TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH

KHOA ĐIỆN – ĐIỆN TỬ

BỘ MÔN KỸ THUẬT MÁY TÍNH – VIỄN THÔNG

BÁO CÁO THIẾT KẾ

BỘ CHUYỂN ĐỔI GIAO THỨC APB – TIMER

26.05.2024

Đinh Đức Anh

21161391

MỤC LỤC

LỜI NÓI ĐẦU

Trong kỷ nguyên công nghệ, việc thiết kế và phát triển hệ thống trên chip ngày càng được quan tâm bởi nhu cầu các thiết bị điện tử ngày nay là nhỏ, thực hiện được nhiều chức năng… trong đó, các giao thức truyền nhận dữ liệu trên bus được sử dụng rộng rãi để truyền tải dữ liệu giữa các thành phần trên chip một cách dễ dàng. Một số vấn đề quan trọng trong thiết kế này là cách thức truyền thông giữa các thành phần của một hệ thống trên chip với nhau.

Đề tài này sử dụng ngôn ngữ mô tả phần cứng Verilog để thực hiện thiết kế hệ thống và xác minh, mô phỏng chức năng bộ chuyển đổi giao thức APB – Timer.

Tôi hy vọng rằng đề tài này sẽ cung cấp một cái nhìn tổng quan về quá trình hoạt động của bộ chuyển đổi, cũng như đưa ra một số kết quả và đánh giá về tính đúng đắn của hệ thống được thiết kế.

TỔNG QUAN THIẾT KẾ

ĐẶT VẤN ĐỀ

System on a Chip (SoC) là một loại vi mạch tích hợp đặc biệt, được thiết kế để chứa các thành phần của một hệ thống điện tử trên một chip duy nhất. Các thành phần này được gọi là lõi sở hữu trí tuệ (IP core), có thể bao gồm CPU, bộ nhớ, bộ điều khiển giao tiếp… Trong đó, các thành phần này là khác nhau về hệ thống bus, vì thế sẽ có một vài trở ngại trong việc giao tiếp giữa các thành phần này như:

- Trong việc đồng bộ dữ liệu: cần phải đồng bộ dữ liệu để tránh xung đột và dẫn đến lỗi hệ thống. Điều này hết sức khó khăn khi các lõi IP có tốc độ và thời gian phản hồi khác nhau.
- Các giao thức giao tiếp khác nhau: như đã đề cập, các lõi sở hữu trí tuệ trong SoC có thể sử dụng các giao thức giao tiếp khác nhau, điều này có thể dẫn đến sự không tương thích và khó khăn trong việc đồng bộ hóa và truyền thông giữa chúng.
- Khả năng mở rộng: việc mở rộng của một SoC có thể dẫn đến thêm nhiều lõi IP và các tài nguyên mới được thêm vào. Điều này có thể gây ra những khó khăn trong việc tăng tính tương thích giữa các lõi IP.

NHIỆM VỤ NGHIÊN CỨU

Trong đề tài thiết kế bộ chuyển đổi giao thức APB – Timer này, các nhiệm vụ cơ bản phải thực hiện được là:

- Trình bày rõ các lý thuyết có liên quan.
- Thiết kế được bộ chuyển đổi giao thức APB – Timer.
- Xác minh tự động cho hệ thống.
- Trình bày các đánh giá hiệu năng của hệ thống.

GIỚI HẠN VÀ ĐỐI TƯỢNG NGHIÊN CỨU

Giới hạn nghiên cứu

Giới hạn nghiên cứu là ngôn ngữ mô tả phần cứng Verilog, kiểm tra và đánh giá chức năng bộ chuyển đổi bộ chuyển đổi giao thức APB – Timer.

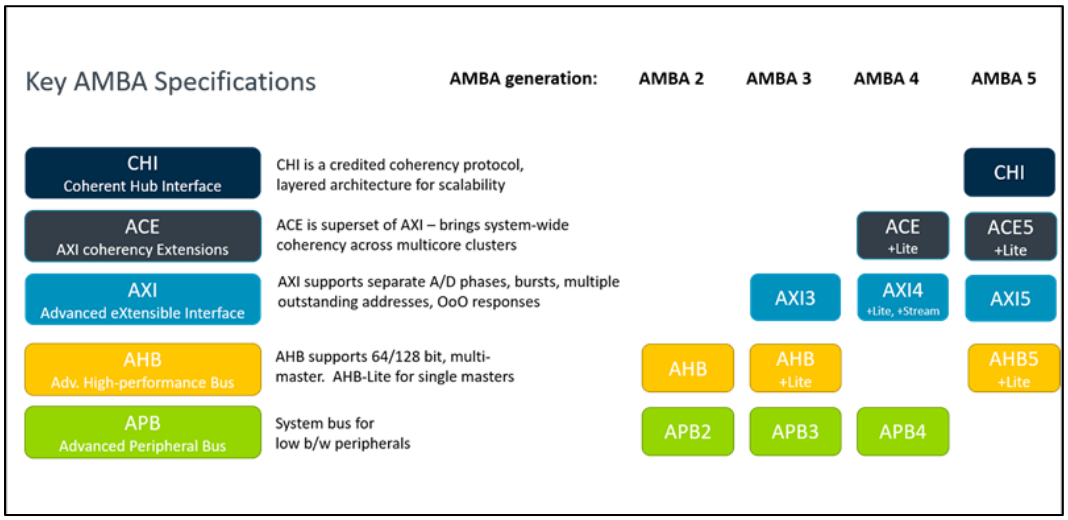
Đối tượng nghiên cứu

Đối tượng nghiên cứu là chuẩn giao tiếp APB bus và thành phần lõi Timer sử dụng phổ biến trong các bộ vi điều khiển.

TỔNG QUAN VỀ AMBA VÀ MỘT SỐ CHUẨN BUS

AMBA là viết tắt của Advanced Microcontroller Bus Architecture, là một chuẩn giao tiếp mở, được giới thiệu lần đầu vào năm 1996. Kiến trúc AMBA được thiết kế bởi ARM (trước đây là Advanced RISC Machines), một công ty sản xuất chip nổi tiếng.

Hình 5.1 dưới đây minh họa một số chuẩn bus và phiên bản của chúng trong họ AMBA.



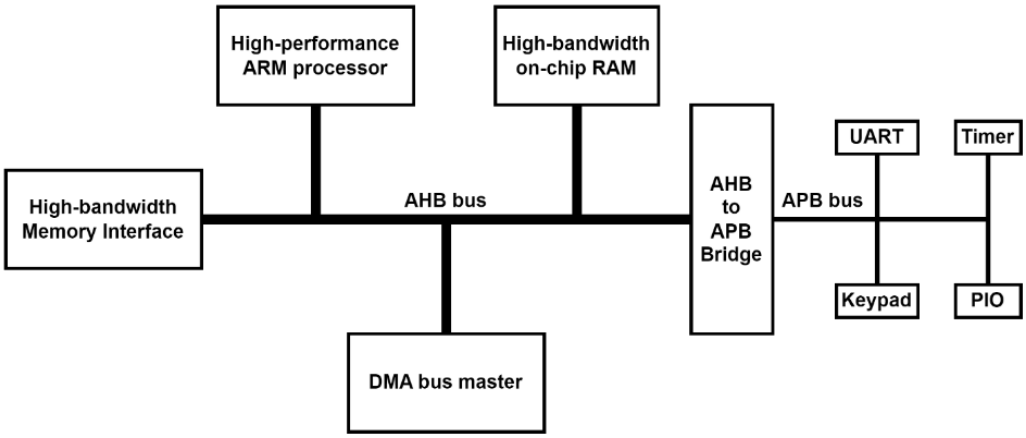
Hình 5.1: Một số chuẩn bus của họ AMBA

AMBA mang lại một số lợi ích sau:

- Linh hoạt: AMBA cung cấp nhiều chuẩn bus khác nhau, có thể kết nối và truyền nhận dữ liệu giữa các thành phần khác nhau trên SoC.
- Hiệu suất cao: với các tính năng pipelining và burst transfer, AMBA giúp tối ưu hóa tốc độ truyền dữ liệu trên bus.
- Độ tin cậy cao: AMBA sử dụng các cơ chế kiểm soát tuyến tính và kiểm soát lỗi để đảm bảo tính toàn vẹn dữ liệu và tránh xung đột trên bus.

- Tiết kiệm chi phí: các nhà thiết kế có thể tận dụng lại các thành phần IP có sẵn, giúp giảm chi phí phát triển và tăng tốc độ sản xuất.
- Dễ dàng tích hợp: AMBA cung cấp các hướng dẫn về cách tích hợp các thành phần vào SoC, giúp các nhà thiết kế dễ dàng phát triển và kiểm tra sản phẩm của họ.

Hình 5.2 minh họa một hệ thống AMBA điển hình.

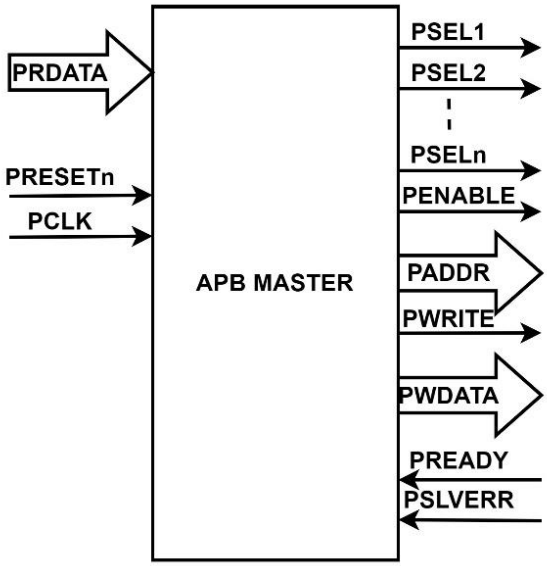


Hình 5.2: Sơ đồ một hệ thống AMBA điển hình

TỔNG QUAN VỀ GIAO THỨC AMBA APB

Thành phần thiết bị chủ APB

Hình 5.3 minh họa sơ đồ khối của thành phần thiết bị chủ trên hệ thống APB.

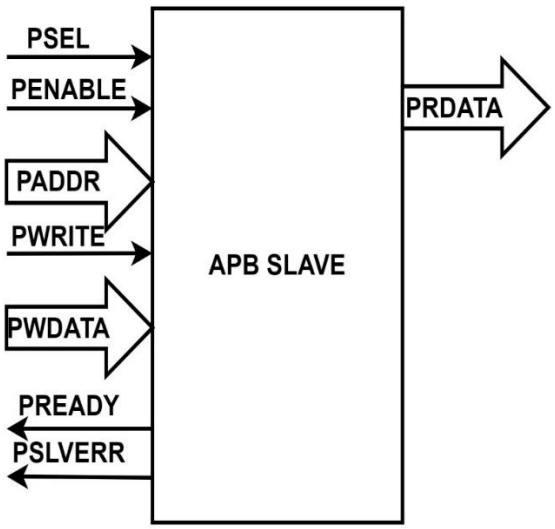


Hình 5.3: Sơ đồ khối của thành phần thiết bị chủ APB

Thành phần này có nhiệm vụ cung cấp dữ liệu, địa chỉ và thông tin điều khiển cho các quá trình giao dịch dữ liệu.

Thành phần thiết bị tớ APB

Hình 5.4 minh họa sơ đồ khối của thành phần thiết bị tớ trên hệ thống APB.



Hình 5.4: Sơ đồ khối của thành phần thiết bị tớ APB

Thành phần này có nhiệm vụ nhận, phản hồi thông tin trong các quá trình giao dịch đọc, ghi dữ liệu từ thành phần thiết bị chủ gửi đến.

Mô tả các đường tín hiệu

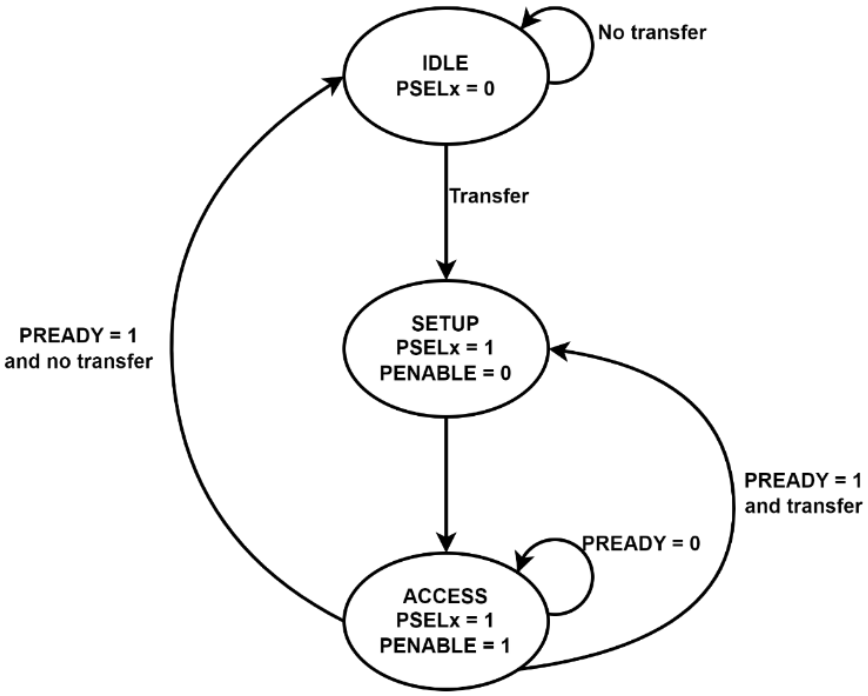
Bảng 2.1: Mô tả một số đường tín hiệu trong AMBA APB

Tên đường tín hiệu	Mô tả
PCLK	Xung đồng hồ, tín hiệu đồng bộ tích cực cạnh lên.
PRESETn	Tín hiệu khởi tạo, tích cực mức thấp.
PADDR	Bus địa chỉ APB, thiết bị chủ gửi địa chỉ đi trên chân này đến thiết bị tớ, chiều rộng tối đa là 32 bits.
PSEL	Tín hiệu lựa chọn, tích cực mức cao. Thiết bị chủ gửi tín hiệu này để lựa chọn thiết bị tớ.

PENABLE	Tín hiệu cho phép, tích cực mức cao. Thiết bị chủ gửi tín hiệu này đến thiết bị tớ để cho phép quá trình đọc hoặc ghi dữ liệu.
PWRITE	Tín hiệu điều khiển quá trình đọc và ghi. Khi tích cực mức thấp, quá trình đọc sẽ được thực hiện, khi tích cực mức cao, quá trình ghi sẽ được thực hiện.
PWDATA	Thiết bị chủ ghi dữ liệu đến thiết bị tớ thông qua chân này, độ rộng tối đa là 32 bits.
PRDATA	Thiết bị chủ đọc dữ liệu được ghi gửi về từ thiết bị tớ thông qua chân này, độ rộng tối đa là 32 bits.
PREADY	Tín hiệu báo hiệu sẵn sàng cho thiết bị chủ, được sử dụng để mở rộng việc truyền tải dữ liệu của APB.
PSELVERR	Là tín hiệu phản hồi lỗi, tín hiệu này cho biết rằng việc truyền tải dữ liệu hiện tại có đúng hay không.

Các trạng thái hoạt động của APB

Hình 5.5 minh họa các trạng thái hoạt động của APB.

