上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

数字逻辑设计

LAB2 – 时序电路设计

姓名： 邱一航

学号： 520030910155

1. 带置位信号的倒计时模块

1.1 实验电路原理

倒计时模块共有 3 个输入（clk、load、data_load），1 个输出（tc）。

其中 clk、load 为 1 位输入，data_load 为 10 位输入；tc 为 1 位输出。clk 为时钟信号，load 为同步置位信号，data_load 为载入数据，tc 为倒计时结束信号（倒计时中为无效即低电平，倒计时结束为有效即高电平）。当 load 有效（即高电平）时，倒计时模块在时钟上升沿时载入数据 data_load，并在之后 data_load 个时钟周期内倒计时，tc 保持为低电平输出；经过 data_load 个时钟周期后，倒计时结束，tc 变为高电平输出。

本次实验中设计的倒计时模块为上升沿触发。

不同输入情况下，倒计时模块的工作状态如下表。

Inputs			**Outputs & Working Mode**	
clk	**load**	**data_load**	**Mode**	**tc**
↑	0	x	countdown	**0** during countdown **1** when countdown terminates
↑	1	0	countdown is terminated	**1**
↑	1	>0	load **data_load** while start the countdown	**0**

表 1　带置位信号的倒计时模块的工作状态表

特别地，当载入数据 0 时，倒计时并不会启动，因此 tc 输出为 1。

1.2 电路实现与关键代码讨论

笔者的实现方式如下：

对每个时钟上升沿，首先判断 load 是否有效，若有效则载入数据，将模块内的计数器赋值为 data_load；接着判断模块内计时器是否大于 0，若仍然大于 0 则继续倒计时，tc 输出 0，若已经为 0 则将 tc 置为 1。

倒计时模块的 Verilog 代码如下：

```
/* ================================[countdown.v]================================ */
`timescale 1ns/100ps
module countdown(
    input wire clk,
    input wire load,
    input wire[9:0] data_load,
    output reg tc);
```

```
reg[9:0] counter;

always @(posedge clk)
begin
    if (load==1'b1)
    // if LOAD signal is valid
        counter = data_load;
    else
    begin
        if (counter>0)
            counter = counter-1;
    end
    if (counter==0)
    // if it is already terminal count.
        tc = 1;
    else
        tc = 0;
end

endmodule
```

```
/* ============================END of [countdown.v]============================ */
```

在倒计时模块的测试平台中，我们测试了所有可能的输入情况，对 load 信号随机生成 0 或 1，载入数据使用随机生成，并特别测试了载入数据为 0 的情况，确保四次测试中至少有一次载入数据为 0。

为了测试我们的 load 信号确实是同步置位信号，我们特别让 load 信号的变化周期与时钟周期不同，每 1.2 个时钟周期随机生成一次 load。

测试平台的 Verilog 代码如下：

```
/* ============================[countdown_tb.v]============================ */

`timescale 1ns/100ps
module countdown_tb;
reg CLK;
reg LD;
reg[2:0] count;
reg[9:0] DATA;
wire OUT;
countdown Counter(
    .clk(CLK),
    .load(LD),
    .data_load(DATA),
    .tc(OUT)
);

initial fork
    CLK = 0;
    LD = 0;
    DATA = 0;
    count = 1;
join

always #5
begin
    CLK = ~CLK;
end

always #6
begin
    LD = {$random}%2;
end

always #50
// if the interval become shorter, it is highly likely that the counter is loading "data_load" all the time and never counts down.
```

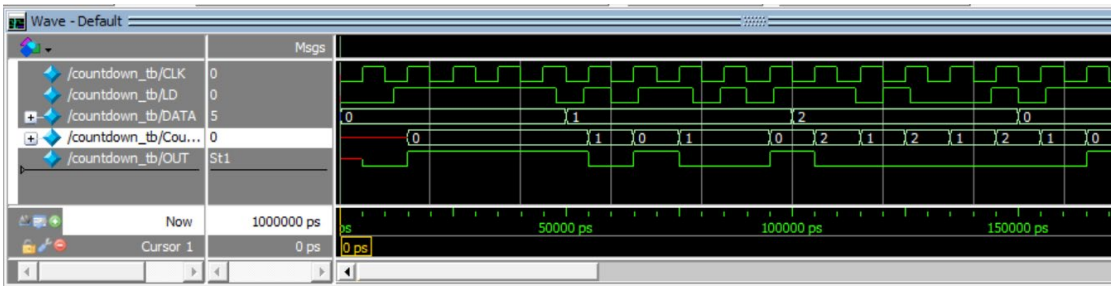
```
begin                                   DATA <= {$random}%1024;
    if (count==3)                     //DATA = {$random}%8 for ploting;
    begin                                 count <= count+1;
        DATA <= 0;                      end
        count <= 0;                   end
    end
    else                              endmodule
    begin

    /* ========================END of [countdown_tb.v]======================== */
```

