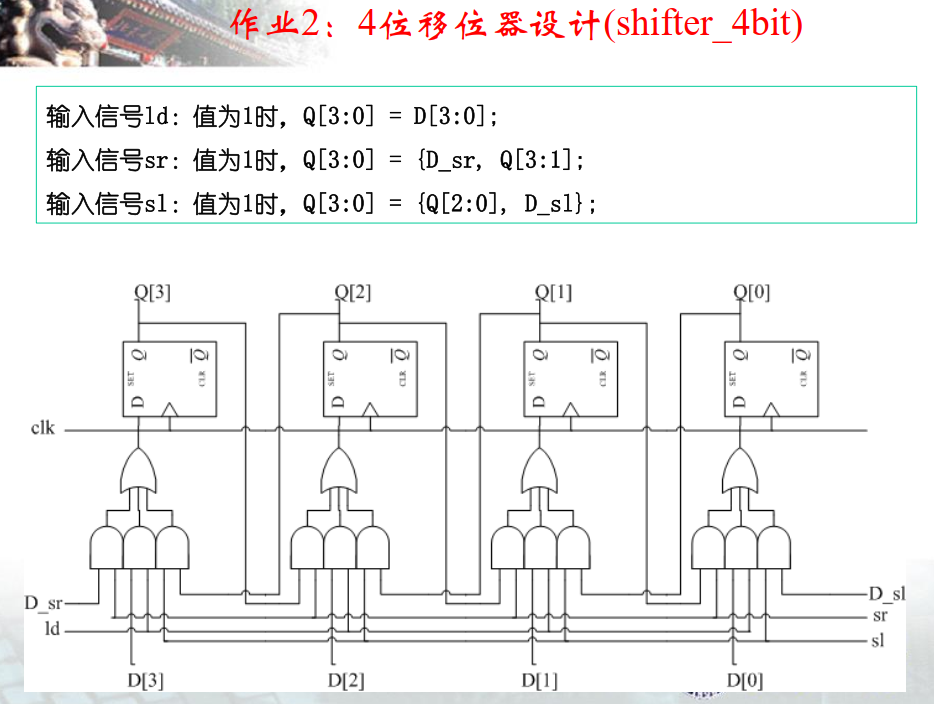
**序列检测器、移位寄存器作业**

**4位移位器设计**

**目标分析**

设计目标为完成一个可支持左移并移入新数据、右移并移入新数据、LOAD功能的4位移位器。



该电路图所描述的电路，**如果输入信号ld、sr、sl有两个或者两个以上信号同时有效，则输出Q为对应情况的结果相或，无法同时得到想要的结果**。并且D触发器的CLR端未接入复位信号。

