EECS 31L: Introduction to Digital Design Lab Lecture 6

Pooria M.Yaghini

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 6: Outline

- Finite State Machine

Lecture 6: Finite State Machine

Digital circuits can be classified as:

- Combinational
- Sequential

Lecture 6: Finite State Machine

Digital circuits can be classified as:

- Combinational
- Sequential

So what is "**finite state machine (FSM)"?**

- A third category?

Lecture 6: Finite State Machine

Digital circuits can be classified as:

- Combinational
- Sequential

So what is "**finite state machine (FSM)"?**

- A third category?
- No! Just a modeling (design) technique for sequential circuits

When is this technique recommended?

- For any sequential circuit?
- For certain sequential circuits?

Lecture 6: Finite State Machine

The FSM approach is recommended when:

Lecture 6: Finite State Machine

The FSM approach is recommended when:

1) The construction of a list with all system states is viable. (i.e., the states are well defined and the list is not too long)

Lecture 6: Finite State Machine

The FSM approach is recommended when:

1) The construction of a list with all system states is viable. (i.e., the states are well defined and the list is not too long)
2) All output values that must be produced by the system (in all states) are easily enumerable. The list must be the same in all states.

Lecture 6: Finite State Machine

The FSM approach is recommended when:

1) The construction of a list with all system states is viable. (i.e., the states are well defined and the list is not too long)
2) All output values that must be produced by the system (in all states) are easily enumerable. The list must be the same in all states.
3) All conditions for the machine to move from one state to another (in all states) are also easily enumerable.

Lecture 6: Finite State Machine

The FSM approach is recommended when:

1) The construction of a list with all system states is viable. (i.e., the states are well defined and the list is not too long)
2) All output values that must be produced by the system (in all states) are easily enumerable. The list must be the same in all states.
3) All conditions for the machine to move from one state to another (in all states) are also easily enumerable.

Lecture 6: Finite State Machine

The FSM approach is recommended when:

1) The construction of a list with all system states is viable. (i.e., the states are well defined and the list is not too long)
2) All output values that must be produced by the system (in all states) are easily enumerable. The list must be the same in all states.
3) All conditions for the machine to move from one state to another (in all states) are also easily enumerable.

Classical example: All sorts of controllers

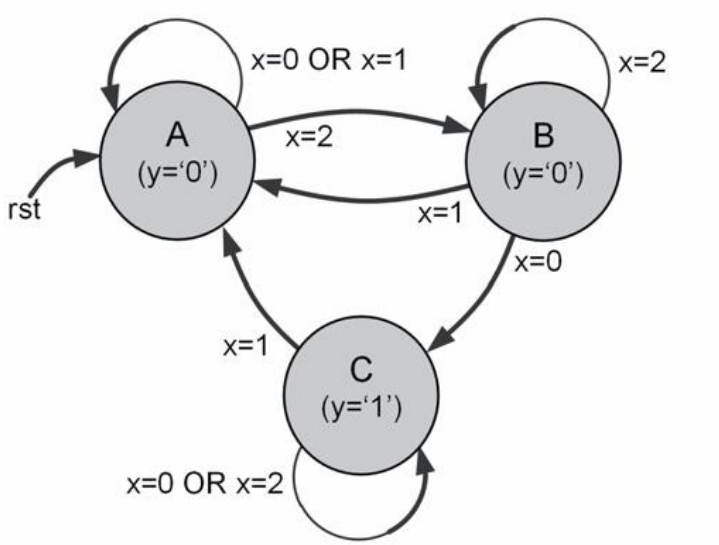
- Traffic-light controller
- Elevator controller
- Control unit for microcontroller datapath

Lecture 6: Finite State Machine

FSM representations:

- From a specifications perspective: **State transition diagram**

State Transition Diagram

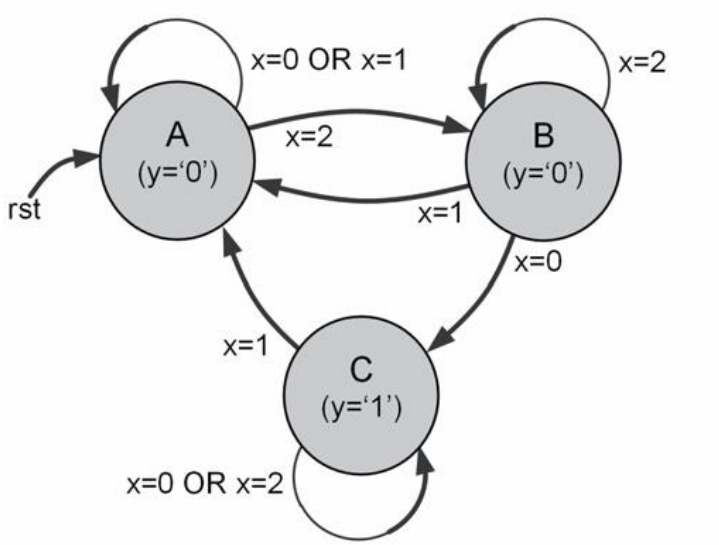


Lecture 6: Finite State Machine

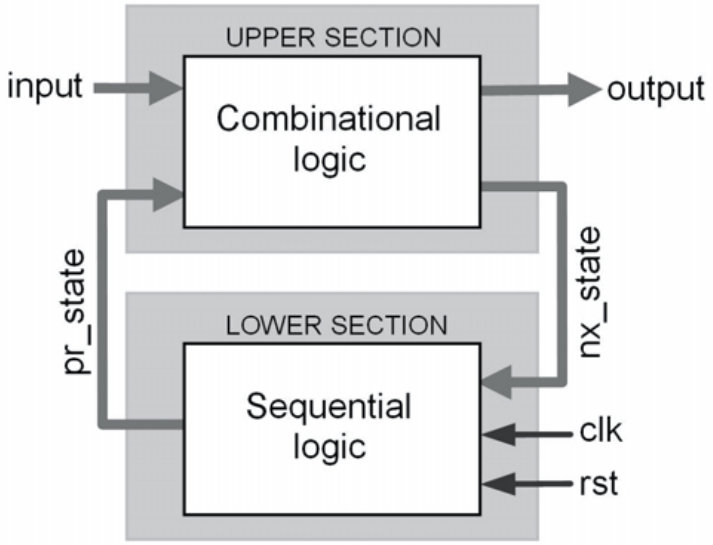
FSM representations:

- From a specifications perspective: **State transition diagram**
- From a hardware perspective: **Sequential** + **combinational sections**

State Transition Diagram

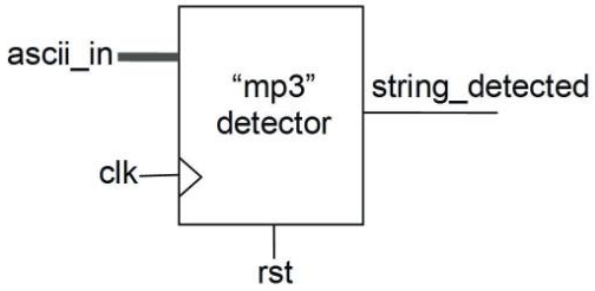


Hardware



Lecture 6: Finite State Machine

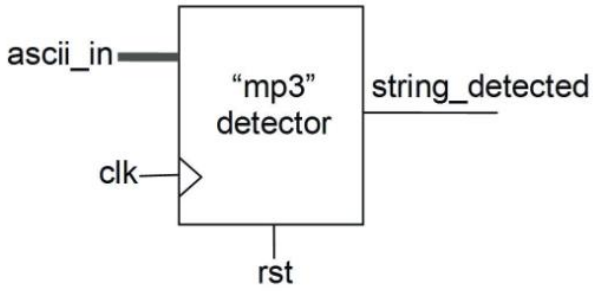
Example: **String detector** (“mp3”)



Data input: ascii_in
Data output: string_detected
Operational inputs: clk, rst

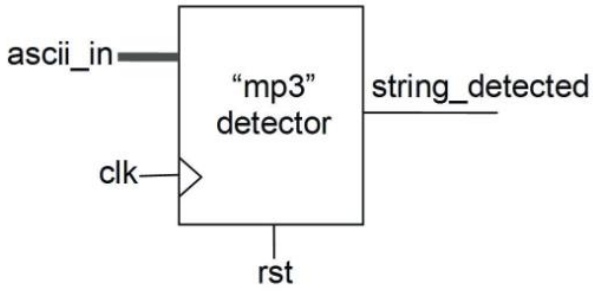
Lecture 6: Finite State Machine

Example: **String detector** (“mp3”)

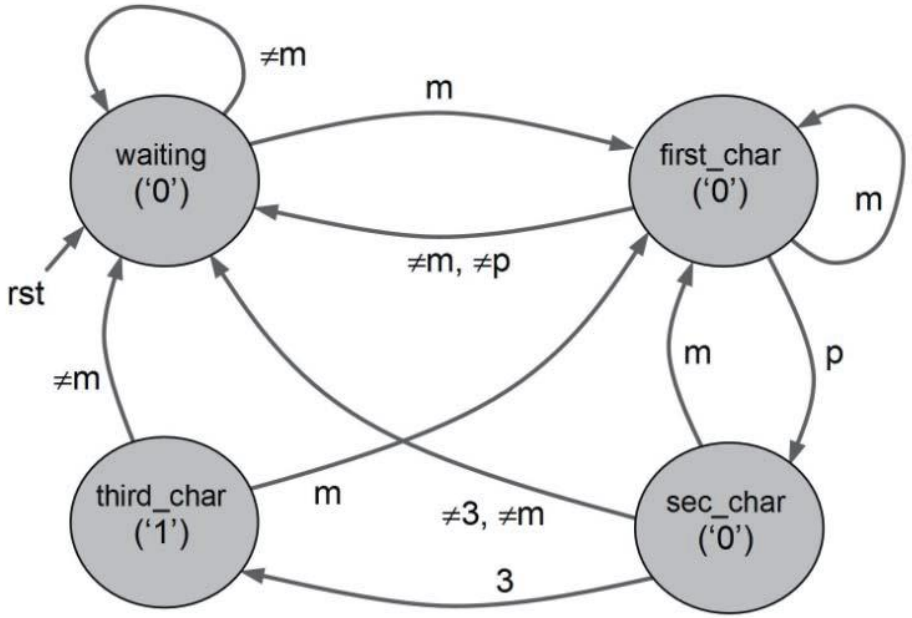


Lecture 6: Finite State Machine

Example: **String detector** (“mp3”)



State transition diagram
(4 states, Moore type)



Data input: ascii_in
Data output: string_detected
Operational inputs: clk, rst

Lecture 6: Finite State Machine

Manual design

Five-step design procedure:

Lecture 6: Finite State Machine

Manual design

Five-step design procedure:

Step 1: Draw state transition diagram.

Lecture 6: Finite State Machine

Manual design

Five-step design procedure:

Step 1: Draw state transition diagram.

