序列检测器

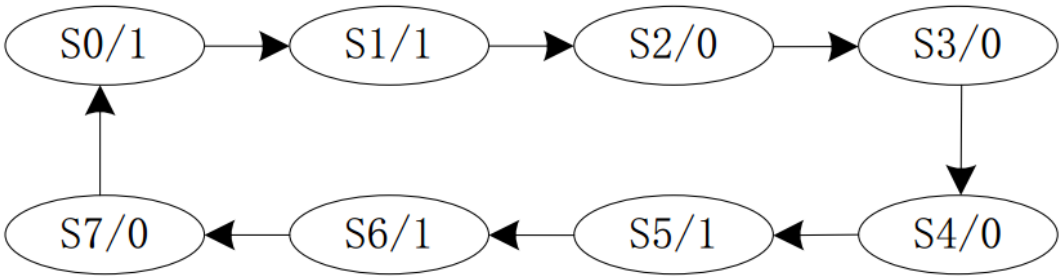
Pb16060240
黄骏达

实验目的：

1. 熟悉并掌握信号发生器的原理与设计
2. 熟悉并掌握状态机的原理与设计
3. 熟悉并掌握序列检测器的原理与设计
4. 熟悉并掌握 VHDL 中原件及其例化
5. 进一步掌握 Altera FPGA 设计

序列产生器与检测器原理：

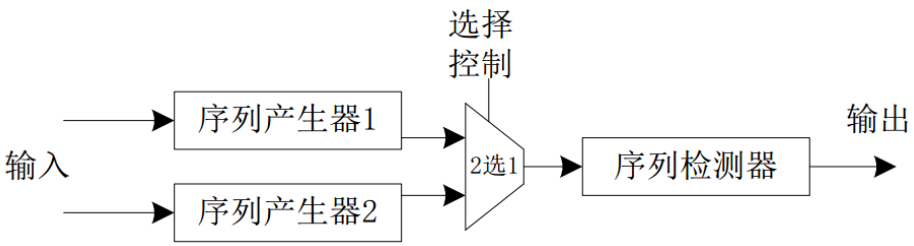
1. 一个时钟控制的序列产生器是由时钟推动的状态转换机，当一个时钟信号到达时，状态机由一个状态切换到下一个状态，并将此信号映射到一个串行输出引脚进行输出。



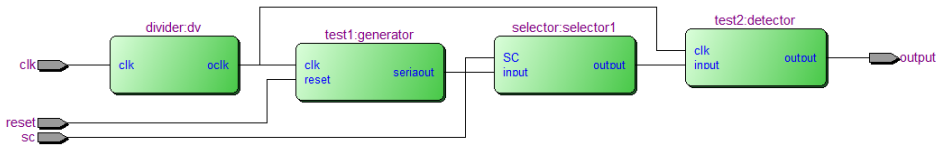
2. 序列检测器串行接收外部来的高低电平信号，然后在检测器内部进行序列匹配，同时，我在此未使用状态转换机的方式来进行序列匹配。

实验设计：

1. 全套的实验架构图如下：由时钟输入推动序列产生器进行状态转换，同时将状态值输出给 2 选 1 选择控制器，然后选择控制器将对应的序列产生器序列串行输出给序列检测器。

