

**VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF SCIENCE
FACULTY OF ELECTRONICS – TELECOMMUNICATIONS**



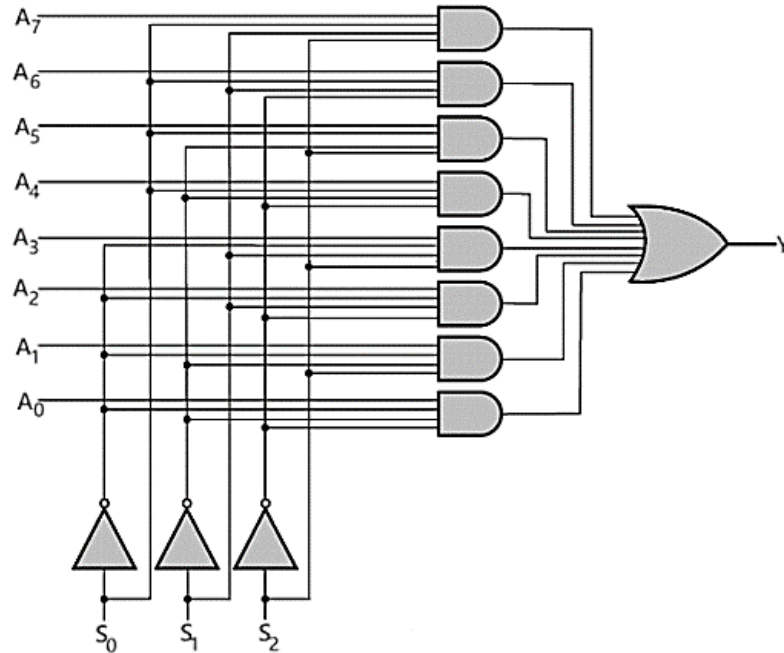
BÁO CÁO THỰC HÀNH LOGIC KHẢ TRÌNH

**GVHD: Mã Khải Minh
SVTH: Đỗ Minh Chương
MSSV: 21207126**

Ho Chi Minh City – April 2024

CÂU 1: Tìm hiểu và thiết kế mạch tổ hợp như hình ở dưới, sử dụng ngôn ngữ Verilog HDL và **lập bảng chân trị** (truth table) phù hợp cho mạch này. Cho biết biết $S_0 \rightarrow S_2$ và $A_0 \rightarrow A_7$ là các ngõ vào, và Y ngõ ra. **Hãy cho biết đây là mạch gì và chức năng của nó?**

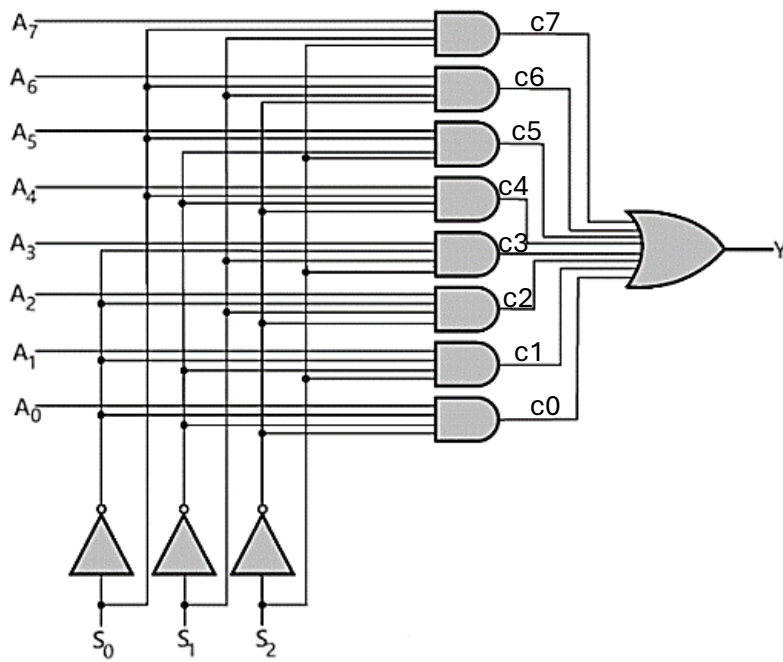
Thực hiện mạch trên board FPGA DE10 với S_0, S_1, S_2 được gán với KEY0, KEY1 và KEY2, $A_0 \rightarrow A_7$ được gán với SW0 \rightarrow SW7, và Y được gán với LED0.



