

**ĐỒ ÁN CUỐI KHÓA**

**THIẾT KẾ 8 BIT TIMER IP VỚI APB BUS**

**VŨ TUẤN HÙNG**

**BIC09092024**

***LỜI CÁM ƠN***

*Trước hết, em muốn bày tỏ lòng biết ơn chân thành đến Trung tâm Semicon vì đã tổ chức khóa học Thiết kế IC cơ bản. Đây là cơ hội tốt để emcó được kiến ​​thức và kỹ năng nền tảng trong thiết kế IC.*

*Tiếp theo, em muốn gửi lời cảm ơn đặc biệt đến anh Tuong Nguyen, giảng viên của chúng lớp BIC09092024, vì sự hướng dẫn tận tình và chia sẻ sâu sắc về kinh nghiệm thực tế. Sự chuyên nghiệp và nhiệt tình của anh đã giúp em hiểu rõ hơn về các khái niệm và quy trình trong thiết kế IC, đồng thời khơi dậy niềm đam mê của em đối với lĩnh vực này.*

*Em hy vọng rằng Trung tâm Semicon và anh Tuong Nguyen sẽ tiếp tục truyền cảm hứng và hỗ trợ nhiều học viên hơn nữa trong tương lai.*

**MỤC LỤC**

[**I.** **LÍ THUYẾT** 4](#_Toc183983475)

[**1.Verilog** 4](#_Toc183983476)

[**2. Lõi IP** 4](#_Toc183983477)

[**3. APB Bus** 6](#_Toc183983478)

[**4. Timer IP** 8](#_Toc183983479)

[**II. Thông số kĩ thuật Timer IP** 10](#_Toc183983480)

[**1. Sơ đồ khối** 10](#_Toc183983481)

[**2. Mô tả I/O** 10](#_Toc183983482)

[**3. Mô tả các thanh ghi** 11](#_Toc183983483)

[**III. Thiết kế và mô phỏng** 13](#_Toc183983484)

[**1. Thiết kế** 13](#_Toc183983485)

[**2. Mô phỏng** 20](#_Toc183983486)

# **LÍ THUYẾT**

## **Verilog**

Verilog, được tiêu chuẩn hóa thành IEEE 1364, là một ngôn ngữ mô tả phần cứng (hardware description language, viết tắt: HDL) được sử dụng để mô hình hóa các hệ thống điện tử. Verilog được ứng dụng rộng rãi trong thiết kế và xác minh các mạch số ở mức thanh ghi (Register Transfer Level – RTL). Ngoài ra, nó còn được sử dụng trong việc kiểm tra các mạch tương tự (analog) và mạch tín hiệu hỗn hợp (mixed signal).

Vào năm 2009, tiêu chuẩn Verilog (IEEE 1364-2005) đã được hợp nhất vào tiêu chuẩn System Verilog, tạo thành IEEE 1800-2009. Từ đó, Verilog chính thức trở thành một phần của ngôn ngữ System Verilog. Phiên bản hiện tại là IEEE 1800-2017.

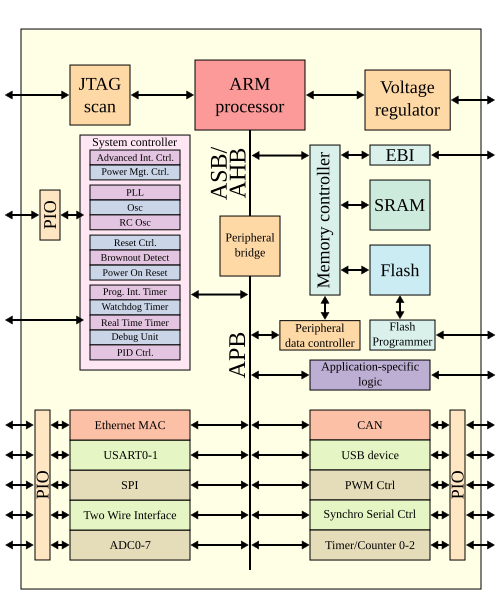
Một ngôn ngữ HDL phổ biến khác bên cạnh Verilog là VHDL (Very High Speed Integrated Circuit HDL). Trước đây, VHDL thường được sử dụng nhiều ở các nước phương Tây, đặc biệt là châu Âu, trong khi Verilog phổ biến hơn tại các nước châu Á. Tuy nhiên, với sự phát triển mạnh mẽ của System Verilog, Verilog hiện nay có phần chiếm ưu thế hơn so với VHDL nhờ khả năng tương thích tốt với System Verilog.