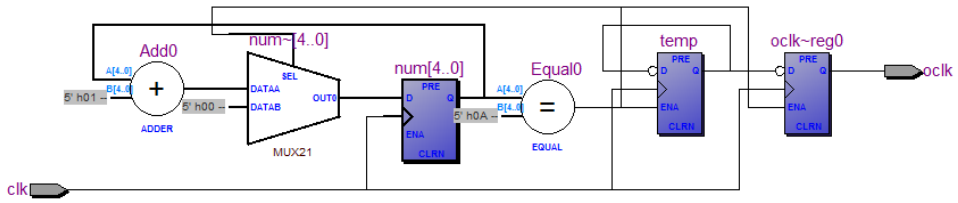


实验电路图：

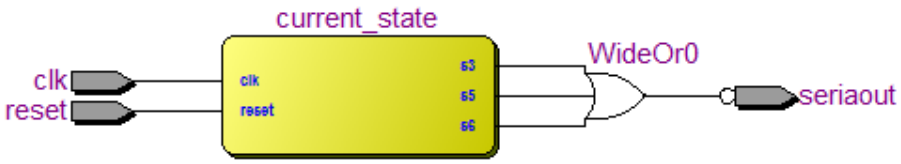


其中只使用一个序列发生器，2 选 1 选择器可选中序列产生器或者选择输出 0，以等效两个序列产生器。

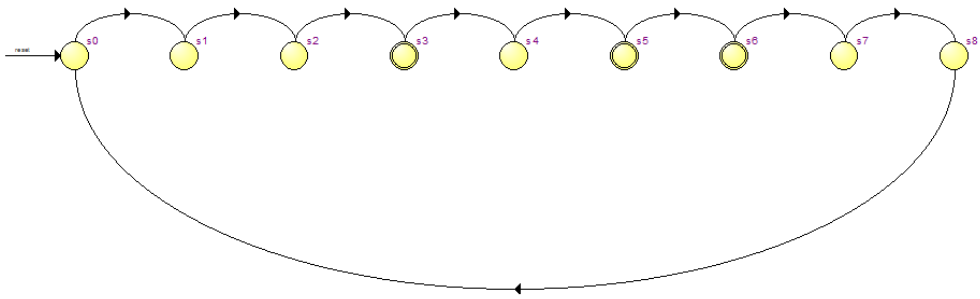
2. 首先时钟信号进入分频器，从而能够让在做实体实验时能够看清 LED 灯的变化。分频器电路图如下：



