C, 1, 1, 32'b10,1);
```

```
    $display("\n=========================[ ALL TEST CASES EXECUTED
]=========================\n");
  end
endtask
```

counter_control_mode.v

```
task run_test;
  begin
    #100;
    $display("=============================================================================
==========");
    $display("|   COUNTER CONTROL TIMER_EN && DIV_EN CHECK  - Test
Results              |");
    $display("=============================================================================
==========");

    //---------------------------------------------------------------------------
---------------------------------
    //                                        DETAILED TEST CASES
    //---------------------------------------------------------------------------
---------------------------------
    $display("\n[DETAILED TEST CASES]");

    // Test Case A:
    // TCR = 32'h0000_0203: div_val = 4'b0010, div_en = 1, timer_en = 1
    // => Expected: Counter should increment (control mode: counting speed
divided by 4)
    counter_control_timer_div_check(1, 12'h00, 1, 1, 32'h00000203);

    // Test Case B:
    // TCR = 32'h0000_0000: div_val = 4'b0000, div_en = 0, timer_en = 0
    // => Expected: Counter remains constant (disabled)
    counter_control_timer_div_check(1, 12'h00, 1, 1, 32'h00000000);

    // Test Case C:
    // TCR = 32'h0000_0001: div_val = 4'b0000, div_en = 0, timer_en = 1
    // => Expected: Counter should increment normally (system clock mode)
    counter_control_timer_div_check(1, 12'h00, 1, 1, 32'h00000001);

    // Test Case D:
    // TCR = 32'h0000_0102: div_val = 4'b0001, div_en = 1, timer_en = 0
    // => Expected: Counter remains constant (timer disabled)
    counter_control_timer_div_check(1, 12'h00, 1, 1, 32'h00000102);

    //---------------------------------------------------------------------------
---------------------------------
    //                                      ADDITIONAL PASS CASES
```

```
    //------------------------------------------------------------------------
--------------------------------
    $display("\n[ADDITIONAL PASS CASES]");

    // Additional Pass Case 1:
    // TCR = 32'h0000_0001: div_val = 4'b0000, div_en = 0, timer_en = 1 (re-
check system clock counting)
    counter_control_timer_div_check(1, 12'h00, 1, 1, 32'h00000001);

    // Additional Pass Case 2:
    // TCR = 32'h0000_0103: div_val = 4'b0001, div_en = 1, timer_en = 1
    // => Expected: Counter should increment (counting speed divided by 2)
    counter_control_timer_div_check(1, 12'h00, 1, 1, 32'h00000103);

    // Additional Pass Case 3:
    // TCR = 32'h0000_0303: div_val = 4'b0011, div_en = 1, timer_en = 1
    // => Expected: Counter should increment (counting speed divided by 8)
    counter_control_timer_div_check(1, 12'h00, 1, 1, 32'h00000303);

    // Additional Pass Case 4:
    // TCR = 32'h0000_0403: div_val = 4'b0100, div_en = 1, timer_en = 1
    // => Expected: Counter should increment (counting speed divided by 16)
    counter_control_timer_div_check(1, 12'h00, 1, 1, 32'h00000403);

    // Additional Pass Case 5:
    // TCR = 32'h0000_0803: div_val = 4'b1000, div_en = 1, timer_en = 1
    // => Expected: Counter should increment (counting speed divided by 256)
    counter_control_timer_div_check(1, 12'h00, 1, 1, 32'h00000803);

    $display("------------------------------------------------------------------
----------");
    $display("[COUNTER CONTROL TIMER_EN && DIV_EN CHECK] All test cases
executed.");
  end
endtask
```

counter_check.v

```
task run_test;
  begin
    #100;
    $display("==========================================================
==================================================");
    $display("|                                                CASE 6: COUNTER
CHECK                                          |");
    $display("==========================================================
==================================================");
```

```
    $display("\n=============================================================
================================================");
    $display("|                                                          SUCCESS
CASE                                          |");
    $display("=============================================================
================================================");