Step 2: Write truth tables for nx_state and output.
Then rearrange truth tables replacing state names with corresponding binary values.

Lecture 6: Finite State Machine

Manual design

Five-step design procedure:

Step 1: Draw state transition diagram.

Step 2: Write truth tables for nx_state and output.
Then rearrange truth tables replacing state names with corresponding binary values.

Step 3: Get boolean expressions for nx_state and output.

Lecture 6: Finite State Machine

Manual design

Five-step design procedure:

Step 1: Draw state transition diagram.

Step 2: Write truth tables for nx_state and output.
Then rearrange truth tables replacing state names with corresponding binary values.

Step 3: Get boolean expressions for nx_state and output.

Step 4: Draw circuit, placing all flip-flops (D-type only) in the lower section and the combinational logic (for the expressions above) in the upper section.

Lecture 6: Finite State Machine

Manual design

Five-step design procedure:

Step 1: Draw state transition diagram.

Step 2: Write truth tables for nx_state and output.
Then rearrange truth tables replacing state names with corresponding binary values.

Step 3: Get boolean expressions for nx_state and output.

Step 4: Draw circuit, placing all flip-flops (D-type only) in the lower section and the combinational logic (for the expressions above) in the upper section.

Step 5 (optional): Add DFFs at the output to eliminate glitches.

Lecture 6: Finite State Machine

Manual design

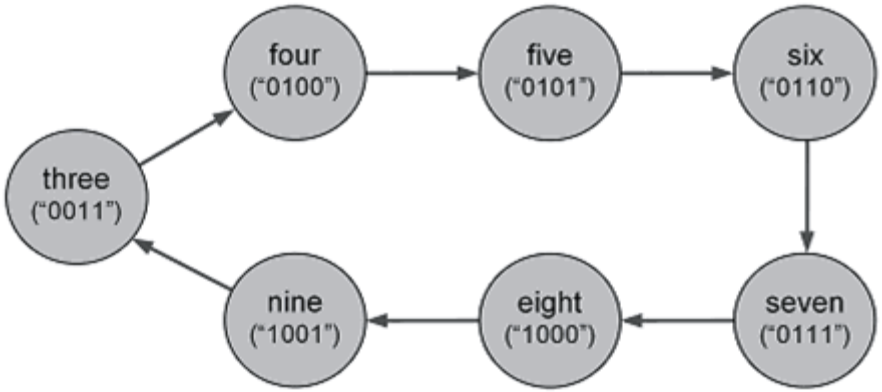
Example: Synchronous 3-to-9 counter

Lecture 6: Finite State Machine

Manual design

Example: Synchronous 3-to-9 counter

Step 1: State transition diagram

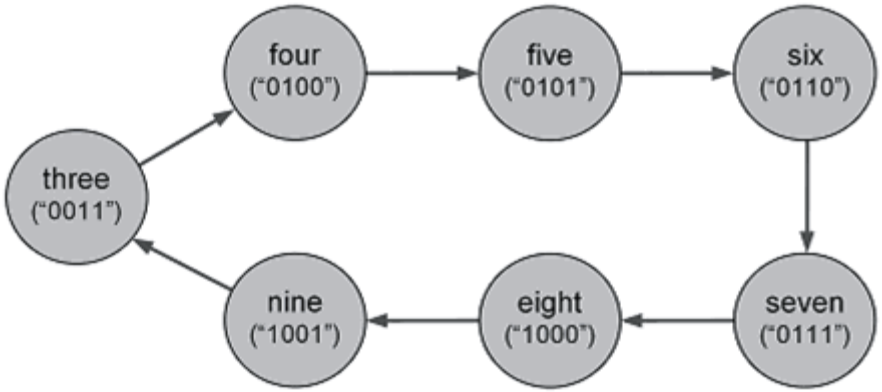


Lecture 6: Finite State Machine

Manual design

Example: Synchronous 3-to-9 counter

Step 1: State transition diagram



Step 2: Truth table

Truth table for nx_state

pr_state $q_3\ q_2\ q_1\ q_0$	nx_state $d_3\ d_2\ d_1\ d_0$
0 0 1 1	0 1 0 0
0 1 0 0	0 1 0 1
0 1 0 1	0 1 1 0
0 1 1 0	0 1 1 1
0 1 1 1	1 0 0 0
1 0 0 0	1 0 0 1
1 0 0 1	0 0 1 1

Lecture 6: Finite State Machine

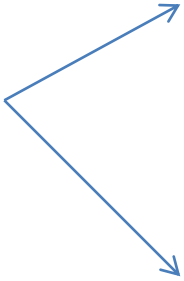
Manual design

Example: Synchronous 3-to-9 counter

Step 3: Boolean expressions

Truth table for nx_state

pr_state $q_3\ q_2\ q_1\ q_0$	nx_state $d_3\ d_2\ d_1\ d_0$
0 0 1 1	0 1 0 0
0 1 0 0	0 1 0 1
0 1 0 1	0 1 1 0
0 1 1 0	0 1 1 1
0 1 1 1	1 0 0 0
1 0 0 0	1 0 0 1
1 0 0 1	0 0 1 1



