Counters

* Designing counters using Finite State Machines

* Designing counters using case statements

* Designing counters using if statements

- Ring counter
- Johnson counter
- Jerky counter

Moore FSM

Example (1) :

Reset

	8-bit output								Decimal # of the output
0	0	0	0	0	0	0	0	1	1
1	0	0	0	0	0	0	1	0	2
2	0	0	0	0	0	0	0	1	1
3	0	0	0	0	0	1	0	0	4
4	0	0	0	0	0	0	0	1	1
5	0	0	0	0	1	0	0	0	8
6	0	0	0	0	0	0	0	1	1
7	0	0	0	1	0	0	0	0	16
8	0	0	0	0	0	0	0	1	1
9	0	0	1	0	0	0	0	0	32
10	0	0	0	0	0	0	0	1	1
11	0	1	0	0	0	0	0	0	64
12	0	0	0	0	0	0	0	1	1
13	1	0	0	0	0	0	0	0	128

```
module counter ( output reg [7:0] count, input clk, reset );
reg [3:0] state, next_state ;
  always @ (posedge clk)
    if (reset) state <= 0 ;
    else       state <= next_state ;
  always @ (state) begin
      next_state = 0 ; count = 1 ;
    case (state)
      0 : begin  next_state = 1 ; count = 1 ; end
      1 : begin  next_state = 2 ; count = 2 ; end
      2 : begin  next_state = 3 ; count = 1 ; end
      3 : begin  next_state = 4 ; count = 4 ; end
      4 : begin  next_state = 5 ; count = 1 ; end
      :
      13 : begin next_state = 0 ; count = 128 ; end
                                            default : begin next_state = 0 ; count = 0 ;
                                                                                  end
                                              endcase
                                              end
                                              endmodule
```

The counter module does not have an input that determines what the next_state would be.

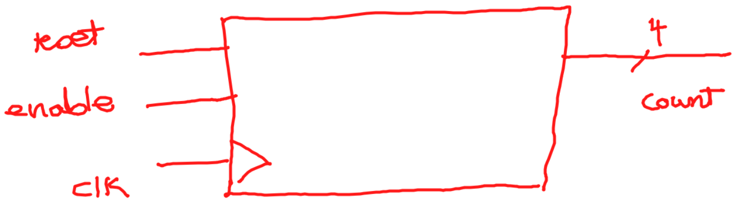
Example (2) :

```
module  Johnson_counter  ( output reg [3:0]  count,
                           input             enable ,
                           input             clock , reset ) ;

always @ (posedge clock , posedge reset )
   if (reset==1)   count <=0 ;
 else if (enable)
     case (count)
       4'b 0000 , 4'b0001 , 4'b0011 , 4'b0111 :  count <= { count [2:0] , 1'b1 } ;
       4'b 1000 , 4'b1110 , 4'b1100 , 4'b 1000 :  count <= { count [2:0] , 1'b0 } ;
       4'b 1111 ,
                        default    :  count <=0 ;

     endcase

endmodule
```

If the counter is enabled

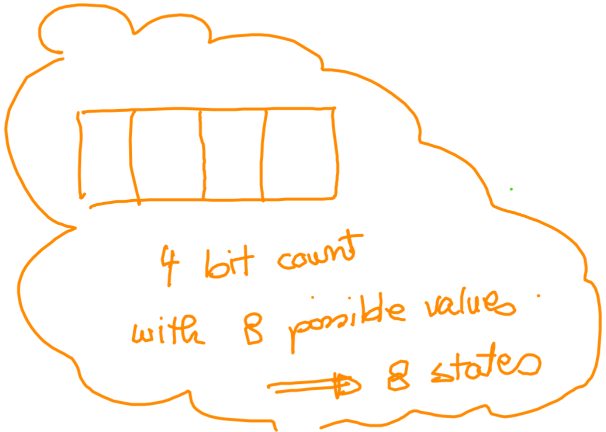
Reset

	State	Binary count				Decimal
000	0	0	0	0	0	0
001	1	0	0	0	1	1
.	2	0	0	1	1	3
.	3	0	1	1	1	7
.	4	1	1	1	1	15
.	5	1	1	1	0	14
.	6	1	1	0	0	12
111	7	1	0	0	0	8



Using a Moore FSM to design the Johnson_counter

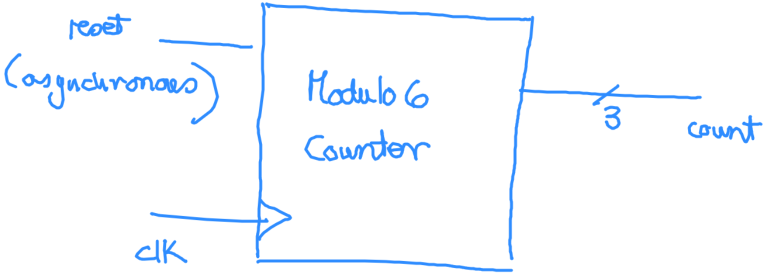
```
module   Johnson_counter_FSM  ( count, clk, reset, enable );
    output reg [3:0]  count ;
    input              clk , reset , enable ;

   reg [2:0]  state , next_state ;

   always @ ( posedge clk , posedge reset )
        if  (reset)    state <= 0 ;
   else if  (enable)   state <= next_state ;
```