    $display("\n/                                      DETAILED TEST CASES");
    // Em sẽ cho Test Case SUCCESS DETAILED 1: Ghi TCR = 32'h00000003 tại địa
chỉ 12'h00 với các trường phụ 4'b0001, 4'b0010, 4'b0000, 4'b0100
    counter_check(1, 12'h00, 1, 1, 32'h00000003, 4'b0001, 4'b0010, 4'b0000,
4'b0100);
    // Em sẽ cho Test Case SUCCESS DETAILED 2: Ghi TCR = 32'h00000103 tại địa
chỉ 12'h00 với các trường phụ 4'b0001, 4'b0010, 4'b0000, 4'b0100
    counter_check(1, 12'h00, 1, 1, 32'h00000103, 4'b0001, 4'b0010, 4'b0000,
4'b0100);
    // Em sẽ cho Test Case SUCCESS DETAILED 3: Ghi TCR = 32'h00000203 tại địa
chỉ 12'h00 với các trường phụ 4'b0001, 4'b0010, 4'b0000, 4'b0100
    counter_check(1, 12'h00, 1, 1, 32'h00000203, 4'b0001, 4'b0010, 4'b0000,
4'b0100);
    // Em sẽ cho Test Case SUCCESS DETAILED 4: Ghi TCR = 32'h00000303 tại địa
chỉ 12'h00 với các trường phụ 4'b0001, 4'b0010, 4'b0000, 4'b0100
    counter_check(1, 12'h00, 1, 1, 32'h00000303, 4'b0001, 4'b0010, 4'b0000,
4'b0100);
    // Em sẽ cho Test Case SUCCESS DETAILED 5: Ghi TCR = 32'h00000403 tại địa
chỉ 12'h00 với các trường phụ 4'b0001, 4'b0010, 4'b0000, 4'b0100
    counter_check(1, 12'h00, 1, 1, 32'h00000403, 4'b0001, 4'b0010, 4'b0000,
4'b0100);

    // Em sẽ cho in tiêu đề cho phần ADDITIONAL PASS CASES
    $display("\n/                                      ADDITIONAL PASS CASES");
    // Em sẽ cho Test Case SUCCESS ADDITIONAL 1: Ghi TCR = 32'h00000503 tại
địa chỉ 12'h00 với các trường phụ 4'b0001, 4'b0010, 4'b0000, 4'b0100
    counter_check(1, 12'h00, 1, 1, 32'h00000503, 4'b0001, 4'b0010, 4'b0000,
4'b0100);
    // Em sẽ cho Test Case SUCCESS ADDITIONAL 2: Ghi TCR = 32'h00000603 tại
địa chỉ 12'h00 với các trường phụ 4'b0001, 4'b0010, 4'b0000, 4'b0100
    counter_check(1, 12'h00, 1, 1, 32'h00000603, 4'b0001, 4'b0010, 4'b0000,
4'b0100);
    // Em sẽ cho Test Case SUCCESS ADDITIONAL 3: Ghi TCR = 32'h00000703 tại
địa chỉ 12'h00 với các trường phụ 4'b0001, 4'b0010, 4'b0000, 4'b0100
    counter_check(1, 12'h00, 1, 1, 32'h00000703, 4'b0001, 4'b0010, 4'b0000,
4'b0100);
    // Em sẽ cho Test Case SUCCESS ADDITIONAL 4: Ghi TCR = 32'h00000803 tại
địa chỉ 12'h00 với các trường phụ 4'b0001, 4'b0010, 4'b0000, 4'b0100
    counter_check(1, 12'h00, 1, 1, 32'h00000803, 4'b0001, 4'b0010, 4'b0000,
4'b0100);
```

```
    $display("\n================================================================
================================================");
    $display("[COUNTER CHECK] All test cases executed.");
  end
endtask
```

div_counter_ctrl.v

```
task run_test;
  begin
    #100;
    $display("================================================================
============================");
    $display("|                                  COUNTER CONTROL DIV VALUE
CHECK                          |");
    $display("================================================================
============================");
  end
$display("================================================================
========================");
$display("|                                      SUCCESS
CASE                                  |");
$display("================================================================
========================");
$display("\n/                                      DETAILED TEST CASES");
// Em sẽ cho Test Case SUCCESS DETAILED 1 với TCR = 32'h0000_0003 tại địa chỉ
12'h00 (dữ liệu hợp lệ)
counter_control_div_val_check(1, 12'h00, 1, 1, 32'h00000003);
// Em sẽ cho Test Case SUCCESS DETAILED 2 với TCR = 32'h0000_0103 tại địa chỉ
12'h00 (dữ liệu hợp lệ)
counter_control_div_val_check(1, 12'h00, 1, 1, 32'h00000103);
// Em sẽ cho Test Case SUCCESS DETAILED 3 với TCR = 32'h0000_0203 tại địa chỉ
12'h00 (dữ liệu hợp lệ)
counter_control_div_val_check(1, 12'h00, 1, 1, 32'h00000203);
// Em sẽ cho Test Case SUCCESS DETAILED 4 với TCR = 32'h0000_0303 tại địa chỉ
12'h00 (dữ liệu hợp lệ)
counter_control_div_val_check(1, 12'h00, 1, 1, 32'h00000303);
// Em sẽ cho Test Case SUCCESS DETAILED 5 với TCR = 32'h0000_0403 tại địa chỉ
12'h00 (dữ liệu hợp lệ)
counter_control_div_val_check(1, 12'h00, 1, 1, 32'h00000403);

$display("\n/                                      ADDITIONAL PASS CASES");
// Em sẽ cho Test Case SUCCESS ADDITIONAL 1 với TCR = 32'h0000_0503 tại địa
chỉ 12'h00 (dữ liệu hợp lệ)
counter_control_div_val_check(1, 12'h00, 1, 1, 32'h00000503);
```