Karnaugh map for d_3

$q_1\ q_0$ \ $q_3\ q_2$	00	01	11	10
00	X	0	X	1
01	X	0	X	0
11	0	1	X	X
10	X	0	X	X

Karnaugh map for d_2

$q_1\ q_0$ \ $q_3\ q_2$	00	01	11	10
00	X	1	X	0
01	X	1	X	0
11	1	0	X	X
10	X	1	X	X

Karnaugh map for d_1

$q_1\ q_0$ \ $q_3\ q_2$	00	01	11	10
00	X	0	X	0
01	X	1	X	1
11	0	0	X	X
10	X	1	X	X

Karnaugh map for d_0

$q_1\ q_0$ \ $q_3\ q_2$	00	01	11	10
00	X	1	X	1
01	X	0	X	1
11	0	0	X	X
10	X	1	X	X

$$d_3 = q_3 \cdot q_0' + q_2 \cdot q_1 \cdot q_0$$

$$d_2 = q_2 \cdot q_1' + q_2 \cdot q_0' + q_2' \cdot q_1$$

$$d_1 = q_1' \cdot q_0 + q_1 \cdot q_0'$$

$$d_0 = q_3 + q_0'$$

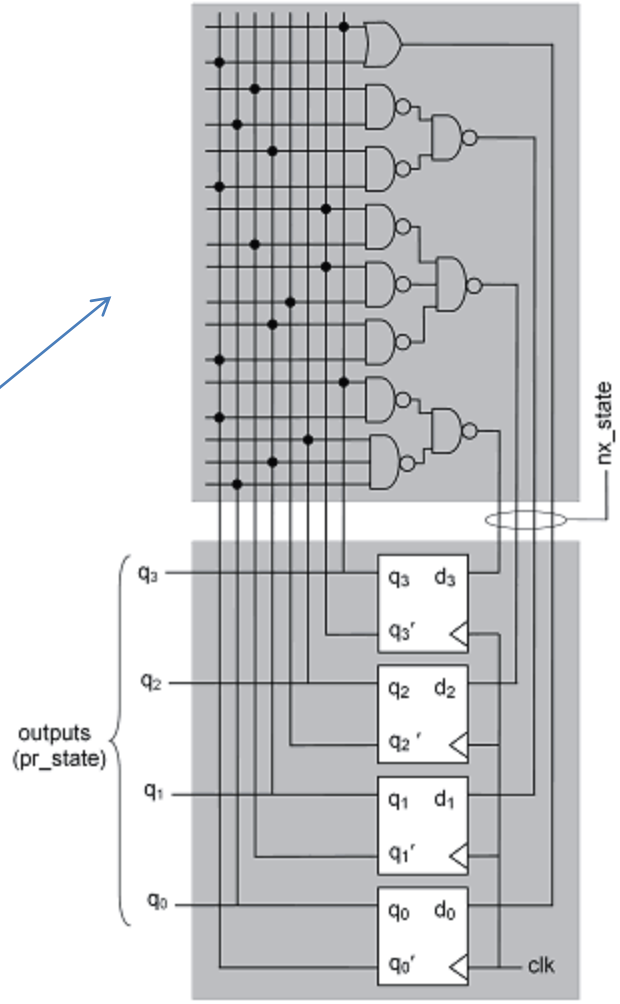
Lecture 6: Finite State Machine

Manual design

Example: Synchronous 3-to-9 counter

Step 4: Circuit

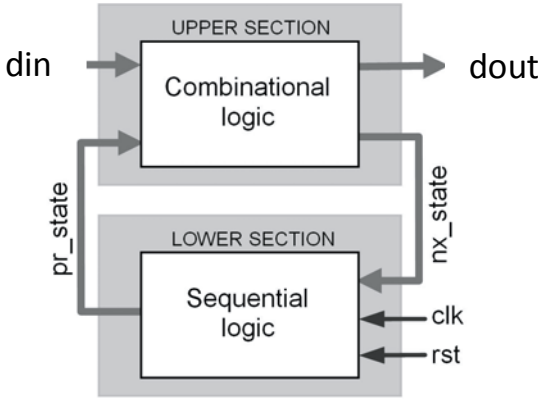
$$d_3 = q_3 \cdot q_0' + q_2 \cdot q_1 \cdot q_0$$
$$d_2 = q_2 \cdot q_1' + q_2 \cdot q_0' + q_2' \cdot q_1$$
$$d_1 = q_1' \cdot q_0 + q_1 \cdot q_0'$$
$$d_0 = q_3 + q_0'$$

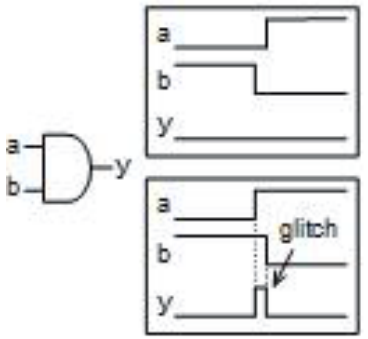


Lecture 6: Finite State Machine

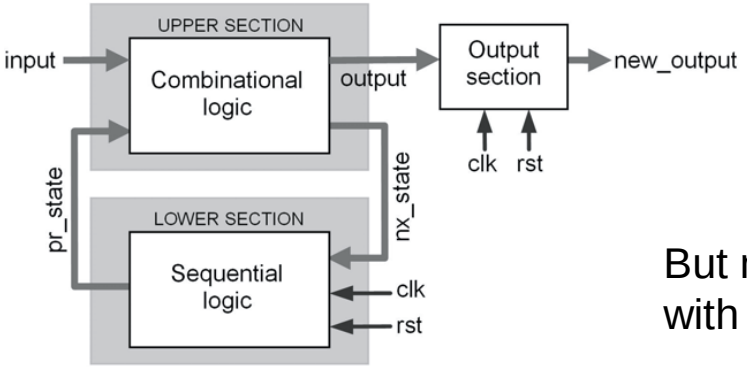
FSM Template

Model subject to glitches:





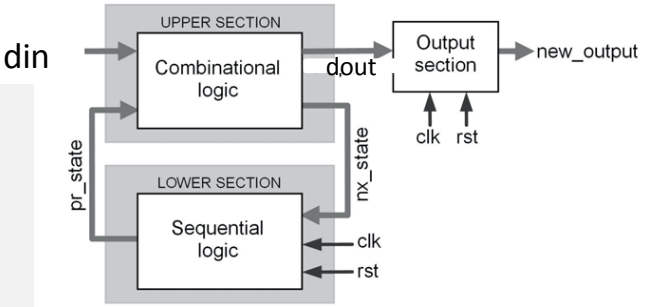
Glitch-free model



But new_output delayed with respect to output

Lecture 6: Finite State Machine

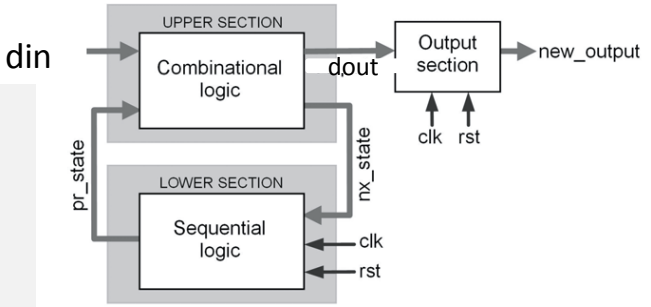
FSM Template




```
module <entity_name>
    ( input  clk,
      input  rst,
      input  <data_type> din,
      output <data_type> din);

    typedef enum logic [3:0]{ A = 1, B = 4, C, D, ...} state_t;
    state_t pr_state, nx_state;


...
```

Lecture 6: Finite State Machine

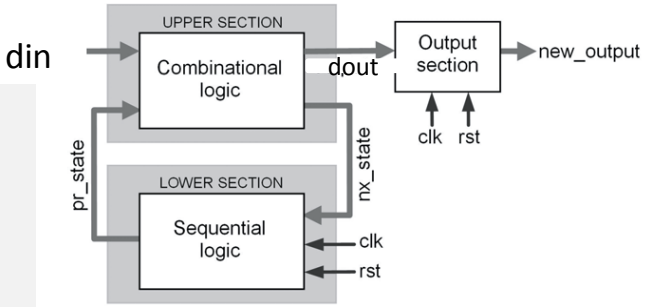
FSM Template




```
/*------ Lower Section -------*/
   always_ff @( posedge clk, negedge rst)
   begin
      if ( !rst )
         pr_state <= A;
      else
         pr_state <= nx_state;
   end



/*...*/
```

Lecture 6: Finite State Machine

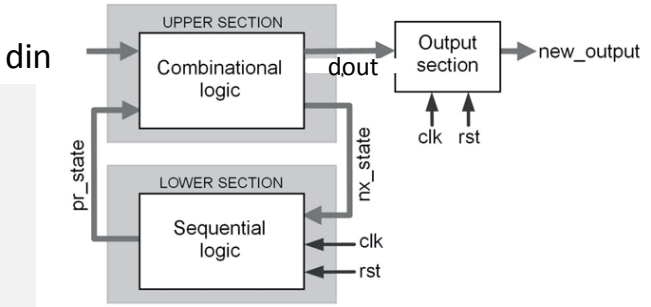
FSM Template




```
/*------ Upper Section -----*/
   always_comb  begin

      nx_state = pr_state;

      unique case ( pr_state ) begin
             A :
                if( din == <value> )
                   nx_state = B;
                else
                   nx_state = A;
              B :
                if( din == <value> )
                   nx_state = C;
                else
                   nx_state = A;
              C : ...
      end
   end
/*...*/
```

Lecture 6: Finite State Machine

FSM Template



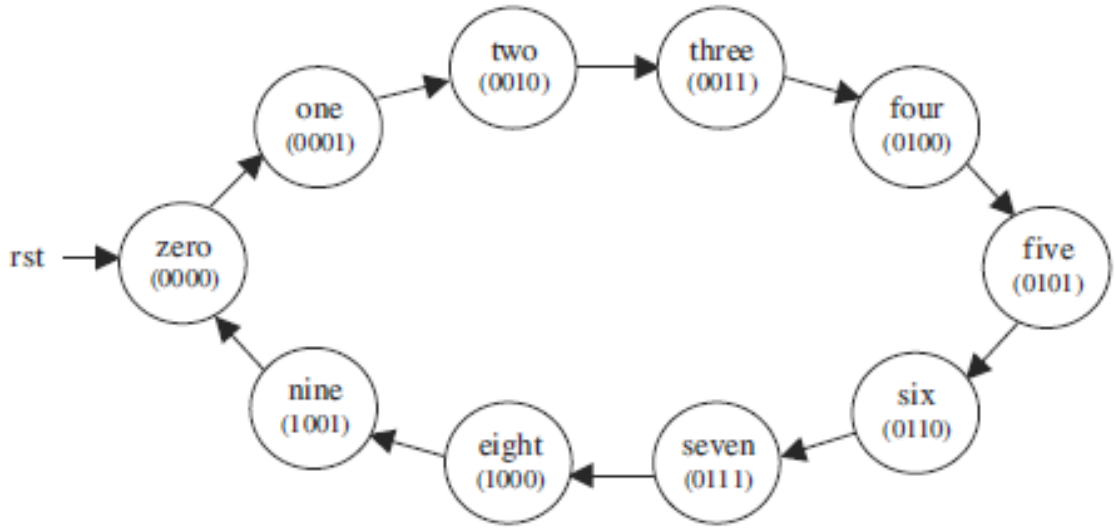

```
/* ------ Output Section ------- */
 always_comb  begin
      dout = 'b0;
      unique case ( pr_state ) begin
             A : dout = <value>;
             B : dout = <value>;
             C : ...
      end
   end


/* ----- Synchronous output ----- */

 always_ff @( posedge clk, negedge rst)
 begin
      if ( !rst )
         new_output = <value>;
      else
         new_output = dout;
 end
endmodule
```

