VHDL 设计

VHDL设计

- 组合逻辑设计
- 时序逻辑设计
- 状态机

组合逻辑

- 数字设计中常用的组合逻辑电路
  - 译码器
  - 编码器
  - 优先编码器
  - 多路选择器
  - 比较器
  - **ALU**

组合逻辑

1.译码器

2－4译码器真值表

选通输入	二进制输入		译码输出			
G	D1	D0	Y0	Y1	Y2	Y3
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1
1	X	X	0	0	0	0

可以直接用布尔方程式设计

```
ENTITY decoder2 IS
PORT(
        d1, d0, g : IN BIT;
        y0, y1, y2, y3 : OUT BIT);
END decoder2;
ARCHITECTURE decoder2_arch OF decoder2 IS
BEGIN
        y0 <= (not d1) and (not d0) and (not g);
        y1 <= (not d1) and ( d0) and (not g);
        y2 <= ( d1) and (not d0) and (not g);
        y3 <= ( d1) and ( d0) and (not g);
END decoder2_arch;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY decoder2a IS
PORT(
        d : IN STD_LOGIC_VECTOR (1 downto 0);
        g : IN STD_LOGIC;
        y : OUT STD_LOGIC_VECTOR (3 downto 0));
END decoder2a;
ARCHITECTURE decoder2a_arch OF decoder2a IS
        SIGNAL inputs : STD_LOGIC_VECTOR (2 downto 0);
BEGIN
        inputs(2) <= g;
        nputs(1 downto 0) <= d;
        WITH inputs SELECT
                y <= “0001” WHEN “000”,
                     “0010” WHEN “001”,
                     “0100” WHEN “010”,
                    “1000” WHEN “011”,
                    “0000” WHEN others;
END decoder2a_arch;
```

可以用并行信号赋值语句来设计

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY decoder2b IS
PORT(
        d : IN INTEGER RANGE 0 to 3;
        g : IN STD_LOGIC;
        y : OUT STD_LOGIC_VECTOR (0 to 3));
END decoder2b;
ARCHITECTURE decoder2b_arch OF decoder2b IS
BEGIN
        y <= “1000” WHEN (d=0 and g=‘0’) ELSE
             “0100” WHEN (d=1 and g=‘0’) ELSE
             “0010” WHEN (d=2 and g=‘0’) ELSE
             “0001” WHEN (d=3 and g=‘0’) ELSE
             “0000”;
END decoder2b_arch;
```

```
ARCHITECTURE decoder2c_arch OF decoder2c IS
BEGIN
        PROCESS(d, g)
        BEGIN
                IF g = '0' THEN
                        IF D = "00" THEN
                                Y <= "0001";
                        ELSIF D = "01" THEN
                                Y <= "0010";
                        ELSIF D = "10" THEN
                                Y <= "0100";
                        ELSIF D = "11" THEN
                                Y <= "1000";
                        ELSE
                                Y <= "0000";
                        END IF;
                ELSE
                        Y <= "0000";
                END IF;
        END PROCESS;
END decoder2c_arch;
```

组合逻辑

2.编码器

8－3编码器真值表

输入								输出		
D(7)	D(6)	D(5)	D(4)	D(3)	D(2)	D(1)	D(0)	Q(2)	Q(1)	Q(0)
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

```
LIBRARY   IEEE;
USE   IEEE.STD_LOGIC_1164.ALL;
ENTITY PRI_ENCODER IS
PORT(
        D : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        Q   : OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END PRI_ENCODER;
```

```
ARCHITECTURE ENCODER_ARCH OF PRI_ENCODER IS
BEGIN
        PROCESS(D)
        BEGIN
                IF D(7) = '1' THEN
                        Q <= "111";
                ELSIF D(6) = '1' THEN
                        Q <= "110";
                ELSIF D(5) = '1' THEN
                        Q <= "101";
                ELSIF D(4) = '1' THEN
                        Q <= "100";
                ELSIF D(3) = '1' THEN
                        Q <= "011";
                ELSIF D(2) = '1' THEN
                        Q <= "010";
                ELSIF D(1) = '1' THEN
                        Q <= "001";
                ELSIF D(0) = '1' THEN
                        Q <= "000";
                ELSE
                        Q <= "000";
                END IF;
        END PROCESS;
END ENCODER_ARCH;
```

```
ENTITY PRI_ENCODER1 IS
PORT(
        d : IN BIT_VECTOR (7 downto 0);
        q : OUT INTEGER RANGE 0 to 7);
END PRI_ENCODER1;
ARCHITECTURE ENCODER1_ARCH OF PRI_ENCODER1 IS
BEGIN
        q <= 7 WHEN d(7)='1' ELSE
             6 WHEN d(6)='1' ELSE
             5 WHEN d(5)='1' ELSE
             4 WHEN d(4)='1' ELSE
             3 WHEN d(3)='1' ELSE
             2 WHEN d(2)='1' ELSE
             1 WHEN d(1)='1' ELSE
             0 WHEN d(0)='1' ELSE
             0;
END ENCODER1_ARCH;
```

组合逻辑

3.多路选择器

四选一选择器真值表

选择信号		输出
S1	S0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