```
// Em sẽ cho Test Case SUCCESS ADDITIONAL 2 với TCR = 32'h0000_0603 tại địa
chỉ 12'h00 (dữ liệu hợp lệ)
counter_control_div_val_check(1, 12'h00, 1, 1, 32'h00000603);
// Em sẽ cho Test Case SUCCESS ADDITIONAL 3 với TCR = 32'h0000_0703 tại địa
chỉ 12'h00 (dữ liệu hợp lệ)
counter_control_div_val_check(1, 12'h00, 1, 1, 32'h00000703);
// Em sẽ cho Test Case SUCCESS ADDITIONAL 4 với TCR = 32'h0000_0803 tại địa
chỉ 12'h00 (dữ liệu hợp lệ)
counter_control_div_val_check(1, 12'h00, 1, 1, 32'h00000803);
// Em sẽ cho Test Case SUCCESS ADDITIONAL 5 với TCR = 32'h0000_0903 tại địa
chỉ 12'h00 (dữ liệu hợp lệ)
counter_control_div_val_check(1, 12'h00, 1, 1, 32'h00000903);
// Em sẽ cho Test Case SUCCESS ADDITIONAL 6 với TCR = 32'h0000_0x03 tại địa
chỉ 12'h00 (dữ liệu hợp lệ)
// Lưu ý: Giá trị 32'h0000_0x03 không hợp lệ trong Verilog, em hãy chỉnh sửa
thành giá trị hợp lệ
counter_control_div_val_check(1, 12'h00, 1, 1, 32'h00000x03);

$display("\n=================================================================
=========================");
$display("[COUNTER CONTROL DIV VALUE CHECK] All test cases executed.");

endtask
// Em sẽ cho bổ sung thêm các trường hợp PASS bổ sung nhằm mở rộng phạm vi
kiểm tra cấu hình hợp lệ của thanh ghi TCR,
// đảm bảo rằng các giá trị TCR khác nhau đều kích hoạt chế độ đếm đúng theo
spec.
```

Interrupt_check.v

```
task run_test;
  begin
    #100;
    $display("=================================================================
================================================");
    $display("|                                        INTERRUPT INT_EN [TIER
(0x14)] CHECK                                |");
    $display("=================================================================
================================================");

    $display("\n---- SUCCESS CASES ----");

    // Em se cho Test Case SUCCESS 1 voi cac gia tri nhu sau
    // pwrite = 1, paddr_TIER = 12'h14, paddr_TCMP0 = 12'h0C, paddr_TCMP1 =
12'h10,
    // paddr_TCR = 12'h00, paddr_TISR = 12'h18, psel = 1, penable = 1,
    // int_en_register = 32'h00000001 (bat ngat), TCMP0_value = 32'h00000005,
TCMP1_value = 32'h00000000,
```

```
    // Counter_Control_Mode = 32'h00000103 (timer_en=1, div_en=1,
count_clr=0), RW1C_en = 1, RW1C_dis = 0.
    force uut.i_inst.match = 1; // Em se cho bat buoc signal match len 1
    #50;
    interrupt_check(
      1,          // pwrite
      12'h14,     // paddr_TIER
      12'h0C,     // paddr_TCMP0
      12'h10,     // paddr_TCMP1
      12'h00,     // paddr_TCR
      12'h18,     // paddr_TISR
      1,          // psel
      1,          // penable
      32'h00000001, // int_en_register (bat ngat)
      32'h00000005, // TCMP0_value (so sanh = 5)
      32'h00000000, // TCMP1_value (0)
      32'h00000103, // Counter_Control_Mode (timer_en=1, div_en=1,
count_clr=0)
      1,          // RW1C_en = 1 (xoa ngat)
      0           // RW1C_dis = 0
    );
    release uut.i_inst.match; // Em se cho giai phong signal match sau khi
test

    // Em se cho Test Case SUCCESS 2 voi cac gia tri tuong tu nhung
TCMP0_value thay doi de kiem tra ngat khac
    force uut.i_inst.match = 1;
    #50;
    interrupt_check(
      1,
      12'h14,
      12'h0C,
      12'h10,
      12'h00,
      12'h18,
      1,
      1,
      32'h00000001,
      32'h0000000A, // TCMP0_value (so sanh = 10)
      32'h00000000,
      32'h00000103,
      1,
      0
    );
    release uut.i_inst.match;

    $display("\n---- ADDITIONAL PASS CASES ----");
```

```
    // Em se cho Test Case ADDITIONAL 1 voi int_en_register tat de kiem tra
khong phat ngat
    force uut.i_inst.match = 1;
    #50;
    interrupt_check(
      1,
      12'h14,
      12'h0C,
      12'h10,
      12'h00,
      12'h18,
      1,
      1,
      32'h00000000, // int_en_register = 0 (tat ngat)
      32'h00000005,
      32'h00000000,
      32'h00000103,
      0,
      0
    );
    release uut.i_inst.match;