3. 通过由分频器输出的分频后的时钟信号输出给序列产生器，从而推动整个数据流的运行，序列发生器的电路图如下所示：

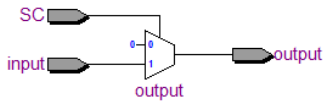


状态转化图如下所示：

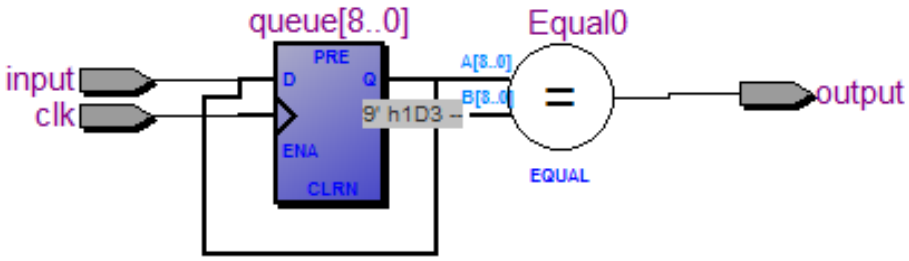


4. 序列发生器的输出输出给 2 选 1 数据选择器，数据选择器选择选中哪个序列发生器，在

此，由于序列发生器相似，因此第二个序列发生器由不断输出数据 0 来进行代替，选择器电路图如下：

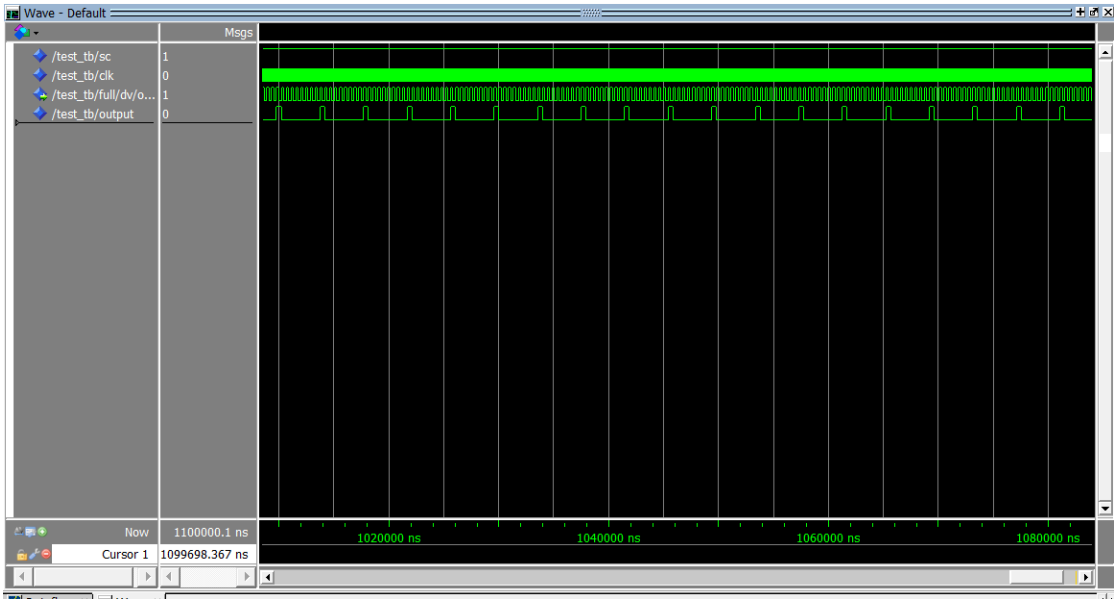


5. 最后从数据选择器输出的信号进入序列检测器，在此，序列检测器主要通过串行信号队列匹配来进行直接的检测，序列检测器电路图如下所示：



仿真结果记录：

顶层实体封装为 clk，sc 和 reset 作为输入，output 作为输出，当检测到匹配的序列时，输出一个高电平，如仿真图所示。



实验源代码：

（divider.vhd）

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity divider is
	port (clk : IN std_logic;
		oclk : OUT std_logic);
end entity;

architecture behave of divider is
begin
	process(clk)
		variable num: integer range 0 to 5000000;
		variable temp:std_logic := '1';
		begin
		if (clk = '1' and clk'event) then
			if (num = 5000000) then
				num := 0; temp := not temp; oclk <= temp;
			else  num := num + 1;
			end if;
		end if;

	end process;
end behave;
```

(selector.vhd)

```
LIBRARY IEEE;
use ieee.std_logic_1164.all;

entity selector is
	port(input,SC : IN std_logic;
		output : OUT std_logic);
END entity selector;

architecture behav of selector is
begin

	process(SC,input)
	begin
		if (SC = '1') then output <= input;
		elsif(SC = '0') then output <= '0';
		end if;
	end process;
end behav;
```

(test1.vhd)序列发生器

```
LIBRARY IEEE;
```

```
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY test1 IS
	port(clk,reset : IN std_logic;
		seriaout : OUT std_logic);
END ENTITY test1;

ARCHITECTURE behav OF test1 IS

TYPE TIMESERIA IS (s0,s1,s2,s3,s4,s5,s6,s7,s8);
SIGNAL current_state :TIMESERIA;

BEGIN

	clk_rise: PROCESS(clk,reset)
	BEGIN
		if (reset = '1') THEN current_state <= s0 ;
		ELSif (clk'EVENT AND clk = '1') THEN
			CASE current_state IS
				WHEN s0 => current_state <= s1;
				WHEN s1 => current_state <= s2;
				WHEN s2 => current_state <= s3;
				WHEN s3 => current_state <= s4;
				WHEN s4 => current_state <= s5;
				WHEN s5 => current_state <= s6;
				WHEN s6 => current_state <= s7;
				WHEN s7 => current_state <= s8;
				WHEN s8 => current_state <= s0;
				when others => null;
			END CASE;
		END IF;
	END PROCESS clk_rise;


	print: PROCESS(current_state)
	BEGIN
		CASE current_state IS
			WHEN s0 => seriaout <= '1';
			WHEN s1 => seriaout <= '1';
			WHEN s2 => seriaout <= '1';
			WHEN s3 => seriaout <= '0';
			WHEN s4 => seriaout <= '1';
			WHEN s5 => seriaout <= '0';
```

