# **SPECIFICATION – TIMER IP**

## a. APB Protocol

* Timer is APB Slave
* Support 32-bit read/write (word access)
* Not support byte access
* Not wait state and error response
* Address registers: 0x00 to 0x1C, 8 main registers in total.
* Base address: 0x4000\_1000 ⇒ Real address 0x4000\_1000 to 0x4000\_101C

## b. Counter

* Type: 64-bit count-up
* Default mode: counting with system clock 200 MHz
* Count control with timer\_en = 1, div\_en = 1 and div\_val = 0 to 8:

|  |  |  |  |
| --- | --- | --- | --- |
| **Timer\_en** | **div\_en** | **div\_val** | **Ý nghĩa tốc độ đếm** |
| 1 | 0 | 0 | Counting with system clock 200 MHz |
| 1 | 1 | 0000 | 4’b0000: Counting speed is not divided |
| 1 | 1 | 0001 | 4’b0001: Counting speed is divided by 2 (default) |
| 1 | 1 | 0010 | 4’b0010: Counting speed is divided by 4 |
| 1 | 1 | ... | 4’b1000: Counting speed is divided by 256 |

Don’t change div\_en or div\_val when timer\_en = 1

## c. Timer Interrupt

* Compare values counter with TCMP0 and TCMP1 (64-bit compare value)
* If TCMP0 and TCMP1 == TDR0 and TDR1 and int\_en = 1, tim\_int signal is enabled.
* TISR.int\_st status flag is set = 1
* Clear by writing 1 to that bit

## d. Control and function registers

|  |  |  |
| --- | --- | --- |
| **Address** | **Register** | **Funtion** |
| 0x00 | TCR – Timer Control | On/off timer, prescale counter |
| 0x04 | TDR0 – Data Register 0 | Lower 32-bits of 64-bits counter |
| 0x08 | TDR1 – Data Register 1 | Upper 32-bits of 64-bits counter |
| 0x0C | TCMP0 – Compare Register 0 | Lower 32-bits of 64-bits compare values |
| 0x10 | TCMP1 – Compare Register 1 | Upper 32-bits of 64-bits compare values |
| 0x14 | TIER – Interrupt Enable | On /off interrupt |
| 0x18 | TISR – Interrupt Status | Status Timer interrupt |
| 0x1C | THCSR – Halt Control | Reserve |

## REGISTER SPECIFICATION

###### Timer Control Register -TCR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bit** | **Name** | **Type** | **Default value** | **Description** |
| 31:12 | Reserve | - | 20’h0 | Reserve |
| 11:8 | Div\_val | RW | 4’b0001 | Counter control mode setting:  • 4’b0000: Counting speed is not divided  • 4’b0001: Counting speed is divided by 2 (default)  • 4’b0010: Counting speed is divided by 4  • 4’b0011: Counting speed is divided by 8  • 4’b0100: Counting speed is divided by 16  • 4’b0101: Counting speed is divided by 32  • 4’b0110: Counting speed is divided by 64  • 4’b0111: Counting speed is divided by 128  • 4’b1000: Counting speed is divided by 256  • Others: reserved, (\*)prohibit settings.  When setting the prohibit value, div\_val is not changed.  Note: user must not change div\_en while timer\_en is High |
| 7:2 | Reserve | RO | 6’b0 | Reserve |
| 1 | Div\_en | RW | 1’b0 | Counter control mode enable.  • 0: Disabled. Counter counts with normal speed based on system clock  • 1: Enabled. The couting speed of counter is controlled based on div\_val |
| 0 | Timer\_en | RW | 1’b0 | Timer enable  • 0: Disabled. Counter does not count.  • 1: Enabled. Counter starts counting. |

###### Timer Data Register 0/1 –TDR0/1 and Timer Compare Register 0/1 –TCMP0/1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bit** | **Name** | **Type** | **Default value** | **Description** |
| 31:0 | TDR0 | RW | 32’h0 | Lower 32-bit of 64-bit counter |
| 31:0 | TDR1 | RW | 32’h0 | Upper 32-bit of 64-bit counter |
| 31:0 | TCMP0 | RW | 32’hffff\_ffff | Lower 32-bit of 64-bit compare value  Interrupt is asserted when counter value is qual to compare value. |
| 31:0 | TCMP1 | RW | 32’hffff\_ffff | Upper 32-bit of 64-bit compare value  Interrupt is asserted when counter value is qual to compare value. |