    // Em se cho Test Case ADDITIONAL 2 voi thay doi TCMP1_value de kiem tra
hoat dong ngat khi gia tri so sanh khac
    force uut.i_inst.match = 1;
    #50;
    interrupt_check(
      1,
      12'h14,
      12'h0C,
      12'h10,
      12'h00,
      12'h18,
      1,
      1,
      32'h00000001,
      32'h00000005,
      32'h00000005, // TCMP1_value thay doi
      32'h00000103,
      1,
      0
    );
    release uut.i_inst.match;

    $display("\n================================================================
================================================");
    $display("[INTERRUPT INT_EN CHECK] All test cases executed.");
  end
```

```
endtask
```

register_mode.v

```
task run_test;
  begin
    #100;
    $display("========================================================================
=====");
    $display("=================[ REGISTER READ/WRITE
CHECK  ]======================");
    $display("========================================================================
=====");

    // Em sẽ cho in phần SUCCESS CASES
    $display("========================================================================
=====");
    $display("                              SUCCESS
CASES                            ");
    $display("========================================================================
=====");

    $display("\n/                                        DETAILED TEST CASES");
    // Em cho Test Case SUCCESS DETAILED 1 với pwrite=1, paddr=12'h00, psel=1,
penable=1, wdata=32'hAAAA_A8AF
    register_rw_check(1, 12'h00, 1, 1, 32'hCCCC_BBBB);
    // Em cho Test Case SUCCESS DETAILED 2 với pwrite=1, paddr=12'h00, psel=1,
penable=1, wdata=32'hAAAA_AAAF
    register_rw_check(1, 12'h00, 1, 1, 32'hBBBB_AAAF);
    // Em cho Test Case SUCCESS DETAILED 3 với pwrite=1, paddr=12'h04, psel=1,
penable=1, wdata=32'hBBBB_BBBB
    register_rw_check(1, 12'h04, 1, 1, 32'hDDDD_EEEE);
    // Em sẽ cho Test Case SUCCESS DETAILED 4 với pwrite=1, paddr=12'h08,
psel=1, penable=1, wdata=32'hCCCC_CCCC
    register_rw_check(1, 12'h08, 1, 1, 32'hCCCC_CCCC);
    // Em cho Test Case SUCCESS DETAILED 5 với pwrite=1, paddr=12'h0C, psel=1,
penable=1, wdata=32'hDDDD_DDDD
    register_rw_check(1, 12'h0C, 1, 1, 32'hDDDD_DDDD);
    // Em scho Test Case SUCCESS DETAILED 6 với pwrite=1, paddr=12'h10,
psel=1, penable=1, wdata=32'hEEEE_EEEE
    register_rw_check(1, 12'h10, 1, 1, 32'hEEEE_EEEE);
    // Em cho Test Case SUCCESS DETAILED 7 với pwrite=1, paddr=12'h14, psel=1,
penable=1, wdata=32'hFFFF_FFFF
    register_rw_check(1, 12'h14, 1, 1, 32'hFFFF_FFFF);
    // Em cho Test Case SUCCESS DETAILED 8 với pwrite=1, paddr=12'h18, psel=1,
penable=1, wdata=32'hAAAA_BBBB
    register_rw_check(1, 12'h18, 1, 1, 32'hAAAA_BBBB);

    $display("\n/                                        ADDITIONAL PASS CASES");
```

```
    // Em cho Test Case SUCCESS ADDITIONAL 1 với pwrite=1, paddr=12'h00,
psel=1, penable=1, wdata=32'h12345678
    register_rw_check(1, 12'h00, 1, 1, 32'h12345678);
    // Em cho Test Case SUCCESS ADDITIONAL 2 với pwrite=1, paddr=12'h04,
psel=1, penable=1, wdata=32'h87654321
    register_rw_check(1, 12'h04, 1, 1, 32'h87654321);