Đối với mạch số, Verilog có 2 chứng năng chính

* Thiết kế mạch số, dùng trong khâu RTL Design
* Kiểm tra xác minh mạch số, dùng trong khâu Design Verification.

## **2. Lõi IP**

Trong ngành công nghiệp bán dẫn, intellectual property (IP) core là một khối logic tái sử dụng, một khối chức năng hoặc một thiết kế layout. Các IP cores này thường được phát triển với mục đích cấp phép cho nhiều nhà cung cấp khác nhau, như các khối xây dựng cơ bản trong thiết kế chip. Trong thời đại thiết kế integrated circuit (IC) ngày nay, các thiết kế system-on-chip (SoC), tích hợp nhiều chức năng hệ thống trên một chip duy nhất, ngày càng phổ biến. Các IP cores được thiết kế sẵn đóng vai trò quan trọng trong phát triển SoC, vì nhiều thiết kế SoC bao gồm bộ xử lý chuẩn và các chức năng hệ thống chuẩn hóa, có thể tái sử dụng qua nhiều dự án sau khi đã phát triển..



*Các IP trong 1 Soc*

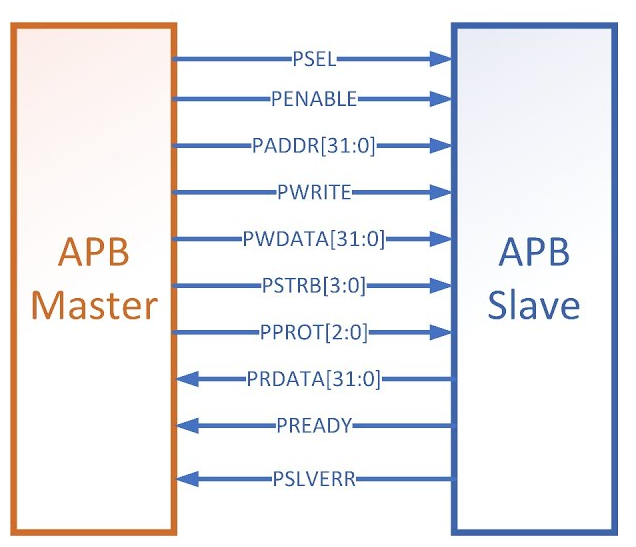
IP cores thường được cấp phép dưới hai dạng chính:

* Soft IP cores:
* Soft IP cores thường được cung cấp dưới dạng các mô hình RTL (register transfer level) có khả năng tổng hợp.
* Các mô hình này được viết bằng các hardware description languages (HDLs) như SystemVerilog hoặc VHDL.
* Trong một số trường hợp, soft IP cores được tổng hợp thành gate-level netlist chung, có thể được ánh xạ sang bất kỳ công nghệ sản xuất nào. Điều này cũng thuộc loại soft IP.
* Hard IP cores:
* Hard IP cores được cung cấp dưới dạng thiết kế layout vật lý, thường ở định dạng như GDSII, đã được ánh xạ sẵn cho một công nghệ sản xuất cụ thể.
* Người dùng có thể tích hợp chúng trực tiếp vào layout cuối cùng của chip, nhưng không thể sửa đổi thiết kế để sử dụng với các công nghệ khác.

Tóm lại, soft IP cores được sử dụng rộng rãi cho logic số vì tính linh hoạt và khả năng tùy chỉnh, trong khi hard IP cores là yếu tố quan trọng cho các chức năng tương tự và tín hiệu hỗn hợp, nơi yêu cầu độ chính xác và thiết kế cố định.