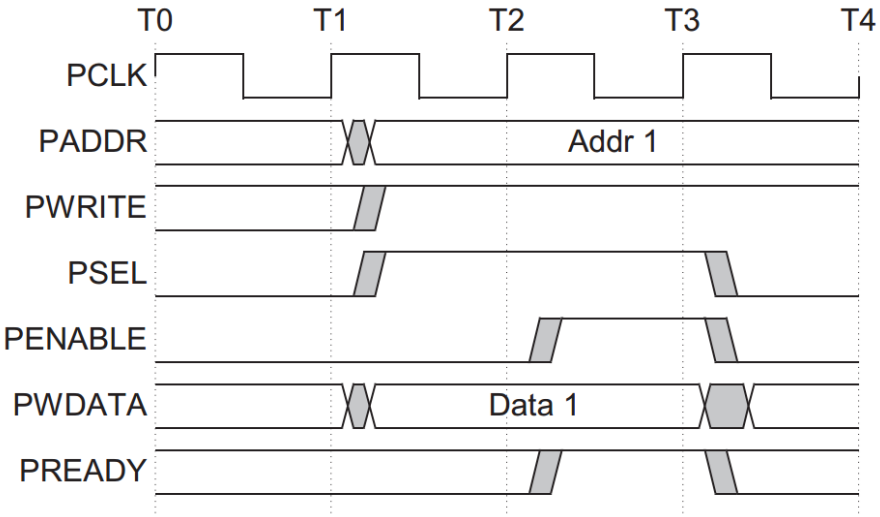


Hình 5.5: Các trạng thái hoạt động của APB

Quá trình ghi dữ liệu trên APB bus

Quá trình ghi không đợi

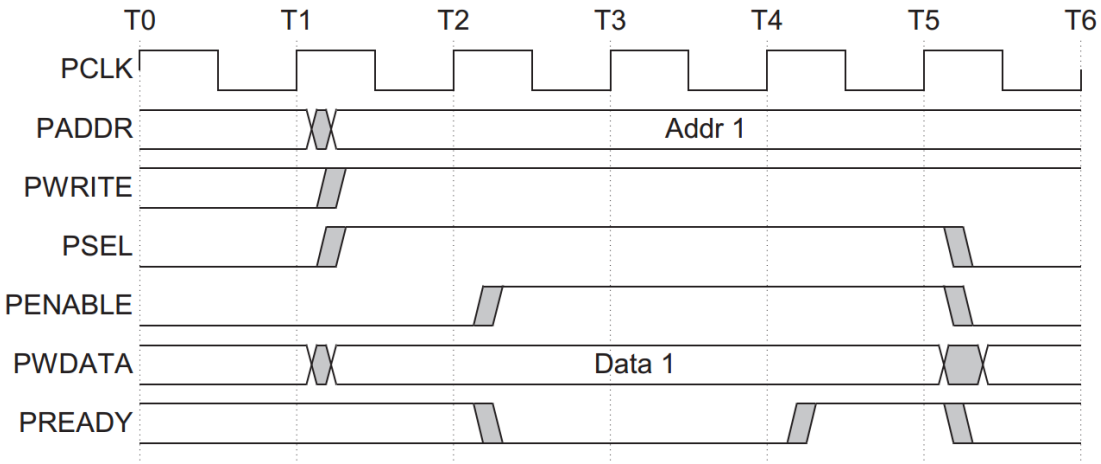
Hình 5.6 minh họa quá trình ghi dữ liệu không đợi trên bus APB.



Hình 5.6: Quá trình ghi dữ liệu không đợi

Quá trình ghi có đợi

Hình 5.7 minh họa quá trình ghi dữ liệu có trạng thái đợi trên bus APB.

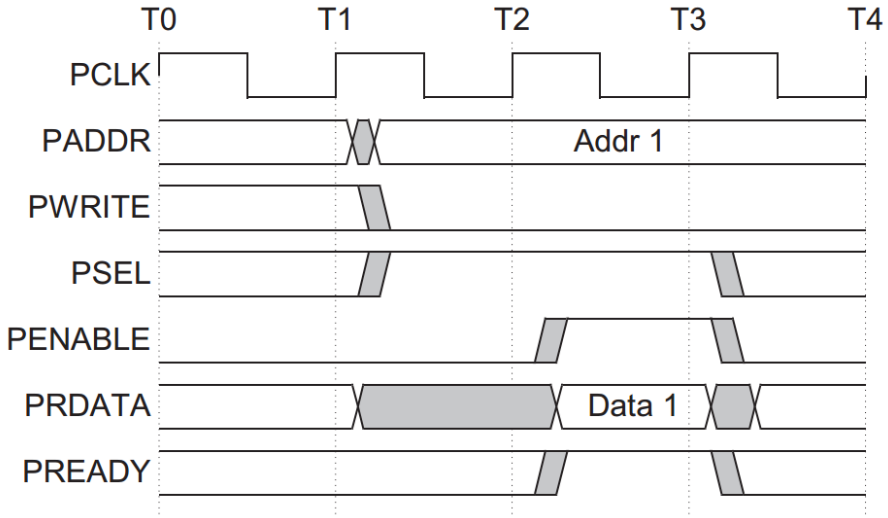


Hình 5.7: Quá trình ghi dữ liệu có đợi

Quá trình đọc dữ liệu trên APB bus

Quá trình đọc không đợi

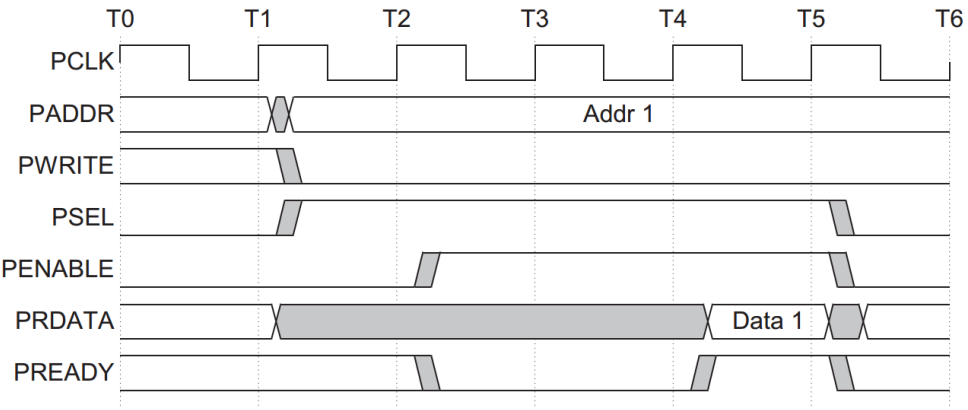
Hình 5.8 minh họa quá trình đọc dữ liệu không có trạng thái đợi trên bus APB.



Hình 5.8: Quá trình đọc dữ liệu không đợi

5.6.5.2 Quá trình đọc có đợi

Hình 5.9 minh họa quá trình đọc dữ liệu có trạng thái đợi trên bus APB.



Hình 5.9: Quá trình đọc dữ liệu có đợi

TỔNG QUAN VỀ LÕI IP TIMER

Timer về cơ bản là một máy trạng thái hữu hạn tăng hoặc giảm một thanh ghi một lần trong 1 chu kì đồng hồ. Tất cả các bộ định thời đều có một thanh ghi chứa giá trị hiện tại của thời gian. Bộ hẹn giờ cũng có một thanh ghi tải lại.

** Time-base unit (Khối cơ sở của bộ Timer)*

– *Counter Register (TIMx_CNT):* Khi hoạt động, thanh ghi này tăng hoặc giảm giá trị theo mỗi xung clock đầu vào. Tùy vào bộ timer mà counter này có thể là 16bit hoặc 32bit.

– *Prescaler Register (TIMx_PSC):* Giá trị của thanh ghi bộ chia tần (16bit) cho phép người dùng cấu hình chia tần số đầu vào (CK_PSC) cho bất kì giá trị nào từ [1- 65536]. Sử dụng kết hợp bộ chia tần của timer và của RCC giúp chúng ta có thể thay đổi được thời gian của mỗi lần CNT thực hiện đếm, giúp tạo ra được những khoảng thời gian, điều chế được độ rộng xung phù hợp với nhu cầu.

– *Auto-Reload Register (TIMx_ARR):* Giá trị của ARR được người dùng xác định sẵn khi cài đặt bộ timer, làm cơ sở cho CNT thực hiện nạp lại giá trị đếm mỗi khi tràn (overflow khi đếm lên – CNT vượt giá trị ARR, underflow khi đếm xuống – CNT bé hơn 0). Tùy vào bộ timer mà counter này có thể là 16bit hoặc 32bit.

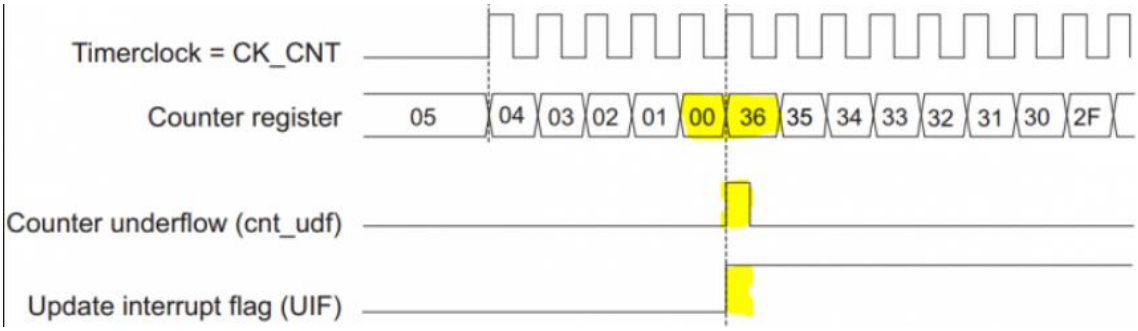
** Các chế độ hoạt động*

– Các chế độ đếm: Mỗi bộ timer đều hỗ trợ 3 chế chế độ đếm sau:

+ Upcounting mode (chế độ đếm lên):

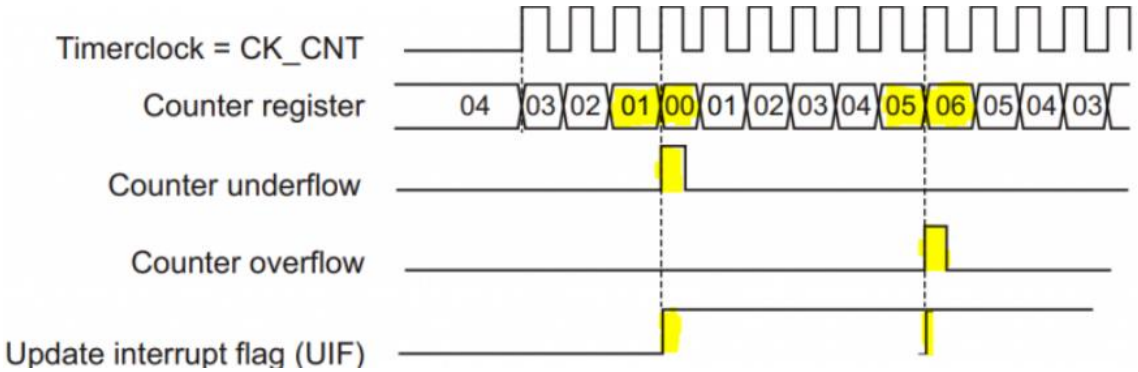
Hình 5.10: Chế độ đếm lên *Upcounting mode*

+ Downcouting mode (chế độ đếm xuống):



Hình 5.11: Chế độ đếm xuống *Downcouting mode*

+ Center-Aligned mode (chế độ đếm lên và xuống):



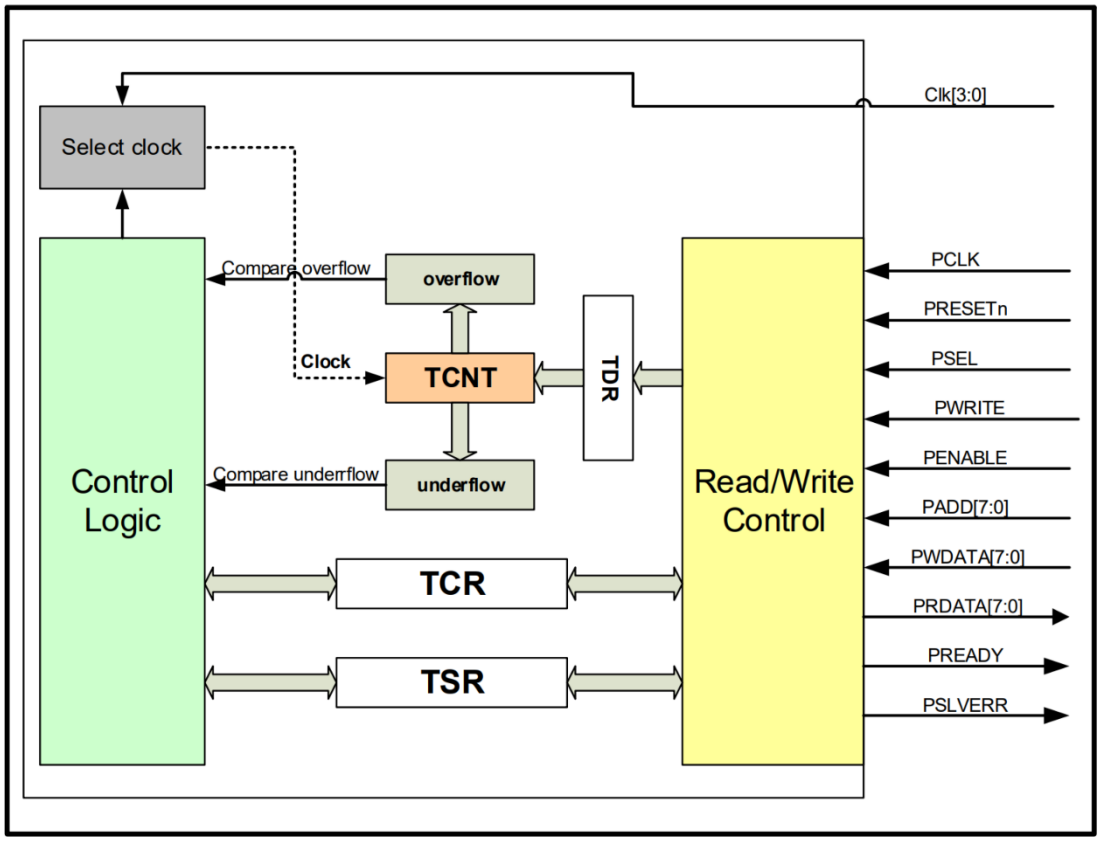
Hình 5.12: Chế độ đếm lên và xuống *Center-Aligned mode*

ĐẶC ĐIỂM THIẾT KẾ

THIẾT KẾ CÁC THÀNH PHẦN CỦA HỆ THỐNG

Thiết kế sơ đồ khối hệ thống

Ở hình 6.1 dưới đây mô tả sơ đồ khối tổng quát của thiết kế



Hình 6.1: Sơ đồ khối hệ thống

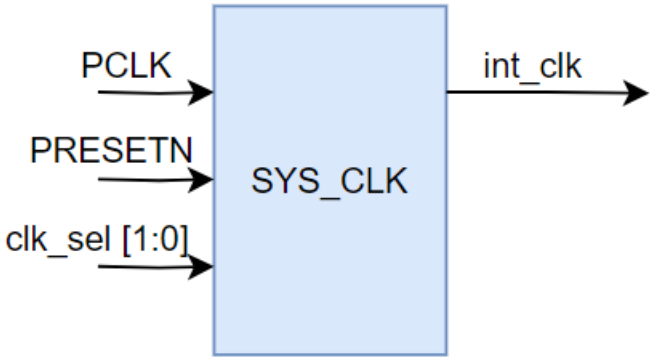
Bộ chuyển đổi giao thức APB – Timer gồm các khối có chức năng sau :

- **Khối Seclect clock:** Bộ chia trước được sử dụng trong thiết kế với các tỉ lệ 2, 4, 8, 16.
- **Khối control logic:** Khối điều khiển logic tạo yêu cầu để bật bit underflow/overflow từ phía phần cứng đồng thời gửi tín hiệu điều khiển cho bộ **Timer counter**.
- **Khối TCNT:** Khối Timer counter thực hiện quá trình tạo bộ đếm cho lõi Timer.
- **Khối Read/Write Control:** Điều khiển cho phép ghi hay đọc dữ liệu ra bus APB.