```
ENTITY mux4 IS
PORT(
        d0, d1, d2, d3 : IN BIT;
        s : IN BIT_VECTOR (1 downto 0);
        y : OUT BIT);
END mux4;
ARCHITECTURE mux4_arch OF mux4 IS
BEGIN
        y<= ((not s(1)) and (not s(0)) and d0)
            or ((not s(1)) and ( s(0)) and d1)
            or (( s(1)) and (not s(0)) and d2)
            or (( s(1)) and ( s(0)) and d3);
END mux4_arch;
```

```
ENTITY mux4a IS
PORT(
        d0, d1, d2, d3 : IN BIT;
        s : IN BIT_VECTOR (1 downto 0);
        y : OUT BIT);
END mux4a;
ARCHITECTURE mux4a_arch OF mux4a IS
BEGIN
        M: WITH s SELECT
                y <= d0 WHEN “00”,
                d1 WHEN “01”,
                d2 WHEN “10”,
                d3 WHEN “11”;
END mux4a_arch;
```

```
ENTITY mux4b IS
PORT(
        d0, d1, d2, d3 : IN BIT;
        s : IN BIT_VECTOR (1 downto 0);
        y : OUT BIT);
END mux4b;
ARCHITECTURE mux4b_arch OF mux4b IS
BEGIN
        PROCESS (s)
        BEGIN
                CASE s IS
                        WHEN “00” => y <= d0;
                        WHEN “01” => y <= d1;
                        WHEN “10” => y <= d2;
                        WHEN “11” => y <= d3;
                        WHEN others => y <= ‘0’;
                END CASE;
        END PROCESS;
END mux4b_arch;
```

组合逻辑

4.比较器

- 两个n－bit的二进制数进行相等和大小比较
- 用VHDL语言可以很容易地实现两个数的比较
  - 可以使用IF语句和关系运算符实现

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY compare4 IS
PORT(
        a, b : IN INTEGER RANGE 0 TO 15;
        agtb, aeqb, altb : OUT STD_LOGIC);
END compare4;
ARCHITECTURE compare4_arch OF compare4 IS
        SIGNAL compare : STD_LOGIC_VECTOR (2 downto 0);
BEGIN
        PROCESS (a,b)
        BEGIN
                IF a <b THEN
                        compare <= “110”;
                ELSIF a=b THEN
                        compare <= “101”;
                ELSIF a>b THEN
                        compare <= “011”;
                ELSE
                        compare <= “111”;
                END IF;
                agtb <= compare(2);
                aeqb <= compare(1);
                altb <= compare(0);
        END PROCESS;
END compare4_arch;
```

时序逻辑设计

- 时钟边沿的描述方法
- 触发器
- 计数器
- 寄存器
- 状态机

时序逻辑

1.时钟边沿的描述方法

- 时序电路通常用时钟进程的形式来描述

时序电路常见的两种描述方式

```
PROCESS(CLK)
BEGIN
   IF（时钟边沿条件）THEN
         ——顺序语句
   END IF;
END PROCESS;
```

```
PROCESS
BEGIN
   WAIT ON CLK UNTIL 时钟边沿条件
   ——顺序语句
END PROCESS;
```

时钟上升沿和下降沿

STD_LOGIC_1164中预定义的函数

```
FUNCTION RISING_EDGE(SIGNAL S : STD_ULOGIC) RETURN BOOLEAN;
FUNCTION FALLING_EDGE(SIGNAL S : STD_ULOGIC) RETURN BOOLEAN
```

用信号的属性来描述

时钟上升沿的检查

```
IF CLK'LASTVALUE = '0' AND CLK'EVENT AND CLK = '1' THEN
IF CLK'LASTVALUE = '0' AND NOT CLK'STABLE AND CLK = '1' THEN
```

时钟下降沿的检查

```
IF CLK'LASTVALUE = '1' AND CLK'EVENT AND CLK = '0' THEN
IF CLK'LASTVALUE = '1' AND NOT CLK'STABLE AND CLK = '0' THEN
```

时序逻辑

2.触发器

D触发器真值表

时钟输入	数据输入	数据输出
CLK	**D**	**Q**
上升沿	**0**	**0**
上升沿	**1**	**1**
0	**X**	不变
1	**X**	不变
下降沿	**X**	不变

```
LIBRARY   IEEE;
USE   IEEE.STD_LOGIC_1164.ALL;
ENTITY DFF IS
PORT(
        CLK : IN STD_LOGIC;
        D : IN STD_LOGIC;
        Q   : OUT STD_LOGIC);
END DFF;
ARCHITECTURE DFF_EXAMPLE OF DFF IS
BEGIN
        PROCESS(CLK)
        BEGIN
                IF CLK'EVENT AND CLK = '1' THEN
                        Q <= D;
                END IF;
        END PROCESS;
END DFF_EXAMPLE;
```