Lecture 6: Finite State Machine

FSM Example #1: BCD Counter

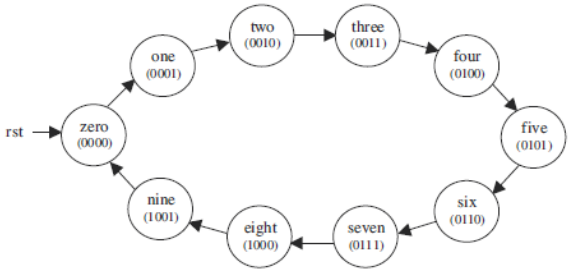
Lecture 6: Finite State Machine

FSM Example #1: BCD Counter



Lecture 6: Finite State Machine

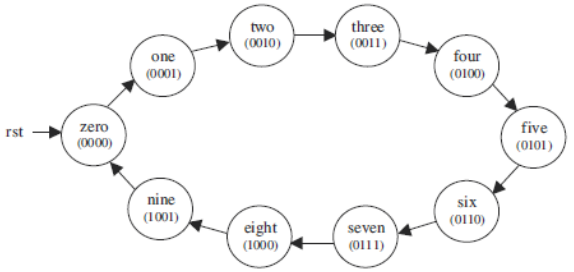
FSM Example #1: BCD Counter




```
module counter
     ( input               clk,
       input               rst,
       output logic [3:0] count
     );
   typedef enum (zero, one, two, three, four, five, six, seven, eight, nine) state_t;
   state_t pr_state, nx_state;

 /* ... */
```

Lecture 6: Finite State Machine

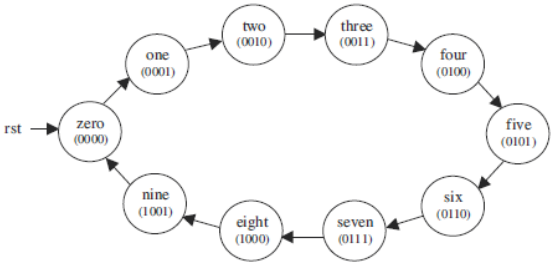
FSM Example #1: BCD Counter




```
module counter
     ( input               clk,
       input               rst,
       output logic [3:0] count
     );
   typedef enum ( zero, one, two, three, four, five, six, seven, eight, nine ) state_t;
   state_t pr_state, nx_state;

/* ----------- Lower section: --------------- */
 always_ff @( posedge clk, negedge rst)
   begin
      if ( !rst )
         pr_state <= zero;
      else
         pr_state <= nx_state;
   end

/* ... Continue to next page */
```

Lecture 6: Finite State Machine

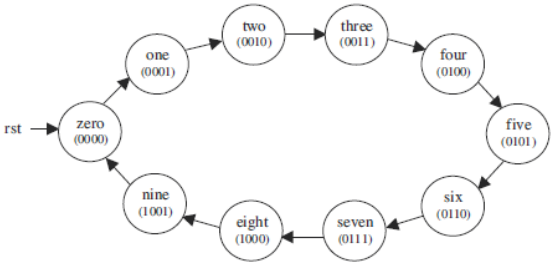
FSM Example #1: BCD Counter




```
/*------ Upper Section -> Next state -----*/

   always_comb  begin

      nx_state = pr_state;

      unique case ( pr_state ) begin
         zero :  nx_state = one;
         one  :  nx_state = two;
         two  :  nx_state = three;
         three:  nx_state = four;
         four :  nx_state = five;
         five :  nx_state = six;
         six  :  nx_state = seven;
         seven:  nx_state = eight;
         eight:  nx_state = nine;
         nine :  nx_state = zero;
      end
   end

/* ... Continue to next page */
```

Lecture 6: Finite State Machine

FSM Example #1: BCD Counter

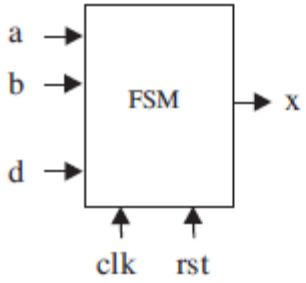


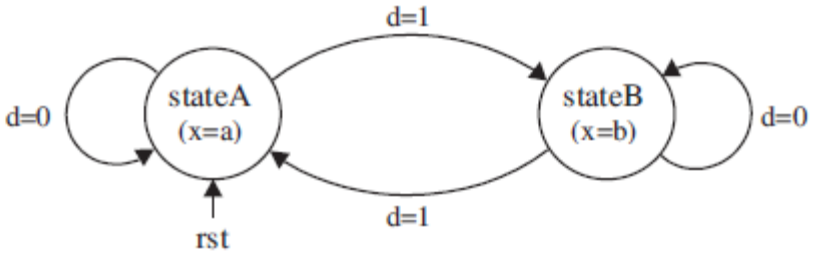

```
/*------ Upper Section -> Output -----*/

   always_comb  begin

      count = 4'b0000;

      unique case ( pr_state ) begin
         zero :  count = 4'b0000;
         one  :  count = 4'b0001;
         two  :  count = 4'b0010;
         three:  count = 4'b0011;
         four :  count = 4'b0100;
         five :  count = 4'b0101;
         six  :  count = 4'b0110;
         seven:  count = 4'b0111;
         eight:  count = 4'b1000;
         nine :  count = 4'b1001;
      end
   end
endmodule
```