在仿真波形时，为了能够在图中完整地显示倒计时周期，我们将 DATA 的生成暂时改为 0~7 范围内的随机数。完整测试中，DATA 则使用 0~1023 范围内的随机数。

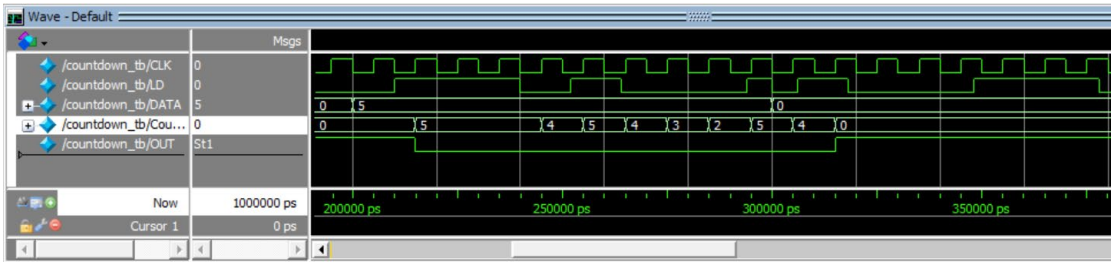
1.3 仿真波形结果讨论

带置位信号的倒计时模块在测试平台上的输出仿真波形如下。

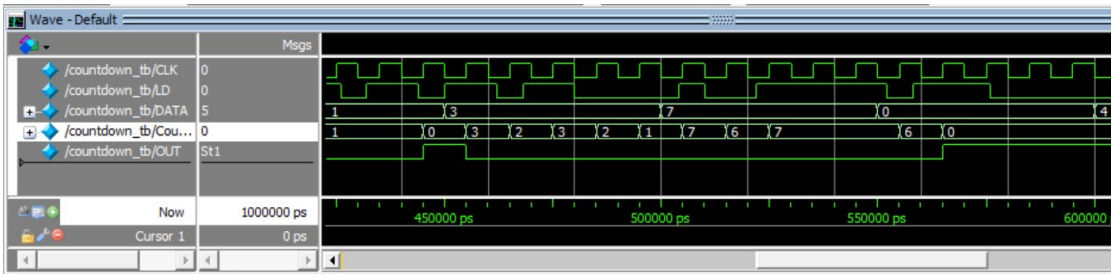
使用软件为 ModelSim 2020.3。



(a)



(b)



(c)

图 1　带置位信号的倒计时模块仿真波形

由图(a)可见 load 信号确实是同步置位信号，在 clk 的上升沿处计时器模块才会开始倒计时（图中的 count 为倒计时模块内部计数器的波形）。此外，载入数值 1 后，当倒计时完成、模块内部计数器变为 0 后，输出 tc 变为高电平。

图(b)中可见，当 data_load 为 0 时，在下一个 load 有效的时刻，模块会将 0 载入计数器，同时输出 tc 变为 1，这点符合要求。

三张图中，计数器在载入非 0 数值后都能正常地进行倒计时，且倒计时过程中输出为 0。

2. 3×3 矩阵转置模块

2.1 实验电路原理

该模块共有 4 个输入、2 个输出，分别为：clk（时钟信号，1-bit），rst_n（异步系统复位信号，1-bit），data_i（输入数据，32-bit），valid_i（输入有效信号，1-bit）；data_o（输出信号，32-bit），valid_o（输出有效信号，1-bit）。

该模块为时钟上升沿触发。

当异步系统复位信号为低电平时，模块应保持复位状态；当复位信号为高电平时，模块才进行正常工作。

当 valid_i 信号有效（即高电平时），data_i 中的数据才会被模块内部记录，认为是真正的输入。模块将按照$A_{11}, A_{12}, A_{13}, A_{21}, A_{22}, A_{23}, A_{31}, A_{32}, A_{33}$的顺序读入矩阵$A$，并在读入第 9 个数值（$A_{33}$）后的下一个时钟上升沿开始转置后矩阵的输出，即通过 data_o 输出$A_{11}, A_{21}, A_{31}, A_{12}, A_{22}, A_{32}, A_{13}, A_{23}, A_{33}$，在输出数据时 valid_o 置为有效状态即高电平。**笔者实现了要求的进阶部分**，即在输出转置后矩阵的同时依然能接受输入的数据，并在模块中记录数据。

不同输入情况下，模块工作状态如下表。

Inputs				Mode
rst_n	clk	valid_i	data_i	
0	x	x	x	Reset
1	↑	0	x	Holding (Might be outputting meanwhile)
1	↑	1	x	Loading data_i (Might be outputting meanwhile)