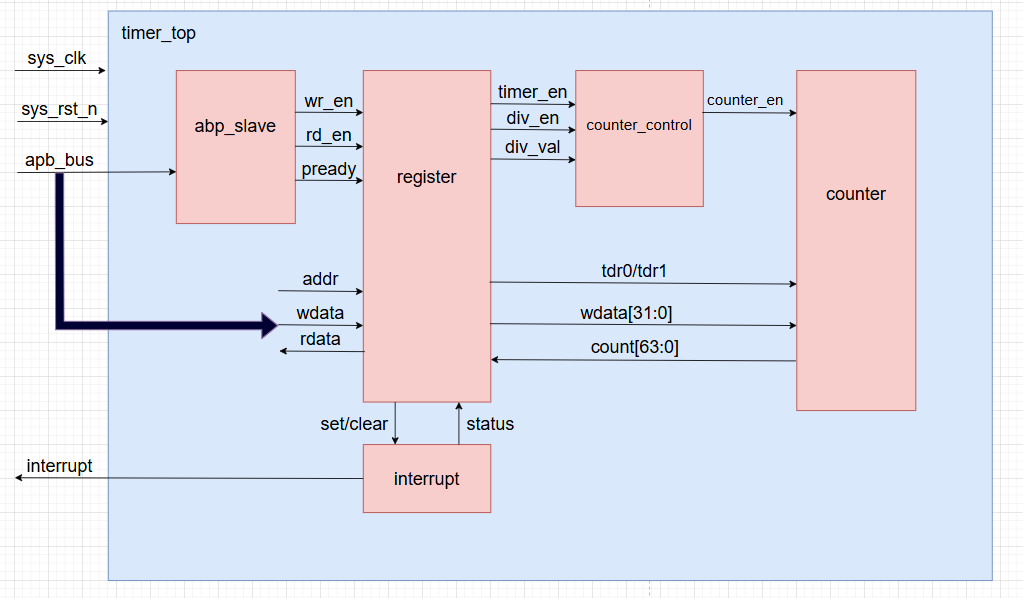
###### Timer Interrupt Enable Register–TIER

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bit** | **Name** | **Type** | **Default value** | **Description** |
| 31:1 | Reserve | RO | 31’h0 | Reserve |
| 0 | Int\_en | R/W | 1’b0 | Timer interrupt enable  0: Timer interrupt is disabled.  1: Timer interrupt is enabled.  When this bit is 0, no timer interrupt is output. When this bit is 1, timer interrupt can be output when reaching trigger condition.  Clearing this bit to 0 while interrupt is asserting will mask the interrupt to 0 but does not affect the interrupt pending bit TISR.int\_st bit |

###### Timer Interrupt Status Register–TISR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bit** | **Name** | **Type** | **Default value** | **Description** |
| 31:1 | Reserve | RO | 31’h0 | Reserve |
| 0 | Int\_st | RW1C | 1’b0 | Timer interrupt trigger condition status bit (interrupt pending bit)  0: the interrupt trigger condition does not  occur.  1: the interrupt trigger condition occurred. Write 1 when this bit is 1 to clear it  Write 0 when this bit is 1 has no effect  Write to this bit when it is 0 has no effect.  Note: When interrupt trigger condition occurred (counter reached compare value), counter continues to count normally. |

# Block diagram



# IO Port list:

## Timer\_top.v

|  |  |  |
| --- | --- | --- |
| Signal name | Width | I/O |
| Sys\_clk |  | Input |
| Sys\_rst\_n |  | Input |
| Tim\_psel |  | Input |
| Tim\_pwrite |  | Input |
| Tim\_penable |  | Input |
| Tim\_paddr | 12 | Input |
| Tim\_pwdata | 32 | Input |
| Tim\_prdata | 32 | Output |
| Tim\_pstrb | 4 | Input |
| Tim\_pready |  | Input |
| Tim\_pslverr |  | Output |
| Tim\_int |  | Output |
| Dbg\_mode |  | Input |

## Abp\_interface.v

|  |  |  |
| --- | --- | --- |
| Signal name | Width | I/O |
| Sys\_clk |  | Input |
| Sys\_rst\_n |  | Input |
| psel |  | Input |
| pwrite |  | Input |
| penable |  | Input |
| pstrb | 4 | Input |
| pready |  | Output |
| Wr\_en |  | Output |
| Rd\_en |  | Output |