Lecture 6: Finite State Machine

FSM Example #2

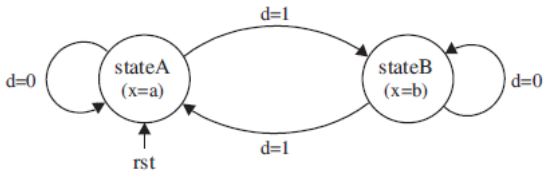
- The system has two states (stateA and stateB), and
- must change from one to the other every time d = '1' is received.
- The desired output is x = a when the machine is in stateA, or x = b when in stateB.
- The initial (reset) state is stateA.





Lecture 6: Finite State Machine

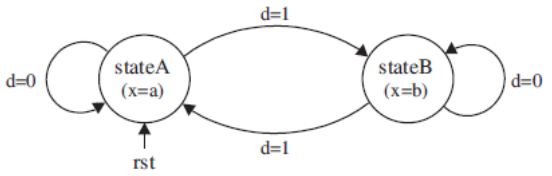
FSM Example #2




```
module simple_fsm
     ( input   clk,
       input   rst,
       input   a,
       input   b,
       input   d,
       output  x
     );
   typedef enum ( stateA, stateB ) state_t;
   state_t pr_state, nx_state;

/* ----------- Lower section: --------------- */
 always_ff @( posedge clk, negedge rst)
   begin
      if ( !rst )
         pr_state <= stateA;
      else
         pr_state <= nx_state;
   end

/* ... Continue to next page */
```

Lecture 6: Finite State Machine

FSM Example #2




```
/*------ Upper Section -> Next state -----*/
   always_comb  begin
      nx_state = pr_state;
      unique case ( pr_state ) begin
         stateA :
              if ( d == 1'b1 )
                nx_state = stateB;
              else
                nx_state = stateA;
         stateB :
              if ( d == 1'b1 )
                nx_state = stateA;
              else
                nx_state = stateB;
      end
   end
/*------ Upper Section -> Output -----*/
   always_comb  begin
      x = 1'b0;
      unique case ( pr_state ) begin
         stateA :  x = a;
         stateB :  x = b;
      end
   end
endmodule
```

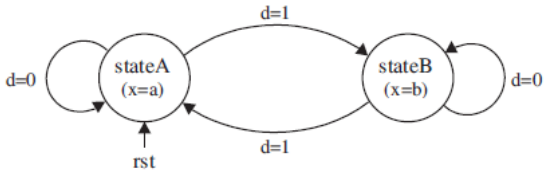
Lecture 6: Finite State Machine

FSM Example #3

- The system has two states (stateA and stateB), and
- must change from one to the other every time d = ‘1’ is received.
- The desired output is x = a when the machine is in stateA, or x = b when in stateB.
- The initial (reset) state is stateA.
- we want the output to be synchronous (to change only when clock rises).

Lecture 6: Finite State Machine

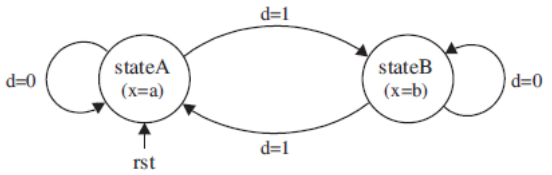
FSM Example #3




```
module simple_fsm
     ( input   clk,
       input   rst,
       input   a,
       input   b,
       input   d,
       output  x
     );
   typedef enum ( stateA, stateB ) state_t;
   state_t pr_state, nx_state;

   logic local_out;

/* ----------- Lower section: --------------- */
 always_ff @( posedge clk, negedge rst)
   begin
      if ( !rst )
         pr_state <= stateA;
         x <= 1'b0;
      else
         pr_state <= nx_state;
         x <= local_out;
   end

/* ... Continue to next page */
```

Lecture 6: Finite State Machine

FSM Example #3

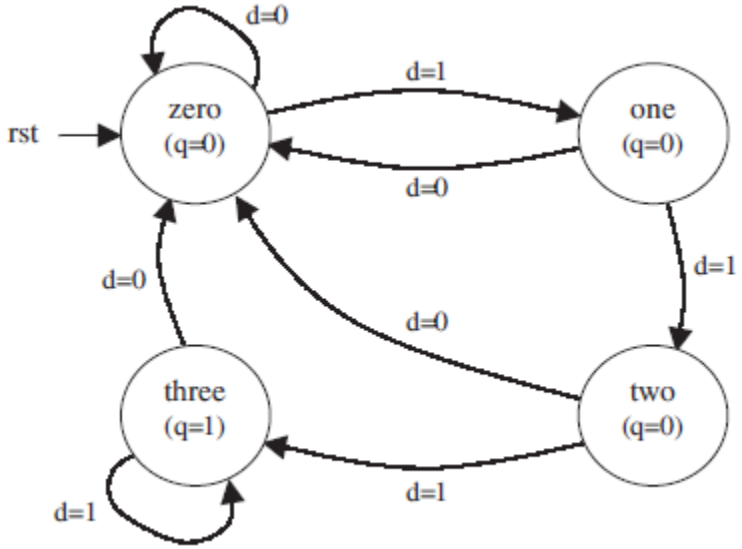



```
/*------ Upper Section -> Next state -----*/
   always_comb  begin
      nx_state = pr_state;
      unique case ( pr_state ) begin
         stateA :
              if ( d == 1'b1 )
                nx_state = stateB;
              else
                nx_state = stateA;
         stateB :
              if ( d == 1'b1 )
                nx_state = stateA;
              else
                nx_state = stateB;
      end
   end
/*------ Upper Section -> Output -----*/
   always_comb  begin
      x = 1'b0;
      unique case ( pr_state ) begin
         stateA : local_out = a;
         stateB : local_out = b;
      end
   end
endmodule
```

