```
module arbiter ( clock, reset, R0, R1, G0, G1);

input clock;

input reset;

input R0;

input R1;

output G0, G1;

parameter [1:0] //synopsys enum states

s0=2'b00,

s1=2'b01,

s2=2'b10,

s3=2'b11,

reg [1:0] /*synposys enum states*/ current_state, next_state;

reg G0, G1;

/*---------sequential logic------*/

always@ ( posedge clock or negedge reset)

if(!reset ) current_state <= s0;

else current_state = next_state;

/*next state logic and output logic combined so as to share state decode logic*/

always@ (current_state or R1 or R0)

begin

G0=0; G1=0; /*defaults to prevent latches*/

case (current_state) // Synopsys full_case parallel_case
```

```
s0:  begin
      G0=0; G1=0;
 if ( R0 and R1) next_state = s3;
else if(R0) next_state = s1;
else if(R1) next_state = s2;
else next_state = s0;
end
s1:  begin
      G0=1; G1=0;
 if ( R0 and R1) next_state = s3;
else if(R0) next_state = s1;
else if(R1) next_state = s2;
else next_state = s0;
end
s2:  begin
      G0=0; G1=1;
 if ( R0 and R1) next_state = s3;
else if(R0) next_state = s1;
else if(R1) next_state = s2;
else next_state = s0;
end
s3:  begin
```

```
      G0=1; G1=0;
 next_state = s2;
end
default : next_state = s0;
endcase
end
endmodule
```