**因此在电路图上，增加以下功能模块设计：**

* 优先级控制电路

由输入信号 [2:0] prior\_con，控制ld、sr、sl的优先级，使得有两个或者两个以上信号同时有效时，只有具有最高优先级的信号有效。

输入信号 [2:0] prior\_con对应的ld、sr、sl优先级关系，如下表所示：



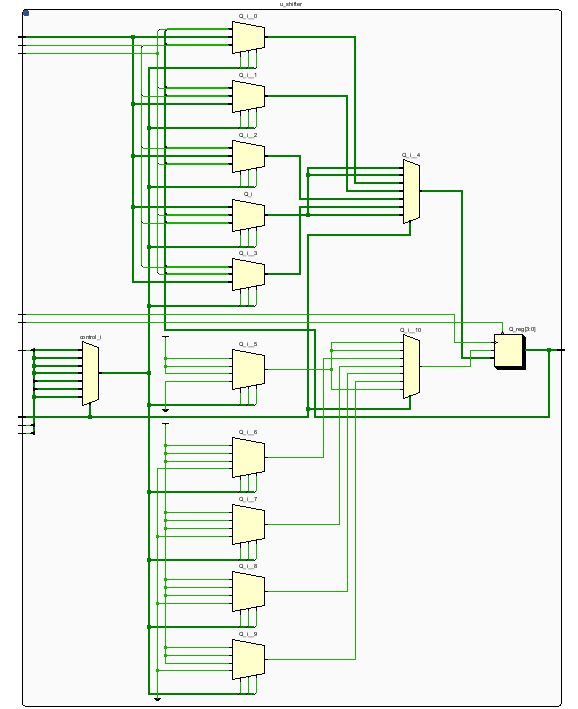
**点击图片可查看完整电子表格**

* 给D触发器加入了异步复位信号CLR

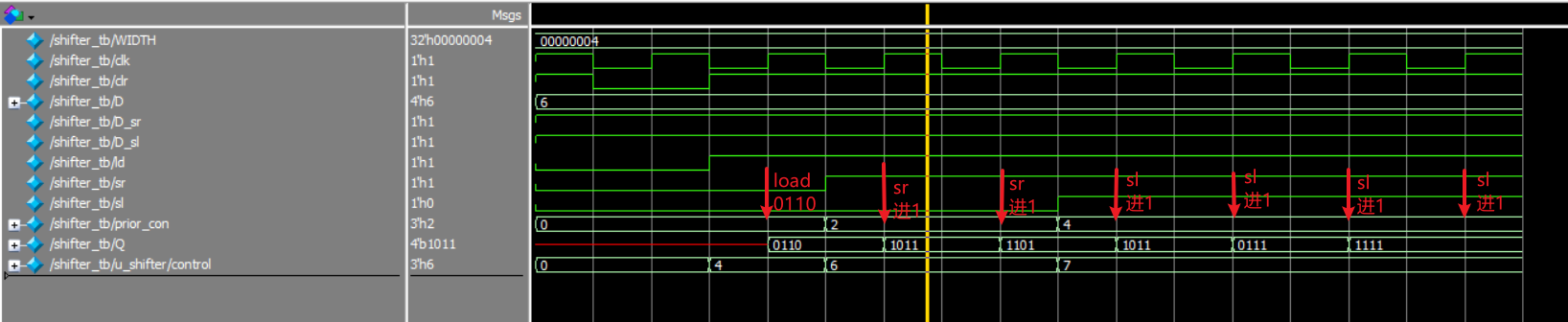
使之符合实际电路具有的复位要求。

**电路结构**

RTL编写完成后，得到的电路结构如下图所示：



**Testbench验证**



分析电路工作时各个cycle：

1. 控制优先级为ld、sr、sl，给一个ld有效信号：Load 4‘b0110，成功
2. 改变优先级为sr、ld、sl，给ld、sr同时有效信号：右移进1，成功
3. 保持2的激励，再次右移进1，成功
4. 改变优先级为sl、ld、sl，给ld、sr、sl同时有效信号：左移进1，成功
5. 保持4的激励，连续三次左移进1，成功

因此优先级控制电路和双向移位器电路功能正常，Testbench通过。

**序列检测器设计**

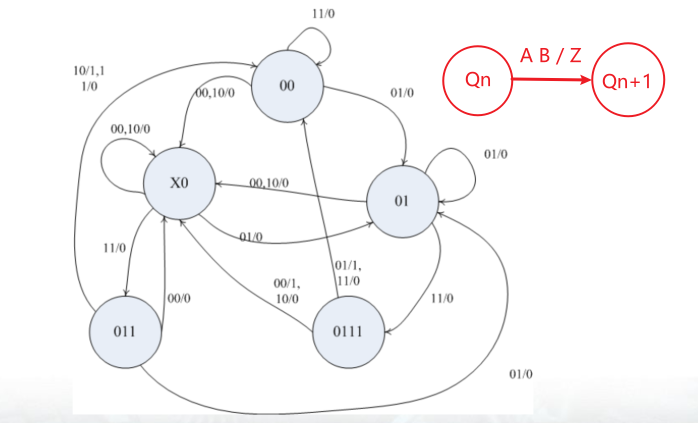
**目标分析**

设计目标为完成一个01110的序列检测电路，数据以两位一组的顺序送入电路，D0送入到A，D1送入到B，以此类推，每发现一个“01110”相邻的“01110”不重叠，在Z输出一个时钟周期宽度的高电平脉冲，也就是说如果序列为“01110 111 01110 01110 x01110”那么应当仅能检测到第一个、第三个、第四个、第五个“01110”序列，因为序列的不重叠特性，因此第二个“01110”序列不应该被检测到。

按照课程所给的米里型状态机状态转换图进行电路设计。使用三段式状态机描述该电路。

**过程注意点**

1. 注意状态转换图的图例



如右上角添加的图例所示，首先应注意到状态转换图中输入的两位数字从左到右分别对应A和B。

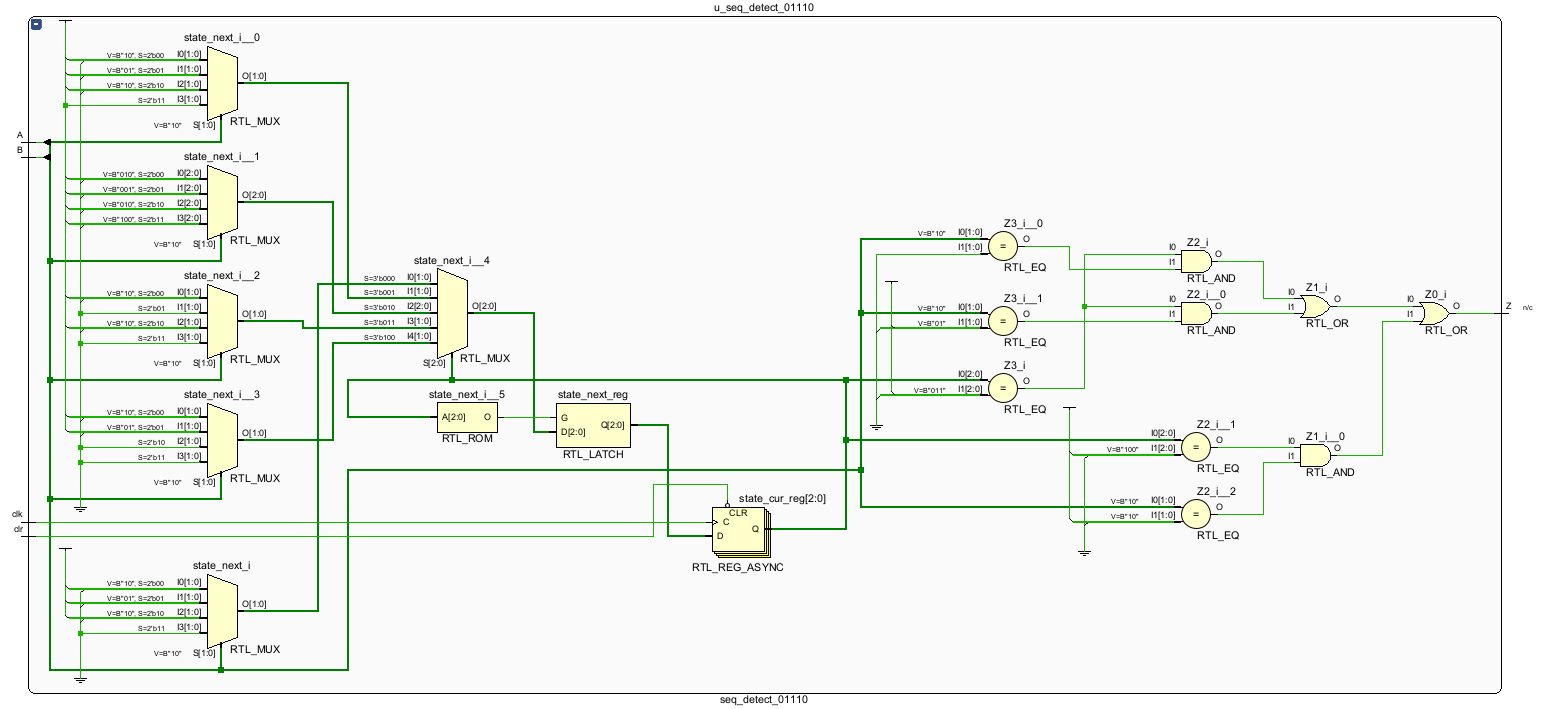
1. 一种常用的在波形中显示当前状态机的状态名称的方法

代码示例：

|  |
| --- |
| reg [ 6 \* 8 - 1 :0] monitor\_state;  // This reg can be used to show current state name in wave.  //Notion: One ASCII code is 8bit, make sure the width of monitor\_state is enough! always @(\*) begin  case(state\_cur)  IDLE : monitor\_state = "S\_00 " ;   S\_01 : monitor\_state = "S\_01 " ;   S\_X0 : monitor\_state = "S\_X0 " ;   S\_0111 : monitor\_state = "S\_0111" ;  S\_011 : monitor\_state = "S\_011 " ;  default : monitor\_state = "? " ;  endcase  end |