    $display("\n================================================================
=======");
    $display("                                FAIL
CASES                              ");
    $display("================================================================
======");
    // Em cho Test Case FAIL 1 với pwrite=0, paddr=12'h00, psel=1, penable=0,
wdata=32'hAAAA_A8AF
    register_rw_check(0, 12'h00, 1, 0, 32'hAAAA_A8AF);
    // Em cho Test Case FAIL 2 với pwrite=0, paddr=12'h04, psel=1, penable=0,
wdata=32'hBBBB_BBBB
    register_rw_check(0, 12'h04, 1, 0, 32'hBBBB_BBBB);
    // Em cho Test Case FAIL 3 với pwrite=0, paddr=12'h08, psel=1, penable=0,
wdata=32'hCCCC_CCCC
    register_rw_check(0, 12'h08, 1, 0, 32'hCCCC_CCCC);
    // Em cho Test Case FAIL 4 với pwrite=0, paddr=12'h0C, psel=1, penable=0,
wdata=32'hDDDD_DDDD
    register_rw_check(0, 12'h0C, 1, 0, 32'hDDDD_DDDD);
    // Em cho Test Case FAIL 5 với pwrite=0, paddr=12'h10, psel=1, penable=0,
wdata=32'hEEEE_EEEE
    register_rw_check(0, 12'h10, 1, 0, 32'hEEEE_EEEE);
    // Em cho Test Case FAIL 6 với pwrite=0, paddr=12'h14, psel=1, penable=0,
wdata=32'hFFFF_FFFF
    register_rw_check(0, 12'h14, 1, 0, 32'hFFFF_FFFF);
    // Em cho Test Case FAIL 7 với pwrite=1, paddr=12'h18, psel=0, penable=0,
wdata=32'hAAAA_BBBB
    register_rw_check(1, 12'h18, 0, 0, 32'hAAAA_BBBB);
    // Em cho Test Case FAIL 8 với pwrite=1, paddr=12'h1C, psel=0, penable=0,
wdata=32'hAAAA_BBBB
    register_rw_check(1, 12'h1C, 0, 0, 32'hAAAA_BBBB);

    $display("\n================================================================
=======");
    $display("[REGISTER READ/WRITE CHECK] All test cases executed.");
  end
endtask
```

register_val_check.v

```
task run_test;
  begin
    #100;
```

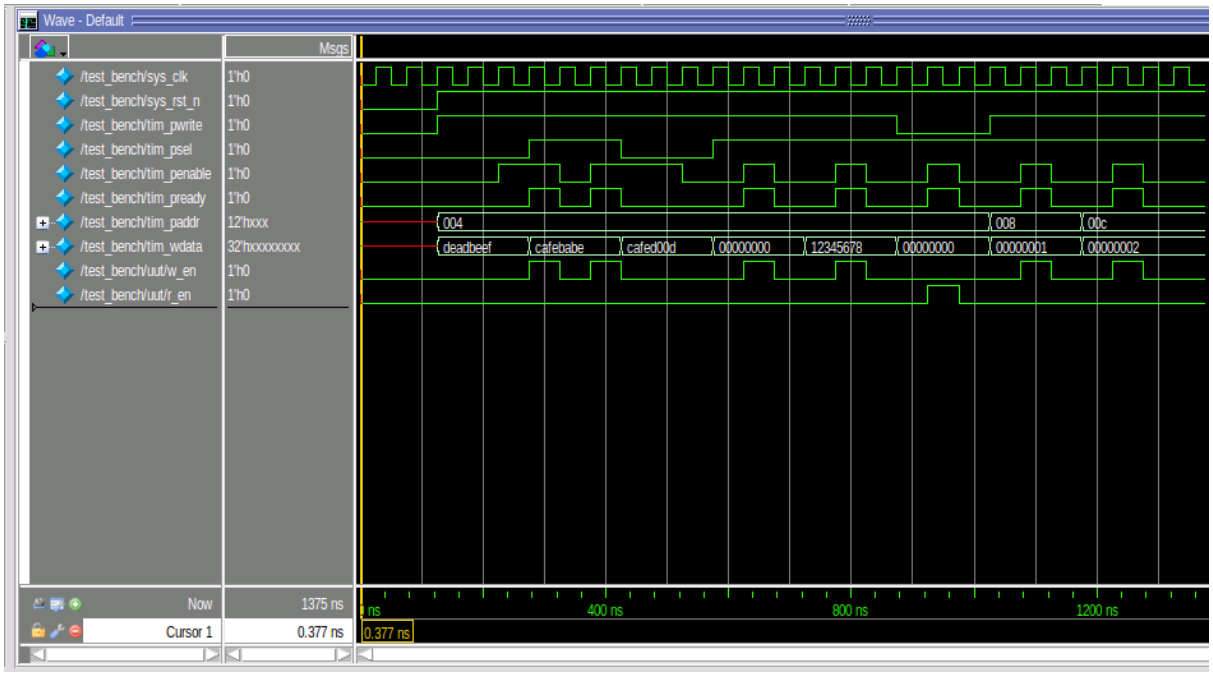
```
    $display("================================================================
=====");
    $display("================[ REGISTER INITIAL VALUE
CHECK  ]==================");
    $display("================================================================
=====");

    $display("================================================================
=====");
    $display("                            PASSS  CASES
    ");
    $display("================================================================
=====");