- **Các khối TDR, TCR , TSR:** Các thanh ghi nội thực hiện từng chức năng trong bộ chuyển đổi giao thức APB – Timer.
- Ở thiết kế này ta lựa chọn thiết kế cố định là bộ Timer 8 bit.

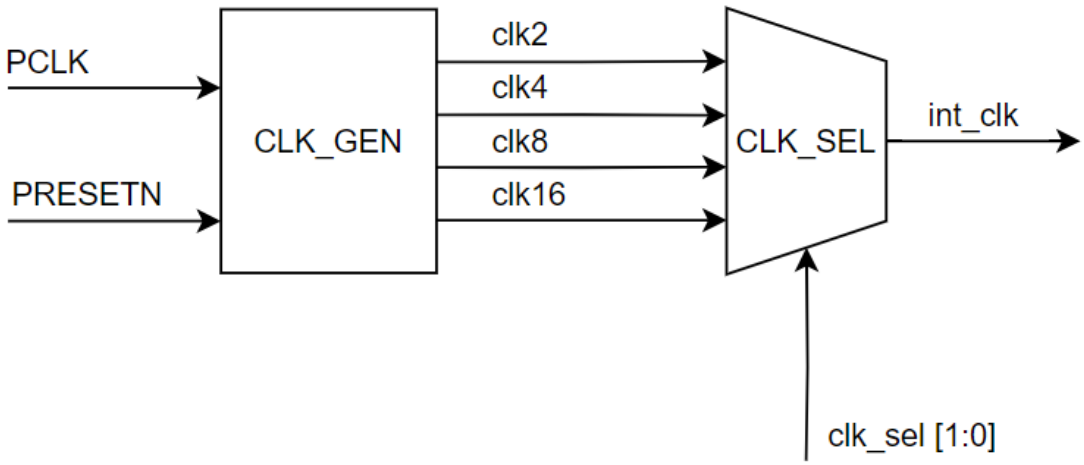
Thiết kế chi tiết từng khối

Thiết kế khối lựa chọn bộ chia trước (clock select)



Hình 6.2: Sơ đồ khối lựa chọn bộ chia trước

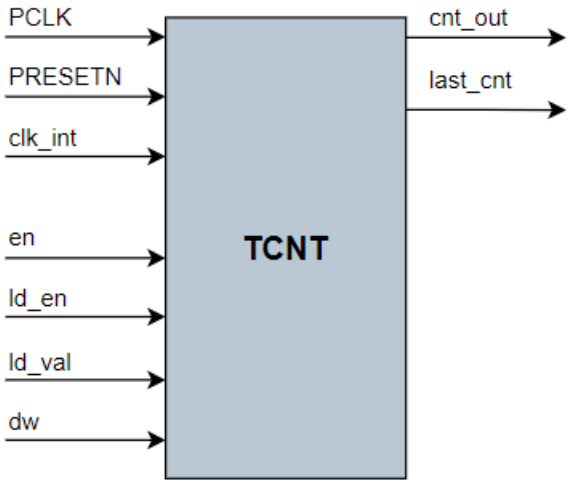
Để tạo ra bộ chia trước lần lượt là 2, 4, 8, 16 ta sử dụng các flip- flop T tạo ra các mạch chia tần số với số flip – flop được sử dụng tối đa là 4.



Hình 6.3: Mạch chọn kênh cho bộ Timer counter

Bộ chia trước sau khi tạo ra các nguồn xung clock với những tần số khác nhau thì cần đưa qua một bộ đa hợp 4 – 1 với tín hiệu clk_sel làm tín hiệu lựa chọn kênh truyền ra cho bộ đếm Timer counter.

Thiết kế khối Timer counter



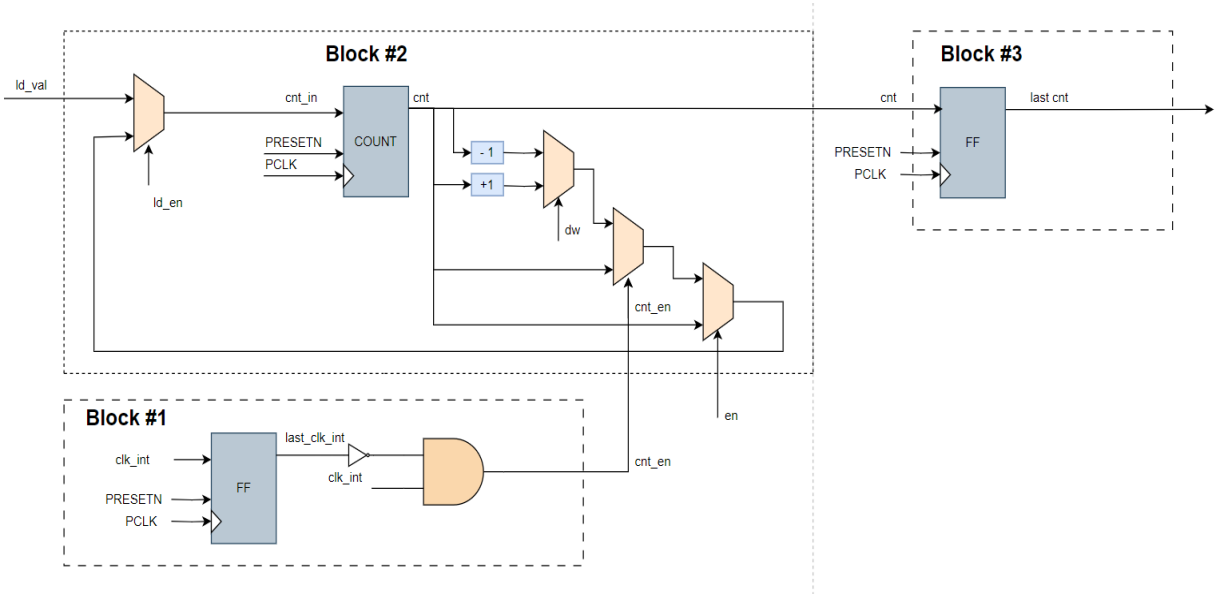
Hình 6.4: Sơ đồ khối tổng quát bộ Timer Counter

Bảng 3.1: Các chân tín hiệu bộ Timer Counter

STT	Chân tín hiệu	Số bit	Phân loại	Mô tả
Tín hiệu bộ Timer Counter				
1	PRESETN	1	Ngõ vào	Tín hiệu khởi động lại thiết kế.
2	PCLK	1	Ngõ vào	Tín hiệu xung clock.
3	clk_int	1	Ngõ vào	Tín hiệu clock từ bộ chia trước.
4	en	1	Ngõ vào	Tín hiệu enable cho bộ đếm.
5	ld_en	1	Ngõ vào	Tín hiệu bật chế độ tải giá trị mới vào bộ đếm.
6	ld_val	8	Ngõ vào	Giá trị tải mới khi ld_en được kích hoạt.

7	dw	1	Ngõ vào	Tín hiệu điều khiển hướng đếm (đếm lên hoặc đếm xuống).
8	cnt_out	1	Ngõ ra	Giá trị hiện tại của bộ đếm.
9	last_cnt	8	Ngõ ra	Giá trị cuối đếm được ngõ ra

Khối này thực hiện chức năng tạo bộ đếm bằng cách thực hiện giống như một mạch đếm.

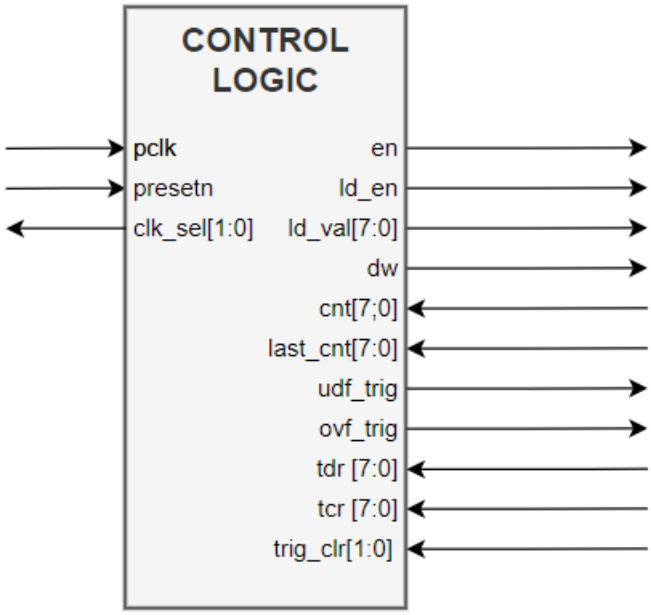


Hình 6.5: Sơ đồ khối logic của Timer Counter

Khi có giá trí **ld_val** và tín hiệu **ld_en** được kích hoạt ngõ ra **cnt_in** sẽ thực hiện đồng bộ với cạnh lên xung PCLK thực hiện quá trình tải các giá trị đếm thông qua bộ flip – flop. Tín hiệu điều khiển en được khi được kích hoạt (**en = 1**) thực hiện cho phép bộ đếm được tiến hành và ngược lại nếu tín hiệu **en** không được kích hoạt (**en = 0**) giữ nguyên trạng thái trước đó. Tín hiệu **cnt_en** được tạo ra bằng phương pháp bắt cạnh lên của xung PCLK, từ tín hiệu **clk_int** từ bộ chia trước thông qua flip – flop tạo ra tín hiệu **last_clk_int** rồi tiến hành đảo tín hiệu và đem giao với bản thân tín hiệu **clk_int,** lợi dụng đặc trưng của cổng logic AND khi có cả 2 tín hiệu bằng 1 thì ngõ ra được tích cực mức 1, vì vậy nếu trong 1 chu kì xung PCLK, tín hiệu **clk_int** được cập nhật ở mức 0 lên mức 1 thì tín hiệu **cnt_en** sẽ

được tích cực 1 lần, tín hiệu này được dùng làm tín hiệu tạo xung tích cực cho mạch đếm có tiếp tục thực hiện đếm giá trị hay không. Cuối cùng, tín hiệu **dw** dùng để điều khiển cập nhật giá trị đếm ở chế độ lựa chọn là đếm lên hay đếm xuống. Giá trị cuối cùng **last_cnt** sẽ được đưa đến khối control logic.

Thiết kế khối Control logic



Hình 6.5: Sơ đồ khối logic của Timer Counter

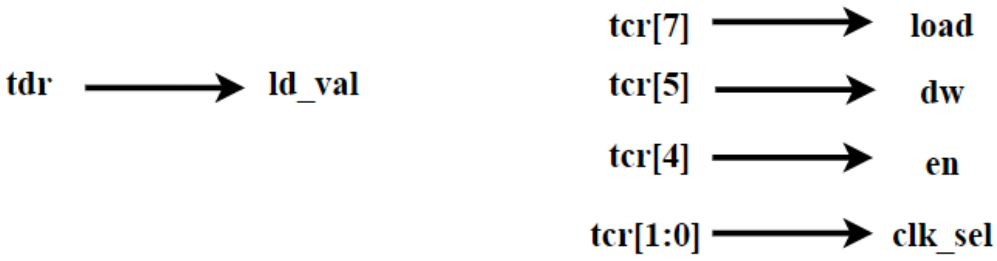
Bảng 3.2: Bảng tín hiệu Timer Counter

STT	Chân tín hiệu	Số bit	Phân loại	Mô tả
Tín hiệu bộ Control logic				
1	presetn	1	Ngõ vào	Tín hiệu khởi động lại thiết kế.
2	pclk	1	Ngõ vào	Tín hiệu xung clock.
3	clk_sel	2	Ngõ ra	Tín hiệu chọn nguồn clock từ bộ chia trước

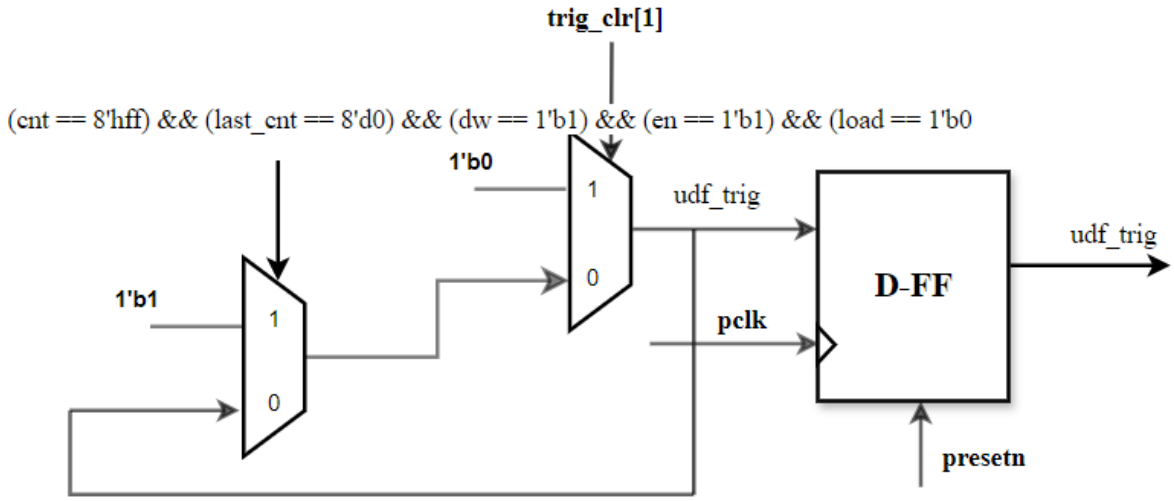
4	en	1	Ngõ ra	Tín hiệu enable cho bộ đếm.
5	ld_en	1	Ngõ ra	Tín hiệu bật chế độ tải giá trị mới vào bộ đếm.
6	ld_val	8	Ngõ ra	Giá trị tải mới khi ld_en được kích hoạt.
7	dw	1	Ngõ ra	Tín hiệu điều khiển hướng đếm (đếm lên hoặc đếm xuống).
8	cnt	8	Ngõ vào	Giá trị đếm của bộ đếm.
9	last_cnt	8	Ngõ vào	Giá trị đếm cuối của bộ đếm
10	udf_trig	8	Ngõ ra	Cờ kích hoạt báo trạng thái tràn dưới
11	ovf_trig	8	Ngõ ra	Cờ kích hoạt báo trạng thái tràn trên
12	tdr	8	Ngõ vào	Thanh ghi Timer Data Register
13	tcr	8	Ngõ vào	Thanh ghi Timer Control Register
14	trig_clr	2	Ngõ vào	Cờ kích hoạt xóa bộ đếm

Khối này nhằm mục đích xử lý các điều kiện của thanh ghi tdr và tcr để thực hiệngửi tín hiệu điều khiển đến khối timer counter. Đồng thời điều khiển các trạng thái của cờ báo tràn trên (overflow), cờ báo tràn dưới (underflow) và nhận sự điều khiển từ tín hiệu trig_clr từ khối Register Control.