**电路结构**

RTL编写完成后，得到的电路结构如下图所示：



**Testbench验证**

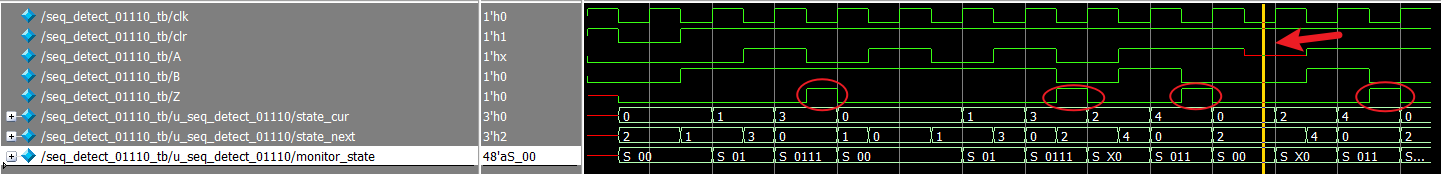
按照输入序列“01110 111 01110 01110 x01110”，编写Testbench进行验证。

值得一提的是两个点：

1. 输入序列“01110 111 01110 01110 x01110”中的x实际也是按照x进行赋值，而非具体的0或者1值。
2. 在TB中加入了显示状态机当前状态名称的以下代码，以增强波形信号的可读性：

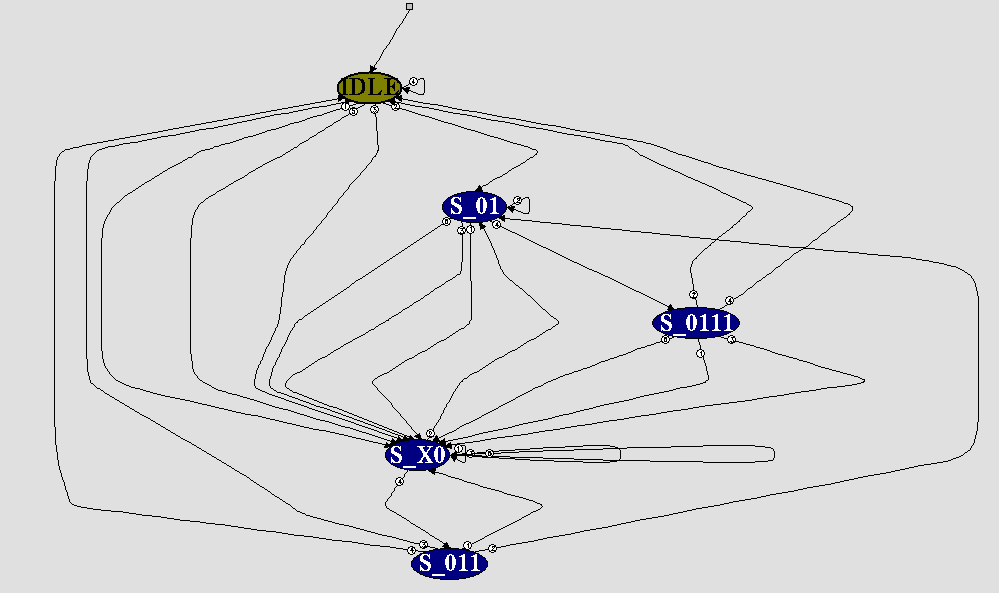
|  |
| --- |
| reg [ 6 \* 8 - 1 :0] monitor\_state; // Reg monitor\_state can be used to show state name in wave. Notion: One ASCII code is 8bit,make sure the width is enough always @(\*) begin  case(u\_seq\_detect\_01110.state\_cur)  IDLE : monitor\_state = "S\_00 " ;   S\_01 : monitor\_state = "S\_01 " ;   S\_X0 : monitor\_state = "S\_X0 " ;   S\_0111 : monitor\_state = "S\_0111" ;  S\_011 : monitor\_state = "S\_011 " ;  default : monitor\_state = "??? " ;  endcase  end |

仿真结果如下图所示：



与之前的分析相同，仿真结果中在正确位置共产生了四次有效输出，且观察到每个时钟上升沿，monitor\_state信号显示的状态变化均符合设计要求，因此电路功能验证成功。

**状态转换图**



用verdi查看的状态转换图如图所示，与预期相符。

**TODO**

以后有时间，可以利用C或者python将随机序列数据写入一个txt文档，再利用查找功能检索有多少个不重叠的“01110”，将结果与通过readmem系统函数读入电路的仿真测试结果进行对比，以验证电路功能。

**在完成作业之后的思考**

* 摩尔型和米里型状态机的区别？

对于状态数量：同样的电路功能，摩尔型需要的状态数量一般比米里型多。例如本次作业，经过推导后，摩尔型需要8个状态描述，而米里型需要5个状态。

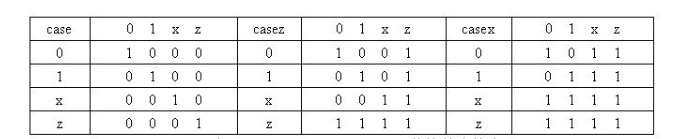
对于输出信号的毛刺：米里型的输出信号与输入信号有关，有可能出现毛刺。而摩尔型不会。

* 做一个可以实现序列数据位重叠情况检测的米里型序列检测器，需要多少个状态？

这个问题，之后有时间自己推导一下。目前凭直觉推测是，允许数据位重叠，那么状态机的状态数量应该比不重叠的要多。

* case，casex，casez在simulation/synthesis的区别？

1. 对于仿真情况，真值表如下图所示



其中，? 是 高阻 z 的另一种写法。

case是全等比较，casez忽略z/？的比较，casex忽略x,z/?的比较。

1. 对于综合情况。case，casez和casex综合的结果是一致的。

虽然case在仿真时是全等比较，但是在综合时，case的索引列表里面的x和z/？，都被综合工具认为是不可达到的状态就被去掉了，也就是说，case和casex的综合结果是相同的。

而casez和casex里面的x , z/？都被认为是don’t care，所以casez和casex的综合结果也是相同的。

1. 因此结论是，在同样的代码下，case、casez、casex的综合结果都相同，综合器都会don’t care 里面的x , z/？。但是当索引包含x , z/？时，case、casez、casex的仿真真值表不同，严格遵守上面的真值表。
2. 举个例子来说，在写优先编码器时，如果用casex或者casez来写，case item的项数会少很多，让代码非常简洁：

|  |
| --- |
| module encoder83( input wire [7:0] a, output reg [2:0] o ); always @(a)  casez(a)  8'b1???\_????: o = 3'b111;  8'b01zz\_zzzz: o = 3'b110;  8'b001?\_????: o = 3'b101;  8'b0001\_????: o = 3'b100;  8'b0000\_1???: o = 3'b011;  8'b0000\_01??: o = 3'b010;  8'b0000\_001?: o = 3'b001;  8'b0000\_0001: o = 3'b000;  default: o = 3'b0;  endcase endmodule |

如果以上代码用casex来写，那么里面的?或者z就可以替换为x，因为x也是被don't care的。代码如下：

|  |
| --- |
| always @(a)   casex(a)  8'b1xxx\_xxxx: o = 3'b111;   8'b01??\_????: o = 3'b110;  8'b001z\_zzzz: o = 3'b101;  8'b0001\_xxxx: o = 3'b100;   8'b0000\_1xxx: o = 3'b011;   8'b0000\_01xx: o = 3'b010;   8'b0000\_001x: o = 3'b001;   8'b0000\_0001: o = 3'b000;  default: o = 3'b0; endcase |

如果以上代码用case来写：

|  |
| --- |
| always @(a)   case(a)  8'b1xxx\_xxxx: o = 3'b111;   8'b01??\_????: o = 3'b110;  8'b001z\_zzzz: o = 3'b101;  8'b0001\_xxxx: o = 3'b100;   8'b0000\_1xxx: o = 3'b011;   8'b0000\_01xx: o = 3'b010;   8'b0000\_001x: o = 3'b001;   8'b0000\_0001: o = 3'b000;  default: o = 3'b0; endcase |

仿真时如果想执行正确的行为，这个模块所需要的激励a的输入是极其不方便的。a里面的0或者1是无法和case item的index当中的x或者z/？对应上的，除非a里面就严格正好是包含x，z/？的，如a=8'b1xxx\_xxxx，那么就能execute 8'b1xxx\_xxxx这个item。所以这给仿真带来很大不便利性，很不实用。因此涉及到到无关值时还是要用casex或者casez来写。

但值得注意的是，上面这三段代码，在综合后，结果均是一样的。