    $display("\n/                                    DETAILED PASS TEST
CASES");
    // Em se cho Test Case SUCCESS DETAILED 1 voi pwrite = 1, paddr = 12'h00,
psel = 1, penable = 1, wdata = 32'h0
    test_bench.register_initial_check(1, 12'h00, 1, 1, 32'h0);
    // Em se cho Test Case SUCCESS DETAILED 2 voi pwrite = 1, paddr = 12'h04,
psel = 1, penable = 1, wdata = 32'h0
    test_bench.register_initial_check(1, 12'h04, 1, 1, 32'h0);
    // Em se cho Test Case SUCCESS DETAILED 3 voi pwrite = 1, paddr = 12'h08,
psel = 1, penable = 1, wdata = 32'h0
    test_bench.register_initial_check(1, 12'h08, 1, 1, 32'h0);
    // Em se cho Test Case SUCCESS DETAILED 4 voi pwrite = 1, paddr = 12'h0C,
psel = 1, penable = 1, wdata = 32'h0
    test_bench.register_initial_check(1, 12'h0C, 1, 1, 32'h0);
    // Em se cho Test Case SUCCESS DETAILED 5 voi pwrite = 1, paddr = 12'h10,
psel = 1, penable = 1, wdata = 32'h0
    test_bench.register_initial_check(1, 12'h10, 1, 1, 32'h0);
    // Em se cho Test Case SUCCESS DETAILED 6 voi pwrite = 1, paddr = 12'h14,
psel = 1, penable = 1, wdata = 32'h0
    test_bench.register_initial_check(1, 12'h14, 1, 1, 32'h0);
    // Em se cho Test Case SUCCESS DETAILED 7 voi pwrite = 1, paddr = 12'h18,
psel = 1, penable = 1, wdata = 32'h0
    test_bench.register_initial_check(1, 12'h18, 1, 1, 32'h0);
    // Em se cho Test Case SUCCESS DETAILED 8 voi pwrite = 1, paddr = 12'h1C,
psel = 1, penable = 1, wdata = 32'h0
    test_bench.register_initial_check(1, 12'h1C, 1, 1, 32'h0);

    $display("\n/                                    ADDITIONAL PASS CASES");
    // Em se cho Test Case SUCCESS ADDITIONAL 1 voi pwrite = 1, paddr =
12'h02, psel = 1, penable = 1, wdata = 32'h0
    test_bench.register_initial_check(1, 12'h02, 1, 1, 32'h0);
    // Em se cho Test Case SUCCESS ADDITIONAL 2 voi pwrite = 1, paddr =
12'h06, psel = 1, penable = 1, wdata = 32'h0
    test_bench.register_initial_check(1, 12'h06, 1, 1, 32'h0);
```

```
    $display("\n================================================================
=======");
    $display("                                FAIL
CASES                              ");
    $display("================================================================
=====");

    // Em se cho Test Case FAIL 1 voi pwrite = 0, paddr = 12'h00, psel = 1,
penable = 0, wdata = 32'h0
    test_bench.register_initial_check(0, 12'h00, 1, 0, 32'h0);
    // Em se cho Test Case FAIL 2 voi pwrite = 0, paddr = 12'h04, psel = 1,
penable = 0, wdata = 32'h0
    test_bench.register_initial_check(0, 12'h04, 1, 0, 32'h0);
    // Em se cho Test Case FAIL 3 voi pwrite = 0, paddr = 12'h08, psel = 1,
penable = 0, wdata = 32'h0
    test_bench.register_initial_check(0, 12'h08, 1, 0, 32'h0);
    // Em se cho Test Case FAIL 4 voi pwrite = 0, paddr = 12'h0C, psel = 1,
penable = 0, wdata = 32'h0
    test_bench.register_initial_check(0, 12'h0C, 1, 0, 32'h0);
    // Em se cho Test Case FAIL 5 voi pwrite = 0, paddr = 12'h10, psel = 1,
penable = 0, wdata = 32'h0
    test_bench.register_initial_check(0, 12'h10, 1, 0, 32'h0);
    // Em se cho Test Case FAIL 6 voi pwrite = 0, paddr = 12'h14, psel = 1,
penable = 0, wdata = 32'h0
    test_bench.register_initial_check(0, 12'h14, 1, 0, 32'h0);
    // Em se cho Test Case FAIL 7 voi pwrite = 0, paddr = 12'h18, psel = 1,
penable = 0, wdata = 32'h0
    test_bench.register_initial_check(0, 12'h18, 1, 0, 32'h0);
    // Em se cho Test Case FAIL 8 voi pwrite = 0, paddr = 12'h1C, psel = 1,
penable = 0, wdata = 32'h0
    test_bench.register_initial_check(0, 12'h1C, 1, 0, 32'h0);

    $display("\n================================================================
=======");
    $display("[REGISTER INITIAL VALUE CHECK] All test cases executed.");
  end
endtask