## **3. APB Bus**

Advanced peripheral bus (APB) là giao thức đơn giản nhất trong họ AMBA (advanced microcontroller bus architecture). APB được thiết kế để kết nối các thiết bị ngoại vi có tần số thấp như GPIO, timer, I2C, SPI và UART.



*APB Bus*

Các loại tín hiệu chính trong APB gồm:

* Tín hiệu hệ thống (system signals):
* PCLK: Tín hiệu clock.
* PRESETn: Tín hiệu reset active-low.
* Tín hiệu điều khiển (control signals):
* PSEL: Chọn slave.
* PENABLE: Kích hoạt quá trình truyền.
* PADDR: Địa chỉ của thanh ghi mục tiêu.
* PWRITE: Xác định hoạt động ghi (1) hoặc đọc (0).
* Tín hiệu dữ liệu (data signals):
* PWDATA: Dữ liệu ghi.
* PRDATA: Dữ liệu đọc.
* Tín hiệu phản hồi (response signals):
* PREADY: Cho biết slave đã sẵn sàng.
* PSLVERR: Báo lỗi từ slave (1 là lỗi, 0 là thành công).

Đặc điểm nổi bật của APB

* Thiết kế đơn giản: APB có thiết kế đơn giản hơn so với các giao thức khác trong họ AMBA như AHB hay AXI, giúp giảm chi phí và độ phức tạp khi tích hợp vào các hệ thống SoC.
* Hiệu suất phù hợp: APB không yêu cầu tốc độ cao mà tập trung vào các thiết bị ngoại vi tần số thấp, đảm bảo hiệu suất và tiết kiệm năng lượng.
* Dễ dàng triển khai: Do không cần xử lý các cơ chế phức tạp như pipeline hoặc burst, APB dễ dàng triển khai và sử dụng trong các hệ thống nhỏ gọn.

APB là lựa chọn lý tưởng cho các thiết bị ngoại vi đơn giản, đóng vai trò hỗ trợ cho các giao thức cao cấp hơn trong hệ thống AMBA.

## **4. Timer IP**

Timer IP là một khối chức năng được thiết kế để đo thời gian, đếm sự kiện hoặc tạo xung với độ chính xác cao. Nó được tích hợp vào các chip (ví dụ: FPGA, ASIC, hoặc vi xử lý) để xử lý các tác vụ liên quan đến thời gian hoặc đồng bộ hóa trong hệ thống.

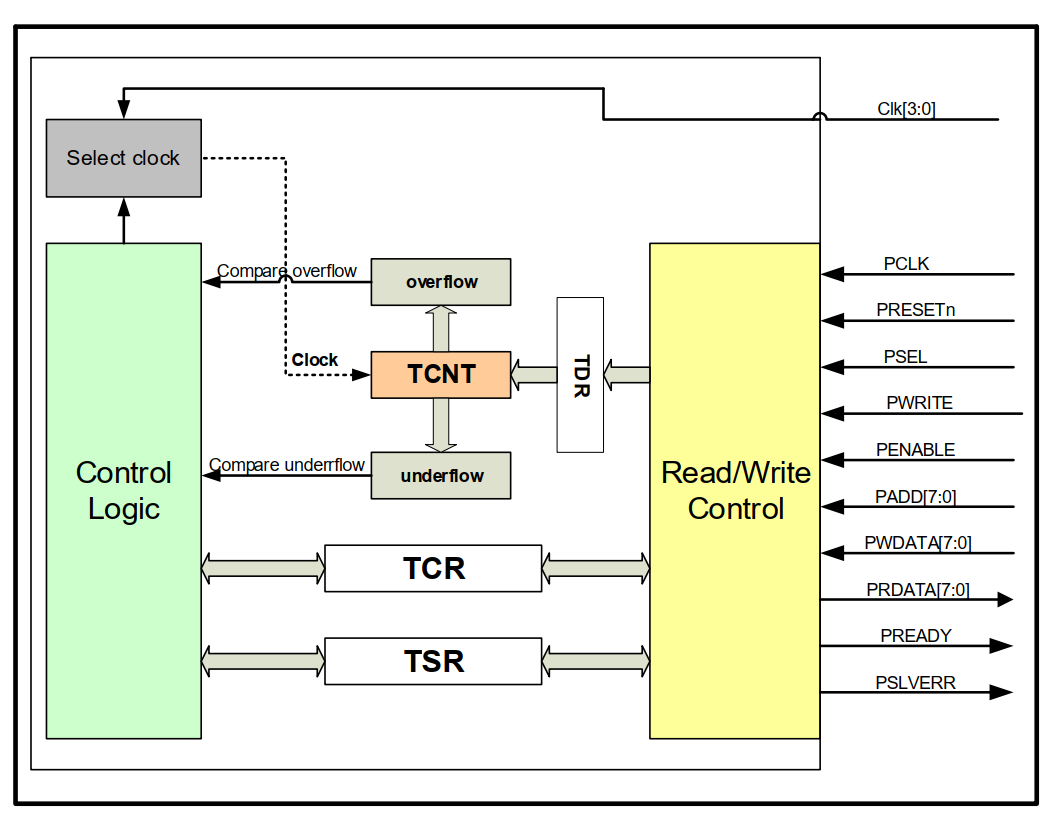
Vai trò của timer IP trong chip:

* Quản lý thời gian chính xác:
* Thực hiện real-time clock (RTC).
* Theo dõi thời gian trong các hệ thống nhúng hoặc vi điều khiển.
* Kiểm soát sự kiện định kỳ:
* Tạo tín hiệu pulse width modulation (PWM) để điều khiển động cơ hoặc LED.
* Lập lịch các tác vụ trong vi xử lý hoặc hệ điều hành nhúng.
* Giám sát hệ thống:
* watchdog timer: Giám sát hoạt động của hệ thống và khởi động lại nếu phát hiện lỗi, ngăn ngừa sự cố hệ thống.
* timeout detection: Phát hiện độ trễ hoặc timeout trong quá trình xử lý dữ liệu hoặc truyền thông.
* Đồng bộ hóa và đo lường:
* Đồng bộ tín hiệu giữa các thành phần, đặc biệt trong các hệ thống đa luồng hoặc đa lõi.
* Đo độ trễ của hệ thống hoặc hiệu năng thời gian thực.
* Ứng dụng trong các giao thức truyền thông:
* Hỗ trợ yêu cầu về thời gian và đồng bộ hóa trong các giao thức như I2C, SPI, UART, Ethernet.
* Hiệu quả tài nguyên và năng lượng:
* Giảm tải các chức năng liên quan đến thời gian từ CPU hoặc các khối logic khác, tự động hóa các tác vụ để tiết kiệm tài nguyên và năng lượng của hệ thống.

Timer IP là một thành phần linh hoạt và cần thiết trong thiết kế chip hiện đại. Nó cung cấp khả năng đo thời gian chính xác, kiểm soát sự kiện, đồng bộ hóa tín hiệu và nâng cao độ tin cậy, hiệu quả của hệ thống trong nhiều ứng dụng khác nhau.

# **II. Thông số kĩ thuật Timer IP**

## **1. Sơ đồ khối**



*Sơ đồ khối IP Timer*

## **2. Mô tả I/O**

|  |  |  |  |
| --- | --- | --- | --- |
| **Port** | **I/O** | **Độ dài bit** | **Mô tả** |
| PCLK | input | 1 | Clock. |
| PRESETn | input | 1 | Reset. Tín hiệu kích hoạt tích cực thấp |
| PSEL | input | 1 | Select. |
| PENABLE | input | 1 | Enable. Tín hiệu này chỉ ra chu kỳ thứ hai và các chu kỳ tiếp theo của quá trình truyền tín hiệu APB. |
| PADDR | input | 8 | Bus địa chỉ |
| PWRITE | input | 1 | Tín hiệu cho phép APB ghi đến TDR, TCR, TSR khi tích cực cao và đọc giá trị khi tích cực thấp |
| PWDATA | input | 8 | Ghi data |
| PRDATA | output | 8 | Đọc data |
| PREADY | output | 1 | Ready. Tín hiệu báo sẵn sàng |
| PSLVERR | output | 1 | Báo lỗi |