表 2　矩阵转置模块的工作状态表

2.2 电路实现与关键代码讨论

模块内部使用两个寄存器数组，一个用于记录正在输入的矩阵，一个用于存放正在输出的矩阵。转置的过程只需要通过数组坐标的变换即可完成。为了能够在输出前一次矩阵的转置结果的同时记录新输入的矩阵，我们在完成矩阵第 9 位输入的同时将输入寄存器数组中的内容转移到输出寄存器数组中，并开始下一次的输出。很显然，在获得第 9 位输入的下一个时钟上升沿，前一次矩阵转置后的结果必定已经完成了输出，因此这样并不会有输出寄存器数组中的数值还没输出，但已经被输入寄存器数组的内容覆盖了的情况。

矩阵转置模块的 Verilog 代码如下：

```
/* ================================[transpose.v]================================ */
`timescale 1ns/100ps
module transpose(
    input wire clk,
    input wire rst_n,
    input wire[31:0] data_i,
    input wire valid_i,
    output reg[31:0] data_o,
    output reg valid_o);
integer inp[8:0];
integer tmp[8:0];
integer out[8:0];
reg[3:0] count_in;
reg[3:0] count_out;

always @(negedge rst_n)
begin
    count_in = 0;
    count_out = 0;
    valid_o = 0;
    data_o = 0;
end

always @(posedge clk)
begin
    if (rst_n==1)
    begin
        if (count_out==0)
        begin
            valid_o = 0;
        end
    else
    begin
        data_o = out[count_out-1];
        valid_o = 1;
        count_out = count_out+1;
        if (count_out>9)   count_out = 0;
    end
    if (valid_i==1)
    begin
        inp[count_in%3*3+count_in/3]
           = data_i;
        count_in = count_in+1;
    end
    if (count_in==9)
    begin
        out[0] <= inp[0];
        out[1] <= inp[1];
        out[2] <= inp[2];
        out[3] <= inp[3];
        out[4] <= inp[4];
        out[5] <= inp[5];
        out[6] <= inp[6];
        out[7] <= inp[7];
        out[8] <= inp[8];
        count_out = 1;
        count_in = 0;
    end
    end
end
endmodule
/* ============================END of [transpose.v]============================ */
```

在矩阵转置模块的 testbench 中，我们需要测试合法输入和非法输入。笔者首先测试了 rst_n 的运行，即在转置模块有输出之后将 rst_n 信号变为 0 再变回 1，观察输出波形是否立刻全部变为 0，且之后的输出与之前的输入没有关联。此外，为了测试复位信号的异步性，rst_n 的改变被设置为 2.1 个时钟周期。

为测试模块对 valid_i 和输入数据 data_i 的处理，笔者随机生成 valid_i 信号，该信号有 80%概率为 1，20%概率为 0。

3×3 矩阵转置模块测试平台的 Verilog 代码如下。

```
/* ==============================[transpose_tb.v]============================== */
`timescale 1ns/100ps
module transpose_tb;

reg CLK;
reg RST;
reg[31:0] num;
reg[4:0] count_rst;
reg V_in;

wire V_out;
wire[31:0] data;

transpose MAT(
    .clk(CLK),
    .rst_n(RST),
    .data_i(num),
    .valid_i(V_in),
    .data_o(data),
    .valid_o(V_out)
);

initial fork
    CLK = 1;
    RST = 0;
    count_rst = 0;
    V_in = 0;
join

always #5
begin
    CLK = ~CLK;
end

always #10
begin
    num = $random % 2147483648;
    if ({$random}%10>2)       V_in = 1;
    else   V_in = 0;
end

always #21  // to test RST (for only twice)
begin
    case (count_rst)
        12:     RST = 1;
        11:     begin
                   RST = 0;
                   count_rst = count_rst+1;
                end
        default: begin
                   RST = 1;
                   count_rst = count_rst+1;
                end
    endcase
end

endmodule

/* ==========================END of [transpose_tb.v]============================ */
```

此外，笔者还尝试了无中断连续输入多个矩阵的情况，testbench 如下。

```
/* ============================[transpose_tb_2.v]============================ */

`timescale 1ns/100ps

module transpose_tb_2;

reg CLK;
reg RST;
reg[31:0] num;
reg V_in;

wire V_out;
wire[31:0] data;

transpose MAT(
    .clk(CLK),
    .rst_n(RST),
    .data_i(num),
    .valid_i(V_in),
    .data_o(data),
    .valid_o(V_out)
);

initial fork
    CLK = 1;
    RST = 0;
    V_in = 0;
join

always #5
begin
    CLK = ~CLK;
    RST = 1;
end

always #10
begin
    num = $random % 2147483648;
    V_in = 1;
end

endmodule

/* ========================END of [transpose_tb_2.v]========================== */
```