## Register.v

|  |  |  |
| --- | --- | --- |
| Signal name | Width | I/O |
| Sys\_clk |  | Input |
| Sys\_rst\_n |  | Input |
| Addr |  | Input |
| Wdata |  | Input |
| Rdata |  | output |
| Cnt | 64 | Input |
| Wr\_en |  | Input |
| Rd\_en |  | Input |
| Tdr0\_wr\_sel |  | output |
| Tdr1\_wr\_sel |  | output |
| Div\_en |  | output |
| Div\_val | 4 | output |
| Timer\_en |  | output |
| Tim\_int |  | output |

## Counter\_control.v

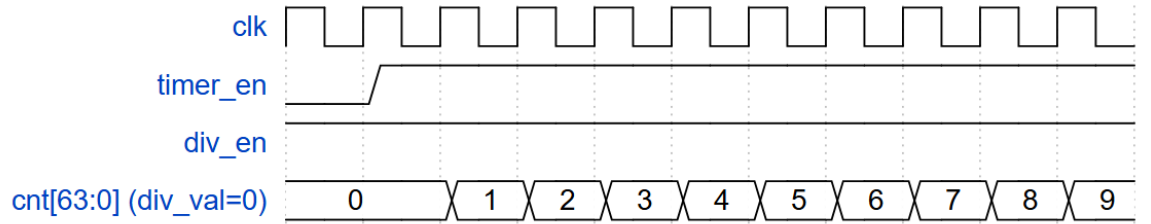
|  |  |  |
| --- | --- | --- |
| Signal name | Width | I/O |
| Sys\_clk |  | Input |
| Sys\_rst\_n |  | Input |
| Wdata | 32 | Input |
| Count\_en |  | output |
| Div\_en |  | input |
| Div\_val | 4 | input |
| Timer\_en |  | input |

## Counter.v

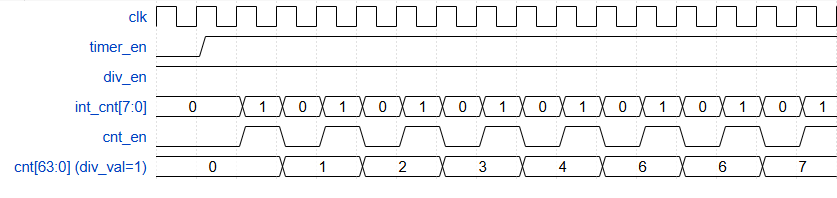
|  |  |  |
| --- | --- | --- |
| Signal name | Width | I/O |
| Sys\_clk |  | Input |
| Sys\_rst\_n |  | Input |
| Wdata | 32 | Input |
| Count\_en |  | input |
| Count | 64 | output |
| Tdr0\_wr\_sel |  | input |
| Tdr1\_wr\_sel |  | input |

# Wave form

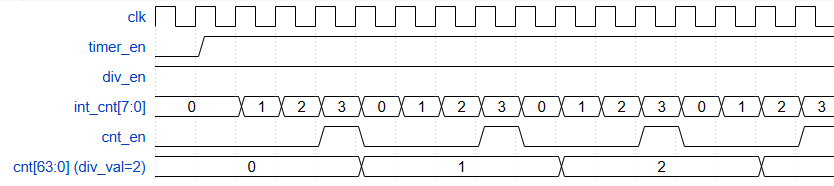
## When timer\_en = 1; div\_en = 0; div\_val = 0; When timer\_en = 1; div\_en = 1; div\_val = 0;