*I/O Timer IP*

## **3. Mô tả các thanh ghi**

* Timer Data Register (TDR): thanh ghi này chứa dữ liệu được sử dụng để cập nhật giá trị của bộ đếm khi thanh ghi này được cập nhật thành giá trị mới.
* Timer counter control Register (TCR):

|  |  |  |
| --- | --- | --- |
| Tên Bit | F/V | Mô tả |
| Load[7] | R/W | Chuyển data từ TDR sang TCNT khi tích cực cao.  1: Chuyển data từ TDR sang TCNT  0: Hoạt động bình thường |
| 6 | Reserved | Không sử dụng |
| Up/Dw[5] | R/W | Điều khiển đếm lên hoặc đếm xuống  0: Đếm lên  1: Đếm xuống |
| En[4] | R/W | 0 : disable  1: enable |
| 3:2 | Không sử dụng | Không sử dụng |
| cks[1:0] | R/W | Chọn tần số  00 : T\*2  01 : T\*4  10 : T\*8  11: T\*16 |

*Mô tả TCR*

- Time Statue Register (TSR):

|  |  |  |
| --- | --- | --- |
| Tên Bit | F/V | Mô tả |
| 7:2 | R | Không sử dụng |
| S\_TMR\_UDF[1] | R/W\* | underflow khi đếm 8’h00 xuống 8’hff |
| S\_TMR\_OVF[0] | R/W\* | overflow khi đếm 8’hFF lên 8’h00 |

*Mô tả TSR*

# **III. Thiết kế và mô phỏng**

## **1. Thiết kế**

module timer:

`timescale 1ns / 1ps

module timer(

input [3:0] Clk,

input PCLK,

input PRESETn,

input PSEL,

input PENABLE,

input PWRITE,

input [7:0] PADDR,

input [7:0] PWDATA,

output [7:0] PRDATA,

output PREADY,output PSLVERR

);

wire [1:0] cks;

wire clk\_in, Load, Up\_Down, EN;

wire s\_tmr\_unf, s\_tmr\_ovf;

wire [7:0] TCNT, TCR, TDR, TSR;

assign cks = TCR[1:0];

assign Load = TCR[7];

assign Up\_Down = TCR[5];

assign EN = TCR[4];

clock\_divider clk\_divider (

.PCLK(PCLK),

.PRESETn(PRESETn),

.cks(cks),

.Clk(Clk), .clk\_in(clk\_in)

);

timer\_counter tc (

.PCLK(PCLK),

.clk\_in(clk\_in),

.PRESETn(PRESETn),

.s\_tmr\_unf(s\_tmr\_unf),

.s\_tmr\_ovf(s\_tmr\_ovf),

.Load(Load),

.Up\_Down(Up\_Down),

.EN(EN),

.TDR(TDR),

.TCNT(TCNT)

);

apb\_control apb\_ctr (

.PCLK(PCLK),

.PRESETn(PRESETn),

.PSEL(PSEL),

.PENABLE(PENABLE),

.PWRITE(PWRITE),

.PADDR(PADDR),

.PWDATA(PWDATA),

.PRDATA(PRDATA),

.PREADY(PREADY),

.PSLVERR(PSLVERR),

.TCR(TCR),

.TDR(TDR),

.TSR(TSR)

);

timer\_status ts (

.PCLK(PCLK),

.PRESETn(PRESETn),

.Up\_Down(Up\_Down),

.TCNT(TCNT),

.s\_tmr\_unf(s\_tmr\_unf),

.s\_tmr\_ovf(s\_tmr\_ovf)