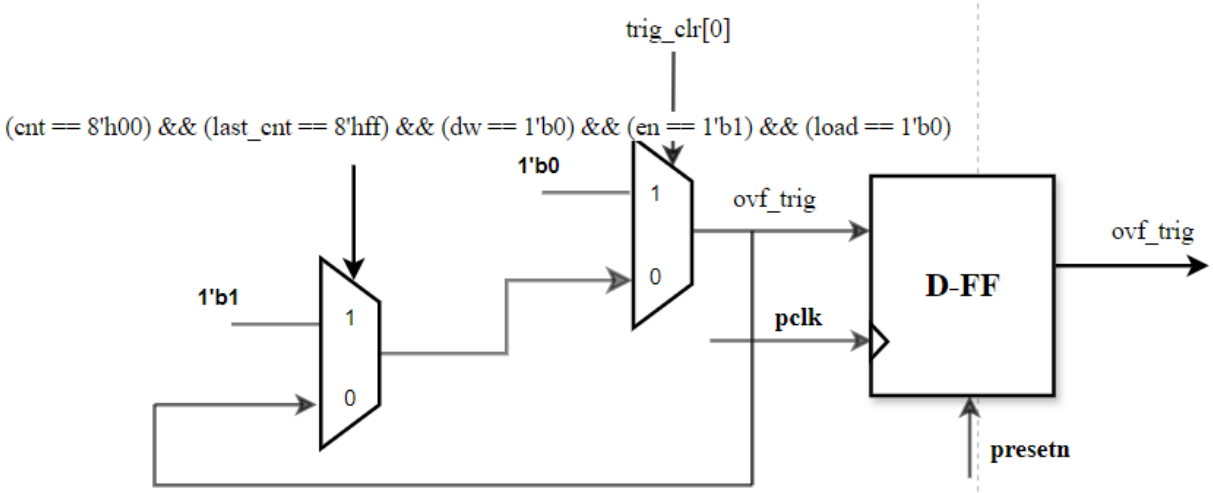
Tiến hành gán các giá trị cần xử lý của các thanh ghi tdr và tcr:



Mạch kiểm tra các điều kiện để thiết lập bit underflow và overflow:



Các điều kiện của tín hiệu trig_clr được điều khiển bởi khối register control nhằm để kiểm tra các điều kiện bật cờ báo tràn trên hay tràn dưới.



Các điều kiện báo tràn trên hay tràn dưới được xem giống như những tín hiệu báo ngắt nhằm đảm bảo mạch được hoạt động chính xác cũng như tiết kiệm được tài nguyên, đạt hiệu quả cao về hiệu suất và năng lượng.

Thiết kế khối Register Control

Khối register control có chứa các tín hiệu giao diện của bus APB và các thanh ghi điều khiển đến các khối control logic. Được sử dụng làm giao diện (interface) để lái điều khiển các hoạt động của hệ thống.

Bảng 3.3: Bảng các tín hiệu điều khiển khối Register Control

STT	Chân tín hiệu	Số bit	Phân loại	Mô tả
Tín hiệu APB BUS				
1	presetn	1	Ngõ vào	Tín hiệu khởi động lại thiết kế.
2	pclk	1	Ngõ vào	Tín hiệu xung clock.
3	penable	1	Ngõ vào	Tín hiệu cho phép hoạt động
4	psel	1	Ngõ vào	Tín hiệu lựa chọn slave
5	pwrite	1	Ngõ vào	Tín hiệu cho phép ghi/ đọc dữ liệu
6	pwdata	8	Ngõ vào	Tín hiệu đầu vào dữ liệu
7	prdata	8	Ngõ ra	Tín hiệu đầu ra dữ liệu
8	pready	1	Ngõ ra	Tín hiệu chờ phản hồi sẵn sàng
9	pslverr	1	Ngõ ra	Tín hiệu phản hồi khi có lỗi địa chỉ
Các tín hiệu nội điều khiển				
10	udf_trig	1	Ngõ vào	Tín hiệu báo ngắt tràn dưới bộ đếm
11	ovf_trig	1	Ngõ vào	Tín hiệu báo ngắt tràn trên bộ đếm
12	tdr	8	Ngõ ra	Thanh ghi dữ liệu
13	tcr	8	Ngõ ra	Thanh ghi điều khiển logic
14	trig_clr	1	Ngõ ra	Tín hiệu xóa cờ khi hết ngắt

Khối này nhằm mục đích giải mã các tín hiệu điều khiển thành các thanh ghi hoạt động nhằm đảm bảo tối ưu tài nguyên sử dụng cũng như dễ dàng trong việc quản lý bởi các khối xử lý trung tâm (CPU). Các thanh ghi được cấu hình sử dụng trong khối:

Thanh ghi Timer Data Register (TDR)

Bit name	F/V	Description
TDR[7:0]	**R/W**	The register contains the data used for updating the value of counter

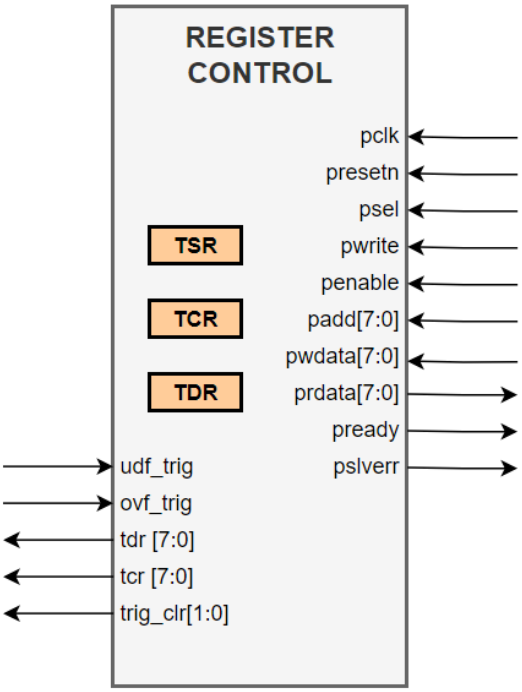
Thanh ghi Timer Control Register (TCR): Chứa các vị trí để điều khiển các khối

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Load	Reserved	Up/down	En	Reserved	Reserved	Cks1	Cks0

Bit name	F/V	Description
Load[7]	**R/W**	Manual load data from TDR to TCNT when itactive High. 1: load data to TCNT 0: Normal operation.
6	**Reserved**	Reserved
Up/Dw[5]	**R/W**	Control counter up or counter down 0: counter up 1: counter down
En[4]	**R/W**	0 : disable 1: enable
3:2	**Reserved**	Reserved
cks[1:0]	**R/W**	Select internal clocks for circuit 00 : T*2 01 : T*4 10 : T*8 11 : T*16

Thanh ghi Timer Status Register (TSR): Chứa các trạng thái báo ngắt tràn trên, tràn dưới của bộ đếm timer.

Bit name	R/W	Description
7:2	**R**	Reserved
s_tmr_udf [1]	**R/W***	Timer counter underflow when counter 8'h00 down to8'hff: This bit is only set by hardware, clear by software
s_tmr_odf [0]	**R/W***	Timer counter overflow when counter 8'hFF to 8'h00:This bit is only set by hardware, clear by software



Hình trên mô tả khối điều khiển thanh ghi của một bộ Timer 8 bit. Khối này thực hiện việc giải mã tín hiệu điều khiển và quản lý các thanh ghi bên trong Timer. Dưới đây là giải thích chi tiết về cách hoạt động của từng thành phần trong hình:

Hoạt động của khối điều khiển thanh ghi

1. Điều khiển truy cập thanh ghi:

- Khi **`psel`** được kích hoạt và **`penable`** được kích hoạt, khối điều khiển thanh ghi sẽ cho phép truy cập các thanh ghi bên trong Timer.

- Nếu **`pwrite`** được kích hoạt, dữ liệu từ **`pwdata[7:0]`** sẽ được ghi vào thanh ghi được chỉ định bởi **`paddr[7:0]`.**

- Nếu **`pwrite`** không được kích hoạt, dữ liệu từ thanh ghi được chỉ định bởi **`paddr[7:0]`** sẽ được đọc ra qua **`prdata[7:0]`**

2. Quản lý các thanh ghi:

- **`TSR`** lưu trữ trạng thái của Timer, bao gồm thông tin về các sự kiện như overflow hoặc underflow.

- **`TCR`** điều khiển hoạt động của Timer, như bắt đầu, dừng, hoặc thiết lập chế độ hoạt động.

- **`TDR`** chứa giá trị hiện tại của Timer, được sử dụng để đếm hoặc tạo xung nhịp.

3. Xử lý các tín hiệu kích hoạt:

- Khi **`udf_trig`** hoặc **`ovf_trig`** được kích hoạt, khối điều khiển sẽ cập nhật trạng thái trong **`TSR`** và có thể tạo ra các ngắt hoặc tín hiệu báo hiệu cho các thành phần khác của hệ thống.

- Tín hiệu **`trig_clr[1:0]`** được sử dụng để xóa các tín hiệu kích hoạt khi cần thiết.

4. Tín hiệu sẵn sàng và lỗi:

- **`pready`** được sử dụng để báo hiệu rằng khối điều khiển đã hoàn thành một thao tác truy cập thanh ghi.

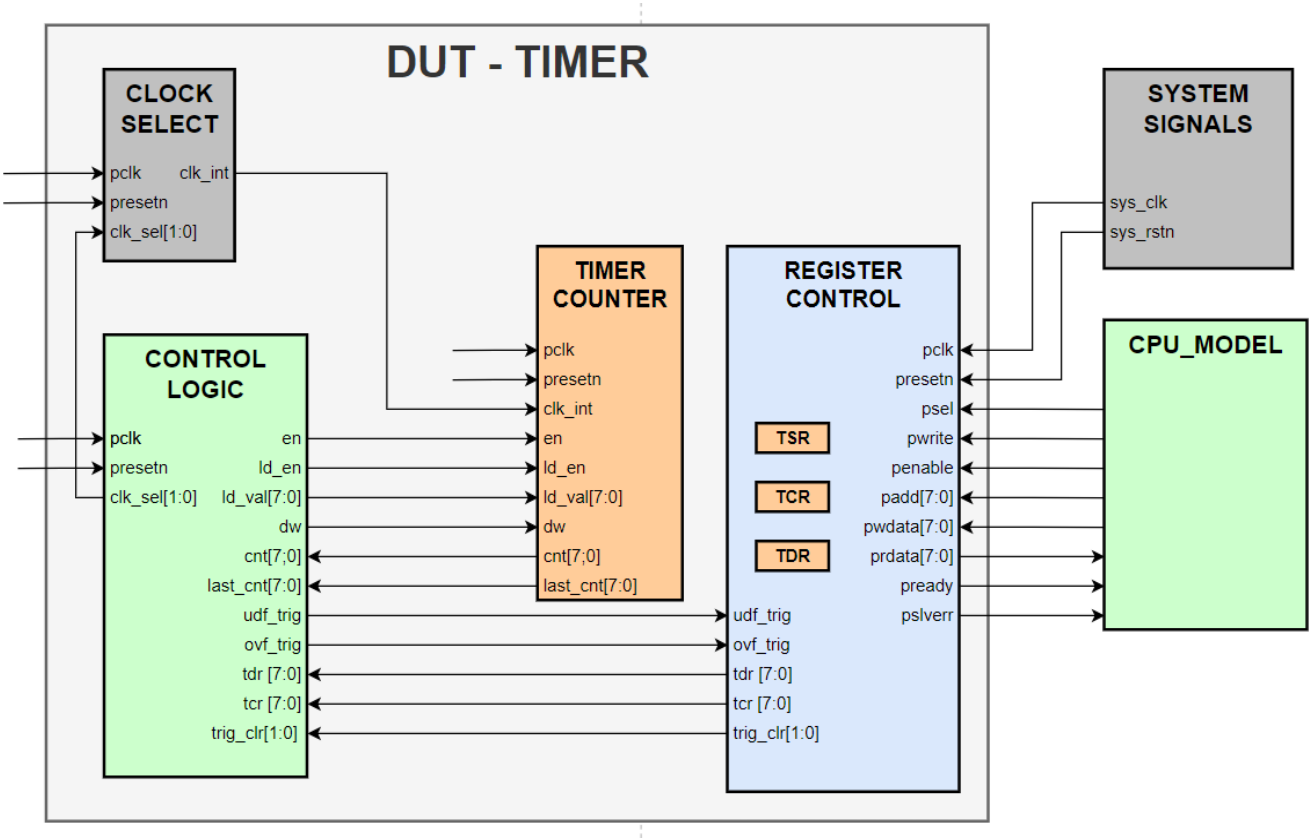
- **`pslverr`** báo hiệu nếu có lỗi xảy ra trong quá trình truy cập.

Khối điều khiển thanh ghi trong hình trên chịu trách nhiệm giải mã và xử lý các tín hiệu điều khiển để truy cập và quản lý các thanh ghi của bộ Timer 8 bit. Nó điều khiển việc ghi và đọc dữ liệu, quản lý trạng thái và điều khiển của Timer, và xử lý các tín hiệu kích hoạt khi xảy ra các sự kiện đặc biệt như overflow và underflow.

XÁC MINH THIẾT KẾ

KẾT QUẢ TỔNG HỢP SƠ ĐỒ KHỐI

Hình 7.1 Sơ đồ xây dựng môi trường kiểm tra



Kết quả tổng hợp sơ đồ khối để tiến hành xác minh (Verification) cho thiết kế:

PHƯƠNG PHÁP THỰC HIỆN MÔ PHỎNG

Xây dựng Mô - đun giả lập

Tiến hành xây dựng các mô – đun System Signals và Cpu Model để làm khối điều khiển chủ giả lập cho hệ thống.

Khối System Signals đóng vai trò là một mô hình hành vi được sử dụng riêng cho việc xác minh, kiểm thử thiết kế, trong khối mô phỏng lại các quá trình điều khiển hành vi cho hệ thống xung clock, khối reset cho toàn bộ hệ thống.

Khối Cpu Model đóng vai trò là một mô hình hành vi mô tả các đặc trưng đọc/ ghi, gửi địa chỉ, các tín hiệu phản hồi từ phía Slave của khối APB Master.

Khối mô tả testbench (tb.v) được xây dựng với cờ fail_flag để khi gọi với các trường hợp kiểm tra sai thì cờ này được bật và cho ra kết quả .

```
    task get_result();
        if(fail_flag) begin
            $display("================================");
            $display("============ FAILED ============");
            $display("================================");
        end
        else begin
            $display("================================");
            $display("============ PASSED ============");
            $display("================================");
        end
    endtask
task test_end();
        begin
            #50;
            get_result();
            $finish();
        end
    endtask
```

Kết quả mô phỏng

Các chương trình kiểm tra mô phỏng thiết kế chi tiết chức năng khối APB – Timer được đánh giá thông qua các testcase được mô tả ở bảng dưới đây:

Tên file test case	Mô tả
Phần 1: Các test case kiểm tra tín hiệu xung clock	
t10_clk_test.v	Kiểm tra phần tạo xung clock từ clock hệ thống
t11_clk_test_rt.v	Kiểm tra phần tạo xung clock từ clock hệ thống với giá trị thời gian thực (realtime)
Phần 2: Các test case kiểm tra thanh ghi	
t20_tdr_rw.v	Kiểm tra thanh ghi giá trị đọc ghi của thanh ghi TDR
t21_tcr_rw.v	Kiểm tra thanh ghi giá trị đọc ghi của thanh ghi TCR

t22_tsr_rw.v	Kiểm tra thanh ghi giá trị đọc ghi của thanh ghi TSR
t23_all_rw.v	Kiểm tra thanh ghi giá trị đọc ghi toàn bộ thanh ghi được cấu hình.
Phần 3: Các test case kiểm tra quá trình đếm lên	
t30_cnt_up_clk2.v	Kiểm tra đếm lên với cấu hình clock * 2
t31_cnt_up_clk4.v	Kiểm tra đếm lên với cấu hình clock * 4
t32_cnt_up_clk8.v	Kiểm tra đếm lên với cấu hình clock * 8
t33_cnt_up_clk16.v	Kiểm tra đếm lên với cấu hình clock * 16
t34_cnt_up_rst.v	Kiểm tra đếm lên với cấu hình có sử dụng reset
t35_cnt_up_ovf.v	Kiểm tra đếm lên với cấu hình đếm đến khi tràn và kiểm tra giá trị đạt được sau khi tràn có thỏa mãn yêu cầu không.
t36_cnt_up_ld.v	Kiểm tra đếm lên với cấu hình sử dụng bit load của thanh ghi TCR để đếm giá trị vào
t37_cnt_up_pause.v	Kiểm tra đếm lên sử dụng bit cấu hình pause trong thanh ghi TCR để dừng giá trị trong thời gian đếm trước khi có tràn
Phần 4: Các Test case kiểm tra quá trình đếm xuống	
t30_cnt_dw_clk2.v	Kiểm tra đếm xuống với cấu hình clock * 2
t31_cnt_dw_clk4.v	Kiểm tra đếm xuống với cấu hình clock * 4
t32_cnt_ dw _clk8.v	Kiểm tra đếm xuống với cấu hình clock * 8
t33_cnt_ dw _clk16.v	Kiểm tra đếm xuống với cấu hình clock * 16
t34_cnt_ dw _rst.v	Kiểm tra đếm xuống với cấu hình có sử dụng reset
t35_cnt_ dw _ovf.v	Kiểm tra đếm xuống với cấu hình đếm đến khi tràn và kiểm tra giá trị đạt được sau khi tràn có thỏa mãn yêu cầu không.
t36_cnt_ dw _ld.v	Kiểm tra đếm xuống với cấu hình sử dụng bit load của thanh ghi TCR để đếm giá trị vào
t37_cnt_ dw _pause.v	Kiểm tra đếm xuống sử dụng bit cấu hình pause trong thanh ghi TCR để dừng giá trị trong thời gian đếm trước khi có tràn

Vì sử dụng môi trường linux dựa trên máy chủ windows nên việc kiểm tra tự động thông qua môi trường linux này cần sử dụng các đoạn script để kết nối giữa các môi trường.