该测试平台类似于“压力测试”，完全不间断地给模块输入矩阵的数值。

2.3 仿真波形结果讨论

3×3 矩阵转置模块在第一个测试平台上的输出波形如下。使用软件为 ModelSim 2020.3。

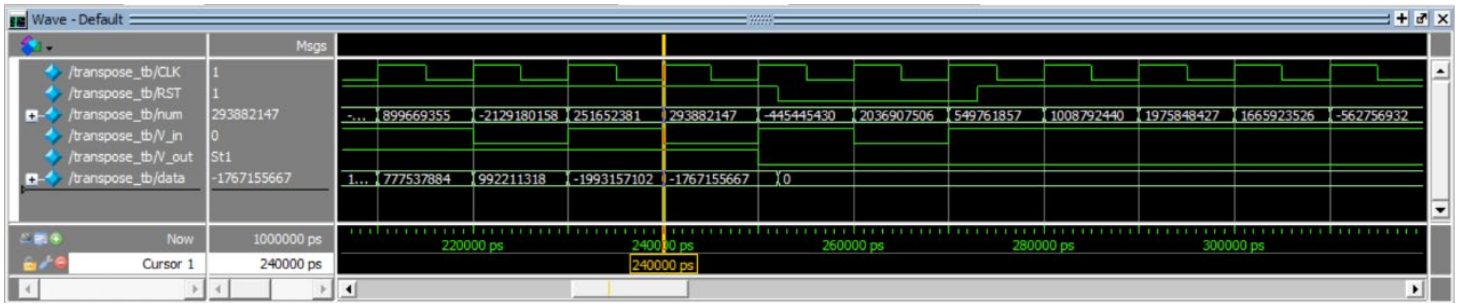


图 2　3×3 矩阵转置模块仿真波形 – 复位信号测试

由图可见，在 241ns 处，rst_n 信号由 1 变为 0，模块的输出在该时刻立刻变为 0（data_o），valid_o 之前就已经变为 0，因此此处继续保持为 0。

在第一个测试平台的后续仿真波形如下。

图 2　3×3 矩阵转置模块仿真波形 – 普通测试

由图可见，在 271ns 处 rst_n 信号由 0 变为 1，模块开始正常工作。

这里我们认为输入的数据是有符号数（实际上是 01 串，毕竟数据的实际类型可能不同，这里为了方便说明统一使用有符号数）。由于 valid_i 为高电平时输入的数据才有效，因此输入的第一个矩阵数据为 $\begin{pmatrix} 1008792440 & 2975848427 & 1665923526 \\ -562756932 & -2049467893 & 1123172741 \\ -1647550405 & 824379234 & 1335973279 \end{pmatrix}$。在最后一个数值输入完成（380ns）后的第一个时钟上升沿（390ns）开始，模块连续九个时钟上升沿 valid_o 输出高电平，同时 data_o 输出了矩阵 $\begin{pmatrix} 1008792440 & -562756932 & -1647550405 \\ 2975848427 & -2049467893 & 824379234 \\ 1665923526 & 1123172741 & 1335973279 \end{pmatrix}$，即输入的矩阵的转置。输出完成后（480ns），valid_o 变为 0。

此外，390ns 开始输入的矩阵 $\begin{pmatrix} 2087561720 & -809216353 & -1379155877 \\ -335626025 & 1260466070 & -517157182 \\ -1149085065 & 921010541 & 265457439 \end{pmatrix}$，在输入最后一个数据（265457439）完成后的下一个时钟上升沿 500ns 时开始输出，输出为

$\begin{pmatrix} 2087561720 & -335626025 & -1149085065 \\ -809216353 & 1260466070 & 921010541 \\ -1379155877 & -517157182 & 265457439 \end{pmatrix}$，即输入矩阵的转置。

在第二个测试平台上，笔者对模块进行了连续输入测试。

图 3　3×3 矩阵转置模块仿真波形 – 连续输入测试

可以发现 100ns~900ns 完成了第一个矩阵的输入，1000ns~1800ns 完成了第二个矩阵的输入。对应的输出（1000ns~1800ns 和 1900ns~2700ns）正是两个输入矩阵的转置。

由测试平台上多种不同输入情况和极端输入情况的输出可见，笔者所写的矩阵转置模块符合要求。

三、实验中的问题与建议

发现如果在 transpose_tb（矩阵转置模块的 testbench）中，将与 data_i 连接的信号 num 改为 integer 类型（也是 32 位）时，转置模块无法正常获得 data_i 的数据，一直输出高阻抗状态。这是由于 integer 没有不定态，不能作为模块输入。

此外，integer 默认是有符号数，reg 默认是无符号数。