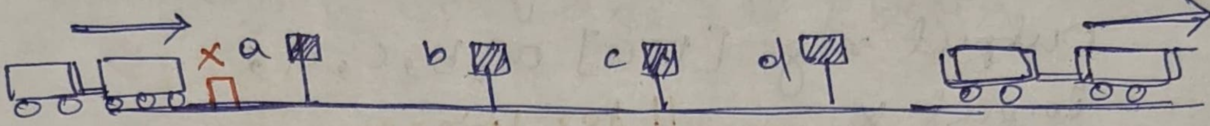
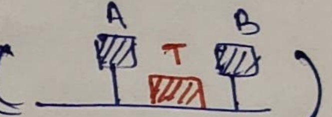
Automated Railway Signalling

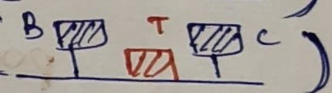
- lets consider 4 Railway signals (a, b, c, d) in a straight track at Equal distance

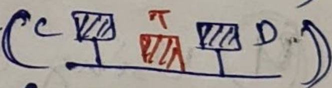


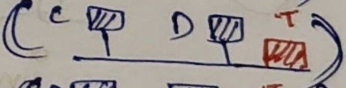
- Each signal has 4-coloured states (Red-R, Caution-YY, Yellow Y, Green-G)
- There is also a Trigger (x) which triggers when a train arrives and sends signal to the 4 light signal. Sequence as follows:-

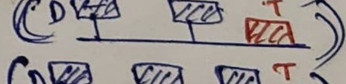
1. initially all the 4 signals will be idle (or all green light) ⇒ S0 ⇒ (a=g, b=g, c=g, d=g)
2. when train hits Trigger idle state overs and series goes like

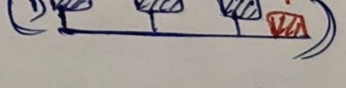
S1 ⇒ a=R, b=g, c=g, d=g ()

S2 ⇒ a=YY, b=R, c=g, d=g ()

S3 ⇒ a=Y, b=YY, c=R, d=g ()

S4 ⇒ a=g, b=Y, c=YY, d=R ()

S5 ⇒ a=g, b=g, c=Y, d=YY ()

S6 ⇒ a=g, b=g, c=g, d=Y ()

(back to ideal) ⇒ S0

($S_0 \xrightarrow{x=1} S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_5 \rightarrow S_6 \xrightarrow{rst} S_0 \xrightarrow{x=1} S_1 \rightarrow S_2 \cdots$)

Verilog Coding

① module descriptⁿ

```
module auto_rail_signal(
    output reg [1:0] a, b, c, d,
    input x,        // for trigger
    input clk, clr);        //rst = clr
```

② parameter defining

```
parameter r = 2'd0;
   "      yy = 2'd1;
   "      y = 2'd2;
   "      g = 2'd3;
```

③ defining States

```
parameter s0 = 3'd0; //(a,b,c,d) = g
   "      s1 = 3'd1;  }
   :      s2 = 3'd2;  }  as per described
   :       :          }  in prev. pg.
parameter s6 = 3'd6;  }
```

④ Using constant intervals

```
integer g2r delay = 10;   // G → R = 10sec
   "    r2yy delay = 10;  // R → YY = 10sec
   "    yy2y delay = 10;  // YY → Y = 10sec
   "    y2g delay = 20;   // Y → g = 20sec
```

(5) declaring state Registers

```
reg [2:0] state, next_state;
```

(6) synchronising state reg

```
always @(posedge clk) begin
  if(clr)
     state <= 0;     // idle state if clr = 1
  else
     state <= next_state;   // if clr 0 → next state
                                            transitⁿ
end
```

(7) o/p logic

```
 always @ (state) begin
    a = g ; b = g ; c = g; d = g;    // default
    case (state)
       s0 : begin
          a = g ; b = g; c = g; d = g;
       end
       s1 : begin
          a = r ; b = g ; c = g; d = g ;
       end
        ⋮     (s2 , s3 , s4 , s5)
       s6 : begin
          a = g ; b = g ; c = g; d = y ;
       end
    endcase
 end
```

8 Next State Logic

```
always @ (state or x) begin
  case (State)
     S0 : begin
        if (x)
           next_state = S1 ;
        else
           next_state = S0;
     end
     S1 : begin
(a)*  {  repeat (g2r delay) @(posedge clk);
      {  next_state = S2;
     end
     S2 : begin
(a)*  {  repeat (r2yy delay) @(posedge clk);
      {  next_state = S3 ;
     end
   * : { S3 => yy2y delay => n.s = S4      //(a)
       { S4 => y2g delay => n.s = S5       //(a)
     S5 : begin
(10 sec)* { repeat ( g2r delay)@(posedge clk)
          { next_state = S6 ;
     end
     S6 : begin                           train there
        if (x)
           next_state = S6 ;     // if x = trigger = 1
        else
           next_state = S0 ;     // else x = 0
     end
     default : next_state = S0 ;
endcase
```