Ở hình dưới đây mô tả cách xây dựng các thư mục trong thư viện thông qua Linux để tiến hành một cách tự động

```
drwxrwxrwx 1 ducanh ducanh 512 Mar  4 11:14 ./
drwxrwxrwx 1 ducanh ducanh 512 Mar  4 11:12 ../
drwxrwxrwx 1 ducanh ducanh 512 Mar  9 15:55 rtl/
drwxrwxrwx 1 ducanh ducanh 512 May  9 19:41 sim/
drwxrwxrwx 1 ducanh ducanh 512 Nov 17  2023 tb/
drwxrwxrwx 1 ducanh ducanh 512 Mar  7 11:18 test/
drwxrwxrwx 1 ducanh ducanh 512 Nov 17  2023 vip/
```

```
drwxrwxrwx 1 ducanh ducanh   512 May 27 14:56 ./
drwxrwxrwx 1 ducanh ducanh   512 Mar  4 11:14 ../
-rwxrwxrwx 1 ducanh ducanh   274 Mar  7 11:12 .log*
-rwxrwxrwx 1 ducanh ducanh    39 Nov  4  2023 compile.f*
-rwxrwxrwx 1 ducanh ducanh   295 Nov 24  2023 define.v*
-rwxrwxrwx 1 ducanh ducanh   280 Nov 22  2023 define.v.bak*
-rwxrwxrwx 1 ducanh ducanh   497 Apr 30 09:15 makefile*
-rwxrwxrwx 1 ducanh ducanh   109 Nov  4  2023 rtl.f*
-rwxrwxrwx 1 ducanh ducanh   548 Apr 30 08:25 setup*
-rwxrwxrwx 1 ducanh ducanh    57 Nov  4  2023 testbench.f*
-rwxrwxrwx 1 ducanh ducanh   650 Apr 30 08:06 testcase.f*
-rwxrwxrwx 1 ducanh ducanh  1929 Nov  4  2023 timer_prj.cr.mti*
-rwxrwxrwx 1 ducanh ducanh 89311 Nov  4  2023 timer_prj.mpf*
```

Trong thư mục ***../../sim*** có chứa các file để tạo tự động

Các file compile.f; rtl.f; testbench.f; testcase.f lần lượt chứa các file phù hợp để tạo một cách tự động.

Để liên kết giữa tools mô phỏng được sử dụng ở đây là Mentor QuestaSim với môi trường linux thì cần tạo thư mục có tên là **makefile** sử dụng ngôn ngữ biên kịch Cshell để tạo ra môi trường một cách tự động.

Các report tổng hợp kết quả mô phỏng được lưu trự thành các file ".log*" mỗi lần ta muốn lấy kết quả để đọc ta có thể vô thư mục này để kiểm tra.

ĐÁNH GIÁ KẾT QUẢ

Kết quả test case kiểm tra xung clock

Thư mục **t10_clk_test.v**	Thư mục **t11_clk_test_rt.v**

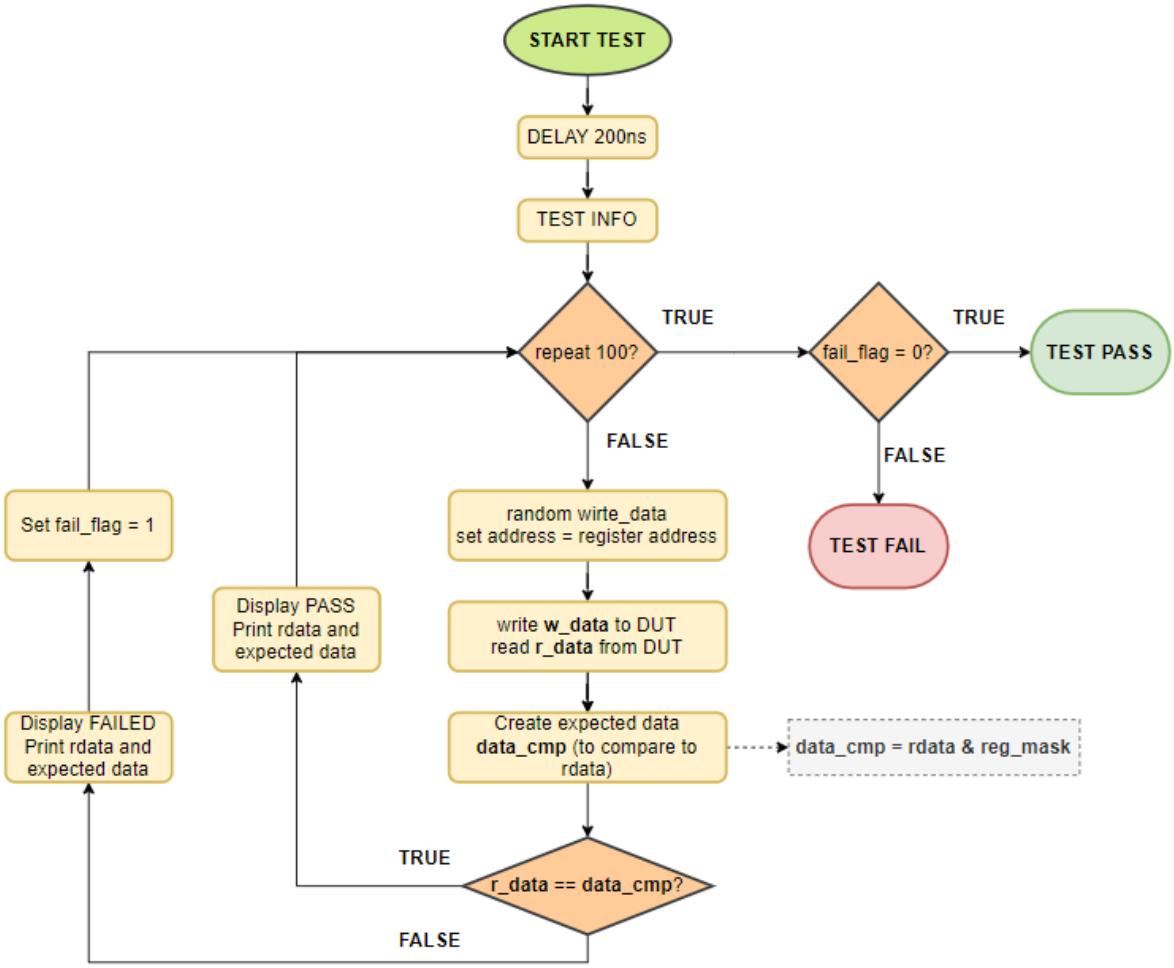
```
# Loading work.control_logic(fast)
# Loading work.register_control(fast)
# Loading work.cpu_model(fast)
# log -r /*
# run -all
# At 20000, Assert reset signal
# At 40000, De-assert reset signal
# ================================================
# ============== CLOCK TEST BEGIN ================
# ================================================
#
# ------------------------------------------------------
#
# At time 200 Clock Clk_2 is testing
# At 201 Start writing wdata = 8'h0 to address = 8'h1
# At 281 Write Transfer has been finished
# At 340 first posedge of clk_2 detected
# At 1140 last posedge of clk_2 detected
# At time 8280 select clk_2, period 2 cycles, expected 2 cycles --PASS--
# ------------------------------------------------------
#
# At time 8280 Clock Clk_4 is testing
# At 8301 Start writing wdata = 8'h1 to address = 8'h1
# At 8381 Write Transfer has been finished
# At 8500 first posedge of clk_4 detected
# At 10100 last posedge of clk_4 detected
# At time 16380 select clk_4, period 4 cycles, expected 4 cycles --PASS--
# ------------------------------------------------------
#
# At time 16380 Clock Clk_8 is testing
# At 16401 Start writing wdata = 8'h2 to address = 8'h1
# At 16481 Write Transfer has been finished
# At 16660 first posedge of clk_8 detected
# At 19860 last posedge of clk_8 detected
# At time 24480 select clk_8, period 8 cycles, expected 8 cycles --PASS--
# ------------------------------------------------------
#
# At time 24480 Clock Clk_16 is testing
# At 24501 Start writing wdata = 8'h3 to address = 8'h1
# At 24581 Write Transfer has been finished
# At 24980 first posedge of clk_16 detected
# At 31380 last posedge of clk_16 detected
# At time 32580 select clk_16, period 16 cycles, expected 16 cycles --PASS--
# ==============================
# ============ PASSED ============
# ==============================
# ** Note: $finish    : ./../test/t10_clk_test.v(31)
#    Time: 32630 ns  Iteration: 0  Instance: /t10_clk_test
# End time: 15:49:40 on May 27,2024, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
mv t10_clk_test.log ./log
cp -rf vsim.wlf wave/t10_clk_test.wlf
ln -sf ./log/t10_clk_test.log sim.log
```

```
# At 20000, Assert reset signal
# At 40000, De-assert reset signal
# ================================================
# ============== CLOCK TEST BEGIN ================
# ================================================
#
# Not any specified clock is testing
# ------------------------------------------------------
#
# At time 200 Clock Clk_x is testing
# At 201 Start writing wdata = 8'h0X to address = 8'h1
# At 281 Write Transfer has been finished
# At time 780 clk_x is not working --FAIL--
# ------------------------------------------------------
#
# At time 780 Clock Clk_2 is testing
# At 801 Start writing wdata = 8'h0 to address = 8'h1
# At 881 Write Transfer has been finished
# At 1020 first posedge of clk_2 detected
# At 1820 last posedge of clk_2 detected
# At time 2180 select clk_2, period 40 ns, expected 40 ns --PASS--
# ------------------------------------------------------
#
# At time 2180 Clock Clk_4 is testing
# At 2201 Start writing wdata = 8'h1 to address = 8'h1
# At 2281 Write Transfer has been finished
# At 2500 first posedge of clk_4 detected
# At 4100 last posedge of clk_4 detected
# At time 4420 select clk_4, period 80 ns, expected 80 ns --PASS--
# ------------------------------------------------------
#
# At time 4420 Clock Clk_8 is testing
# At 4441 Start writing wdata = 8'h2 to address = 8'h1
# At 4521 Write Transfer has been finished
# At 4820 first posedge of clk_8 detected
# At 8020 last posedge of clk_8 detected
# At time 8340 select clk_8, period 160 ns, expected 160 ns --PASS--
# ------------------------------------------------------
#
# At time 8340 Clock Clk_16 is testing
# At 8361 Start writing wdata = 8'h3 to address = 8'h1
# At 8441 Write Transfer has been finished
# At 8980 first posedge of clk_16 detected
# At 15380 last posedge of clk_16 detected
# At time 15620 select clk_16, period 320 ns, expected 320 ns --PASS--
# ==============================
# ============ PASSED ============
# ==============================
# ** Note: $finish    : ./../tb/tb.v(62)
#    Time: 15670 ns  Iteration: 0  Instance: /t11_clk_test_rt
# End time: 16:21:12 on May 27,2024, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
mv t11_clk_test_rt.log ./log
cp -rf vsim.wlf wave/t11_clk_test_rt.wlf
ln -sf ./log/t11_clk_test_rt.log sim.log
```

Nhận xét: Đạt kết đầu ra kết quả 100%

Kết quả các test case kiểm tra thanh ghi

Xây dựng lưu đồ ý tưởng cho các test case:



Bằng cách kiểm tra ở 2 phía đầu nhận data và đầu gửi data, tiến hành gán giá trị địa chỉ thanh ghi cần kiểm tra. Sau đó, lấy giá trị thanh ghi có tên là bit mask để cùng kiểm tra với giá trị data đầu vào được test random lấy đó là giá trị data thảo mãn. Tiến hành lấy giá trị data thỏa mãn so sánh với giá trị data thu được nếu bằng nhau thì thông báo quá trình test thành công. Ngược lại, bật cờ báo test không hoàn thành và in ra kết quả báo FAILED ra màn hình.

Thư mục **t20_tdr_rw.v**

```
# At 18801 Start writing wdata = 8'h7e to address = 8'h0
# At 18881 Write Transfer has been finished
# At 18901 Start reading data at  address = 8'h0
# At 18981 Read Transfer has been finished
# At time 18981, wdata = 8'h7e, rdata = 8'h7e, exp_data = 8'h7e ---PASS---
#
# TEST No.95
# At 19001 Start writing wdata = 8'h6d to address = 8'h0
# At 19081 Write Transfer has been finished
# At 19101 Start reading data at  address = 8'h0
# At 19181 Read Transfer has been finished
# At time 19181, wdata = 8'h6d, rdata = 8'h6d, exp_data = 8'h6d ---PASS---
#
# TEST No.96
# At 19201 Start writing wdata = 8'h39 to address = 8'h0
# At 19281 Write Transfer has been finished
# At 19301 Start reading data at  address = 8'h0
# At 19381 Read Transfer has been finished
# At time 19381, wdata = 8'h39, rdata = 8'h39, exp_data = 8'h39 ---PASS---
#
# TEST No.97
# At 19401 Start writing wdata = 8'h1f to address = 8'h0
# At 19481 Write Transfer has been finished
# At 19501 Start reading data at  address = 8'h0
# At 19581 Read Transfer has been finished
# At time 19581, wdata = 8'h1f, rdata = 8'h1f, exp_data = 8'h1f ---PASS---
#
# TEST No.98
# At 19601 Start writing wdata = 8'hd3 to address = 8'h0
# At 19681 Write Transfer has been finished
# At 19701 Start reading data at  address = 8'h0
# At 19781 Read Transfer has been finished
# At time 19781, wdata = 8'hd3, rdata = 8'hd3, exp_data = 8'hd3 ---PASS---
#
# TEST No.99
# At 19801 Start writing wdata = 8'h85 to address = 8'h0
# At 19881 Write Transfer has been finished
# At 19901 Start reading data at  address = 8'h0
# At 19981 Read Transfer has been finished
# At time 19981, wdata = 8'h85, rdata = 8'h85, exp_data = 8'h85 ---PASS---
#
# TEST No.100
# At 20001 Start writing wdata = 8'h78 to address = 8'h0
# At 20081 Write Transfer has been finished
# At 20101 Start reading data at  address = 8'h0
# At 20181 Read Transfer has been finished
# At time 20181, wdata = 8'h78, rdata = 8'h78, exp_data = 8'h78 ---PASS---
# ================================
# ============ PASSED ============
# ================================
# ** Note: $finish    : ./../test/t20_tdr_rw.v(45)
#    Time: 20231 ns  Iteration: 0  Instance: /t20_tdr_rw
# End time: 16:58:54 on May 27,2024, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
mv t20_tdr_rw.log ./log
cp -rf vsim.wlf wave/t20_tdr_rw.wlf
ln -sf ./log/t20_tdr_rw.log sim.log
```