);

endmodule

- module timer\_status:

module timer\_status(

input PCLK,

input PRESETn,

input Up\_Down,

input [7:0] TCNT,

output reg s\_tmr\_unf,

output reg s\_tmr\_ovf

);

initial begin

s\_tmr\_unf = 0;

s\_tmr\_ovf = 0;

end

always @(posedge PCLK or negedge PRESETn) begin

if (!PRESETn) begin

s\_tmr\_unf <= 0;

s\_tmr\_ovf <= 0;

end else begin

if ((TCNT == 8'b00000000)&& (Up\_Down==1)) begin

s\_tmr\_unf <= 1;

s\_tmr\_ovf <= 0;

end else if ((TCNT == 8'b11111111)&&(Up\_Down==0)) begin

s\_tmr\_unf <= 0;

s\_tmr\_ovf <= 1;

end else begin

s\_tmr\_unf <= 0;

s\_tmr\_ovf <= 0;

end

end

end

endmodule

- module clock\_divider:

module clock\_divider(

input PCLK,

input PRESETn,

input [1:0] cks,

input [3:0] Clk,

output reg clk\_in

);

reg [3:0] counter = 0;

initial begin

clk\_in = 0;

end

always @(posedge PCLK or negedge PRESETn) begin

if (!PRESETn) begin

counter <= Clk;

clk\_in <= PCLK;

end else begin

case (cks)

2'b00: begin

if (counter + 2 > 15) begin

if (Clk == (counter + 2) - 16) begin

clk\_in <= ~clk\_in;

counter <= Clk;

end

end else if (Clk == counter + 2) begin

clk\_in <= ~clk\_in;

counter <= Clk;

end

end

2'b01: begin

if (counter + 4 > 15) begin

if (Clk == (counter + 4) - 16) begin

clk\_in <= ~clk\_in;

counter <= Clk;

end

end else if (Clk == counter + 4) begin

clk\_in <= ~clk\_in;

counter <= Clk;

end

end

2'b10: begin

if (counter + 8 > 15) begin

if (Clk == (counter + 8) - 16) begin

clk\_in <= ~clk\_in;

counter <= Clk;

end

end else if (Clk == counter + 8) begin

clk\_in <= ~clk\_in;

counter <= Clk;

end

end

2'b11: begin

if (counter + 16 > 15) begin

if (Clk == (counter + 16) - 16) begin

clk\_in <= ~clk\_in;

counter <= Clk;

end

end else if (Clk == counter + 16) begin

clk\_in <= ~clk\_in;

counter <= Clk;

end

end

default: begin counter <= Clk;

clk\_in <= PCLK;

end

endcase

end

end

endmodule

- module apb\_control:

module apb\_control(

input PCLK,

input PRESETn,

input PSEL,

input PENABLE,

input PWRITE,

input [7:0] PADDR,

input [7:0] PWDATA,

output reg [7:0] PRDATA,

output reg PREADY,

output reg PSLVERR,

output reg [7:0] TCR,

output reg [7:0] TDR,

output reg [7:0] TSR

);

always @(posedge PCLK or negedge PRESETn) begin

if (!PRESETn) begin

TCR <= 8'b0;

TDR <= 8'b0;

TSR <= 8'b0;

PRDATA <= 8'b0;

PREADY <= 1'b0;

PSLVERR <= 1'b0;

end else if (PSEL && PENABLE) begin

PREADY <= 1'b1;

if (PWRITE) begin

case (PADDR)

8'h00: TCR <= PWDATA;

8'h01: TDR <= PWDATA;

8'h02: TSR <= PWDATA;

default: PSLVERR <= 1'b1;

endcase

end else begin

case (PADDR)

8'h00: PRDATA <= TCR;

8'h01: PRDATA <= TSR;

default: PSLVERR <= 1'b1;

endcase

end

end else begin

PREADY <= 1'b0;

end

end

endmodule

- module timer\_counter:

module timer\_counter(

input PCLK,

input clk\_in,

input PRESETn,

input s\_tmr\_unf,

input s\_tmr\_ovf,

input Load,

input Up\_Down,

input EN,

input [7:0] TDR,

output reg [7:0] TCNT

);

reg allow\_count;

always @(posedge PCLK or negedge PRESETn) begin

if (!PRESETn) begin

TCNT <= 0;

allow\_count <= 0;

end else begin

if (Load) begin

TCNT <= TDR;

end else if (s\_tmr\_unf && allow\_count && clk\_in) begin

TCNT <= 8'b11111111;

end else if (s\_tmr\_ovf && allow\_count && clk\_in) begin

TCNT <= 8'b0;