4 bit count
with 8 possible values
⟹ 8 states

```
          always @ (state) begin

           case (state)
     0 : begin next_state = 1 ; count = 0 ;
     1 : begin next_state = 2 ; count = 1 ;
     2 : begin next_state = 3 , count = 3 ;
     3 : begin next_state = 4 ; count = 7 ;
     4 : begin next_state = 5 ; count = 15 ;
     5 : begin next_state = 6 ; count = 14 ;
     6 : begin next_state = 7 ; count = 12 ;
     7 : begin next_state = 0 ; count = 8 ;
default : begin next_state = 0 ; count = 0 ; end
       endcase  end
   endmodule.
```

Example (3) :



Reset →

0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1

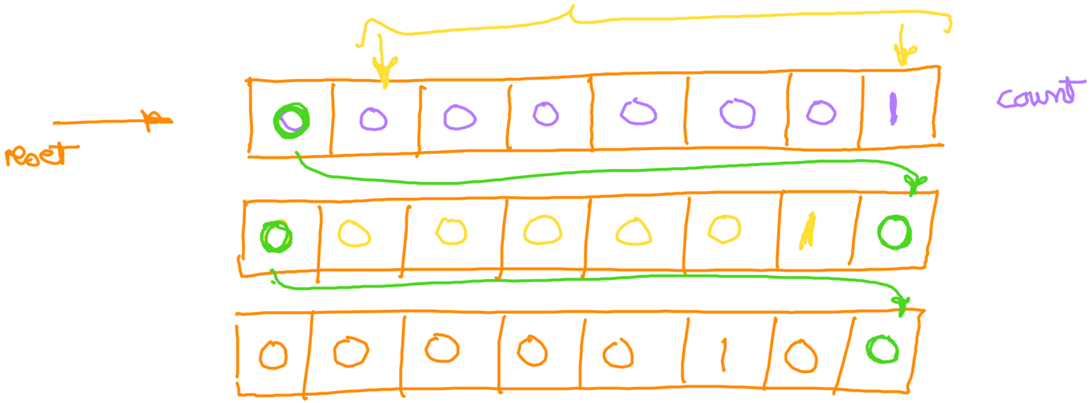
1. FSM (Moore)
2. Case structure that depends on the count.
3. if statement

```
module    Modulo_6_Counter (output reg [2:0] count,
                            input clk, reset);

always @ (posedge clk , posedge reset)
    if (reset)          count <= 0 ;
else if (count < 5)     count <= count + 1 ;
else                    count <= 0 ;

endmodule
```

Example (4) :

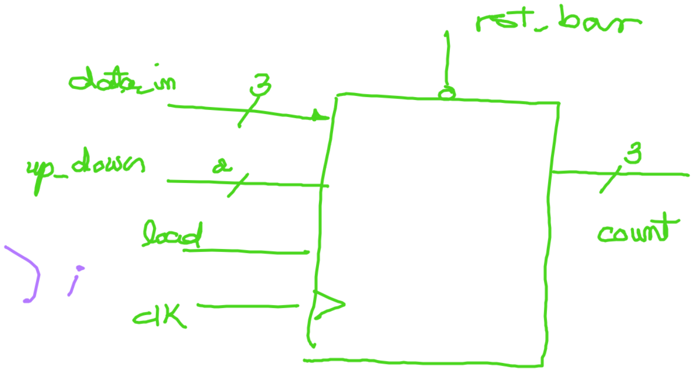
```
module Ring_counter #( parameter word_size = 8)(
                      output reg [word_size -1 : 0] count ,
                      input enable , clk , reset );

always @ (posedge clk , posedge reset)
    if      (reset)  count <= { {(word_size-1){1'b0}}, 1'b1 };
    else if (enable) count <= { count [word_size -2 : 0] , count [word_size -1] };

endmodule
```

MSB of the count looped back to the LSB position



Example (5) :

```
module    counter   ( output reg   [2:0]   count ,
                      input        [2:0]   data_in ,
                      input        [1:0]   up_down ,
                      input                load , clk , rst_bar ) ;

always @ (negedge clk , negedge rst_bar)
     if (rst_bar == 1'b0)                                   count <= 0 ;
else if (load == 1'b1)                                      count <= data_in ;
else if (up_down == 2'b00) || (up_down == 2'b11)            count <= count ;
else if (up_down == 2'b01)                                  count <= count + 1 ;
else if (up_down == 2'b10)                                  count <= count - 1 ;

endmodule
```


rst_bar
data_in
3
up_down
2
load
clk
3
count