同步和异步复位置位

- 同步复位或置位：在复位或置位信号有效而且在时钟边沿到来时，触发器才被复位或置位
- 异步复位或置位：一旦复位或置位信号有效，触发器立即被复位或置位

带异步复位的D触发器真值表

异步复位	时钟输入	数据输入	数据输出
RST	**CLK**	**D**	**Q**
0	**X**	**X**	**0**
1	上升沿	**0**	**0**
1	上升沿	**1**	**1**
1	**0**	**X**	不变
1	**1**	**X**	不变
1	下降沿	**X**	不变

```
LIBRARY   IEEE;
USE   IEEE.STD_LOGIC_1164.ALL;
ENTITY DFF IS
PORT(
        CLK : IN STD_LOGIC;
        RST : IN STD_LOGIC;
        D : IN STD_LOGIC;
        Q   : OUT STD_LOGIC);
END DFF;
ARCHITECTURE DFF_EXAMPLE OF DFF IS
BEGIN
        PROCESS(CLK, RST)
        BEGIN
                IF RST = '0' THEN
                        Q <= '0';
                ELSIF CLK'EVENT AND CLK = '1' THEN
                        Q <= D;
                END IF;
        END PROCESS;
END DFF_EXAMPLE;
```

带异步复位和同步置位D触发器真值表

异步复位	同步置位	时钟输入	数据输入	数据输出
RST	**PSET**	**CLK**	**D**	**Q**
0	**X**	**X**	**X**	**0**
1	**0**	上升沿	**X**	**1**
1	**1**	上升沿	**0**	**0**
1	**1**	上升沿	**1**	**1**
1	**1**	**0**	**X**	不变
1	**1**	**1**	**X**	不变
1	**1**	下降沿	**X**	不变

```
LIBRARY   IEEE;
USE   IEEE.STD_LOGIC_1164.ALL;
ENTITY DFF IS
PORT(
        CLK : IN STD_LOGIC;
        RST : IN STD_LOGIC;
        PSET : IN STD_LOGIC;
        D : IN STD_LOGIC;
        Q   : OUT STD_LOGIC);
END DFF;
ARCHITECTURE DFF_EXAMPLE OF DFF IS
BEGIN
        PROCESS(CLK, RST)
        BEGIN
                IF RST = '0' THEN
                        Q <= '0';
                ELSIF CLK'EVENT AND CLK = '1' THEN
                        IF PSET = '0' THEN
                                Q <= '1';
                        ELSE
                                Q <= D;
                        END IF;
                END IF;
        END PROCESS;
END DFF  EXAMPLE;
```

时序逻辑

3.计数器

- 是对脉冲进行计数的电路，可以用于如事件计数、定时、分频和控制等
- 同步计数器在时钟脉冲的控制下，构成计数器的各触发器的状态同时发生变化
- 异步计数器是把上一级计数器的输出作为下一级计数器的时钟输入，多个计数器串联起来就构成了异步计数器
  - 结构设计方法：设计加减法器，电路连接，一般描述计数器不必描述到这一层次
  - 最简单常用的方法：＋、－算术运算符

带异步复位的4-bit同步计数器真值表

复位输入	时钟输入	输出端			
RST	**CLK**	**Q3**	**Q2**	**Q1**	**Q0**
0	**X**	**0**	**0**	**0**	**0**
1	**0**	不变	不变	不变	不变
1	**1**	不变	不变	不变	不变
1	下降沿	不变	不变	不变	不变
1	上升沿	计数值加1			