Thư mục **t21_tcr_rw.v**

```
# At 18801 Start writing wdata = 8'h7e to address = 8'h1
# At 18881 Write Transfer has been finished
# At 18901 Start reading data at  address = 8'h1
# At 18981 Read Transfer has been finished
# At time 18981, wdata = 8'h7e, rdata = 8'h32, exp_data = 8'h32 ---PASS---
#
# TEST No.95
# At 19001 Start writing wdata = 8'h6d to address = 8'h1
# At 19081 Write Transfer has been finished
# At 19101 Start reading data at  address = 8'h1
# At 19181 Read Transfer has been finished
# At time 19181, wdata = 8'h6d, rdata = 8'h21, exp_data = 8'h21 ---PASS---
#
# TEST No.96
# At 19201 Start writing wdata = 8'h39 to address = 8'h1
# At 19281 Write Transfer has been finished
# At 19301 Start reading data at  address = 8'h1
# At 19381 Read Transfer has been finished
# At time 19381, wdata = 8'h39, rdata = 8'h31, exp_data = 8'h31 ---PASS---
#
# TEST No.97
# At 19401 Start writing wdata = 8'h1f to address = 8'h1
# At 19481 Write Transfer has been finished
# At 19501 Start reading data at  address = 8'h1
# At 19581 Read Transfer has been finished
# At time 19581, wdata = 8'h1f, rdata = 8'h13, exp_data = 8'h13 ---PASS---
#
# TEST No.98
# At 19601 Start writing wdata = 8'hd3 to address = 8'h1
# At 19681 Write Transfer has been finished
# At 19701 Start reading data at  address = 8'h1
# At 19781 Read Transfer has been finished
# At time 19781, wdata = 8'hd3, rdata = 8'h93, exp_data = 8'h93 ---PASS---
#
# TEST No.99
# At 19801 Start writing wdata = 8'h85 to address = 8'h1
# At 19881 Write Transfer has been finished
# At 19901 Start reading data at  address = 8'h1
# At 19981 Read Transfer has been finished
# At time 19981, wdata = 8'h85, rdata = 8'h81, exp_data = 8'h81 ---PASS---
#
# TEST No.100
# At 20001 Start writing wdata = 8'h78 to address = 8'h1
# At 20081 Write Transfer has been finished
# At 20101 Start reading data at  address = 8'h1
# At 20181 Read Transfer has been finished
# At time 20181, wdata = 8'h78, rdata = 8'h30, exp_data = 8'h30 ---PASS---
# ================================
# ============ PASSED ============
# ================================
# ** Note: $finish    : ./../test/t21_tcr_rw.v(41)
#    Time: 20231 ns  Iteration: 0  Instance: /t21_tcr_rw
# End time: 17:08:06 on May 27,2024, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
mv t21_tcr_rw.log ./log
cp -rf vsim.wlf wave/t21_tcr_rw.wlf
ln -sf ./log/t21_tcr_rw.log sim.log
```

Thư mục **t22_tsr_rw.v**

```
#
# TEST No.95
# At 19001 Start writing wdata = 8'h1 to address = 8'h2
# At 19081 Write Transfer has been finished
# At 19101 Start reading data at  address = 8'h2
# At 19181 Read Transfer has been finished
# At time 19181, wdata = 8'h1, rdata = 8'h0, ---PASS---
#
# TEST No.96
# At 19201 Start writing wdata = 8'h1 to address = 8'h2
# At 19281 Write Transfer has been finished
# At 19301 Start reading data at  address = 8'h2
# At 19381 Read Transfer has been finished
# At time 19381, wdata = 8'h1, rdata = 8'h0, ---PASS---
#
# TEST No.97
# At 19401 Start writing wdata = 8'h3 to address = 8'h2
# At 19481 Write Transfer has been finished
# At 19501 Start reading data at  address = 8'h2
# At 19581 Read Transfer has been finished
# At time 19581, wdata = 8'h3, rdata = 8'h0, ---PASS---
#
# TEST No.98
# At 19601 Start writing wdata = 8'h3 to address = 8'h2
# At 19681 Write Transfer has been finished
# At 19701 Start reading data at  address = 8'h2
# At 19781 Read Transfer has been finished
# At time 19781, wdata = 8'h3, rdata = 8'h0, ---PASS---
#
# TEST No.99
# At 19801 Start writing wdata = 8'h1 to address = 8'h2
# At 19881 Write Transfer has been finished
# At 19901 Start reading data at  address = 8'h2
# At 19981 Read Transfer has been finished
# At time 19981, wdata = 8'h1, rdata = 8'h0, ---PASS---
#
# TEST No.100
# At 20001 Start writing wdata = 8'h0 to address = 8'h2
# At 20081 Write Transfer has been finished
# At 20101 Start reading data at  address = 8'h2
# At 20181 Read Transfer has been finished
# At time 20181, wdata = 8'h0, rdata = 8'h0, ---PASS---
# ================================
# ============ PASSED ============
# ================================
# ** Note: $finish    : ./../test/t22_tsr_rw.v(46)
#    Time: 20231 ns  Iteration: 0  Instance: /t22_tsr_rw
# End time: 17:16:26 on May 27,2024, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
mv t22_tsr_rw.log ./log
cp -rf vsim.wlf wave/t22_tsr_rw.wlf
ln -sf ./log/t22_tsr_rw.log sim.log
```

Thư mục **t23_all_rw.v**

```
# TEST No.88
# INVALID ADDRESS
#
# TEST No.89
# INVALID ADDRESS
#
# TEST No.90
# INVALID ADDRESS
#
# TEST No.91
# INVALID ADDRESS
#
# TEST No.92
# At 6201 Start writing wdata = 8'h37 to address = 8'h1
# At 6281 Write Transfer has been finished
# At 6301 Start reading data at  address = 8'h1
# At 6381 Read Transfer has been finished
# At time 6381, wdata = 8'h37, rdata = 8'h33, exp_data = 8'h33 ---PASS---
#
# TEST No.93
# INVALID ADDRESS
#
# TEST No.94
# INVALID ADDRESS
#
# TEST No.95
# INVALID ADDRESS
#
# TEST No.96
# INVALID ADDRESS
#
# TEST No.97
# INVALID ADDRESS
#
# TEST No.98
# INVALID ADDRESS
#
# TEST No.99
# At 6401 Start writing wdata = 8'h61 to address = 8'h2
# At 6481 Write Transfer has been finished
# At 6501 Start reading data at  address = 8'h2
# At 6581 Read Transfer has been finished
# At time 6581, wdata = 8'h97, rdata = 8'h0, ---PASS---
#
# TEST No.100
# INVALID ADDRESS
# ================================
# ============ PASSED ============
# ================================
# ** Note: $finish    : ./../test/t23_all_rw.v(86)
#    Time: 6631 ns  Iteration: 0  Instance: /t23_all_rw
# End time: 17:22:46 on May 27,2024, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
mv t23_all_rw.log ./log
cp -rf vsim.wlf wave/t23_all_rw.wlf
ln -sf ./log/t23_all_rw.log sim.log
```

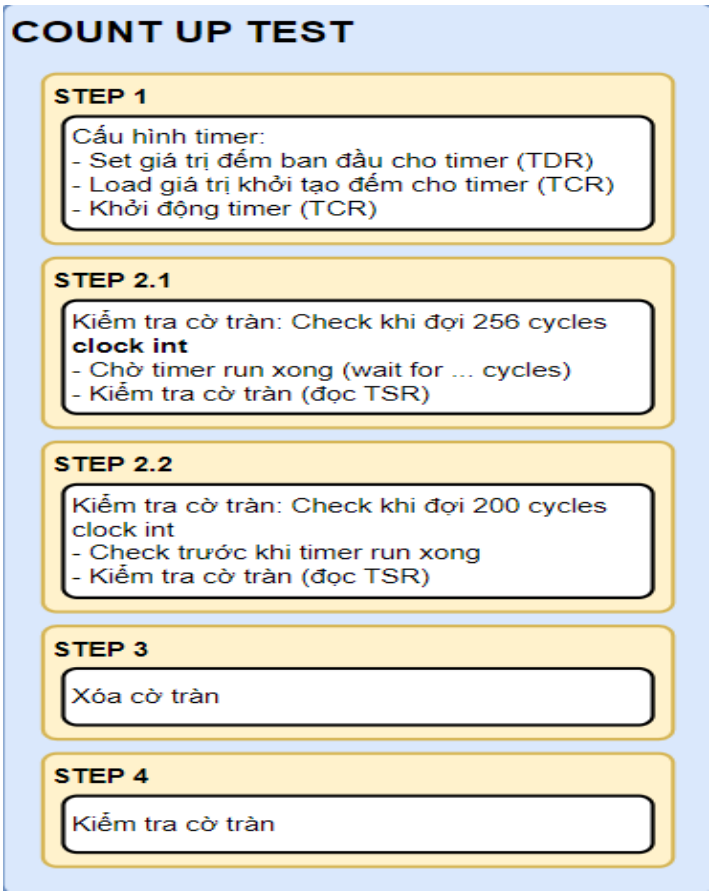
Nhận xét: Các test case kiểm tra đều chính xác 100% yêu cầu

Kết quả các test case kiểm tra đếm lên

Xây dựng ý tưởng cho các trường hợp kiểm tra đếm lên:

Các bước thực hiện:

- Tiến hành khởi tạo bằng cách set giá trị đếm cho thanh ghi đếm TDR
- Bật bit load của thanh ghi TCR để khởi tạo đếm.
- Kiểm tra cờ và kiểm tra trước khi chạy xong
- Kiểm tra cờ có tràn không ở thanh ghi TSR
- Xóa cờ tràn và kiểm tra cờ báo trạng thái đạt đúng hay sai.



t30_cnt_up_clk2.v	t31_cnt_up_clk4.v

```
# At 20000, Assert reset signal
# At 40000, De-assert reset signal
# ================================================
# ============== COUNT UP TEST CLOCK2==============
# ================================================
# ------------------------------------------------
# STEP1: TIMER CONFIGRUATION
# At time 200, write TCR to start timer
# At 201 Start writing wdata = 8'h10 to address = 8'h1
# At 281 Write Transfer has been finished
#
# ------------------------------------------------
# STEP2: CHECK OVERFLOW FLAG
#
# At time 281, waiting for ovf
# At time 9080, after 220 clk_cnt, read TSR
# At 9101 Start reading data at  address = 8'h2
# At 9181 Read Transfer has been finished
# At time 9181, TSR = 8'h0, NOT OVERFLOW --PASS--
#
# At time 10520, after 256 clk_cnt, read TSR (STEP 2.2)
# At 10541 Start reading data at  address = 8'h2
# At 10621 Read Transfer has been finished
# At time 10621, TSR = 8'h1, OVERFLOW --PASS--
#
# ------------------------------------------------
# STEP 3: CLEAR TSR
# At time 10621, clear TSR
# At 10641 Start writing wdata = 8'h0 to address = 8'h2
# At 10721 Write Transfer has been finished
#
# ------------------------------------------------
# STEP 4: CLEAR TSR
# At time 10721, read TSR
# At 10741 Start reading data at  address = 8'h2
# At 10821 Read Transfer has been finished
# At time 10821, TSR = 8'h0
# BIT OVERFLOW CLEARED --PASS--
# ============================
# ============ PASSED ============
# ============================
# ** Note: $finish    : ./../tb/tb.v(62)
#    Time: 10871 ns  Iteration: 0  Instance: /t30_cnt_up_clk2
# End time: 17:46:07 on May 27,2024, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
mv t30_cnt_up_clk2.log ./log
cp -rf vsim.wlf wave/t30_cnt_up_clk2.wlf
ln -sf ./log/t30_cnt_up_clk2.log sim.log
```

t32_cnt_up_clk8.v

```
# At 20000, Assert reset signal
# At 40000, De-assert reset signal
# ================================================
# ============== COUNT UP TEST CLOCK4==============
# ================================================
# ------------------------------------------------
# STEP1: TIMER CONFIGRUATION
# At time 200, write TCR to start timer
# At 201 Start writing wdata = 8'h11 to address = 8'h1
# At 281 Write Transfer has been finished
#
# ------------------------------------------------
# STEP2: CHECK OVERFLOW FLAG
#
# At time 281, waiting for ovf
# At time 17880, after 220 clk_cnt, read TSR
# At 17901 Start reading data at  address = 8'h2
# At 17981 Read Transfer has been finished
# At time 17981, TSR = 8'h0, NOT OVERFLOW --PASS--
#
# At time 20760, after 256 clk_cnt, read TSR (STEP 2.2)
# At 20781 Start reading data at  address = 8'h2
# At 20861 Read Transfer has been finished
# At time 20861, TSR = 8'h1, OVERFLOW --PASS--
#
# ------------------------------------------------
# STEP 3: CLEAR TSR
# At time 20861, clear TSR
# At 20881 Start writing wdata = 8'h0 to address = 8'h2
# At 20961 Write Transfer has been finished
#
# ------------------------------------------------
# STEP 4: CLEAR TSR
# At time 20961, read TSR
# At 20981 Start reading data at  address = 8'h2
# At 21061 Read Transfer has been finished
# At time 21061, TSR = 8'h0
# BIT OVERFLOW CLEARED --PASS--
# ============================
# ============ PASSED ============
# ============================
# ** Note: $finish    : ./../tb/tb.v(62)
#    Time: 21111 ns  Iteration: 0  Instance: /t31_cnt_up_clk4
# End time: 17:46:54 on May 27,2024, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
mv t31_cnt_up_clk4.log ./log
cp -rf vsim.wlf wave/t31_cnt_up_clk4.wlf
ln -sf ./log/t31_cnt_up_clk4.log sim.log
```

t33_cnt_up_clk16.v

```
# At 20000, Assert reset signal
# At 40000, De-assert reset signal
# ================================================
# ============== COUNT UP TEST CLOCK8==============
# ================================================
# ------------------------------------------------
# STEP1: TIMER CONFIGRUATION
# At time 200, write TCR to start timer
# At 201 Start writing wdata = 8'h12 to address = 8'h1
# At 281 Write Transfer has been finished
#
# ------------------------------------------------
# STEP2: CHECK OVERFLOW FLAG
#
# At time 281, waiting for ovf
# At time 35480, after 220 clk_cnt, read TSR
# At 35501 Start reading data at  address = 8'h2
# At 35581 Read Transfer has been finished
# At time 35581, TSR = 8'h0, NOT OVERFLOW --PASS--
#
# At time 41240, after 256 clk_cnt, read TSR (STEP 2.2)
# At 41261 Start reading data at  address = 8'h2
# At 41341 Read Transfer has been finished
# At time 41341, TSR = 8'h1, OVERFLOW --PASS--
#
# ------------------------------------------------
# STEP 3: CLEAR TSR
# At time 41341, clear TSR
# At 41361 Start writing wdata = 8'h0 to address = 8'h2
# At 41441 Write Transfer has been finished
#
# ------------------------------------------------
# STEP 4: CLEAR TSR
# At time 41441, read TSR
# At 41461 Start reading data at  address = 8'h2
# At 41541 Read Transfer has been finished
# At time 41541, TSR = 8'h0
# BIT OVERFLOW CLEARED --PASS--
# ============================
# ============ PASSED ============
# ============================
# ** Note: $finish    : ./../tb/tb.v(62)
#    Time: 41591 ns  Iteration: 0  Instance: /t32_cnt_up_clk8
# End time: 17:47:46 on May 27,2024, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
mv t32_cnt_up_clk8.log ./log
cp -rf vsim.wlf wave/t32_cnt_up_clk8.wlf
ln -sf ./log/t32_cnt_up_clk8.log sim.log
```