Bảng chân trị:

Select inputs			Output
S_0	S_1	S_2	Y
0	0	0	A_0
0	0	1	A_1
0	1	0	A_2
0	1	1	A_3
1	0	0	A_4
1	0	1	A_5
1	1	0	A_6
1	1	1	A_7

Mạch trên là mạch Multiplexer-MUX 8 sang 1 hay còn gọi là mạch đa hợp 8 sang 1. Chức năng của mạch là chọn một trong số 8 đầu vào để chuyển đến đầu ra dựa trên 3 tín hiệu điều khiển.



Từ mạch bên, ta thiết kế được các biểu thức logic:

$$c0 = A_0 \& \bar{S}_0 \& \bar{S}_1 \& \bar{S}_2 = SW[0] \& \sim KEY[0] \& \sim KEY[1] \& \sim KEY[2]$$

$$c1 = A_1 \& \bar{S}_0 \& \bar{S}_1 \& S_2 = SW[1] \& \sim KEY[0] \& \sim KEY[1] \& KEY[2]$$

$$c2 = A_2 \& \bar{S}_0 \& S_1 \& \bar{S}_2 = SW[2] \& \sim KEY[0] \& KEY[1] \& \sim KEY[2]$$

$$c3 = A_3 \& \bar{S}_0 \& S_1 \& S_2 = SW[3] \& \sim KEY[0] \& KEY[1] \& KEY[2]$$

$$c4 = A_4 \& S_0 \& \bar{S}_1 \& \bar{S}_2 = SW[4] \& KEY[0] \& \sim KEY[1] \& \sim KEY[2]$$

$$c5 = A_5 \& S_0 \& \bar{S}_1 \& S_2 = SW[5] \& KEY[0] \& \sim KEY[1] \& KEY[2]$$

$$c6 = A_6 \& S_0 \& S_1 \& \bar{S}_2 = SW[6] \& KEY[0] \& KEY[1] \& \sim KEY[2]$$

$$c7 = A_7 \& S_0 \& S_1 \& S_2 = SW[7] \& KEY[0] \& KEY[1] \& KEY[2]$$

$$\rightarrow Y = c0 \mid c1 \mid c2 \mid c3 \mid c4 \mid c5 \mid c6 \mid c7$$

Từ sơ đồ mạch và biểu thức logic trên ta dễ dàng thiết kế code cho mạch:

```
module Cau1(
    input [0:2] KEY,      //KEY[] = S[]
    input [7:0] SW,       //SW[] = A[]
    output LED0           //LED0 = Y
);

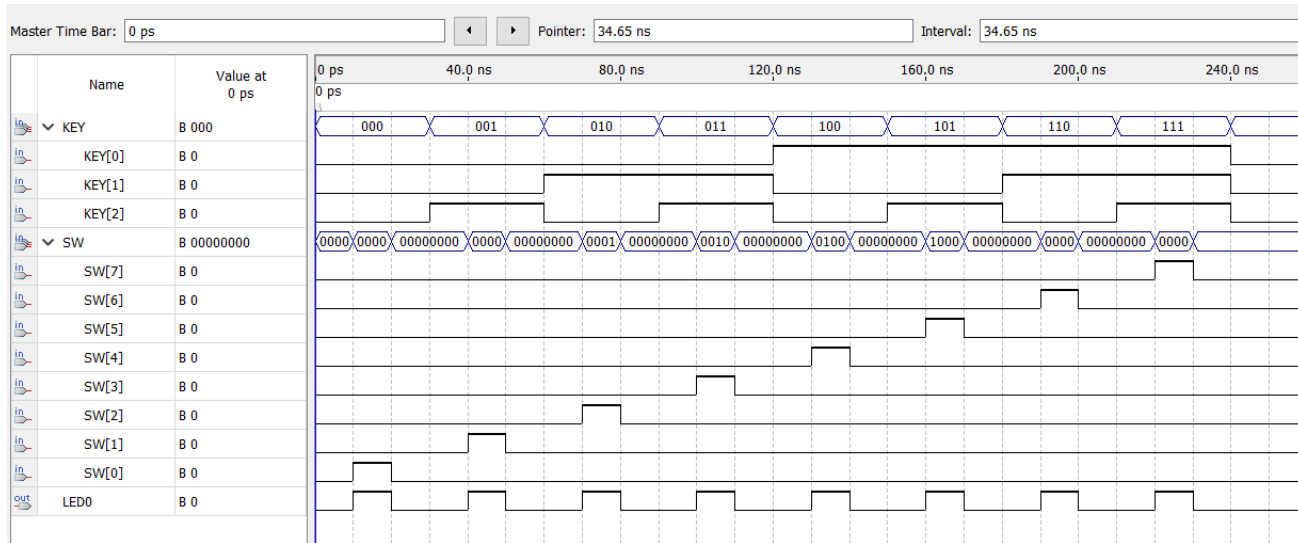
wire c0, c1, c2, c3, c4, c5, c6, c7;

assign c0 = SW[0] & ~KEY[0] & ~KEY[1] & ~KEY[2];
assign c1 = SW[1] & ~KEY[0] & ~KEY[1] & KEY[2];
assign c2 = SW[2] & ~KEY[0] & KEY[1] & ~KEY[2];
assign c3 = SW[3] & ~KEY[0] & KEY[1] & KEY[2];
assign c4 = SW[4] & KEY[0] & ~KEY[1] & ~KEY[2];
assign c5 = SW[5] & KEY[0] & ~KEY[1] & KEY[2];
assign c6 = SW[6] & KEY[0] & KEY[1] & ~KEY[2];
assign c7 = SW[7] & KEY[0] & KEY[1] & KEY[2];

assign LED0 = c0 | c1 | c2 | c3 | c4 | c5 | c6 | c7;

endmodule
```