// Em se cho test case SUCCESS va FAIL, em lam cho cac test case duoc thuc
hien de kiem tra chuc nang doc/ghi register, em tao ra test case PASS
DETAILED, ADDITIONAL PASS, va FAIL.
```

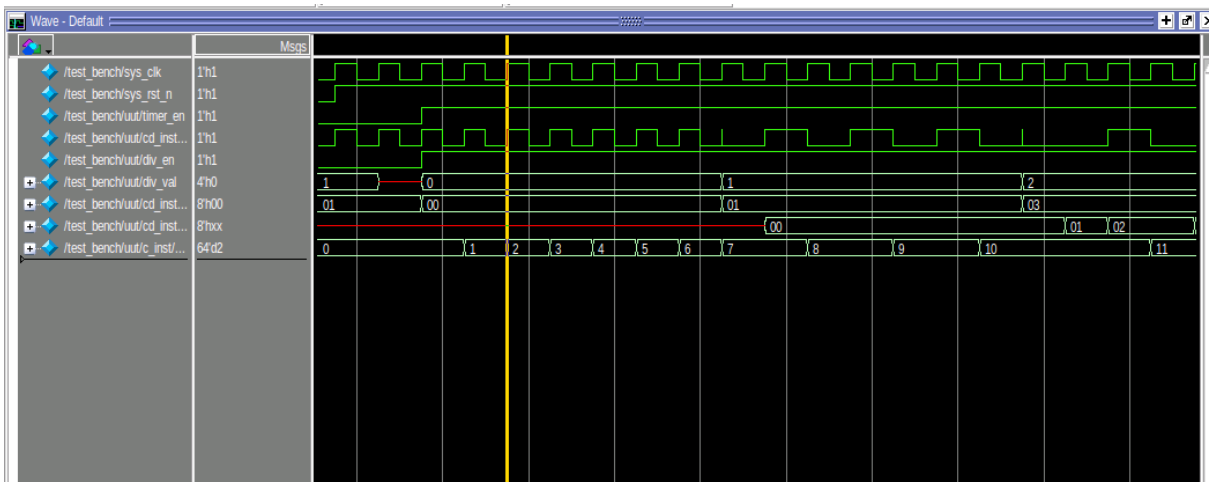
CHAPTER 5: SIMULATION RESULT

5.1 Check Wave

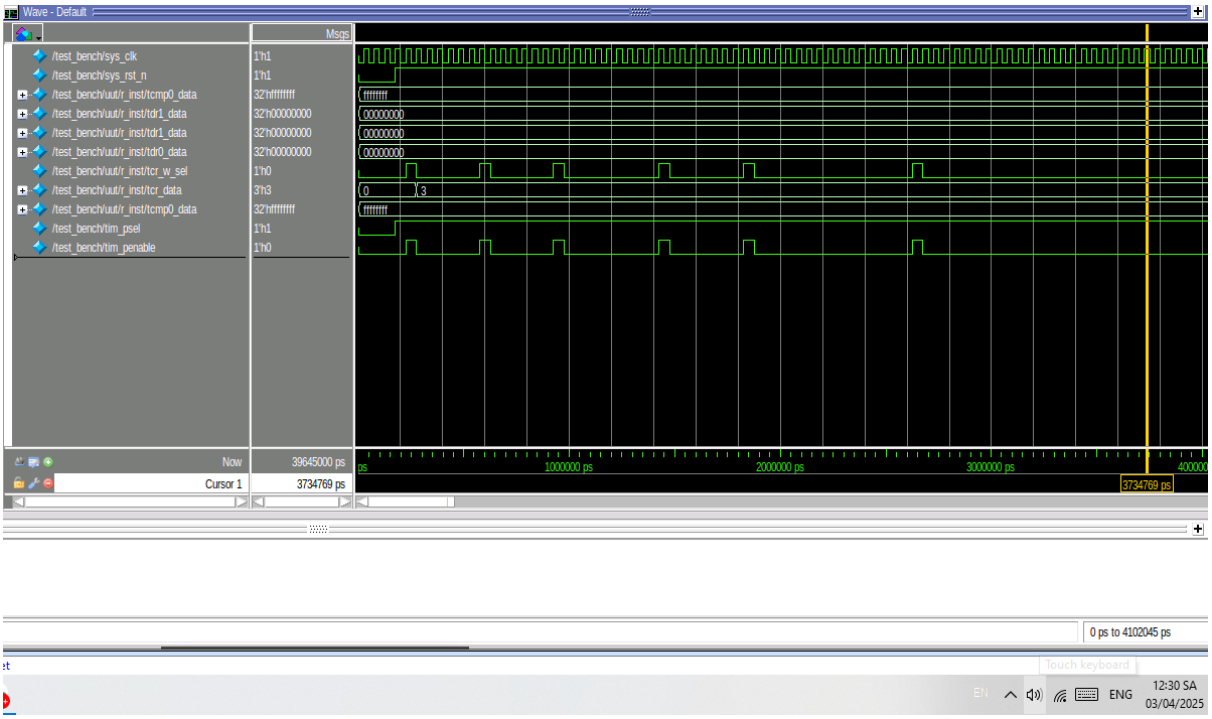
APB Register Read/Write Transaction Waveform



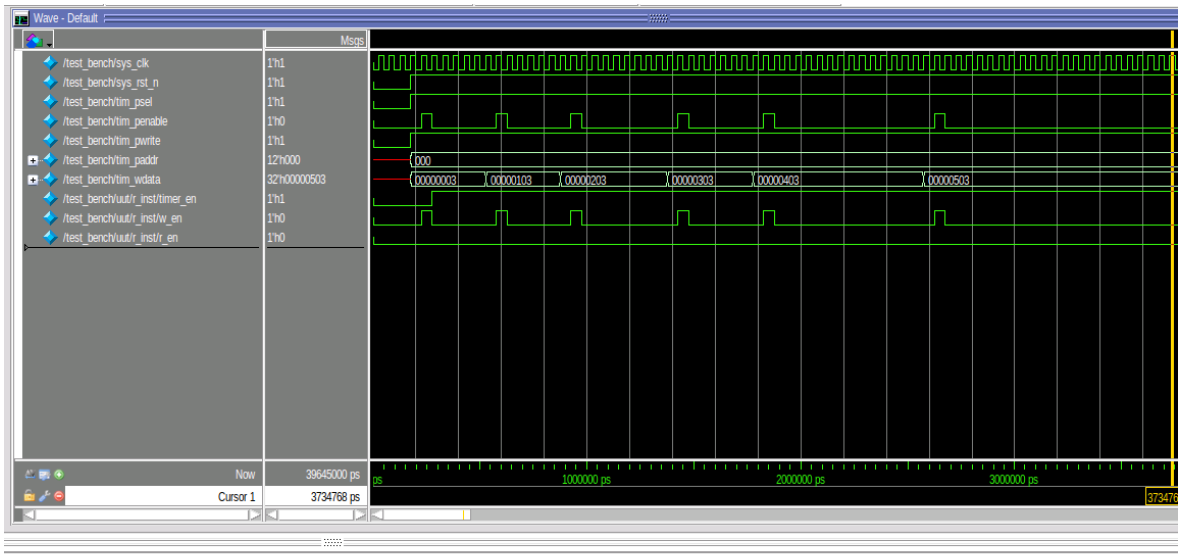
Counter Control Mode

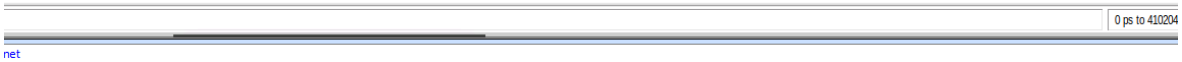


Interrupt Verification



Register Read/Write





REFERENCES

[1] ICTC, *Basic IC Design (RTL + DV) – Lecture Slides*, Institute of Chip Technology & Communication (ICTC), Vietnam, 2023. *(Unpublished course materials provided by ICTC for the final project.)*

[2] ICTC, "ICTC Official Website." [Online]. Available: https://ictc.edu.vn. [Accessed: 03, 2025]. *(Reference for additional course information and resources.)*

APPENDIX 1