t34_cnt_up_rst.v

```
# At 20000, Assert reset signal
# At 40000, De-assert reset signal
# ================================================
# ============== COUNT UP TEST CLOCK16==============
# ================================================
# ------------------------------------------------
# STEP1: TIMER CONFIGRUATION
# At time 200, write TCR to start timer
# At 201 Start writing wdata = 8'h13 to address = 8'h1
# At 281 Write Transfer has been finished
#
# ------------------------------------------------
# STEP2: CHECK OVERFLOW FLAG
#
# At time 281, waiting for ovf
# At time 70680, after 220 clk_cnt, read TSR
# At 70701 Start reading data at  address = 8'h2
# At 70781 Read Transfer has been finished
# At time 70781, TSR = 8'h0, NOT OVERFLOW --PASS--
#
# At time 82200, after 256 clk_cnt, read TSR (STEP 2.2)
# At 82221 Start reading data at  address = 8'h2
# At 82301 Read Transfer has been finished
# At time 82301, TSR = 8'h1, OVERFLOW --PASS--
#
# ------------------------------------------------
# STEP 3: CLEAR TSR
# At time 82301, clear TSR
# At 82321 Start writing wdata = 8'h0 to address = 8'h2
# At 82401 Write Transfer has been finished
#
# ------------------------------------------------
# STEP 4: CLEAR TSR
# At time 82401, read TSR
# At 82421 Start reading data at  address = 8'h2
# At 82501 Read Transfer has been finished
# At time 82501, TSR = 8'h0
# BIT OVERFLOW CLEARED --PASS--
# ============================
# ============ PASSED ============
# ============================
# ** Note: $finish    : ./../tb/tb.v(62)
#    Time: 82551 ns  Iteration: 0  Instance: /t33_cnt_up_clk16
# End time: 17:48:48 on May 27,2024, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
mv t33_cnt_up_clk16.log ./log
cp -rf vsim.wlf wave/t33_cnt_up_clk16.wlf
ln -sf ./log/t33_cnt_up_clk16.log sim.log
```

t35_cnt_up_ovf.v

```
# At 40000, De-assert reset signal
# ================================================
# ============== COUNT UP WITH RESET ==============
# ================================================
# ------------------------------------------------
# STEP1: TIMER CONFIGRUATION
# At time 200, write TCR to start timer
# At 201 Start writing wdata = 8'h10 to address = 8'h1
# At 281 Write Transfer has been finished
#
# ------------------------------------------------
# STEP2: CHECK OVERFLOW FLAG
# At time 9080, after 220 clk_cnt, read TSR
# At 9101 Start reading data at  address = 8'h2
# At 9181 Read Transfer has been finished
# At time 9181, TSR = 8'h0, NOT OVERFLOW --PASS--
#
# ------------------------------------------------
# STEP 3: RESET TIMER
# At time 9181, assert reset: reset value = 0
# At time 9381, de-assert reset: reset value = 1
#
# --------------------RESET DONE--------------------
#
# ------------------------------------------------
# STEP 4: START TIMERR AGAIN
# At 9401 Start writing wdata = 8'h10 to address = 8'h1
# At 9481 Write Transfer has been finished
#
# ------------------------------------------------
# STEP 5: WAIT FOR OVF
#
# ------------------------------------------------
# STEP 6: CHECK OVF
# At time 19720, read TSR
# At 19741 Start reading data at  address = 8'h2
# At 19821 Read Transfer has been finished
# At time 19821, TSR = 8'h1, OVERFLOW --PASS--
#
# ------------------------------------------------
# STEP 7: CLEAR TSR
# At time 19821, clear TSR
# At 19841 Start writing wdata = 8'h0 to address = 8'h2
# At 19921 Write Transfer has been finished
#
# ------------------------------------------------
# STEP 8: READ TSR AND CHECK
# At time 19921, read TSR
# At 19941 Start reading data at  address = 8'h2
# At 20021 Read Transfer has been finished
# At time 20021, TSR = 8'h0
# BIT OVERFLOW CLEARED --PASS--
# ================================
# ============ PASSED ============
# ================================
```

t36_cnt_up_ld.v

```
# ------------------------------------------------
# STEP 4: CLEAR TSR
# At time 523481, read TSR
# At 523501 Start reading data at  address = 8'h2
# At 523581 Read Transfer has been finished
# At time 523581, TSR = 8'h0
# BIT OVERFLOW CLEARED --PASS--
#
#
# ..................................................
# .......Test No.20........
# ..................................................
# ------------------------------------------------
# STEP1: TIMER CONFIGRUATION
# At 523601 Start writing wdata = 8'h0 to address = 8'h0
# At 523681 Write Transfer has been finished
# At 523701 Start writing wdata = 8'h80 to address = 8'h1
# At 523781 Write Transfer has been finished
# At time 523781, write TCR to start timer
# At 523801 Start writing wdata = 8'h13 to address = 8'h1
# At 523881 Write Transfer has been finished
#  At time 523881, timer run with clk( 16 )
#
# ------------------------------------------------
# STEP2: CHECK OVERFLOW FLAG
#
# At time 523881, waiting for ovf
# At time 594280, after 220 clk_cnt, read TSR
# At 594301 Start reading data at  address = 8'h2
# At 594381 Read Transfer has been finished
# At time 594381, TSR = 8'h0, NOT OVERFLOW --PASS--
#
# At time 605800, after 256 clk_cnt, read TSR (STEP 2.2)
# At 605821 Start reading data at  address = 8'h2
# At 605901 Read Transfer has been finished
# At time 605901, TSR = 8'h1, OVERFLOW --PASS--
#
# ------------------------------------------------
# STEP 3: CLEAR TSR
# At time 605901, clear TSR
# At 605921 Start writing wdata = 8'h0 to address = 8'h2
# At 606001 Write Transfer has been finished
#
# ------------------------------------------------
# STEP 4: CLEAR TSR
# At time 606001, read TSR
# At 606021 Start reading data at  address = 8'h2
# At 606101 Read Transfer has been finished
# At time 606101, TSR = 8'h0
# BIT OVERFLOW CLEARED --PASS--
# ================================
# ============ PASSED ============
# ================================
```

t37_cnt_up_pause.v

```
# At 40000, De-assert reset signal
# ================================================
# =============== COUNT UP WITH LOAD ===============
# ================================================
# ------------------------------------------------
# STEP1: TIMER CONFIGRUATION
# At time 200, Generated init value for Timer Counter
#  At time 200, init_value is smaller than 50 --> add 50
#  At time 200, wait_cycles_1 = 150
#  At time 200, wait_cycles_2 = 170
#
# At 200 Starting counting to TDR at 86
# At 201 Start writing wdata = 8'h56 to address = 8'h0
# At 281 Write Transfer has been finished
#
# At 281 Start load count to TCR at 86
# At 301 Start writing wdata = 8'h80 to address = 8'h1
# At 381 Write Transfer has been finished
# At time 381, write TCR to start timer
# At 401 Start writing wdata = 8'h10 to address = 8'h1
# At 481 Write Transfer has been finished
#
# ------------------------------------------------
# STEP2: CHECK OVERFLOW FLAG
#
# At time 481, waiting for ovf
# At time 6480, after 220 clk_cnt, read TSR
# At 6501 Start reading data at  address = 8'h2
# At 6581 Read Transfer has been finished
# At time 6581, TSR = 8'h0, NOT OVERFLOW --PASS--
#
# At time 7280, after 256 clk_cnt, read TSR (STEP 2.2)
# At 7301 Start reading data at  address = 8'h2
# At 7381 Read Transfer has been finished
# At time 7381, TSR = 8'h1, OVERFLOW --PASS--
#
# ------------------------------------------------
# STEP 3: CLEAR TSR
# At time 7381, clear TSR
# At 7401 Start writing wdata = 8'h0 to address = 8'h2
# At 7481 Write Transfer has been finished
#
# ------------------------------------------------
# STEP 4: CLEAR TSR
# At time 7481, read TSR
# At 7501 Start reading data at  address = 8'h2
# At 7581 Read Transfer has been finished
# At time 7581, TSR = 8'h0
# BIT OVERFLOW CLEARED --PASS--
# ================================
# ============ PASSED ============
# ================================
# ** Note: $finish    : ./../tb/tb.v(62)
```

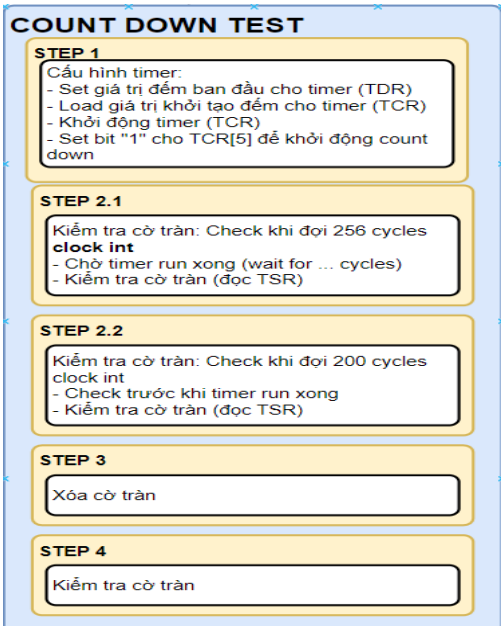
```
# At 20000, Assert reset signal
# At 40000, De-assert reset signal
# ================================================
# ============== COUNT UP WITH PAUSE  ==============
# ================================================
#
# ------------------------------------------------
# STEP1: TIMER CONFIGRUATION
# At time 200, pause_after_n_cycles is 36
# At time 200, write TCR to start timer with clk2
# At 201 Start writing wdata = 8'h10 to address = 8'h1
# At 281 Write Transfer has been finished
# ------------------------------------
# STEP 2: Pause timer after 36 cycles
# At time 281, wait for 36 cycles
# At time 1720, write TCR to pause timer
# At 1741 Start writing wdata = 8'h0 to address = 8'h1
# At 1821 Write Transfer has been finished
# At time 2021, write TCR to start timer again
# At 2041 Start writing wdata = 8'h10 to address = 8'h1
# At 2121 Write Transfer has been finished
# ------------------------------------
# STEP 3: Wait for timer to finish counting
# At time 2121, wait for 220 cycles
# ------------------------------------
# STEP 4: Check TSR
# At time 10920, after 220 clk_int, read TSR
# At 10941 Start reading data at  address = 8'h2
# At 11021 Read Transfer has been finished
# At time 11021, TSR = 8'h1, OVERFLOW --PASS--
#
# ------------------------------------------------
# STEP 5: CLEAR TSR AND DOUBLE CHECK TSR
# At time 11021, clear TSR
# At 11041 Start writing wdata = 8'h0 to address = 8'h2
# At 11121 Write Transfer has been finished
# At time 11121, read TSR
# At 11141 Start reading data at  address = 8'h2
# At 11221 Read Transfer has been finished
# At time 11221, TSR = 8'h0
# BIT OVERFLOW CLEARED --PASS--
# ================================
# ============ PASSED ============
# ================================
# ** Note: $finish    : ./../tb/tb.v(62)
#    Time: 11271 ns  Iteration: 0  Instance: /t38_cnt_up_pause
# End time: 17:55:19 on May 27,2024, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
```

Kết quả các test case kiểm tra đếm xuống

Xây dựng ý tưởng cho các trường hợp kiểm tra đếm xuống:

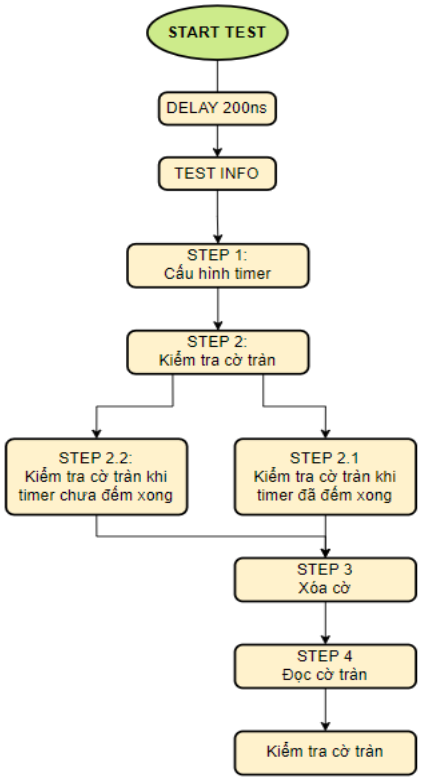
Các bước thực hiện:

- Tiến hành khởi tạo bằng cách set giá trị đếm cho thanh ghi đếm TDR
- Bật bit load của thanh ghi TCR để khởi tạo đếm.
- Kiểm tra cờ và kiểm tra trước khi chạy xong
- Kiểm tra cờ có tràn không ở thanh ghi TSR
- Xóa cờ tràn và kiểm tra cờ báo trạng thái đạt đúng hay sai.



Với trường hợp test case đếm xuống có trạng thái dừng (pause) ta cũng có:

- Tiến hành cấu hình cho timer và bắt đầu kiểm tra cờ tràn,
- Tiến hành kiểm tra và đem ra kết quả khi chưa đếm xong và khi đã đếm xong đem ra liệu có đạt được kết quả mong muốn (expected data).
- Tiến hành xóa cờ đọc cờ và kiểm tra lại cờ tràn.