Mô phỏng waveform của mạch trên:



Khi **KEY[0:2] = 3'b000** thì ngõ ra **LED0 = SW[0]**, tương tự:

KEY[0:2] = 3'b001 thì ngõ ra **LED0 = SW[1]**,

KEY[0:2] = 3'b010 thì ngõ ra **LED0 = SW[2]**,

KEY[0:2] = 3'b011 thì ngõ ra **LED0 = SW[3]**,

KEY[0:2] = 3'b100 thì ngõ ra **LED0 = SW[4]**,

KEY[0:2] = 3'b101 thì ngõ ra **LED0 = SW[5]**,

KEY[0:2] = 3'b110 thì ngõ ra **LED0 = SW[6]**,

KEY[0:2] = 3'b111 thì ngõ ra **LED0 = SW[7]**

CÂU 2: Sử dụng ngôn ngữ Verilog HDL **thiết kế mạch dịch led** sử dụng 10 LED đỏ, và **được điều khiển bởi SW0 và KEY0**. Trong đó:

- LED chạy theo hướng được điều khiển bởi SW0 (tự quy định), khi LED dịch đến cạnh thì quay trở lại từ LED đầu tiên
- LED được dịch với tần số tùy chọn
- Khi nhấn KEY0, led sẽ đứng yên tại chỗ cho đến khi KEY0 được nhấn lần nữa.

Yêu cầu và gợi ý:

- Động tác nhấn KEY0 được tính là nhấn rồi thả, không phải nhấn và giữ
- Có thể thêm tín hiệu reset (tùy chọn/không bắt buộc)

CÁCH THIẾT KẾ











1. Thiết kế mạch dịch led

Dịch led từ trái sang phải: $LEDR \leq \{LEDR[8:0], LEDR[9]\}$

Chuỗi bit LEDR ban đầu sẽ được khởi tạo là 10'b1

$LEDR = 10'b000000001$, sau khi ghép $\{LEDR[8:0], LEDR[9]\}$ ta được: $10'b000000001_0$

Tương tự cho các lần ghép chuỗi bit tiếp theo:

$LEDR \leq \{LEDR[8:0], LEDR[9]\}$										
LEDR \ Clock	0	0	0	0	0	0	0	0	0	1
	0	0	0	0	0	0	0	0	1	0
	0	0	0	0	0	0	0	1	0	0
	0	0	0	0	0	0	1	0	0	0
	0	0	0	0	0	1	0	0	0	0
	0	0	0	0	1	0	0	0	0	0
	0	0	0	1	0	0	0	0	0	0
	0	0	1	0	0	0	0	0	0	0
	0	1	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	1