```
                WHEN s6 => seriaout <= '0';
                WHEN s7 => seriaout <= '1';
                WHEN s8 => seriaout <= '1';
                when others => null;
            END CASE;
        END PROCESS print;


END behav;
```

(test2.vhd)序列检测器

```
library ieee;
use ieee.std_logic_1164.all;

entity test2 is
        port(input,clk: in std_logic;
--              list : out std_logic_vector(0 to 8);
                output: out std_logic);
end test2;

architecture behav of test2 is
begin

        process(input,clk)
        variable queue:std_logic_vector(0 to 8);
        variable temp:std_logic;
        begin
            if (clk'event and clk = '1') then
                temp := input;
                queue(0 to 7) := queue(1 to 8);
                queue(8):=temp;
            end if;
--          list(0 to 8) <= queue(0 to 8);
            if(queue = "111010011") then output <= '1';
            else  output <= '0';
            end if;

        end process;

end behav;
```

(testx.vhd)顶层实体

```
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.std_logic_unsigned.all;

entity testx is
	port(clk, reset, sc : in std_logic;
		output : out std_logic);
end entity;

architecture sa of testx is
component divider
	port (clk : IN std_logic;
		oclk : OUT std_logic);
end component;

component test1
	port(clk,reset : IN std_logic;
		seriaout : OUT std_logic);
end component;

component selector
	port(input,SC : IN std_logic;
		output : OUT std_logic);
end component;

component test2
	port(input,clk: in std_logic;
		output: out std_logic);
end component;

signal fclk, a1, a2: std_logic;

begin
	dv: divider port map(clk,fclk);
	generator: test1 port map(fclk, reset, a1);
	selector1 : selector port map(a1, sc,a2);
	detector : test2 port map(a2,fclk, output);

end architecture sa;
```

(test_tb.vhd)test bench

```
LIBRARY IEEE;
use ieee.std_logic_1164.all;
--
ENTITY test_tb IS
END ENTITY test_tb;
```

```
ARCHITECTURE shell of test_tb IS
component testx is
	port(clk, reset, sc : in std_logic;
			output : out std_logic);
end component;

signal reset, sc ,clk:std_logic;
signal output:std_logic;

begin
	full :testx port map (clk, reset,sc,output);
	process
		begin
		clk <= '1';
		wait for 10 ns;
		clk <= '0';
		wait for 10 ns;
	end process;

	process(clk)
		begin
		sc <= '1';
--		input <= '1';
--		wait for 10 ns;
--		input <= '1';
--		wait for 10 ns;
--		input <= '1';
--		wait for 10 ns;
--		input <= '0';
--		wait for 10 ns;
--		input <= '1';
--		wait for 10 ns;
--		input <= '0';
--		wait for 10 ns;
--		input <= '0';
--		wait for 10 ns;
--		input <= '1';
--		wait for 10 ns;
--		input <= '1';
--		wait for 10 ns;
--		input <= '1';
--		wait for 10 ns;
--		input <= '1';
```

```
--          wait for 10 ns;
--          input <= '1';
--          wait for 10 ns;
    end process;
END shell;
```

FPGA 验证结果记录：

此实验中，人机接口为 LED，当选中序列发生器之后，LED 周期性闪动，当选中数据 0 输出时，LED 灯熄灭。

实验总结：

1. 首先遇到的问题是序列发生器中，当使用 current_state 和 next_state 进行状态转化时，无法生成状态转换图，并且在仿真中显示，虽然能够进行序列转换，但是序列输出产生一个二分频。这个的原因应该是若引入双状态变量，需要打开两个进程。因此，在此实验中，选择将 next_state 去除，以降低复杂度，从而实现了要求输出。