## When timer\_en = 1; div\_en = 1; div\_val = 1;

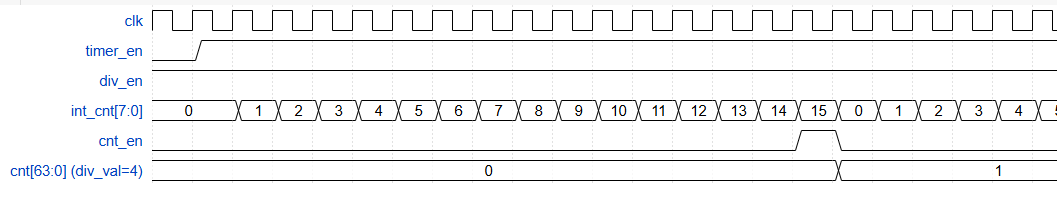


## When timer\_en = 1; div\_en = 1; div\_val = 2;

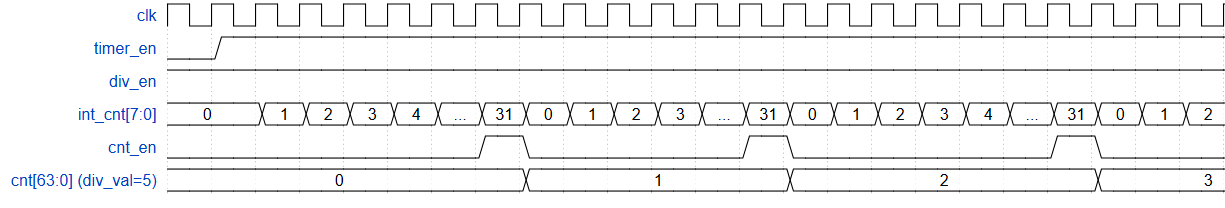


## When timer\_en = 1; div\_en = 1; div\_val = 3;

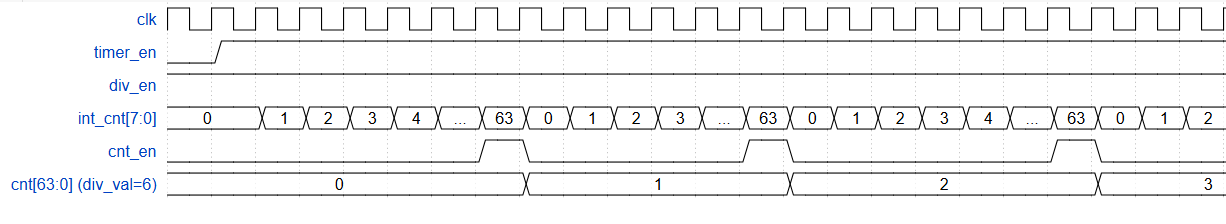
## When timer\_en = 1; div\_en = 1; div\_val = 4;



## When timer\_en = 1; div\_en = 1; div\_val = 5;

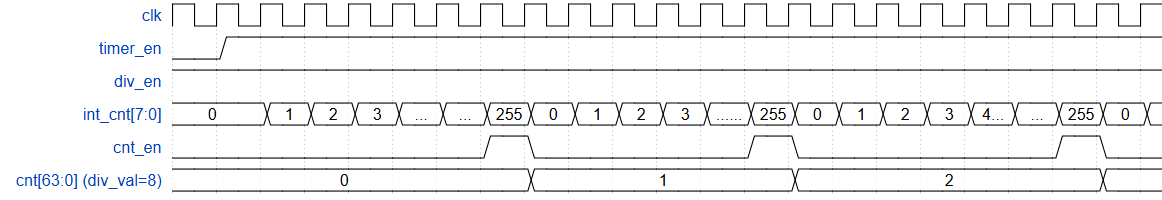


## When timer\_en = 1; div\_en = 1; div\_val = 6;



## When timer\_en = 1; div\_en = 1; div\_val = 7;

## When timer\_en = 1; div\_en = 1; div\_val = 8;



## Write psel, penable, pwrite, paddr, pwdata to Register TCR, TDR0/1, TCMP0/1, TIER and TISR

## Read psel, penable, pwrite, paddr, pwdata to Register TCR, TDR0/1, TCMP0/1, TIER and TISR

## Code wavedrom counter

{ signal: [

{ name: "clk", wave: "p...................................." },

{ name: "timer\_en", wave: "01........................" },

{ name: "div\_en", wave: "1........................." },

{ name: "int\_cnt[7:0]", wave: "=.========================", data: ["0","1","2","3","...","...","255","0","1","2","3","......","255","0","1","2","3","4...","...","255","0","1","2","7","8","4",] },

{ name: "cnt\_en", wave: "0......10....10.....10..1..0..10101" },

{ name: "cnt[63:0] (div\_val=8)", wave: "=.......=.....=......=....=....=....=.=.=.", data: ["0","1","2","3","4","6","6","7","8"] }

## Code wave drom write and read

{ signal: [

{ name: "clk", wave: "p...................................." },

{ name: "timer\_en", wave: "01...........0............" },

{ name: "div\_en", wave: "1........................." },

{ name: "int\_cnt[7:0]", wave: "=.===========xxxxxxxxxxxxx", data: ["0","1","0","1","0","1","0","1","0","1","0","1"] },