Dùng phương pháp ghép LED này để khi LEDR dịch tới cạnh thì LEDR sẽ được ghép quay trở lại LEDR đầu tiên, sẽ tối ưu hơn phép dịch trái (<<)

Tương tự cho dịch led từ phải sang trái: $LEDR \leq \{LEDR[0], LEDR[9:1]\}$

2. Thiết kế nút nhấn KEY0, khi nhấn thì led sẽ đứng yên và khi nhấn lần nữa led sẽ chạy lại.

```
reg flag = 1'b1;
always @(negedge KEY0)
begin
    flag = ~flag;
end
```

Vì **KEY0** không khi được nhấn sẽ ở mức **0** nên ta dùng tín hiệu cạnh xuống **negedge**. Khi có tín hiệu cạnh xuống của **KEY0** thì đảo tín hiệu của **flag**, cờ **flag** này sẽ điều khiển việc dừng chạy led khi **flag = 1'b0** và tiếp tục chạy led khi **flag = 1'b1**.

Code sau khi hoàn thiện:

```
module Cau2(
    input CLOCK_50,
    input SW0,
    input KEY0,
    output reg [9:0] LEDR
);
    reg [24:0] counter;
    reg flag = 1'b1;

    always @(posedge CLOCK_50)
        counter <= counter + 1'b1;

    always @(negedge KEY0)
    begin
        flag = ~flag;
    end

    always @(posedge counter[22])
    begin
        if(flag) begin
            if(LEDR == 10'b0)
                LEDR <= 10'b1;
            else if(SW0)
                LEDR <= {LEDR[8:0], LEDR[9]};
            else
                LEDR <= {LEDR[0], LEDR[9:1]};
        end
    end
end
endmodule
```

Giải thích code:

```
module Cau2(  
    input CLOCK_50,  
    input SW0,  
    input KEY0,  
    output reg [9:0] LEDR  
);
```

Đây là các đầu vào cho module **Cau2**, Khai báo các cổng đầu vào và đầu ra của module. CLOCK_50 là tín hiệu clock, SW0 là switch, KEY0 là nút nhấn, và LEDR là các LED.

```
always @(posedge CLOCK_50)  
    counter <= counter + 1'b1;
```

Luôn tăng giá trị của biến đếm counter mỗi khi xảy ra cạnh dương của tín hiệu clock CLOCK_50 để làm bộ chia tần số cho LED chạy.

```
always @(negedge KEY0)  
begin  
    flag = ~flag;  
end
```

Đảo ngược giá trị của biến flag khi xảy ra cạnh âm của tín hiệu từ nút nhấn KEY0.

```
always @(posedge counter[22])  
begin  
    if(flag) begin  
        if(LEDR == 10'b0)  
            LEDR <= 10'b1;  
        else if(SW0)  
            LEDR <= {LEDR[8:0], LEDR[9]};  
        else  
            LEDR <= {LEDR[0], LEDR[9:1]};  
    end  
end
```

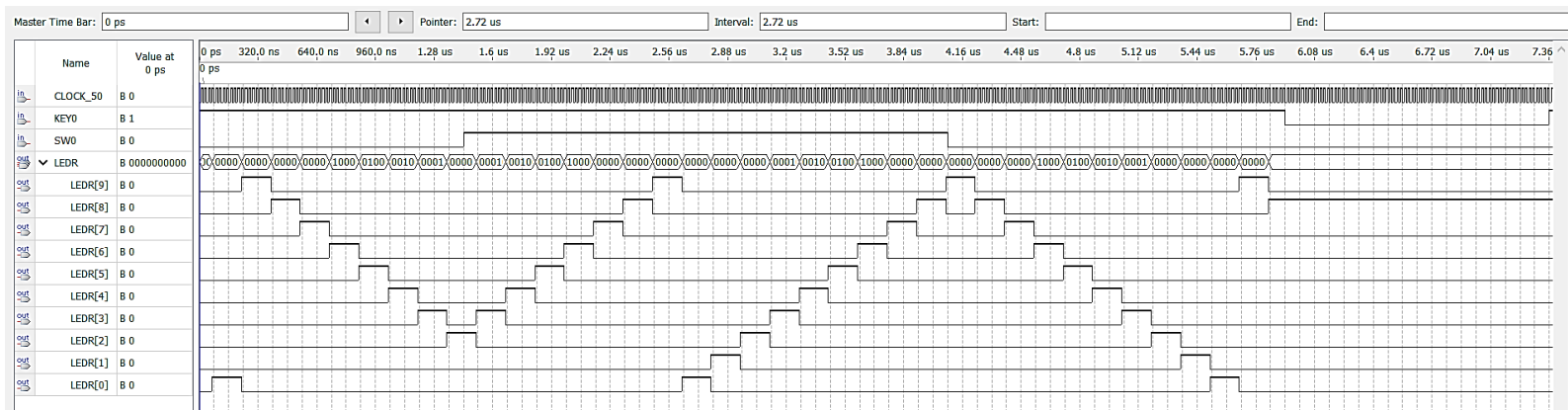
Đoạn code này là một mảng kích hoạt được kích hoạt mỗi khi cạnh dương của bit thứ 22 của bộ đếm counter được nhận. Dòng `always @(posedge counter[22])` xác định rằng block code này sẽ được thực thi mỗi khi có sự kiện cạnh dương trên bit thứ 22 của counter.