end

if (!clk\_in) begin

allow\_count <= 1;

end

if (!Load && EN && allow\_count && clk\_in) begin

if (Up\_Down) begin

TCNT <= TCNT - 1;

end else begin

TCNT <= TCNT + 1;

end

allow\_count <= 0;

end

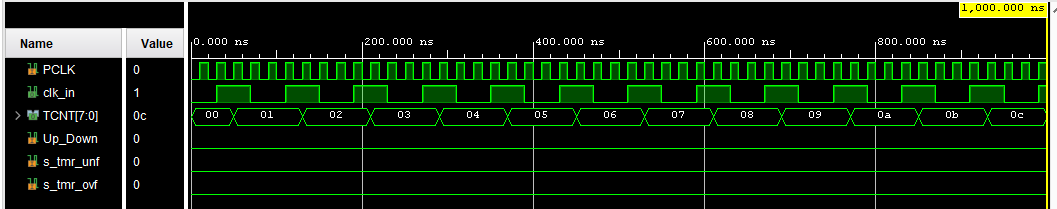
end

end

endmodule

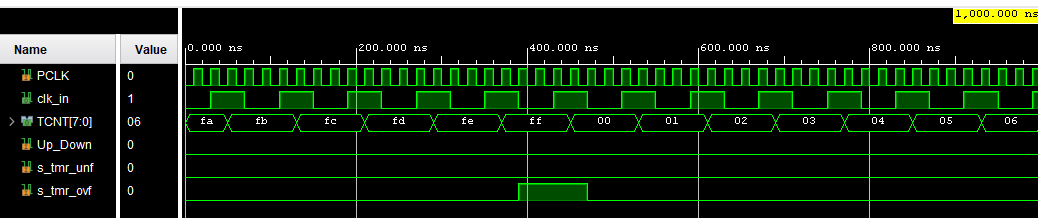
## **2. Mô phỏng**

Testcase 1: Đếm lên từ 8’h0



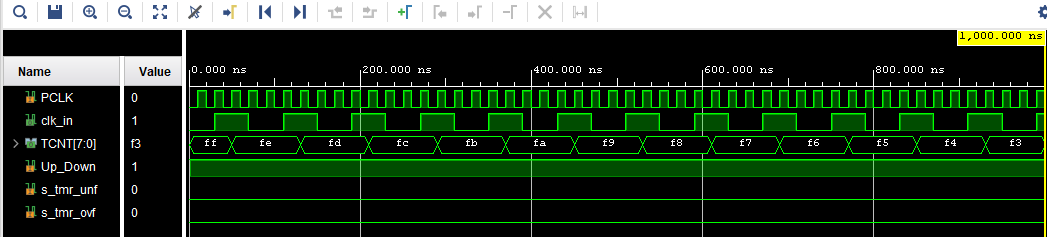
*Đếm lên*

Testcase2 : Đếm lên từ 8’FA cho đến khi overflow



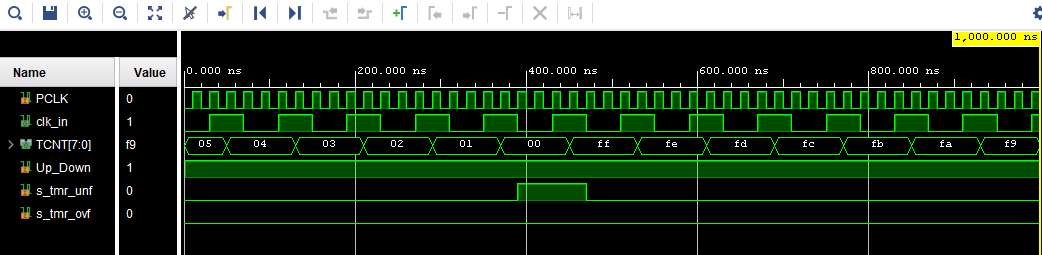
*Overflow*

Testcase 3: Đếm xuống từ 8h’FF



*Đếm xuống*

Testcase 4: Đếm xuống từ 8h’05 đến khi underflow



*Underflow*