

第一章

- 8421码和二进制不完全一样、8421码的17是“0001 0111”
- 8421加法：判9加6
- 奇偶校验码中、奇校验码要多用一个异或门（用来异或0）
- 方框也是与门 方框里“ \vee ”=1”是或门
- 同或在偶数个门时相当于异或非
- OC门并联具有“线与”的功能
- 冗余律 $AB + \overline{A}C + BC = AB + \overline{A}C$
- 求反函数用反演律
 - 所有变量自己先取反
 - 与或互换
 - 运算顺序不变（加括号）
 - 大非号不变
- 对偶式用于由已知等式证明未知等式
 - 变量本身不变、其余与反演相同
- 消去律 $A + \overline{A}B = A + B$
- 卡诺图
 - 无论行还是列都是1243的顺序 一定要记住
 - 与或式形式：直接填
 - 或与式形式：求反演、填0
 - 合并
 - 可以合并1、2、4、8个，别忘了边和四个角
 - 圈尽可能少、尽可能大
 - 最简结果不唯一
 - 无关项可以用、但不能违反圈尽可能少的原则

第二章

- 半加器：两个数相加 全加器：三个数相加
- 组合逻辑分析：写电路图的表达式~化简~真值表~猜功能
- 组合逻辑分析：列完整真值表~一个输出一个卡诺图~画电路
- 数据选择器
 - 地址线是几、输出哪根线
 - 实现组合逻辑的时候可以拎任意根做地址
- 数据分配器
 - 地址是几、把数据送到哪根线
- 译码器
 - 类型：38译码器-74138
 - 地址是几、哪根线是0（低有效）
 - 地址范围：把被限制的线先配好（如使能或要求的地址）、剩下的看范围
 - 实现组合逻辑：直接把要的最小项对应的输出用与非门连起来即可
 - 译码器做数据分配器：数据接低有效

- 编码器
 - Eo: 无输入标识、只有在使能正常但是没有输入的时候为0（低有效）
 - Gs: 有效输出标识、正常输出0~7时为0
 - I7~I0从高到低、哪个输入先为0、输出其二进制反码
 - 扩展要把输入输出表写全、然后慢慢分析
 - 有优先性
- 数值比较器
 - 输入两个数以及级连比较结果（平的时候咋算）、输出结果高有效
- 加法器
 - 串型加法器
 - 进位逐级传递
 - 结构简单
 - 速度慢
 - 超前进位加法器
 - 速度快，进位直接全算出来
 - 占用空间大
- 减法器
 - 加补码（按位取反+1
 - 有进位输出的差是正数、无进位输出的差是负数
- 奇偶校验器
 - 偶校验位发生器：用奇输出
 - 奇校验位发生器：异或1、用奇输出
 - （有异或1:奇、没有：偶）
 - 接收：哪个输出接收哪个
- 竞争冒险
 - 逻辑冒险：一个信号和自己的非或（0冒险）/与（1冒险）
 - 功能冒险：多个输入信号同时发生变化、产生毛刺
 - 译码器容易出现毛刺
 - 使能端稳定后开启可以避免毛刺
 - 格雷码可以避免毛刺
 - 判别冒险情况
 - 检查是否有原、反变量同时出现
 - 其他变量赋值、看有没有可能出现上面的情况
 - 消除冒险方法
 - 加一个冗余项
 - 选通、有冒险的时候关门
 - 加个旁路电容、滤波

第三章

- RS触发器：次态方程要知道
 - 注意同时为0的‘不定’

- D锁存器会空翻（电平触发）
- 寄存器（边沿触发）——记得看清上升沿下降沿
 - D: 不变
 - JK: JK同0维持、同1翻转、不同听J, 次态方程要知道
 - T: 0维持1翻转
- 时序电路分析
 - 时序电路的特点: 输出状态不仅取决于此时的输入信号、还与前一时刻电路的状态有关
 - 分类
 - 同步/异步
 - 莫尔型（输出无输入信号）/米勒型（输出受外部输入影响）
 - 分析方法:
 - 确定输入输出信号
 - 列方程
 - 输出方程（输出从哪几根线）
 - 驱动方程（进触发器的J啊K啊T啊都是谁）
 - 状态方程（触发器自己的次态方程）
 - 列真值表（现态Q2、Q1、Q0 | 次态Q2、Q1、Q0 | 输出）
 - 画状态转移图
 - 写逻辑功能（一般是计数器或者序列信号检测器）
 - 主循环
 - 有效状态、无效状态
 - 能不能自启动
- 锁存器（立即响应、电平触发）
 - OE: 输出使能、输出或高阻
 - LE: 门门控制、高则可变、低则锁存
 - 输入稳定滞后与锁存信号
- 寄存器（等时钟相应、边沿触发）
 - 先有效数据、再有脉冲信号打进去
- 环形计数器、例4个D寄存器
 - 四进制计数器、模N
 - 不需要译码、方便
 - 无自启动能力、电路利用率低、开始时预置1000作为起点
- 扭环形计数器（最后一个从非输出）
 - 8进制计数器、模2N
 - 利用率高很多、有个格雷码
 - 无自启动能力、异步清零进入格雷码主循环
- 中规模计数器
 - 计数器同步级连: 时钟信号公用、低位的进位输出控制高位的使能端
 - 每有一个进位让高位有效一下记一个数
 - 异步十进制74LS160
 - 异步二进制74LS161

- 同步十进制74LS162
- 同步二进制74LS163
- 十进制计数器的位权：80、40、20、10、8、4、2、1
- 任意进制计数器
 - 清零法（例10进制）
 - 同步：到9就复位
 - 异步：到10就复位（在10不停留、立马清零）
 - 一次预置法
 - 多次预置法
- 同步时序电路的设计
 - 对问题进行逻辑抽象、得到状态转移图
 - 状态化简
 - 状态编码，得到真值表
 - 一般就直接二进制按顺序编就行
 - 驱动方程和输出方程
 - 分别进卡诺图
 - 画逻辑图
 - 检查自启动
 - 把无效状态带入驱动方程

第四章

- 特殊存储部件
 - 寄存器堆
 - 多端口寄存器、可同时读写、同时输出两个数
 - 寄存器堆列FIFO
 - 指令队列
 - 寄存器栈LIFO
 - 减少函数调用时堆内存的访问
 - 双向移位寄存器构成
- 随机读写寄存器RAM
 - 能读能写、易失
 - 外部连线
 - 单向地址线
 - 双向数据线
 - 单向控制线
 - 片选线
- 只读寄存器ROM
 - 只读不能写、不易失
 - 逻辑构成
 - 与门（地址译码器）、或门（存储矩阵）
- 存储器容量：单元数（字数）*每个单元的位数（字长）

- 单位：B——8bit、K—— 2^{10} 、M—— 2^{20} 、G—— 2^{30}
- 例：8KB=8K*8= 2^{13} *8——13根地址线、8根数据线
- RAM分类
 - SRAM：速度快集成度低容量小
 - 触发器、同时送行列地址
 - DRAM：速度慢集成度高容量大
 - 电容、分两次传送行列地址
- 求组成存储的芯片要几片、就直接 \div 就行
- FPGA是查找表、其他的如CPLD都猜与阵列可编程

第五章

- PLD基本概念
 - PLD分类
 - PROM：与阵列固定、或阵列可编程、输出电路固定
 - PLA：与阵列可编程、或阵列可编程、输出电路固定
 - PAL：与阵列可编程、或阵列固定、输出电路固定
 - GLA：与阵列可编程、或阵列固定、输出电路可组态
 - CPLD
 - 与或结构复杂
 - 集成度低、多用于组合逻辑、成本低
 - FGBA
 - 查找表LUT
 - 集成度高、多用于时序逻辑、成本高
- VHDL语言
 - 合法标识符：字母开头、下划线不可连续不可开头结尾
 - 调用库部分
 - library ieee;
 - use ieee.std_logic_1164.all; (有时会用unsigned)
 - 实体部分
 - entity xxx is
 - port
 - (
 - xxx : in std_logic;
 - xxx : in std_logic_vector(2 downto 0);
 - xxx : out std_logic_vector(7 downto 0) (这里没有分号)
 -);
 - end xxx;
 - 结构体部分
 - architecture xxx of xxx is
 - signal xxx; xxx: std_logic (这里不需要in或者out)
 - begin

- `xxx <= not(xxx and xxx);` (这里语句都是并行、不分先后)
 - `end xxx;`
- vhdl操作符
 - 逻辑操作符
 - `and`、`or`、`nand`、`nor`、`xor`、`not`
 - 无优先级、需要括号 (`and`不优先于`or`)
 - 关系操作符
 - `=`、`/=`、`<`、`<=`、`>`、`>=` (后四个只用于整数、枚举类型、逻辑矢量)
 - 如`if (en='0')` ——所有的0必须加'
 - 算数操作符
 - `+`、`-`、`*`、`/`、`&`(并置符, 用于拼接字符串)
 - 前四个智能用于整数运算
 - 重载操作符
 - 加载`unsigned`库之后、两个矢量可以用算数运算符了、并会自动转化为整数
 - 例: 实现4位二进制全加器: `sum<=('0'&a)+b+c` (`sum`5位、后面自动转换)
 - `cout<=sumt(4);` (取最高位)
 - `sum<=sumt(3 downto 0);` (取低四位)

- vhdl语句

- 并行语句

- 赋值 (简单、条件、选择)
- 进程
- 例化


- 赋值语句

- 简单并行赋值: 信号`<=`表达式
- 选择赋值`with-select-when`

- 格式:

- `with` 选择表达式 `select`
- 信号名 `<=` 表达式1 `when` 选择值1,
表达式2 `when` 选择值2, (这里是逗号)

表达式n `when others`;



```
with sel select
  f <= d0 when "00",
  d1 when "01",
  d2 when "10",
  d3 when others;
```

- 无先后顺序

- 必须在进程外用

- 条件赋值`when-else`


- 格式:

- 信号名 `<=` 表达式1 `when` 条件式1 `else`
表达式2 `when` 条件式2 `else`

表达式n;

- 有先后顺序、类似`if`

- 进程语句



```
A <= "000" when I(7) = '0' else
  "001" when I(6) = '0' else
  "010" when I(5) = '0' else
  "011" when I(4) = '0' else
  "100" when I(3) = '0' else
  "101" when I(2) = '0' else
  "110" when I(1) = '0' else
  "111";
```


▸ 格式

- process (敏感信号表)
 - 局部变量声明 (如 variable d: std_logic)
- begin
 - 顺序语句
- end process

◦ 顺序语句

▸ 赋值语句

- 信号 <= 表达式;
 - 赋值在一个过程结束之后才会发生变化
 - 不能在一个process里对一个信号多次赋值
- 变量 := 表达式;
 - 赋值没有延迟, 立即变化

▸ if语句

- 不完全if格式
 - if 条件式 then
 - 顺序处理语句;
 - end if
- 这里没有else, 不完全的时候将形成触发器, 实现“不满足条件则保持”
- 边沿触发: if (clk'event and clk = '1') then

▸ 多选择if

- 格式
 - if 条件表达式1 then
 - 顺序语句
 - elsif 条件表达式2 then
 - 顺序语句
 - elsif
 - . . .
 - else (如果是组合逻辑必须有else, 否则变成寄存器了)
 - 顺序语句
 - end if;
- 有先后顺序、可用于优先编码器

```
architecture art of coder is
begin
    process(input)
    begin
        if input(7)='0' then
            output<="000";
        elsif input(6)='0' then
            output<="001";
        elsif input(5)='0' then
            output<="010";
        elsif input(4)='0' then
            output<="011";
        elsif input(3)='0' then
            output<="100";
        elsif input(2)='0' then
            output<="101";
        elsif input(1)='0' then
            output<="110";
        else
            output<="111";
        end if;
    end process;
end art;
```



- case语句
 - 格式
 - case 选择表达式 is
 - when 分支值1 => 顺序语句;
 - when 分支值2 => 顺序语句;
 - when others => 顺序语句;
 - end case
 - 必须穷尽

```
CASE s IS
  WHEN "00" => z<=d0;
  WHEN "01" => z<=d1;
  WHEN "10" => z<=d2;
  WHEN others => z<=d3;
END CASE;
```

- 循环语句
 - 格式
 - for 循环变量 in 循环范围 loop
 - 顺序语句
 - end loop
 - 循环范围min to max

```
FOR n IN 0 TO 7 LOOP
  --FOR循环语句
  tmp := tmp XOR d(n);
END LOOP;
```

• 基本设计

- 组合逻辑
 - 高阻断开: a<='Z' / a<="ZZZZ"
 - if else写全
- 时序逻辑
 - 同步: 敏感信号表只有时钟、复位在时钟判断里
 - 异步: 敏感信号还有别的信号、复位begin后就判断
 - 判断时钟电路边沿没有else
 - 计数器
 - 需要中间信号完成又读又写
 - temp到输出的赋值在process外

◦ 状态机

- 用户自定义数据类型
- 格式
 - type 数据类型名 is 数据类型定义 (枚举)
- 整体结构
 - 时序进程 (敏感信号时钟)
 - if (时钟上升沿) then
 - 现态<=次态
 - 组合进程 (现态、输入)
 - case 现态 is
 - when s0 => 输出赋值——莫尔型 (输出独善其身)
 - if 输入=xxx then
 - 次态<=xxx
 - end if
 - when s0 => if 输入=xxx then——米莉型 (输出还受输入影响)
 - 次态<=xxx;

type statetype 取值范围

```
type statetype is (s0, s1, s2, s3);
signal present_state, next_state : statetype;
..... 现态 次态
present_state <= S0;
```


◦ 输出赋值

◦ end case

◦ 原件例化

- 在architecture下面把别的元件的实体部分摘过来进行元件声明
- 格式
 - entity改成component
 - is去掉
 - 结尾改成end component
- 中间信号还是要声明
- 先画图，然后按照图上的顺序接线
- 格式
 - u1:xxx port map (xx, xx)

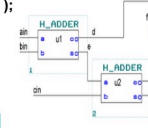
```

LIBRARY IEEE; --或门逻辑描述
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY or2a IS
  PORT (a, b : IN STD_LOGIC;
        c : OUT STD_LOGIC);
END or2a;
ARCHITECTURE art1 OF or2a IS
  BEGIN
    c <= a OR b;
END art1;

LIBRARY IEEE; --半加器描述
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY h_adder IS
  PORT (a, b : IN STD_LOGIC;
        co, so : OUT STD_LOGIC);
END h_adder;
ARCHITECTURE art2 OF h_adder IS
  BEGIN
    so <= a XOR b;
    co <= a AND b;
END art2;

LIBRARY IEEE; --1位二进制全加器顶层设计描述
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY f_adder IS
  PORT (ain, bin, cin : IN STD_LOGIC;
        cout, sum : OUT STD_LOGIC);
END f_adder;
ARCHITECTURE art3 OF f_adder IS
  COMPONENT h_adder
    PORT (a, b : IN STD_LOGIC;
          co, so : OUT STD_LOGIC);
  END COMPONENT; -- 元件声明
  COMPONENT or2a
    PORT (a, b : IN STD_LOGIC;
          c : OUT STD_LOGIC);
  END COMPONENT;
  SIGNAL d, e, f : STD_LOGIC;
  BEGIN
    u1 : h_adder PORT MAP (ain, bin, d, e);
    u2 : h_adder PORT MAP (a=>e, b=>cin, co=>f, so=>sum);
    u3 : or2a PORT MAP (d, f, cout); -- 元件例化
  END art3;

```



必须化中顺序一致

第六章

• 数字系统

- 五大部件：运算器、控制器、存储器、输入、输出

• ASM图

- 状态框：矩形
 - 代表一个状态
 - 装控制命令
- 条件判断框：菱形
 - 对于控制器输入信号进行判断
 - 装检测条件
- 条件输出框：平行四边形
 - 判断后产生的结果
 - 装控制命令、功能相当于状态框、但从属于状态框
- 一个状态是2T，数据在1T的时候打入（稳定）
- 而所有“▲”的命令都是电位型命令、电位命令会随着状态转移而消失
 - 一般电位型命令不会单独出现、会配合判断或者打入脉冲

• 计数器型控制器（状态数多、n=>2^n）

- 给ASM图状态码编码
 - 00、01、10、11...(几个数就是几个触发器)
- 由ASM图得到控制信号表达式

$$C_1 = \overline{Q_2} \overline{Q_1} \overline{X}$$

$$C_2 = \overline{Q_2} \overline{Q_1} X$$

◦ 控制器的状态转移表

- 00到01、现态到次态怎么转换

◦ 触发器驱动方程

- 把是1的现态全并起来、整理

- 电路实现
- 在写控制信号表达式的时候、所有脉冲信号（L? ）后面都要加与T2
 - 代表脉冲信号在状态周期的中间打入
 - 电位信号不需要
- 定序型控制器（n个触发器n个状态）
 - 给ASM图的状态框分配触发器
 - 分配Q1、Q2、Q3、Q4
 - 由ASM图得控制信号表达式
 - 控制器的MDS表
 - 现态次态（直接是Q1、Q2 不是00、01）
 - 触发器的次态方程、激励函数
- 多路选择器型控制器（n个触发器 2^n 个控制状态、还需要n个数据选择器）
 - 给ASM图的状态框编码
 - 00、01...
 - 由ASM图得到控制信号表达式
 - 和第一种一样
 - 控制器的状态转移表
 - 表也一样、同样把触发器的驱动方程写出来
 - 数据选择器的数据端
 - 看转移表次态的两列、每一列的4种情况分别填入对应数据端

2 个四选一的选择器	$Q_1^n \quad Q_0^n$	$Q_1^{n+1} \quad Q_0^{n+1}$	转移条件
$A_1 A_0 = Q_1^n \quad Q_0^n$	0 0	1 0	\overline{X}
		1 1	X
	0 1	0 0	
MUX1 -- D1	1 0	0 0	
MUX0 -- D0	1 1	0 0	
MUX1 (d0) = $\overline{X} + X = 1$; MUX0 (d0) = X			
MUX1 (d1) = 0 ; MUX0 (d1) = 0			
MUX1 (d2) = 0 ; MUX0 (d2) = 0			
MUX1 (d3) = 0 ; MUX0 (d3) = 0			