Trong block code này, điều kiện `if (flag)` kiểm tra xem biến `flag` có đang ở trạng thái logic 1 hay không. Nếu điều kiện này đúng, nghĩa là `flag` đang ở trạng thái logic 1, block code sẽ được thực thi.

Trong trường hợp block code được thực thi, chương trình sẽ kiểm tra giá trị của biến `LEDR`:

- Nếu **LEDR** bằng **10'b0** (tức là toàn bộ các bit đều bằng 0), chương trình sẽ gán giá trị **10'b1** cho **LEDR**.
- Nếu không, chương trình sẽ kiểm tra giá trị của cờ **SW0**. Nếu **SW0** đang ở trạng thái logic 1, chương trình sẽ thực hiện quá trình *dịch trái 1 bit cho LEDR*.
- Trong trường hợp **SW0** không ở trạng thái logic 1, chương trình sẽ thực hiện quá trình *dịch phải 1 bit cho LEDR*.

Mô phỏng waveform của mạch trên:



Ban đầu $KEY0 = 1$ (tức là chưa nhấn) và $SW0 = 0$ thì `LEDR` đang dịch phải từ bit 9 xuống

Khi $SW0 = 1$ thì `LEDR` bắt đầu dịch trái dịch từ bit 2 lên (như trong hình)

Ở cuối thì `KEY0` được nhấn lúc này tín hiệu `LEDR` được giữ nguyên tại `LEDR[8]`.

CÂU 3: Thiết kế mạch **đồng hồ đếm thời gian gồm phút và giây** hiển thị lên HEX3, HEX2, HEX1, HEX0 và được điều khiển bởi các KEY trên board FPGA DE10, sử dụng ngôn ngữ Verilog HDL. Trong đó:

- KEY0 dùng để reset đồng hồ về “00:00”
- KEY1 dùng để bắt đầu hoặc tạm dừng đếm thời gian (đồng hồ tiếp tục đếm khi KEY1 được nhấn lần nữa)

Yêu cầu và gợi ý:

- Dùng một bộ đếm và thực hiện so sánh để tạo **xung clock chính xác 1Hz** từ clock 50MHz (để tạo ra 1 giây thời gian thực chính xác nhất); bộ đếm có thể được thiết kế riêng thành module riêng và gọi vào thiết kế (instantiate)
- Các chức năng giải mã hiển thị ra HEX được thiết kế thành module riêng

CÁCH THIẾT KẾ

Để thiết kế **đồng hồ đếm thời gian gồm phút và giây** trên chúng ta cần thiết các vấn đề sau:

- Xung clock chính xác 1Hz
- Bộ đếm thời gian gồm 2 biến second và minute
- Bộ decoder led 7 đoạn
- Nút reset KEY[0] và nút bắt đầu hoặc tạm dừng KEY[1]

1. Thiết kế xung clock 1Hz từ xung clock 50MHz

Để tạo ra xung clock 1Hz tức là $T = 1s$ thì chúng ta cần tạo 1 biến đếm counter đếm ở mỗi cạnh lên của xung clock 50MHz đủ 50.000.000 lần thì sẽ tạo được $T =$

$$50.000.000 * \frac{1}{50MHz} = 1s \text{ hay } f = 1/T = 1Hz.$$

Ý tưởng code của phần này:

```
reg [25:0] counter, FREQUENCY;
initial FREQUENCY = 26'd50_000_000;
always @(posedge CLOCK_50)
begin
    if (counter >= FREQUENCY) begin
        counter <= 26'b0;
        // Code bộ đếm thời gian sẽ được đặt ở đây
    end
    else
        counter <= counter + 1'b1;
end
```

2. Thiết kế bộ đếm thời gian second và minute

Ý tưởng code:

```
reg [5:0] second = 0, minute = 0;
if(second == 59) begin
    second <= 0;
    if(minute == 59)
        minute <= 0;
    else
        minute <= minute + 1'b1;
end
else
    second <= second + 1'b1;
```

Như các bộ đếm thời gian khác biến **second** và **minute** luôn ≤ 59 , vì vậy ta tạo điều kiện if khi **second** ≤ 59 thì tăng **second** lên 1 đơn vị cho tới khi **second** đã = 59 ta đặt **second** về 0 lại và tăng **minute** lên 1 đơn vị và chú ý điều kiện của **minute**.

3. Thiết kế bộ giải mã led 7 đoạn hiển thị ra HEX từ 0 -> 9

```
module decoder_7seg(bin, HEX);
    input [3:0] bin;
    output reg [0:6] HEX;
always @(*)
    begin
        HEX = (bin == 4'b0000) ? 7'b0000001 : // 0
              (bin == 4'b0001) ? 7'b1001111 : // 1
              (bin == 4'b0010) ? 7'b0010010 : // 2
              (bin == 4'b0011) ? 7'b0000110 : // 3
              (bin == 4'b0100) ? 7'b1001100 : // 4
              (bin == 4'b0101) ? 7'b0100100 : // 5
              (bin == 4'b0110) ? 7'b0100000 : // 6
              (bin == 4'b0111) ? 7'b0001111 : // 7
              (bin == 4'b1000) ? 7'b0000000 : // 8
              (bin == 4'b1001) ? 7'b0000100 : // 9
              7'b1111111;
    end
endmodule
```

Module decoder_7seg:

- Module này chuyển đổi một số nhị phân mã hóa thập phân (BCD) 4 bit thành đầu ra hiển thị 7 đoạn.
- Có hai cổng đầu vào: **bin** (đầu vào BCD 4 bit) và **HEX** (đầu ra hiển thị 7 đoạn 7 bit).
- Trong khối **always**, lệnh **case** được dùng để ánh xạ các giá trị đầu vào BCD vào các giá trị hiển thị 7 đoạn tương ứng.

4. Thiết kế nút reset KEY[0] và nút bắt đầu hoặc tạm dừng KEY[1]

```
reg [25:0] counter, FREQUENCY;
KEY0 <= KEY[0];
KEY1 <= KEY[1];
if (KEY0 && ~KEY[0])
begin
    second = 0;
    minute = 0;
end
if (KEY1 && ~KEY[1])
begin
    start = ~start;
end
```

- Tạo thêm **KEY0** và **KEY1** để khi thả nút thì chức năng mới hoạt động
- Khi **KEY[0]** không nhấn thì sẽ ở mức 1 vì thế ta để **~KEY[0]** trong điều kiện if vì khi nhấn nó sẽ về mức 0 và khi **KEY[0]** được nhấn thì sẽ reset second, minute về 0. Khi chúng ta chỉ ghi 0 hoặc 1 mà không để định dạng 1'b0 hay 1'b1 thì phần mềm sẽ tự hiểu đó là kiểu số Decimal (Hệ 10).
- Tương tự cho **KEY[1]** và biến start sẽ đảo lại cho mỗi lần nhấn **KEY[1]** để chuyển qua lại giữa 2 chức năng đếm và dừng đếm.

Viết lại code của các phần trên ta được:

```
module Cau3(
    input CLOCK_50,
    input [1:0] KEY,
    output [0:6] HEX0,
    output [0:6] HEX1,
    output [0:6] HEX2,
    output [0:6] HEX3
);

reg [25:0] counter, FREQUENCY;
reg [5:0] second = 0, minute = 0;
reg start, KEY0, KEY1;
initial FREQUENCY = 26'd50_000_000;
```

```

always @(posedge CLOCK_50)
begin
    KEY0 <= KEY[0];
    KEY1 <= KEY[1];

    if (KEY0 && ~KEY[0])
    begin
        second = 0;
        minute = 0;
    end

    if (KEY1 && ~KEY[1])
    begin
        start = ~start;
    end

    if (counter >= FREQUENCY)
    begin
        counter <= 26'b0;
        if (start) begin
            if(second == 59) begin
                second <= 0;
                if(minute == 59)
                    minute <= 0;
                else
                    minute <= minute + 1'b1;

                end
            else
                second <= second + 1'b1;
            end
        end
        else
            counter <= counter + 1'b1;
    end

    decoder_7seg((second % 10), (HEX0));
    decoder_7seg((second / 10), (HEX1));
    decoder_7seg((minute % 10), (HEX2));
    decoder_7seg((minute / 10), (HEX3));

endmodule

```

Giải thích code:

```
module Cau3(  
    input CLOCK_50,  
    input [1:0] KEY,  
    output [0:6] HEX0,  
    output [0:6] HEX1,  
    output [0:6] HEX2,  
    output [0:6] HEX3  
);
```

Đây là các đầu vào cho module **Cau3**, **CLOCK_50** là xung tín hiệu 50MHz của DE10, **KEY[0]** là đầu vào nút reset về 00:00, **KEY[1]** là đầu vào nút stop/count và các biến **HEX0**, **HEX1**, **HEX2**, và **HEX3** là các đầu ra hiển thị 7 đoạn được thiết lập bằng các giá trị từ module **decoder_7seg**.

```
reg [25:0] counter, FREQUENCY;  
reg [5:0] second = 0, minute = 0;  
reg start, KEY0, KEY1;  
initial FREQUENCY = 26'd50_000_000;
```

Các register sử dụng trong module:

- **counter**: để đếm số xung clock.
- **FREQUENCY**: để lưu trữ một giá trị tần số.
- **second**: để lưu trữ giá trị giây (khởi tạo là 0).
- **minute**: để lưu trữ giá trị phút (khởi tạo là 0).
- **start**: 1 bit để điều khiển bắt đầu/dừng đồng hồ.
- **KEY0, KEY1**: bit để lưu trạng thái của hai nút.

```
always @(posedge CLOCK_50)
```

Khối **always** này được kích hoạt bởi cạnh dương của tín hiệu **CLOCK_50**, tức là mỗi khi có một xung đồng hồ từ **CLOCK_50**.

```
KEY0 <= KEY[0];  
KEY1 <= KEY[1];  
if (KEY0 && ~KEY[0])  
begin  
    second = 0;  
    minute = 0;  
end  
if (KEY1 && ~KEY[1])  
begin  
    start = ~start;  
end
```

Đoạn code này kiểm tra xem nút bấm đã được nhấn hay không.

Nếu **KEY0** đã được nhấn và trạng thái trước đó của **KEY[0]** là thấp (**~KEY[0]**), tức là nút này chuyển từ không được nhấn sang được nhấn, thì chương trình thực hiện: *Đặt second và minute về 0 để reset giờ và phút về 0.*

Nếu **KEY1** đã được nhấn và trạng thái trước đó của **KEY[1]** là thấp (**~KEY[1]**), tức là nút này chuyển từ không được nhấn sang được nhấn, thì chương trình thực hiện: *Đảo ngược giá trị của biến start, để bắt đầu hoặc dừng đồng hồ tùy thuộc vào trạng thái hiện tại của nó.*

```
if (counter >= FREQUENCY)
```

Kiểm Tra Điều Kiện Tần Số Đếm:

Dòng này kiểm tra xem giá trị hiện tại của bộ đếm counter có lớn hơn hoặc bằng giá trị tần số **FREQUENCY** không.

```
begin
    counter <= 26'b0;
    if (start) begin
        if(second == 59) begin
            second <= 0;
            if(minute == 59)
                minute <= 0;
            else
                minute <= minute + 1'b1;
        end
        else
            second <= second + 1'b1;
    end
end
```

Nếu bộ đếm counter vượt quá giá trị tần số **FREQUENCY**, tức là đã đến thời điểm đếm mới:

- Bộ đếm counter được reset về 0.

Nếu biến start là logic 1 (đồng hồ đang chạy), thì chương trình tiến hành đếm thời gian:

- Nếu giây second đã đếm đến 59, thì giây được reset về 0.
- Nếu phút minute đã đếm đến 59, thì phút được reset về 0, ngược lại thì giá trị phút tăng thêm 1.
- Nếu giây chưa đến 59, thì giá trị giây tăng thêm 1.

```
counter <= counter + 1'b1;
end
```

Nếu chưa đến thời điểm cần đếm (giá trị counter chưa vượt quá **FREQUENCY**), thì bộ đếm counter vẫn tiếp tục tăng lên.

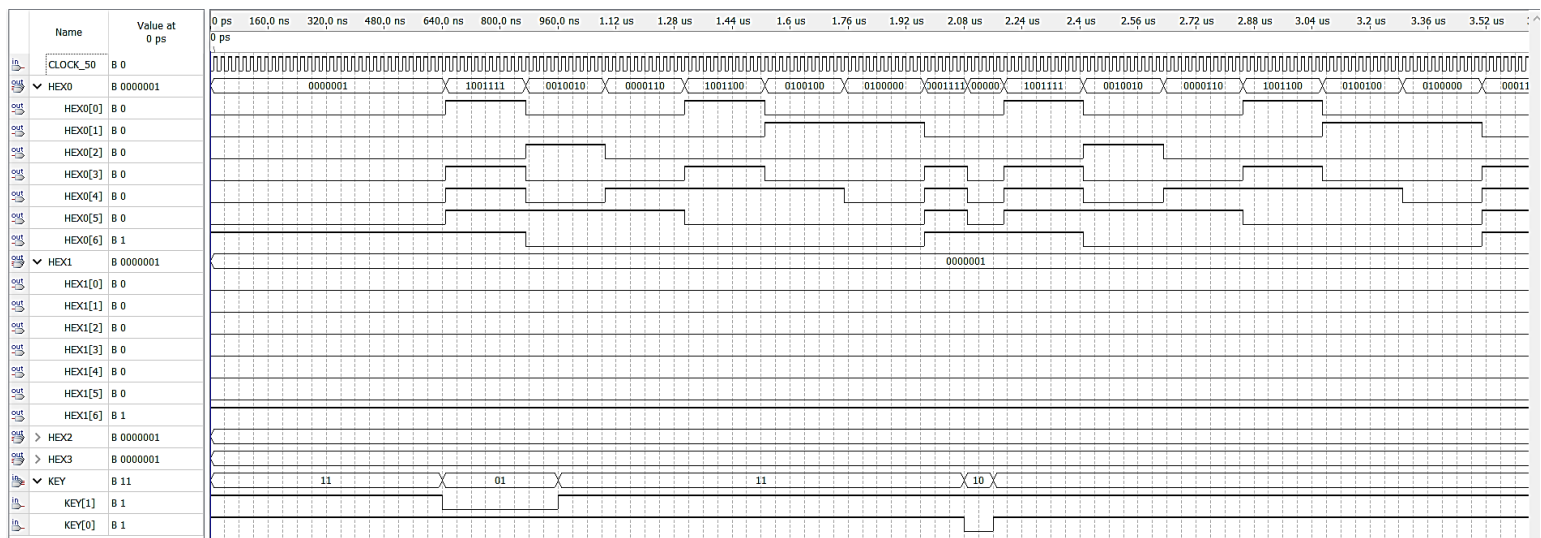
```
decoder_7seg((second % 10), (HEX0));
decoder_7seg((second / 10), (HEX1));
decoder_7seg((minute % 10), (HEX2));
decoder_7seg((minute / 10), (HEX3));
```

Khối này gọi module **decoder_7seg** bốn lần để giải mã các giá trị của second và minute thành các mã hiển thị 7 đoạn tương ứng.

Các giá trị đã giải mã sau đó được gán cho các đầu ra **HEX0, HEX1, HEX2, và HEX3**.

- second % 10: chữ số hàng đơn vị của giây
- second / 10: chữ số hàng chục của giây
- minute % 10: chữ số hàng đơn vị của phút
- minute / 10: chữ số hàng chục của phút

Kết quả mô phỏng waveform và link video mô phỏng trên FPGA



Link mô phỏng:

<https://drive.google.com/drive/folders/1sKRlPb9A1FmA3PE7tdc0IApY64ualpTV>