t40_cnt_dw_clk2.v

```
# At 20000, Assert reset signal
# At 40000, De-assert reset signal
# ================================================
# ============== COUNT DW TEST CLOCK2==============
# ================================================
# ------------------------------------------------
# STEP1: TIMER CONFIGRUATION
# At 201 Start writing wdata = 8'hff to address = 8'h0
# At 281 Write Transfer has been finished
# At 301 Start writing wdata = 8'h80 to address = 8'h1
# At 381 Write Transfer has been finished
# At time 381, write TCR to start timer
# At 401 Start writing wdata = 8'h30 to address = 8'h1
# At 481 Write Transfer has been finished
#
# ------------------------------------------------
# STEP2: CHECK UNDERFLOW FLAG
#
# At time 481, waiting for udf
# At time 9280, after 220 clk_cnt, read TSR
# At 9301 Start reading data at  address = 8'h2
# At 9381 Read Transfer has been finished
# At time 9381, TSR = 8'h0, NOT UNDERFLOW --PASS--
#
# At time 10720, after 256 clk_cnt, read TSR (STEP 2.2)
# At 10741 Start reading data at  address = 8'h2
# At 10821 Read Transfer has been finished
# At time 10821, TSR = 8'h2, UNDERFLOW --PASS--
#
# ------------------------------------------------
# STEP 3: CLEAR TSR
# At time 10821, clear TSR
# At 10841 Start writing wdata = 8'h0 to address = 8'h2
# At 10921 Write Transfer has been finished
#
# ------------------------------------------------
# STEP 4: CLEAR TSR
# At time 10921, read TSR
# At 10941 Start reading data at  address = 8'h2
# At 11021 Read Transfer has been finished
# At time 11021, TSR = 8'h0
# BIT UNDERFLOW  CLEARED --PASS--
# ==============================
# ============ PASSED ===========
# ==============================
```

t41_cnt_dw_clk4.v

```
# At 20000, Assert reset signal
# At 40000, De-assert reset signal
# ================================================
# ============== COUNT DW TEST CLOCK4==============
# ================================================
# ------------------------------------------------
# STEP1: TIMER CONFIGRUATION
# At 201 Start writing wdata = 8'hff to address = 8'h0
# At 281 Write Transfer has been finished
# At 301 Start writing wdata = 8'h80 to address = 8'h1
# At 381 Write Transfer has been finished
# At time 381, write TCR to start timer
# At 401 Start writing wdata = 8'h31 to address = 8'h1
# At 481 Write Transfer has been finished
#
# ------------------------------------------------
# STEP2: CHECK UNDERFLOW FLAG
#
# At time 481, waiting for udf
# At time 18080, after 220 clk_cnt, read TSR
# At 18101 Start reading data at  address = 8'h2
# At 18181 Read Transfer has been finished
# At time 18181, TSR = 8'h0, NOT UNDERFLOW --PASS--
#
# At time 20960, after 256 clk_cnt, read TSR (STEP 2.2)
# At 20981 Start reading data at  address = 8'h2
# At 21061 Read Transfer has been finished
# At time 21061, TSR = 8'h2, UNDERFLOW --PASS--
#
# ------------------------------------------------
# STEP 3: CLEAR TSR
# At time 21061, clear TSR
# At 21081 Start writing wdata = 8'h0 to address = 8'h2
# At 21161 Write Transfer has been finished
#
# ------------------------------------------------
# STEP 4: CLEAR TSR
# At time 21161, read TSR
# At 21181 Start reading data at  address = 8'h2
# At 21261 Read Transfer has been finished
# At time 21261, TSR = 8'h0
# BIT UNDERFLOW  CLEARED --PASS--
# ==============================
# ============ PASSED ===========
# ==============================
```

t42_cnt_dw_clk8.v

```
# At 20000, Assert reset signal
# At 40000, De-assert reset signal
# ================================================
# ============== COUNT DW TEST CLOCK8==============
# ================================================
# ------------------------------------------------
# STEP1: TIMER CONFIGRUATION
# At 201 Start writing wdata = 8'hff to address = 8'h0
# At 281 Write Transfer has been finished
# At 301 Start writing wdata = 8'h80 to address = 8'h1
# At 381 Write Transfer has been finished
# At time 381, write TCR to start timer
# At 401 Start writing wdata = 8'h32 to address = 8'h1
# At 481 Write Transfer has been finished
#
# ------------------------------------------------
# STEP2: CHECK UNDERFLOW FLAG
#
# At time 481, waiting for udf
# At time 35680, after 220 clk_cnt, read TSR
# At 35701 Start reading data at  address = 8'h2
# At 35781 Read Transfer has been finished
# At time 35781, TSR = 8'h0, NOT UNDERFLOW --PASS--
#
# At time 41440, after 256 clk_cnt, read TSR (STEP 2.2)
# At 41461 Start reading data at  address = 8'h2
# At 41541 Read Transfer has been finished
# At time 41541, TSR = 8'h2, UNDERFLOW --PASS--
#
# ------------------------------------------------
# STEP 3: CLEAR TSR
# At time 41541, clear TSR
# At 41561 Start writing wdata = 8'h0 to address = 8'h2
# At 41641 Write Transfer has been finished
#
# ------------------------------------------------
# STEP 4: CLEAR TSR
# At time 41641, read TSR
# At 41661 Start reading data at  address = 8'h2
# At 41741 Read Transfer has been finished
# At time 41741, TSR = 8'h0
# BIT UNDERFLOW  CLEARED --PASS--
# ==============================
# ============ PASSED ===========
# ==============================
```

t43_cnt_dw_clk16.v

```
# At 20000, Assert reset signal
# At 40000, De-assert reset signal
# ================================================
# ==============COUNT DW TEST CLOCK16==============
# ================================================
# ------------------------------------------------
# STEP1: TIMER CONFIGRUATION
# At 201 Start writing wdata = 8'hff to address = 8'h0
# At 281 Write Transfer has been finished
# At 301 Start writing wdata = 8'h80 to address = 8'h1
# At 381 Write Transfer has been finished
# At time 381, write TCR to start timer
# At 401 Start writing wdata = 8'h33 to address = 8'h1
# At 481 Write Transfer has been finished
#
# ------------------------------------------------
# STEP2: CHECK UNDERFLOW FLAG
#
# At time 481, waiting for udf
# At time 70880, after 220 clk_cnt, read TSR
# At 70901 Start reading data at  address = 8'h2
# At 70981 Read Transfer has been finished
# At time 70981, TSR = 8'h0, NOT UNDERFLOW --PASS--
#
# At time 82400, after 256 clk_cnt, read TSR (STEP 2.2)
# At 82421 Start reading data at  address = 8'h2
# At 82501 Read Transfer has been finished
# At time 82501, TSR = 8'h2, UNDERFLOW --PASS--
#
# ------------------------------------------------
# STEP 3: CLEAR TSR
# At time 82501, clear TSR
# At 82521 Start writing wdata = 8'h0 to address = 8'h2
# At 82601 Write Transfer has been finished
#
# ------------------------------------------------
# STEP 4: CLEAR TSR
# At time 82601, read TSR
# At 82621 Start reading data at  address = 8'h2
# At 82701 Read Transfer has been finished
# At time 82701, TSR = 8'h0
# BIT UNDERFLOW  CLEARED --PASS--
# ==============================
# ============ PASSED ===========
# ==============================
```

t44_cnt_dw_rst.v

t45_cnt_dw_ovf.v

```
#
# ------------------------------------------------
# STEP2: CHECK UNDERFLOW FLAG
# At time 9280, after 220 clk_cnt, read TSR
# At 9301 Start reading data at  address = 8'h2
# At 9381 Read Transfer has been finished
# At time 9381, TSR = 8'h0, NOT UNDERFLOW --PASS--
#
# ------------------------------------------------
# STEP 3: RESET TIMER
# At time 9381, assert reset: reset value = 0
# At time 9581, de-assert reset: reset value = 1
#
# ---------------------RESET DONE----------------------
#
# ------------------------------------------------
# STEP 4: START TIMER AGAIN
# At 9601 Start writing wdata = 8'h30 to address = 8'h1
# At 9681 Write Transfer has been finished
#
# ------------------------------------------------
# STEP 5: WAIT FOR UNDERFLOW
#
# ------------------------------------------------
# STEP 6: CHECK UNDERFLOW
# At time 19920, read TSR
# At 19941 Start reading data at  address = 8'h2
# At 20021 Read Transfer has been finished
# At time 20021, TSR = 8'h2, UNDERFLOW --PASS--
#
# ------------------------------------------------
# STEP 7: CLEAR TSR
# At time 20021, clear TSR
# At 20041 Start writing wdata = 8'h0 to address = 8'h2
# At 20121 Write Transfer has been finished
#
# ------------------------------------------------
# STEP 8: READ TSR AND CHECK
# At time 20121, read TSR
# At 20141 Start reading data at  address = 8'h2
# At 20221 Read Transfer has been finished
# At time 20221, TSR = 8'h0
# BIT UNDERFLOW CLEARED --PASS--
# ================================
# ============ PASSED ============
# ================================
# ** Note: $finish    : ./../tb/tb.v(62)
```

```
#
# ................................................
# ...Test No.20...
# ................................................
# ------------------------------------------------
# STEP1: TIMER CONFIGRUATION
# At 523601 Start writing wdata = 8'hff to address = 8'h0
# At 523681 Write Transfer has been finished
# At 523701 Start writing wdata = 8'h80 to address = 8'h1
# At 523781 Write Transfer has been finished
# At time 523781, write TCR to start timer
# At 523801 Start writing wdata = 8'h33 to address = 8'h1
# At 523881 Write Transfer has been finished
#  At time 523881, timer run with clk( 16 )
#
# ------------------------------------------------
# STEP2: CHECK UNDERFLOW FLAG
#
# At time 523881, waiting for udf
# At time 594280, after 220 clk_cnt, read TSR
# At 594301 Start reading data at  address = 8'h2
# At 594381 Read Transfer has been finished
# At time 594381, TSR = 8'h0, NOT UNDERFLOW --PASS--
#
# At time 605800, after 256 clk_cnt, read TSR (STEP 2.2)
# At 605821 Start reading data at  address = 8'h2
# At 605901 Read Transfer has been finished
# At time 605901, TSR = 8'h2, UNDERFLOW --PASS--
#
# ------------------------------------------------
# STEP 3: CLEAR TSR
# At time 605901, clear TSR
# At 605921 Start writing wdata = 8'h0 to address = 8'h2
# At 606001 Write Transfer has been finished
#
# ------------------------------------------------
# STEP 4: CLEAR TSR
# At time 606001, read TSR
# At 606021 Start reading data at  address = 8'h2
# At 606101 Read Transfer has been finished
# At time 606101, TSR = 8'h0
# BIT UNDERFLOW CLEARED --PASS--
# ================================
# ============ PASSED ============
# ================================
```

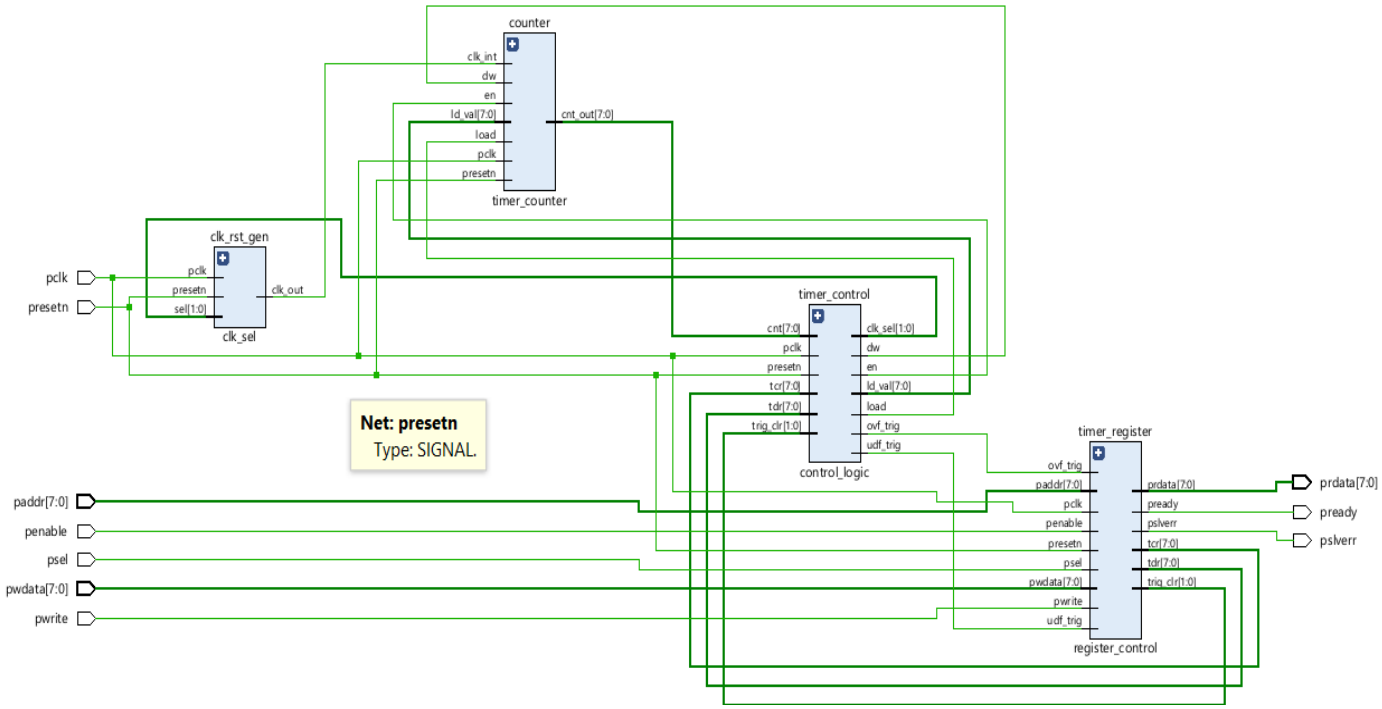
t46_cnt_dw_ld.v

```
# At 20000, Assert reset signal
# At 40000, De-assert reset signal
# ================================================
# ============== COUNT DW WITH LOAD ==============
# ================================================
# ------------------------------------------------
# STEP1: TIMER CONFIGRUATION
# At time 200, Generated init value for Timer Counter
#  At time 200, init_value is smaller than 50 --> add 50
#  At time 200, wait_cycles_1 = 70
#  At time 200, wait_cycles_2 = 170
#
# At 200 Starting counting to TDR at 86
# At 201 Start writing wdata = 8'h56 to address = 8'h0
# At 281 Write Transfer has been finished
#
# At 281 Start load count to TCR at 86
# At 301 Start writing wdata = 8'h80 to address = 8'h1
# At 381 Write Transfer has been finished
# At time 381, write TCR to start timer
# At 401 Start writing wdata = 8'h30 to address = 8'h1
# At 481 Write Transfer has been finished
#
# ------------------------------------------------
# STEP2: CHECK UNDFLOW FLAG
#
# At time 481, waiting for ovf
# At time 3280, after 220 clk_cnt, read TSR
# At 3301 Start reading data at  address = 8'h2
# At 3381 Read Transfer has been finished
# At time 3381, TSR = 8'h0,  NOT UNDFLOW --PASS--
#
# At time 7280, after 256 clk_cnt, read TSR (STEP 2.2)
# At 7301 Start reading data at  address = 8'h2
# At 7381 Read Transfer has been finished
# At time 7381, TSR = 8'h2, UNDFLOW --PASS--
#
# ------------------------------------------------
# STEP 3: CLEAR TSR
# At time 7381, clear TSR
# At 7401 Start writing wdata = 8'h0 to address = 8'h2
# At 7481 Write Transfer has been finished
#
# ------------------------------------------------
# STEP 4: CLEAR TSR
# At time 7481, read TSR
# At 7501 Start reading data at  address = 8'h2
# At 7581 Read Transfer has been finished
# At time 7581, TSR = 8'h0
# BIT UNDERFLOW CLEARED --PASS--
# ================================
# ============ PASSED ============
# ================================
# ** Note: $finish    : ./../tb/tb.v(62)
```

t47_cnt_dw_pause.v

```
# At 20000, Assert reset signal
# At 40000, De-assert reset signal
# ================================================
# ============== COUNT DW WITH PAUSE  ==============
# ================================================
#
# ------------------------------------------------
# STEP1: TIMER CONFIGRUATION
# At time 200, pause_after_n_cycles is 36
# At time 200, write TCR to start timer with clk2
# At 201 Start writing wdata = 8'h30 to address = 8'h1
# At 281 Write Transfer has been finished
# ------------------------------------
# STEP 2: Pause timer after 36 cycles
# At time 281, wait for 36 cycles
# At time 1720, write TCR to pause timer
# At 1741 Start writing wdata = 8'h0 to address = 8'h1
# At 1821 Write Transfer has been finished
# At time 2021, write TCR to start timer again
# At 2041 Start writing wdata = 8'h30 to address = 8'h1
# At 2121 Write Transfer has been finished
# ------------------------------------
# STEP 3: Wait for timer to finish counting
# At time 2121, wait for 220 cycles
# ------------------------------------
# STEP 4: Check TSR
# At time 10920, after 220 clk_int, read TSR
# At 10941 Start reading data at  address = 8'h2
# At 11021 Read Transfer has been finished
# At time 11021, TSR = 8'h2, UNDERFLOW --PASS--
#
# ------------------------------------------------
# STEP 5: CLEAR TSR AND DOUBLE CHECK TSR
# At time 11021, clear TSR
# At 11041 Start writing wdata = 8'h0 to address = 8'h2
# At 11121 Write Transfer has been finished
# At time 11121, read TSR
# At 11141 Start reading data at  address = 8'h2
# At 11221 Read Transfer has been finished
# At time 11221, TSR = 8'h0
# BIT UNDERFLOW CLEARED --PASS--
# ================================
# ============ PASSED ============
# ================================
```

Kết quả sau tổng hợp được:



Kết quả mạch thực thi với các cổng logic: