数逻实验报告

姓名：宁晨然
日期：2018/12/20
学号：17307130178
课程：COMP130003.01

第一次实验：

1）必做实验：3-8 译码器

1.实验目的

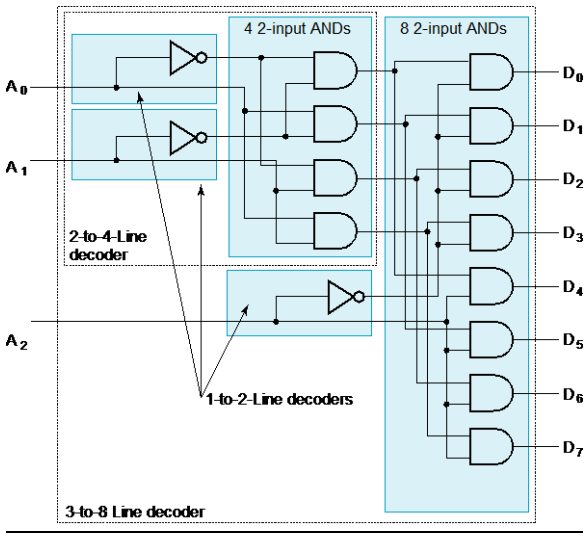
熟悉 74LS138 的功能，了解其每个引脚的作用。

了解 3-8 译码器的功能及作用。

熟悉 verilog 硬件编程语言。

2.实验原理

①实验原理图：



②实验真值表：

输入				输出							
EN	*C*	*B*	*A*	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	0	1	1	1	1	1	1
1	0	1	0	1	1	0	1	1	1	1	1
1	0	1	1	1	1	1	0	1	1	1	1
1	1	0	0	1	1	1	1	0	1	1	1
1	1	0	1	1	1	1	1	1	0	1	1
1	1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	0

③功能：

G1 高电平、G2AG2B 同时低电平时，正常显示译码；否则全为高电平。

3.实验过程

根据真值表，使用 assign 语句对每个 Y 进行赋值，比如 assign y[0] = ~(~s[2] & ~s[1] & ~s[0]) | ~(G1 & ~G2A & ~G2B);decoder 将每个 Y 都赋值后，在 top 文件中调用一次 decoder 模块，输入为[5:0]SW，输出为[7:0]LED，使得 LED 亮。

4.实验结论

当{G1,G2A,G2B}=3'b100 时，led 灯正常按照真值表显示；其他时候所有输出均为高电平，不会亮灯。

2）选做实验：4-16 译码器

1.实验目的

利用 1 中的设计的 3-8 译码器和与非门，设计一个 4-16 译码器。

2.实验原理

①实验原理：

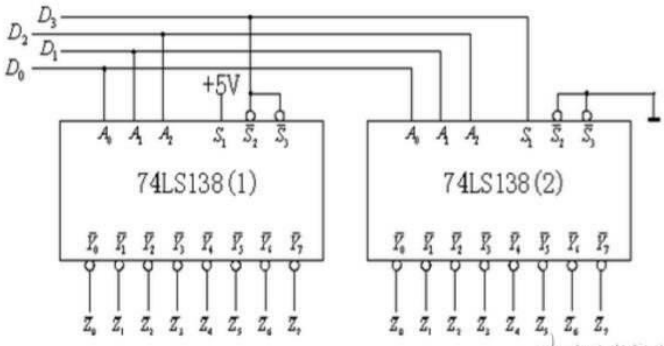


图 2-1 为 4 线-16 线译码器逻辑图

②实验真值表：

输入				输出															
A3	A2	A1	A0	Y15	Y14	Y13	Y12	Y11	Y10	Y9	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
0	1	1	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

③实验功能：

直接利用 3-8 译码器和与非门实现 4-16 译码器的全部功能。

3.实验过程

将 3-8 译码器中的 8 个输出分别与 SW[3]和~SW[3]排列组合，可以得到 2*8 个输出结果，即 4-16 译码器。

4.实验结论

16 个输出灯泡按照真值表正常显示。

3）选做实验：8-3 普通编码器

1.实验目的

设计一个表中逻辑功能的 8-3 普通编码器。

2.实验原理

①实验真值表

输入								输出		
I0	I1	I2	I3	I4	I5	I6	I7	F0	F1	F2
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

②实验功能

将 8 个输入数据编码压缩为 3 个输出

3.实验过程

按照真值表，利用 assign 语句，将 F 与 I 一一对应起来。输入为 SW，输出为 LED。

4.实验结论

LED 按照真值表正常显示。

4）选做实验：8-3 优先编码器

1.实验目的

实现一个 8-3 优先编码器

2.实验原理

在选做 3 中实验的真值表中，写出带优先级的真值表，再利用真值表和 assign 语句实现优先编码器。

3.实验过程

利用 always 语句中的“assign 语句”，比如 8'b1???????:out=3'b000;可以直接进行三位输出的一次性赋值。案例中，其中？表示优先级低从而不需要直到它的值，当 SW[0]==1 时，直接进行赋值。

4.实验结论

与选做 3 对比可知，此实验中的编码器能够实现优先级。

5.实验感想

由于是第一次使用 vivado 和 verilog 语言来实现，所以写代码时、运行时非常吃力。本次实验中的原理都不难，逻辑图也很容易理解，我也全部弄懂了，就是在实现代码时有些不熟练，比如 Case 语句可以极大减少代码量，相比使用逻辑门的代码方式更加简洁；比如 always 语句的语法规则又使我错误了好几次。以后做实验时需要更加了解 verilog 的语法特点，利用代码的便利实现我熟知的逻辑结构图。

写代码时，也明白了，主要步骤就是先画真值表、画逻辑图，然后使用 verilog 代码写出来即可。

第二次实验：

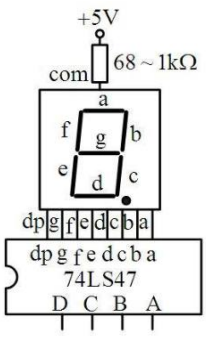
1）必做实验：BCD-7 段显示译码器

1.实验目的

设计一个 BCD-7 段显示译码器。

2.实验原理

①实验原理图



②实验真值表

输入				输出						
D	C	B	A	A	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X

③实验功能

输入 0-15 的数字，输出对应的数字（<=9），其他情况(>9)输出 default 状态

3.实验过程

主要是利用 always 当中，用 case 语句进行分情况讨论，case([3:0]SW)，然后根据输入的值，分成 11 种情况讨论，分别是 0-9 和 default 状态，每种状态输出对应的数字。每个数字都会对应七段显示管的显示部分，如下。

```
'h0: led = 7'b0000001;
'h1: led = 7'b1001111;
'h2: led = 7'b0010010;
'h3: led = 7'b0000110;
'h4: led = 7'b1001100;
'h5: led = 7'b0100100;
'h6: led = 7'b0100000;
'h7: led = 7'b0001111;
'h8: led = 7'b0000000;
'h9: led = 7'b0001100;
default: led = 7'b1111110;
```

4.实验结论

七段显示管正常显示，输入 0-9 的数字，七段显示管显示对应的数字。

5.实验感想

本次实验主要考验的是 assign 语句、case 语句、always 语句的语法用法，思路非常简单。并且了解 xdc 更多的开关，这次实验中就用到了七段显示管，而不是第一次实验中的 LED。根据真值表可以找到思路，然后按照真值表分情况讨论即可。

第三次实验：

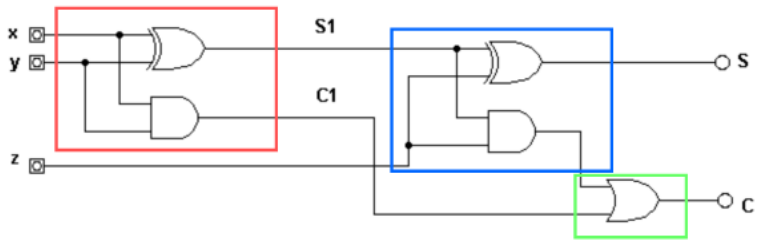
1）必做实验：一位全加器

1.实验目的

用与非门和反相器设计一个一位全加器

2.实验原理

①实验原理图



②实验逻辑公式

$$S=A\oplus B\oplus Cin$$

$$C = AB + Cin * （A \oplus B）$$

3.实验过程

题目要求只能使用与非门和反相器，所以我先用与非门实现了异或门（module XR），再将原全加器的图改写为使用异或门和与非门、反相器的图，逻辑表达式如上。根据表达式使用门元件进行连接。由于输出只有 0、1，我使用了 led 灯作为输出。

4.实验结论

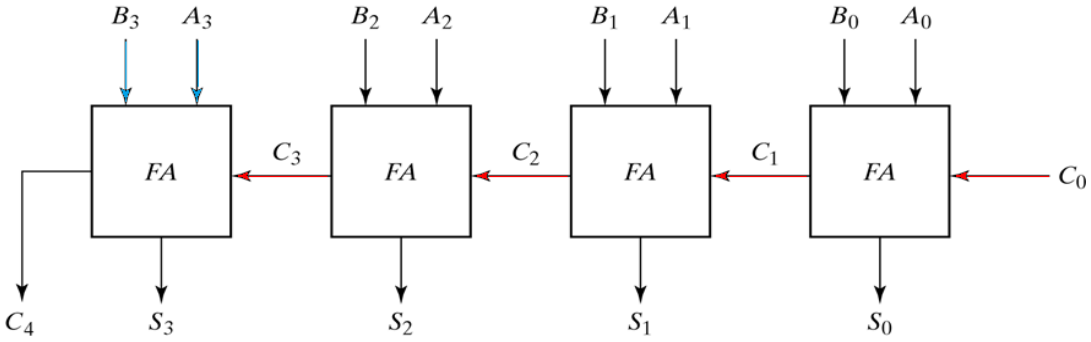
全加器功能正常。

2）选做实验：4 位行波进位加法器

1.实验目的

以上述一位全加器为模板，设计一个 4 位行波进位加法器

2.实验原理



3.实验过程

根据教材上的行波进位加法器的图，可知将四个一位全加器串联起来即可。则定义三个中间连接线 wire，实例化全加器四次。使用了模块例化的方法。

4.实验结论

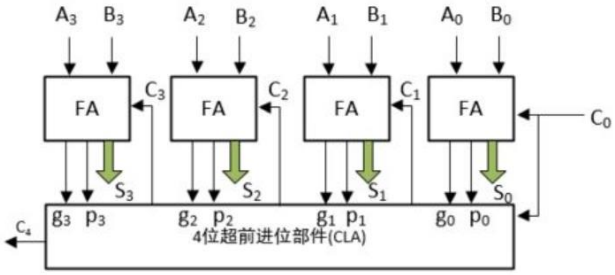
四位行波加法器正常使用。

3）选做实验：超前进位 4 位全加器

1.实验目的

设计一个含超前进位电路的 4 位加法器，对比上面行波进位加法器的延时

2.实验原理



超前进位加法器的实验重要的一点就是提前计算 P、G，公式如下:

P = a_i ⊕ b_i

G = a_i & b_i

$C_0 = cin$

C_i = G_{i-1} | (P_{i-1} & C_{i-1})　　//展开即可

S_i = P_i ⊕ C_i

3.实验过程

我使用了 add.v 中定义的 XR（也可以直接用库）和与门，再利用超前进位加法器 C 和 S 的公式，先用 C 的公式算出 cout，再由公式直接算出每一个 S。利用表达式，c 我使用 assign 语句直接赋值，s 使用了异或门（XR），公式如上。

4.实验结论

对比第二个实验，这个实验看起来并没有明显的缩减“延时”，应该是时间频率都比较快难以看出来。

5.实验感想

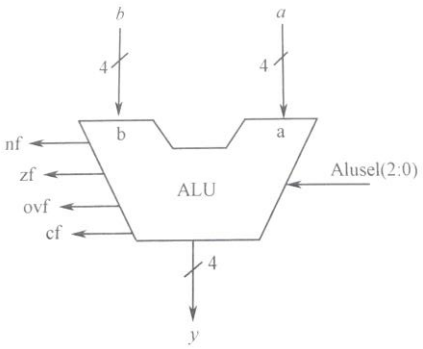
更深刻地理解了 verilog 语言中的模块功能，模块与 c++中的函数类似，都是将各种功能封装。一开始为了实现异或门，不知道如何使用与非门，觉得特别麻烦，后来写出了公式后，使用一个 XR 模块封装起来，每次只需要实例化即可。这样做非常节省了代码空间，使得结构更加有条理。第二个实验直接使用四个全加器模块例化就特别方便。

还有就是一定要在实验前就把逻辑表达式、实验原理图想清楚，提前思考清楚使用什么门、assign 等逻辑运算。注释也很重要，在第三个实验中完成了很多复杂的操作（主要是实例化）后，代码会显得很多，这个时候一定要分块写清楚这一个部分是在干什么。

第四次实验：

1）必做实验：算术逻辑单元的设计

1.实验目的



实现 ALU 算术逻辑单元。

2.实验原理

S1	S0	M=0 逻辑运算	M=1 算术运算	
			Cn=0	Cn=1
0	0	F=not A	A+B	A+B+1
0	1	F=A and B	A-B	A-B-1
1	0	F=A or B		
1	1	F=A xor B		

ALU 是算术逻辑单元，根据 case 语句和 if else 语句可以控制输出的逻辑运算。

3.实验过程

首先根据书上的控制条件可知，首先定义三位控制 S，用 case 语句判断，进行什么运算。对于与、或、非、异或的计算，直接用 verilog 中对应符号计算即可。关键是在于加减法。

关于加减法，首先这个输入的 A 和 B 是有符号数，所以定义是写上 signed 关键字，同理输出 F 也是有符号。做加法时，可知可能越位，所以将 F 用 cout 扩充一位，并且加上 cin 进位，即{cout,F} = A+B+cin，这个时候 F 记录的是一个有符号数，如果溢出、进位会显示在 cout 上。进位发生在正数相加越位时。另外，做减法时同理，{cout,F} = A-B-cin，会进行有符号的扩展一位的运算。在实现 N（负数）、V（溢出）、Z（零）时，零判断很简单，直接判断最终结果是否为全 0，即 F == 0；负数判断 N = F[3]；溢出判断为几个异或判断，三个符号位异或后与 cout 异或。

关于减法是否借位的问题，我一开始考虑了很多，有把 signed 改成 unsigned 后扩位进行无符号运算，后来发现可以直接用{cout,F}扩位后进行有符号位的运算，并且会自动判断借位、进位。

4.实验结论

ALU 逻辑所有功能均正常使用。

5.实验感想

这次试验中没有怎么运用门做，直接运用了 verilog 自带定义的加减法，所以比较方便。比较复杂的是思考有符号数和无符号数的运算的不同，有符号数的加减自动变为有符号数，无符号数的加减或许需要扩位进行运算。比如一开始我做的无符号运算，用了一个扩位后进行了加减法。

同样这次的控制条件、开的 LED 灯、开的开关很多，这种情况需要在程序中备注清楚每个开关、LED 灯是管控的什么，在后面进行测试的时候比较方便的看出结果对不对。这次试验写了几个版本的控制，最终选择了 verilog 自带的加减法（一开始甚至用了上次课的全加器），说明在做实验之前可以把 verilog 的语法先巩固一下，有些 signed 标识符没有学习到，需要去查阅资料、自己加强。

第五次实验：

1）必做实验：74LS163

1.实验目的

用 verilog 语言描述 74LS163。

2.实验原理

①实验真值表

输 入									输 出			
C_R	CP	L_D	EP	ET	D_3	D_2	D_1	D_0	Q_3	Q_2	Q_1	Q_0
0	↑	x	x	x	x	x	x	x	0	0	0	0
1	↑	0	x	x	D	C	B	A	D	C	B	A
1	↑	1	0	x	x	x	x	x	Q_3	Q_2	Q_1	Q_0
1	↑	1	x	0	x	x	x	x	Q_3	Q_2	Q_1	Q_0
1	↑	1	1	1	x	x	x	x	计数脉冲			

②功能描述

74LS163 的真值表和功能可知，当清零端“CR=0”，计数器输出 Q[3:0]为全 0，这个时候是异步复位的功能。当 CR=1，LOAD=0 时，在时钟上升沿时，Q = A，输出等于输入，相当于同步置数功能。当 CR=LOAD=P=T 时，每一个时钟上升沿时计数器加 1，进位端 RCO=Q0&Q1&Q2&Q3&T。利用 74LS163，可以实现 16 进制以下的各种分频的进制时钟。

3.实验过程

在 always 结构中利用 ifelse 语句，先对 CR 进行讨论，此时为全 0，异步复位，然后讨论 LOAD=1 时的同步置数，再讨论 P、T 的情况，此时均为异步的加法。注意要写上进位端 RCO。

4.实验结论

对照真值表和同步异步功能，结合实验原理，CR=0 时全部清 0，置数功能状态 Q=A，当 P=T=1 时，根据时钟计数器每次加一，并且 RCO 输出正常，全亮后紧接着是全灭，说明实验正常显示。同步置数和异步复位均正常。

2）选做实验：数字时钟

1.实验目的

实现一个数字时钟，用分秒模拟时分。为了检查方便用了近似 1hz 的频率。

时为 24 进制，分为 60 进制。

需要用一个 RESET 控制清零返回预设时间。RESET 是同步置数功能，为了检查方便设置为 23：50。

2.实验原理

在做实验二 BCD 管时可以发现，对于每个 AN，BCD 的七段显示相同，只能显示相同的数字。所以利用分频，每次只显示一个数字、利用视觉残留，感觉像是同时显示四个数字，这个频率称之为扫描频率。使用了一个 190hz 的扫描频率使得分频显示 BCD 管。

对于进位的问题，只需要讨论需要进位的情况，就可以模拟 60 进制和 24 进制，具体实现过程中用 ifelse 语句多加判断。

3.实验过程

1）进位判断

①xx:x9->xx:y0 ②xx:59->xy:00 ③x9:59->y0:00 ④23:59->00:00

2）RESET : xx:xx->23:50

3）BCD7 利用实验二中的设置，已经显示好 0-9 的数字

4）设置 hour1/hour0/min1/min0 四个值保存时钟四个数字

5）计数分频使用 4hz/3 当作计数时间（近似 1s），显示分频使用 190hz 作为扫描频率

4.实验结论

最后时钟正常走动，并且成功进位（23：59->00：00 也正常运行）

5.实验感想

①实验一完成 74LS163 时，更加学会了查看一个部件的设计图、设计原理还有其真值表、波形图等，在阅读完后能够在脑中形成一个完整的框架。并且学会了分频的方法，能够把预设的 CLK100MHZ 分成更低频率，使每个时钟上升沿间隔变长。

②实验一让我学会了同步赋值和异步赋值，即<=和=的区别，并且做的时候遇到了很多 reg、wire 利用失误的小细节，更加深刻理解了 reg 和 wire 的作用

③实验二实现时钟时，学会了分频显示可以达到“视觉暂留”的有趣效果，更深刻的理解了频率的分频。在时钟部分，最主要的模块是进位的判断，这里有很多细节的进位操作，这就需要在草稿纸中先把草稿打好，逻辑思路必须清晰，并且写代码的时候、代码的风格必须一致，以免出现 begin 和 end、if 和 else 没有对应的情况

④对于多种功能的实现使用模块化更好，比如分频显示数字、计算下一个数字等，功能不同互不干涉的模块，可以分开写，这样逻辑更加清晰。

第六次实验：

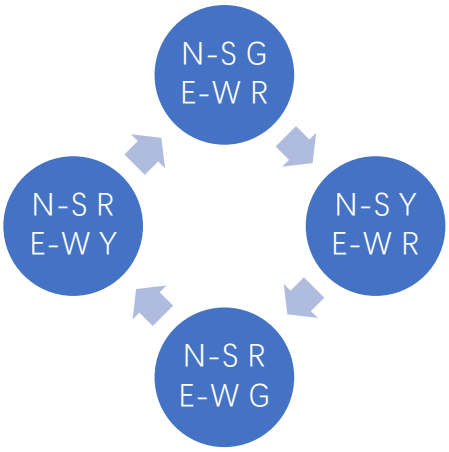
1）必做实验：利用有限状态机实现交通灯

1.实验目的

用 verilog 语言描述交通灯控制器的及 3s、9s 定时输入信号的逻辑功能。

2.实验原理

利用一个有限状态机，实现一个交通灯控制器，要求灯会按照绿-黄-红循环变化，并且绿-45s，黄-15s，红-60s。



根据实验原理图，可知有四个状态，且每个状态分别触发的条件为：

①N-S G/E-W R -> N-S Y/E-W R : counter up to 9s

②N-S Y/E-W R -> N-S R/E-W G : counter up to 3s

③N-S R/E-W G -> N-S R/E-W Y : counter up to 9s

④N-S R/E-W Y -> N-S G/E-W R : counter up to 3s

default : counter++

说明我们需要一个计数器，根据 1hz 的分频信号，TS 判断是否达到 9s，TL 判断是否达到 3s 根据计数器，作为判断状态变化的条件，当到达上述条件，state 即可变化

3.实验过程

具体实验细节需要三个步骤：clkdiv / counter / state changes
①clkdiv 选择 q[25]近似等于 1hz
②counter 中输入信号为 ST，输出信号为 TS，TL
输入 ST：清零计数器 / 输出 TS：是否达到 3s / 输出 TL：是否达到 9s
注意采用了 posedge clk or posedge ST，时钟上升沿或清零计数器操作都可以进入 always
③state changes
用六个 LED 表示 6 盏灯，分别是 N-S(R,Y,G) E-W(R,Y,G)
根据状态图定义了 4 个状态，分别是 S0 = 6'b001100,S1 = 6'b010100,
S2 = 6'b100001,S3 = 6'b100010
state 之间变化的条件为 TS/TL/TS/TL

4.实验结论

实验结果是 LED 灯按照状态图所示正常变化。

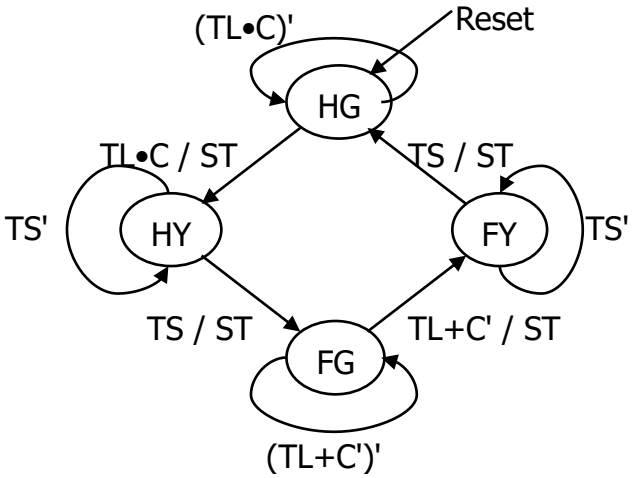
2）选做实验：交通灯控制器 2

1.实验目的

设计高速公路/农场十字路口相交的交通灯控制器

2.实验原理

利用一个有限状态机，实现一个交通灯控制器，要求灯会按照绿-黄-红循环变化，并且绿-45s，黄-15s，红-60s。并且该红绿灯控制器有优先级，用结构模块实现。



按照状态图中的条件变化。
该实验与上一个实验类似，加上了一个 Car 的优先级，即当农场方向有车来时，才开始正常红绿灯运作，当在状态 3 没有车来时，则立刻回复黄灯，具体细节见实验过程。

3.实验过程

①输入与输出
输入：TS：时间是否达到 9s　TL：时间是否达到 3s　RESET:清零　C：是否有车
输出：ST：counter 清零　lights：LED 灯
②counter 文件与上一个实验相同，利用计数器的原理，判断时间到达多少。注意 ST 是清零端，当 ST=1 时，则清理 TS 和 TL 当前值
③状态图
根据状态图可知有四个状态：
HG = 6'b001100　HY = 6'b010100　FG = 6'b100001　FY = 6'b100010

状态变化的条件分别为：
HG -> HY：时间到达 9s 且此时有车 TL&C
HY -> FG：时间到达 3s TS
FG -> FY：时间到达 9s 或此时没有车 TL|~C
FY -> HG：时间到达 3s TS
注意：当在 FG 状态时，此时没有车则立刻 F 变为黄灯
④ST 的控制
每次状态发生变化时，ST 均置为 1，使得 counter 清零，可以保证下一个 TS 和 TL 的输入
⑤parameter
用 parameter 保存一个常二进制量非常方便，可以直接用于 state 的赋值，很好用

4.实验结论

对比上一个实验可知，在状态 2、3 的变化中有优先级的区别。结果跟预期一样。

5.实验感想

这次实验充分理解了摩尔状态机和米里状态机如何用 verilog 语言实现，并且成功实现了一个红绿灯控制器。这次实验也加深了对状态机的理解，并且能够使用更加规整的模块化语言描述一个部件。比如这次的 clkdiv 是使用了上次实验中的分频文件，由于加深了对时间分频的理解，所以自己写了一个 1hz 的分频方式，然后去除了冗余的分频输入和输出内容。对于 counter 的设计，有点类似 LS163 的方式，不过这次学会了用 TS 和 TL 仅仅表示是否达到 counter 中的特殊值 9s 和 3s，这种方式是由于不需要了解 counter 实现细节，只需要了解触发状态变化的时间条件特殊值即可。

对于 traffic 实现细节中，学会用了 parameter 表示一个比较复杂的状态，然后使用了 posedge clk or posedge reset 的方法，既不影响 clk 的正常使用还能直接进行清零的操作，加强了 verilog 语法的理解。

第七次实验：

1）必做实验：总线实验

1.实验目的

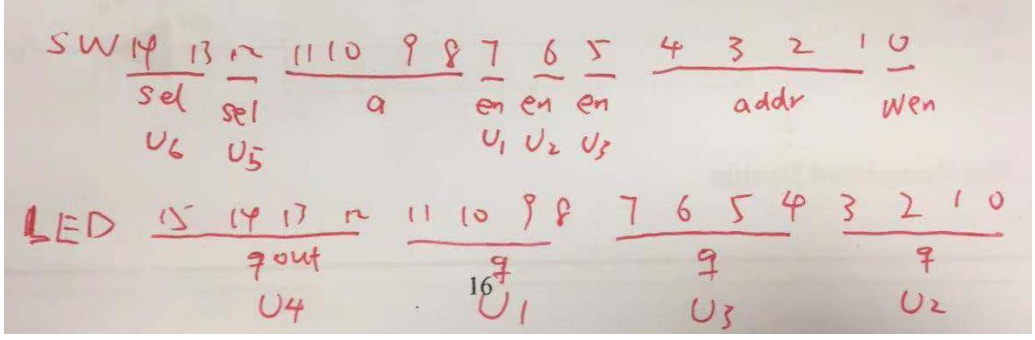
实现一个系统，系统由三个 74377 寄存器（U1，U2，U3），一片 RAM（U4）一个 2：1 多路器（U5），一个 4：1 多路器（U6）及时钟分频电路（U0）组成。

2.实验原理

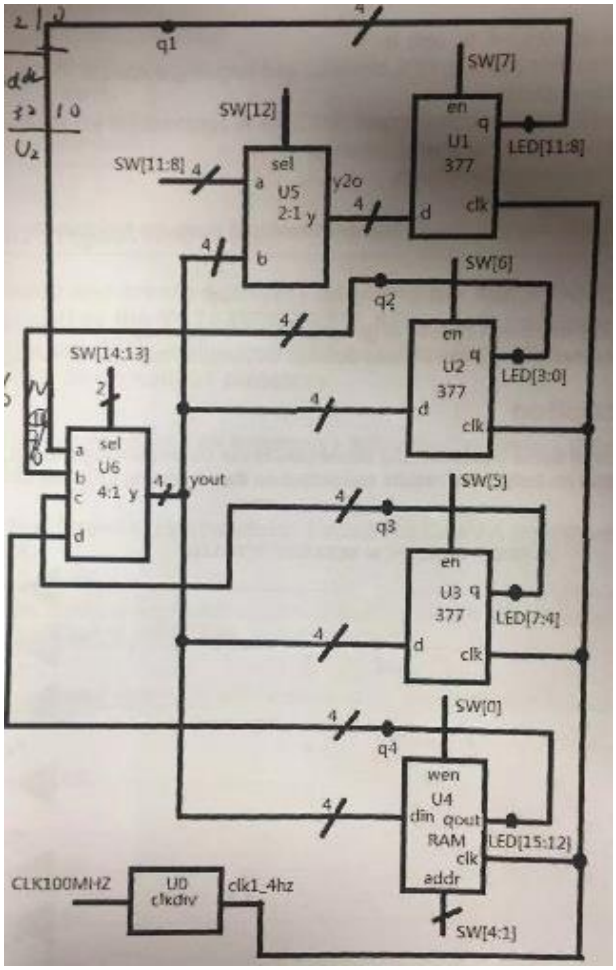
①输入和输出

利用模块例化，分别使用 U0-U6 几个模块，分别实现一些功能，使得最终能够完成“x 存入 U2，把 y 存入 U3；以 U4 为暂存处，实现 U2 及 U3 寄存器内容的交换”功能。

输入和输出对应：



②总线原理图



3.实验过程

利用模块化的思想分别实现各个模块，再进行整合。

①74377 寄存器：思路简单，当 en=1 时保持原值，当 en=0 时进行赋值。

②RAM：和寄存器比较类似，wen 控制是否保持原值，addr 表示储存的位置，赋值方式使用 always 和 case 语句。

③2:1 多路器：sel 选项控制输出 a 还是 b

④4:1 多路器：sel 选项控制输出四个中的哪个。

⑤分频电路：clkdiv.v 用于实现 100MHZ 分频为 1hz

⑥总线：将每个模块，按照原理图中的输入输出链接起来即可。中间定义临时变量 ABCD。

4.实验结论

先利用 U5 和 U1 输入 temp1，保存在寄存器 U2 中，然后输入 temp2 保存在寄存器 U3 中，再调整 4:1 多路选择器，将 U2 输入到内存中，再用 U3 赋值 U2，用内存中保存的赋值 U3，最终实现了 temp1 和 temp2 的交换。

5.实验感想

本实验再次充分体现了模块例化的方法，并且发挥到了极致。按照题目要求，先分别实现每个小模块的功能，每个小功能，比如寄存器、多路复用器等都是我们熟知之前实验做过的，所以比较容易就能写出来。关键部分就是把这几个模块链接起来，需要模块例化几个不同的实例，利用临时变量 reg 或 wire 将他们连起来，最终再控制输入输出，就比较容易地做出来。

虽然最后还是没有做出这个实验的选做实验，利用有限状态机和总线进行改进，但是原理大概能够理解，只是代码实现比较繁琐，在代码改过去改过来的过程中，出现了很多应该避免的 bug。

总的来说这次实验理解还是比较透彻了，代码实现方面还有待提升。