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY COUNTER IS
PORT(
        CLK : IN STD_LOGIC;
        RST : IN STD_LOGIC;
        Q   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END COUNTER;
ARCHITECTURE COUNT_EXAMPLE OF COUNTER IS
        SIGNAL TEMP : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
        PROCESS(CLK, RST)
        BEGIN
                IF RST = '0' THEN
                        TEMP <= "0000";
                ELSIF CLK'EVENT AND CLK = '1' THEN
                        IF TEMP = "1111" THEN
                                TEMP <= "0000";
                        ELSE
                                TEMP <= TEMP + 1;
                        END IF;
                END IF;
        END PROCESS;
        Q <= TEMP;
END COUNT_EXAMPLE;
```

双向计数器真值表

复位输入	方向控制	时钟输入	输出端			
RST	**DIRCTION**	**CLK**	**Q3**	**Q2**	**Q1**	**Q0**
0	**X**	**X**	**0**	**0**	**0**	**0**
1	**X**	**0**	不变	不变	不变	不变
1	**X**	**1**	不变	不变	不变	不变
1	**X**	下降沿	不变	不变	不变	不变
1	**1**	上升沿	计数值加1			
1	**0**	上升沿	计数值减1			

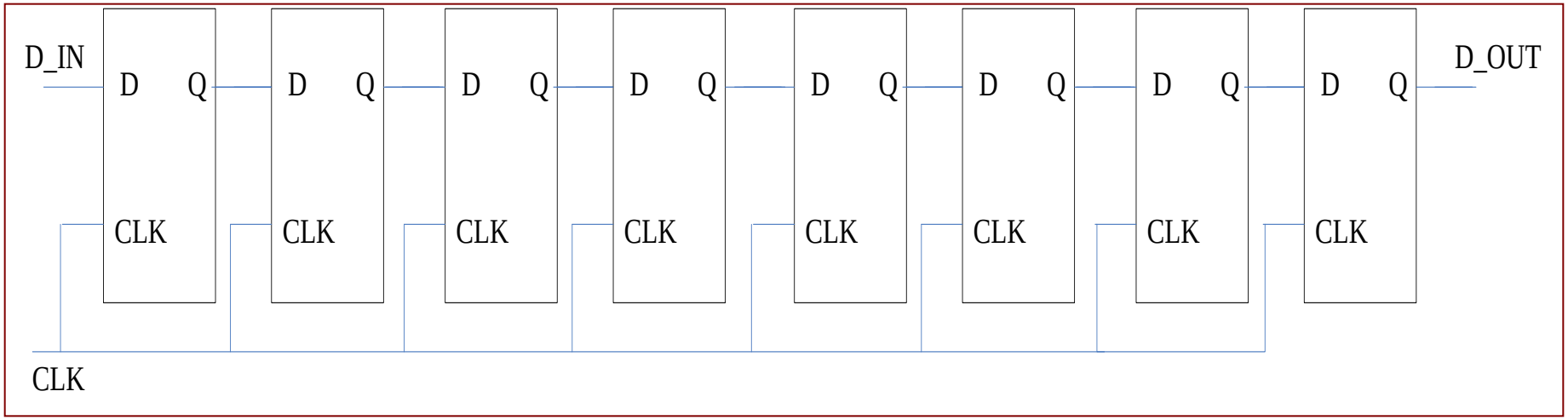
```
LIBRARY   IEEE;
USE   IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY BIDIR_COUNTER IS
PORT(
        CLK : IN STD_LOGIC;
        RST : IN STD_LOGIC;
        DIR : IN STD_LOGIC;
        Q   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END BIDIR_COUNTER;
```

```
ARCHITECTURE COUNT_EXAMPLE OF BIDIR_COUNTER IS
        SIGNAL TEMP : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
        PROCESS(CLK, RST)
        BEGIN
                IF RST = '0' THEN
                        TEMP <= "0000";
                ELSIF CLK'EVENT AND CLK = '1' THEN
                        IF DIR = '1' THEN
                                IF TEMP = "1111" THEN
                                        TEMP <= "0000";
                                ELSE
                                        TEMP <= TEMP + 1;
                                END IF;
                        ELSE
                                IF TEMP = "0000" THEN
                                        TEMP <= "1111";
                                ELSE
                                        TEMP <= TEMP – 1;
                                END IF;
                        END IF;
                END IF;
        END PROCESS;
        Q <= TEMP;
END COUNT_EXAMPLE;
```

时序逻辑

4.寄存器

- 寄存器通常由多个触发器连接而成
  - 通用寄存器
  - 移位寄存器



8-bit串入/串出移位寄存器

8-bit通用寄存器

```
LIBRARY   IEEE;
USE   IEEE.STD_LOGIC_1164.ALL;
ENTITY REGISTER IS
PORT(
        CLK : IN STD_LOGIC;
        D : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        Q   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END REGISTER;
ARCHITECTURE REG_EXAMPLE OF REGISTER IS
BEGIN
        PROCESS(CLK)
        BEGIN
                IF CLK'EVENT AND CLK = '1' THEN
                        Q <= D;
                END IF;
        END PROCESS;
END REG_EXAMPLE;
```

```
LIBRARY   IEEE;
USE   IEEE.STD_LOGIC_1164.ALL;
ENTITY SHIFT8 IS
PORT(
        CLK : IN STD_LOGIC;
        D_IN : IN STD_LOGIC;
        D_OUT   : OUT STD_LOGIC);
END SHIFT8;
ARCHITECTURE SHIFT_EXAMPLE OF SHIFT8 IS
        COMPONENT DFF
        PORT(
                CLK : IN STD_LOGIC;
                D : IN STD_LOGIC;
                Q : OUT STD_LOGIC);
        END COMPONENT;
        SIGNAL TEMP : STD_LOGIC_VECTOR(8 DOWNTO 0);
BEGIN
        TEMP(0) <= D_IN;
        GEN_DFF : FOR I IN 0 TO 7 GENERATE
                DFFS : DFF PORT MAP(
                        D => TEMP(I),
                        CLK => CLK,
                        Q => TEMP(I + 1));
        END GENERATE;
        D_OUT <= TEMP(8);
END SHIFT_EXAMPLE;
```

8-bit 串入/串出移位寄存器

```
ARCHITECTURE SHIFT_EXAMPLE OF SHIFT8 IS
        SIGNAL TEMP : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
        TEMP (0) <= D_IN;
        PROCESS(CLK)
        BEGIN
                IF CLK'EVENT AND CLK = '1' THEN
                        TEMP (1) <= TEMP (0);
                        TEMP (2) <= TEMP (1);
                        TEMP (3) <= TEMP (2);
                        TEMP (4) <= TEMP (3);
                        TEMP (5) <= TEMP (4);
                        TEMP (6) <= TEMP (5);
                        TEMP (7) <= TEMP (6);
                        D_OUT <= TEMP(7);
                END IF;
        END PROCESS;
END SHIFT_EXAMPLE;
```

循环左移移位寄存器功能表

输入					输出							
S(2)	S(1)	S(0)	LOAD	CLK	Q(7)	Q(6)	Q(5)	Q(4)	Q(3)	Q(2)	Q(1)	Q(0)
X	X	X	0	上升沿	D(7)	D(6)	D(5)	D(4)	D(3)	D(2)	D(1)	D(0)
0	0	0	1	上升沿	不循环，保持原值							
0	0	1	1	上升沿	循环左移1位							
0	1	0	1	上升沿	循环左移2位							
0	1	1	1	上升沿	循环左移3位							
1	0	0	1	上升沿	循环左移4位							
1	0	1	1	上升沿	循环左移5位							
1	1	0	1	上升沿	循环左移6位							
1	1	1	1	上升沿	循环左移7位							

```
LIBRARY   IEEE;
USE   IEEE.STD_LOGIC_1164.ALL;
ENTITY BARREL_SHIFT IS
PORT(
        CLK : IN STD_LOGIC;
        LOAD : IN STD_LOGIC;
        S : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
        D : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        Q   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END BARREL_SHIFT;
```

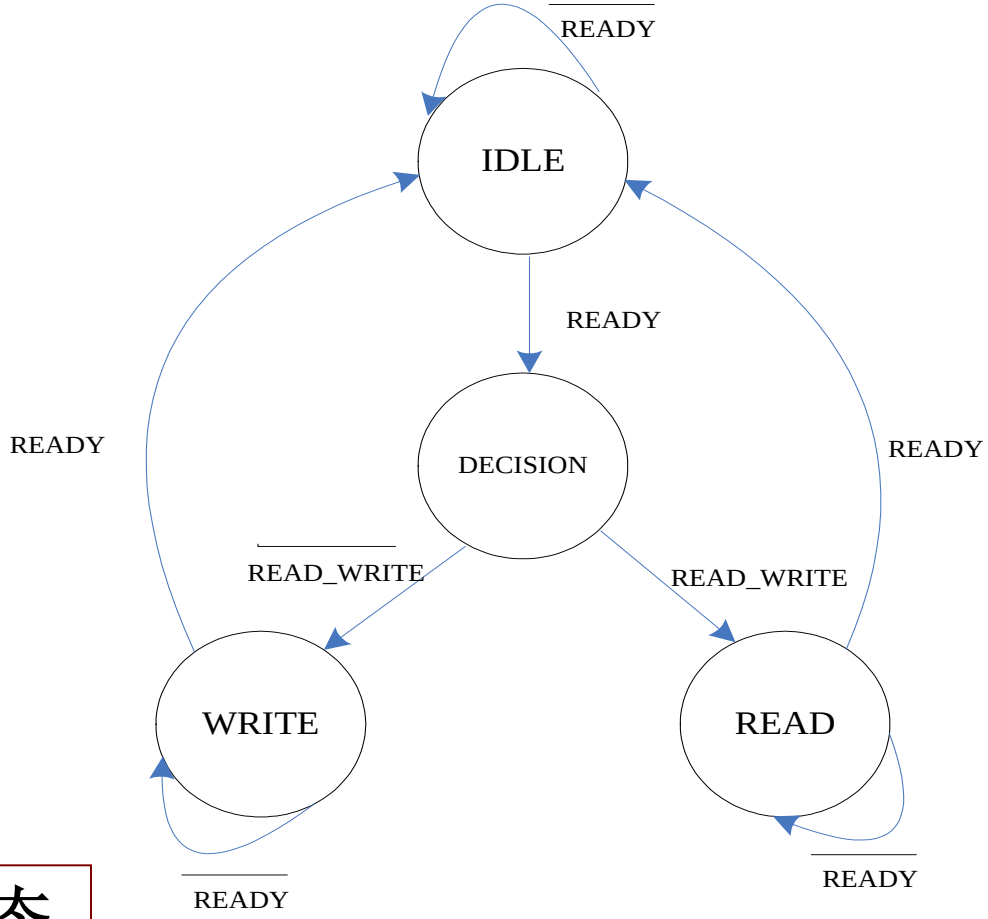
```
ARCHITECTURE BARREL_EXAMPLE OF BARREL_SHIFT IS
  SIGNAL TEMP_Q : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
  PROCESS(CLK, LOAD, S)
  BEGIN
    IF CLK'EVENT AND CLK = '1' THEN
          IF LOAD = '0' THEN
            TEMP_Q <= D;
          ELSE
            CASE S IS
                  WHEN "000" =>
                    TEMP_Q <= TEMP_Q;
                  WHEN "001" =>
                    TEMP_Q <= TEMP_Q(6 DOWNTO 0) & TEMP_Q(7);
                  WHEN "010" =>
                    TEMP_Q <= TEMP_Q(5 DOWNTO 0) & TEMP_Q(7 DOWNTO 6);
                  WHEN "011" =>
                    TEMP_Q <= TEMP_Q(4 DOWNTO 0) & TEMP_Q(7 DOWNTO 5);
                  WHEN "100" =>
                    TEMP_Q <= TEMP_Q(3 DOWNTO 0) & TEMP_Q(7 DOWNTO 4);
                  WHEN "101" =>
                    TEMP_Q <= TEMP_Q(2 DOWNTO 0) & TEMP_Q(7 DOWNTO 3);
                  WHEN "110" =>
                    TEMP_Q <= TEMP_Q(1 DOWNTO 0) & TEMP_Q(7 DOWNTO 2);
                  WHEN "111" =>
                    TEMP_Q <= TEMP_Q(0) & TEMP_Q(7 DOWNTO 1);
                  WHEN OTHERS =>
                    TEMP_Q <= TEMP_Q;
            END CASE;
          END IF;
    END IF;
  END PROCESS;
  Q <= TEMP_Q;
END BARREL_EXAMPLE;
```

状态机

传统状态机设计步骤

- 分析和定义问题
- 根据分析画出状态图，并由此列出状态表，包括所有的当前状态、输入、和输出
- 根据状态转移表，可以进一步推出次态方程组
- 对布尔方程进行简化，得到次态方程和输出逻辑，画出逻辑电路

用VHDL语言设计步骤



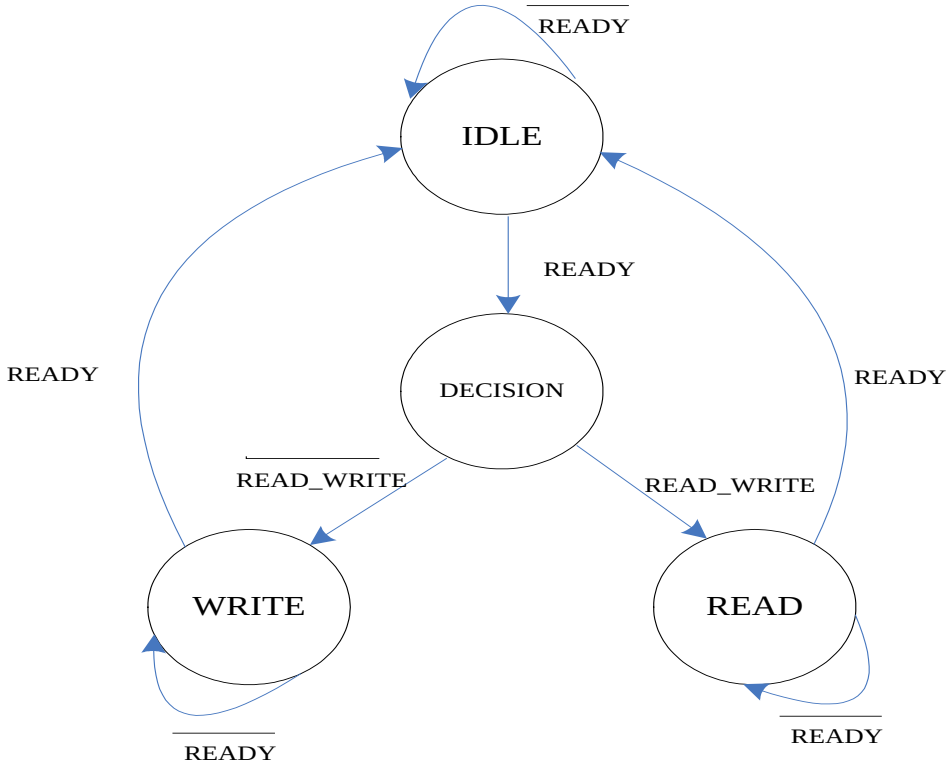
- 用枚举类型定义信号的状态

TYPE STATE_TYPE IS (IDLE, DECISION, READ, WRITE);

SIGNAL CURRENT_STAE, NEXT_STATE : STATE_TYPE;

- 建立一个进程，把当前态和输入信号作为进程的敏感信号
  - 在进程中定义状态的转移
  - 通过CASE－WHEN语句写出状态的转移流程