```
/ictc/student_data/minhnhut/final_project/TIMER_IP
```

```
Coverage Report Summary Data by instance
================================================================================
===
=== Instance: /test_bench/uut/apb_inst
=== Design Unit: work.APB_slave
================================================================================
===
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         4         4         0   100.00%
    Expressions                     12        11         1    91.66%
    Statements                      11        11         0   100.00%
    Toggles                         10        10         0   100.00%

================================================================================
===
=== Instance: /test_bench/uut/r_inst
=== Design Unit: work.Register
================================================================================
===
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        40        40         0   100.00%
    Conditions                       2         1         1    50.00%
    Expressions                     21        21         0   100.00%
    Statements                      49        49         0   100.00%
    Toggles                        660       399       261    60.45%

================================================================================
===
=== Instance: /test_bench/uut/cd_inst
=== Design Unit: work.Clock_Divider
================================================================================
===
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        16        16         0   100.00%
    Expressions                      4         4         0   100.00%
    Statements                      19        19         0   100.00%
    Toggles                         78        78         0   100.00%
================================================================================
===
```

```
=== Instance: /test_bench/uut/c_inst
=== Design Unit: work.Counter
================================================================================
===
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         6         6         0   100.00%
    Statements                       6         6         0   100.00%
    Toggles                        256        36       220    14.06%

================================================================================
===
=== Instance: /test_bench/uut/i_inst
=== Design Unit: work.Interrupt
================================================================================
===
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         5         5         0   100.00%
    Conditions                       2         1         1    50.00%
    Statements                       6         6         0   100.00%
    Toggles                          4         4         0   100.00%

================================================================================
===
=== Instance: /test_bench/uut
=== Design Unit: work.Timer_top
================================================================================
===
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                        224       114       110    50.89%

================================================================================
===
=== Instance: /test_bench
=== Design Unit: work.test_bench
================================================================================
===
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                       117        75        42    64.10%
    Conditions                      63        11        52    17.46%
    Statements                     460       399        61    86.73%
    Toggles                        422       181       241    42.89%


Total Coverage By Instance (filtered view): 67.08%
```