{ name: "cnt\_en", wave: "0.10101010101xxxxxxxxxxxxx" },

{ name: "cnt[63:0] (div\_val=0)", wave: "=..=.=.=.=.=.=.=.=.=.=.=.=.=", data: ["0","1","2","3","4","5","5","5","5","5","5","5","5","5"] },

{ name: "tim\_psel", wave: "010..........10.....10.....10" },

{ name: "tim\_penable", wave: "010..........10.....10.....10" },

{ name: "tim\_pwrite", wave: "010..........10.....10.....10" },

{ name: "tim\_paddr", wave: "===..........==.....==.....==...", data: ["0","12'h0","0","12'h4","0","12'h14","0","12'18","0"] },

{ name: "tim\_pwdata", wave: "==.=.=.=.=.=.=......=......=...============", data: ["0","2","1","2","3","4","5","5","1","1","5","5","5","5","1"] },

]}

{ signal: [

{ name: "clk", wave: "p...................................." },

{ name: "timer\_en", wave: "01...........0............" },

{ name: "div\_en", wave: "1........................." },

{ name: "int\_cnt[7:0]", wave: "=.===========xxxxxxxxxxxxx", data: ["0","1","0","1","0","1","0","1","0","1","0","1"] },

{ name: "cnt\_en", wave: "0.10101010101xxxxxxxxxxxxx" },

{ name: "cnt[63:0] (div\_val=0)", wave: "=..=.=.=.=.=.=.=.=.=.=.=.=.=", data: ["0","1","2","3","4","5","5","5","5","5","5","5","5","5"] },

{ name: "tim\_psel", wave: "010..........10.....10.....10" },

{ name: "tim\_penable", wave: "010..........10.....10.....10" },

{ name: "tim\_pwrite", wave: "0............................." },

{ name: "tim\_paddr", wave: "===..........==.....==.....==...", data: ["0","12'h0","0","12'h4","0","12'h14","0","12'18","0"] },

{ name: "tim\_pwdata", wave: "==.=.=.=.=.=.=......=......=...============", data: ["0","2","1","2","3","4","5","5","1","1","5","5","5","5","1"] },

]}

# Logic diagram

# Test Plan for IP Timer

#### **1. Test case: Verify default values after reset**

* **Expectation:** All registers (TCR, TDR0, TDR1, TCMP0, TCMP1, TIER, TISR) return default values when read after reset.
* **Test flow:**
  + Step 1: Set sys\_rst\_n = 0 for 25ns.
  + Step 2: Set sys\_rst\_n = 1 to release reset.
  + Step 3: Perform read transactions for addresses 12'h000 (TCR), 12'h004 (TDR0), 12'h008 (TDR1), 12'h00c (TCMP0), 12'h010 (TCMP1), 12'h014 (TIER), 12'h018 (TISR), and 12'hfff (invalid address).
  + Step 4: Verify tim\_prdata matches expected default values.

#### **2. Test case: Verify counter operation with timer\_en = 0 (disabled)**

* **Expectation:** Counter stops incrementing, TDR0 and TDR1 hold last values, no interrupt triggers.
* **Test flow:**
  + Step 1: Write 32'h0000\_0000 to 12'h000 (TCR) to set timer\_en = 0.
  + Step 2: Write 32'h0000\_0048 to 12'h00c (TCMP0) and 32'h0000\_0000 to 12'h010 (TCMP1).
  + Step 3: Wait 10ns, then read 12'h004, 12'h008, 12'h00c, 12'h010, and 12'h018 (TISR).
  + Step 4: Write 32'h0000\_0001 to 12'h018 to clear interrupt (if any).
  + Step 5: Verify TDR0 and TDR1 do not change, and tim\_int remains low.

#### **3. Test case: Verify counter operation with timer\_en = 1, div\_en = 0, div\_val = 0**

* **Expectation:** Counter increments continuously, TDR0 and TDR1 reflect the count, and interrupt triggers when count matches TCMP0/TCMP1.
* **Test flow:**
  + Step 1: Write 32'h0000\_0001 to 12'h000 (TCR) to set timer\_en = 1, div\_en = 0, div\_val = 0.
  + Step 2: Read 12'h004 (TDR0) and 12'h008 (TDR1) to verify initial values.
  + Step 3: Write 32'h0000\_0027 to 12'h00c (TCMP0) and 32'h0000\_0000 to 12'h010 (TCMP1).
  + Step 4: Write 32'h0000\_0001 to 12'h014 (TIER) to enable interrupt.
  + Step 5: Wait 10ns, then read 12'h004, 12'h008, 12'h00c, 12'h010, and 12'h018 (TISR) to check interrupt status.
  + Step 6: Write 32'h0000\_0001 to 12'h018 to clear interrupt.
  + Step 7: Verify tim\_int and tim\_prdata from 12'h018 reflect interrupt clearing.

#### **4. Test case: Verify counter with div\_en = 1, div\_val = 0**

* **Expectation:** Counter increments with default mode (no division), interrupt triggers when count matches TCMP0/TCMP1.
* **Test flow:**
  + Step 1: Write 32'h0000\_0002 to 12'h000 (TCR) to set timer\_en = 1, div\_en = 1, div\_val = 0.
  + Step 2: Read 12'h004 (TDR0) and 12'h008 (TDR1).
  + Step 3: Write 32'h0000\_0048 to 12'h00c (TCMP0) and 32'h0000\_0000 to 12'h010 (TCMP1).
  + Step 4: Wait 10ns, then read 12'h004, 12'h008, 12'h00c, 12'h010, and 12'h018 (TISR).
  + Step 5: Write 32'h0000\_0001 to 12'h018 to clear interrupt.
  + Step 6: Verify interrupt behavior.

#### **5. Test case: Verify counter with various div\_val (1 to 9)**

* **Expectation:** Counter increments based on div\_val, with prescaler limit applied, and interrupt triggers accordingly.
* **Test flow:**
  + Step 1: For each div\_val (1 to 9):
    - Write 32'h0X\_03 to 12'h000 (TCR) where X is div\_val (e.g., 01\_03 for div\_val = 1).
    - Read 12'h004 and 12'h008.
    - Write appropriate TCMP0 (e.g., 0x48, 0xb9, etc.) to 12'h00c and 0x0 to 12'h010.
    - Wait delay (e.g., #40 for div\_val = 3, #640 for div\_val = 7).
    - Read 12'h004, 12'h008, 12'h00c, 12'h010, and 12'h018.
    - Write 32'h0000\_0001 to 12'h018 to clear interrupt.
* Step 2: Verify count increments match limit from div\_val.
* Step 3: Turn off timer\_en and read again.

#### **6. Test case: Verify interrupt enable/disable**

* **Expectation:** Interrupt (tim\_int) triggers only when TIER = 1 and count matches TCMP0/TCMP1.
* **Test flow:**
  + Step 1: Write 32'h0000\_0001 to 12'h014 (TIER) to enable interrupt.
  + Step 2: Write 32'h0000\_0048 to 12'h00c (TCMP0) and configure timer.
  + Step 3: Wait and read 12'h018 (TISR) to verify tim\_int.
  + Step 4: Write 32'h0000\_0000 to 12'h014 to disable interrupt.
  + Step 5: Verify tim\_int remains low despite count match.
  + Step 6: Re-enable interrupt and clear with 12'h018.

#### **7. Test case: Verify TCMP0/TCMP1 write/read**

* **Expectation:** Written values (0xffff\_ffff) are correctly read back.
* **Test flow:**
  + Step 1: Write 32'hffff\_ffff to 12'h00c (TCMP0) and 12'h010 (TCMP1).
  + Step 2: Read 12'h00c and 12'h010 to verify values.

#### **8. Test case: Verify TDR0/TDR1 write/read**

* **Expectation:** Written values (0xffff\_ffff) are correctly read back after write.
* **Test flow:**
  + Step 1: Write 32'h0000\_0001 to 12'h000 (TCR) to enable timer.
  + Step 2: Write 32'hffff\_ffff to 12'h004 (TDR0) and wait 16ns.
  + Step 3: Read 12'h004 to verify.
  + Step 4: Write 32'hffff\_ffff to 12'h008 (TDR1) and wait 1000ns.
  + Step 5: Read 12'h008 to verify.

#### **10. Test case: Verify invalid address behavior**

* **Expectation:** Reading/writing to invalid address (e.g., 12'h020) returns 0 or no effect.
* **Test flow:**
  + Step 1: Write 32'hffff\_ffff to 12'h020.
  + Step 2: Read 12'h020 to verify response.
  + Step 3: Write to valid addresses and re-read 12'h020.