```
STATE_TRANS : PROCESS(CURRENT_STATE, READ, WRITE);
BEGIN
        ……
END PROCESS;
```



```
STATE_TRANS : PROCESS(CURRENT_STATE, READ_WRITE, READY);
BEGIN
CASE CURRENT_STATE IS
        WHEN IDLE =>
                OE <= '0';
                WE <= '0';
                IF READY = '1' THEN
                        NEXT_STATE <= DECISION;
                ELSE
                        NEXT_STATE <= IDLE;
                END IF;
        WHEN DECISION =>
                OE <= '0';
                WE <= '0';
                IF READ_WRITE = '1' THEN
                        NEXT_STATE <= READ;
                ELSE
                        NEXT_STATE <= WRITE;
                END IF;
        WHEN READ =>
                OE <= '1';
                WE <= '0';
                IF READY = '1' THEN
                        NEXT_STATE <= IDLE;
                ELSE
                        NEXT_STATE <= READ;
                END IF;
        WHEN WRITE =>
                OE <= '0';
                WE <= '1';
                IF READY = '1' THEN
                        NEXT_STATE <= IDLE;
                ELSE
                        NEXT_STATE <= WRITE;
                END IF;
                        END CASE;
END PROCESS;
```

- 次态变为当前态
  - 状态的变化过程和时钟的上升沿同步

```
STATE_CLOCKED : PROCESS(CLK)
BEGIN
        IF CLK'EVENT AND CLK = '1' THEN
                CURRENT_STATE <= NEXT_STATE ;
        END IF;
END PROCESS;
```

- 两个进程来定义状态机，称为双进程的状态机描述方式
- 一个进程定义了组合逻辑部分
- 一个进程定义了状态转移对时钟的同步处理

- 状态机的复位
  - 同步复位时，可以在状态转移进程的开始对复位信号RESET进行判断
  - 异步复位时，可以在STATE_CLOCKED进程中进行

```
STATE_TRANS: PROCESS(CURRENT_STATE, READ_WRITE, READY);
BEGIN
        IF RESET = '1' THEN
                OE <= '0';
                WE <= '0';
                NEXT_STATE <= IDLE;
        ELSE
                CASE CURRENT_STATE IS
                ……
                END CASE;
        END IF;
END PROCESS;
```

同步复位

异步复位

```
STATE_CLOCKED: PROCESS(CLK, RESET)
BEGIN
        IF RESET = '1' THEN
                CURRENT_STATE <= IDLE;
        ELSIF CLK'EVENT AND CLK = '1' THEN
                CURRENT_STATE <= NEXT_STATE;
        END IF;
END PROCESS;
```

双进程状态机的完整代码

```
LIBRARY   IEEE;
USE   IEEE.STD_LOGIC_1164.ALL;
ENTITY STATE_MACHINE IS
PORT(
        CLK : IN STD_LOGIC;
        RESET : IN STD_LOGIC;
        READY : IN STD_LOGIC;
        READ_WRITE : IN STD_LOGIC;
        OE : OUT STD_LOGIC;
        WE : OUT STD_LOGIC);
END STATE_MACHINE;
ARCHITECTURE STATE_EXAMPLE OF STATE_MACHINE IS
        TYPE STATE_TYPE IS (IDLE, DECISION, READ, WRITE);
        SIGNAL CURRENT_STATE, NEXT_STATE : STATE_TYPE;
BEGIN
STATE_TRANS : PROCESS(CURRENT_STATE, READ_WRITE, READY);
BEGIN
CASE CURRENT_STATE IS
        WHEN IDLE =>
                OE <= '0';
                WE <= '0';
                IF READY = '1' THEN
                        NEXT_STATE <= DECISION;
                ELSE
                        NEXT_STATE <= IDLE;
                END IF;
```

```
                WHEN DECISION =>
                        OE <= '0';
                        WE <= '0';
                        IF READ_WRITE = '1' THEN
                                NEXT_STATE <= READ;
                        ELSE
                                NEXT_STATE <= WRITE;
                        END IF;
                                WHEN READ =>
                        OE <= '1';
                        WE <= '0';
                        IF READY = '1' THEN
                                NEXT_STATE <= IDLE;
                        ELSE
                                NEXT_STATE <= READ;
                        END IF;
                                WHEN WRITE =>
                        OE <= '0';
                        WE <= '1';
                        IF READY = '1' THEN
                                NEXT_STATE <= IDLE;
                        ELSE
                                NEXT_STATE <= WRITE;
                        END IF;
                                END CASE;
                END PROCESS;
                STATE_CLOCKED: PROCESS(CLK, RESET)
                BEGIN
                        IF RESET = '1' THEN
                                CURRENT_STATE <= IDLE;
                        ELSIF CLK'EVENT AND CLK = '1' THEN
                                CURRENT_STATE <= NEXT_STATE;
                        END IF;
                END PROCESS;
END STATE_EXAMPLE;
```

```
ARCHITECTURE STATE_EXAMPLE1 OF STATE_MACHINE IS
        TYPE STATE_TYPE IS (IDLE, DECISION, READ, WRITE);
        SIGNAL STATE: STATE_TYPE;
BEGIN
        STATE_TRANS: PROCESS(CLK, RESET, READ_WRITE, READY)
        BEGIN
                IF RESET = '1' THEN
                        STATE <= IDLE;
                ELSIF CLK'EVENT AND CLK = '1' THEN
                        CASE STATE IS
                                WHEN IDLE =>
                                        IF READY = '1' THEN
                                                STATE <= DECISION;
                                        ELSE
                                                STATE <= IDLE;
                                        END IF;
                                WHEN DECISION =>
                                        IF READ_WRITE = '1' THEN
                                                STATE <= READ;
                                        ELSE
                                                STATE <= WRITE;
                                        END IF;
                                WHEN READ =>
                                        IF READY = '1' THEN
                                                STATE <= IDLE;
                                        ELSE
                                                STATE <= READ;
                                        END IF;
                                WHEN WRITE =>
                                        IF READY = '1' THEN
                                                STATE <= IDLE;
                                        ELSE
                                                STATE <= WRITE;
                                        END IF;
                        END CASE;
                END IF;
        END PROCESS;
        WITH STATE SELECT
                OE <= '1' WHEN READ,
                      '0' WHEN OTHERS;
        WITH STATE SELECT
                WE <= '1' WHEN WRITE,
                      '0' WHEN OTHERS;
END STATE_EXAMPLE1;
```

单进程状态机的代码

•状态的定义可以直接用常量来指定各个状态变量的取值

```
ARCHITECTURE STATE_EXAMPLE1 OF STATE_MACHINE IS
        SIGNAL STATE : STD_LOGIC_VECTOR(2 DOWNTO 0);
        CONSTANT IDLE : STD_LOGIC_VECTOR(2 DOWNTO 0) := “000”;
        CONSTANT DECISION : STD_LOGIC_VECTOR(2 DOWNTO 0) := “001”;
        CONSTANT READ : STD_LOGIC_VECTOR(2 DOWNTO 0) := “010”;
        CONSTANT WRITE : STD_LOGIC_VECTOR(2 DOWNTO 0) := “100”;
BEGIN
```

```
STATE_TRANS: PROCESS(CLK, RESET, READ_WRITE, READY)
BEGIN
        IF RESET = '1' THEN
                STATE <= IDLE;
        ELSIF CLK'EVENT AND CLK = '1' THEN
                CASE STATE IS
                        WHEN IDLE =>
                                IF READY = '1' THEN
                                        STATE <= DECISION;
                                ELSE
                                        STATE <= IDLE;
                                END IF;
                        WHEN DECISION =>
                                IF READ_WRITE = '1' THEN
                                        STATE <= READ;
                                ELSE
                                        STATE <= WRITE;
                                END IF;
                        WHEN READ =>
                                IF READY = '1' THEN
                                        STATE <= IDLE;
                                ELSE
                                        STATE <= READ;
                                END IF;
                        WHEN WRITE =>
                                IF READY = '1' THEN
                                        STATE <= IDLE;
                                ELSE
                                        STATE <= WRITE;
                                END IF;
                END CASE;
        END IF;
END PROCESS;
OE <= STATE(1);
WE <= STATE(2);
END STATE_EXAMPLE1;
```

状态机的容错

- 使状态机转移出非法状态
  - 确定系统中非法状态的数量，非法状态的数量是状态位数量的2次方减去已定义状态数。
  - 在定义状态的枚举类型时包括对非法状态的定义
  - 使用**WHEN OTHERS**语句来定义当进入这些非法状态后，脱离它们并进入到指定的状态

```
CASE CURRENT_STATE IS
        ……
        WHEN UNDEFINED =>
                NEXT_STATE <= IDLE;
END CASE;
```

```
CASE CURRENT_STATE IS
        ……
        WHEN OTHERS =>
                NEXT_STATE <= S0;
END CASE;
```