Lecture 6: Finite State Machine

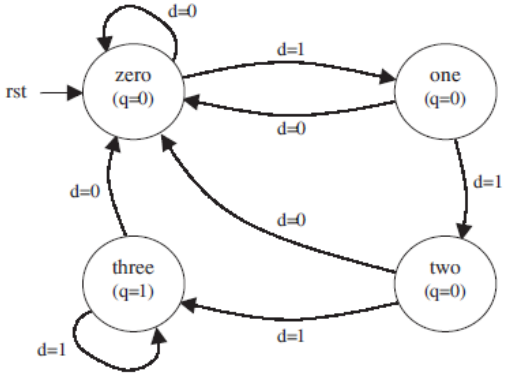
FSM Example #4: String Detector

- design a circuit that takes as input a serial bit stream and
- outputs a ‘1’ whenever the sequence ‘‘111’’ occurs
- Overlaps must also be considered
  - ✓ 0111110 occurs, than the output remains active for three consecutive clock cycles.



Lecture 6: Finite State Machine

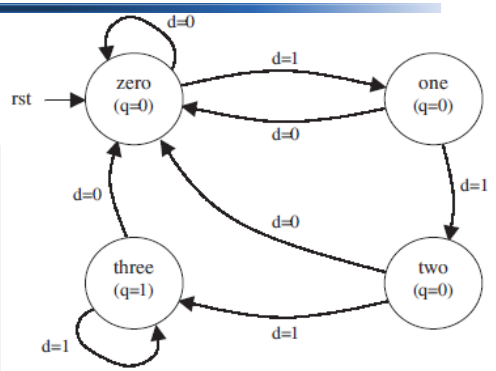
FSM Example #4: String Detector




```
module string_detector
     ( input   clk,
       input   rst,
       input   d,
       input   q
     );
   typedef enum ( zero, one, two, three ) state_t;
   state_t pr_state, nx_state;

/* ----------- Lower section: --------------- */
 always_ff @( posedge clk, negedge rst)
   begin
      if ( !rst )
         pr_state <= zero;
      else
         pr_state <= nx_state;
   end

/* Continue to the next page ... */
```

Lecture 6: Finite State Machine

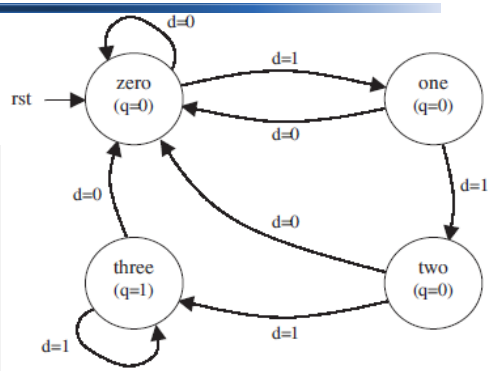
FSM Example #4: String Detector




```
/*------ Upper Section -> Next state -----*/
   always_comb  begin
      nx_state = pr_state;
      unique case ( pr_state ) begin
         zero :
              if ( d == 1'b1 )
                nx_state = one;
              else
                nx_state = zero;
         one :
              if ( d == 1'b1 )
                nx_state = two;
              else
                nx_state = zero;
         two :
              if ( d == 1'b1 )
                nx_state = three;
              else
                nx_state = zero;
         three:
              if ( d == 1'b1 )
                nx_state = zero;
              else
                nx_state = three;
      end
   end
/* Continue to the next page ... */
```

Lecture 6: Finite State Machine

FSM Example #4: String Detector

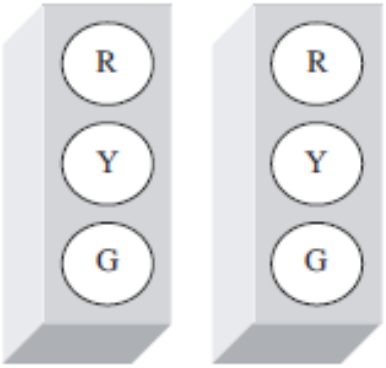



```
 /*------ Upper Section -> Output -----*/
   always_comb  begin

      q = 1'b0;

      unique case ( pr_state ) begin
         zero : q = 1'b0;
         one  : q = 1'b0;
         two  : q = 1'b0;
         three: q = 1'b1;
      end
   end
endmodule
```

Lecture 6: Finite State Machine

FSM Exercise : Traffic Light Controller

- Three modes of operation: Regular, Test, and Standby.
- Regular mode: four states, each with an independent, programmable time, passed to the circuit by means of a parameter.
- Test mode: allows all pre-programmed times to be overwritten (by a manual switch) with a small value, such that the system can be easily tested during maintenance (1 second per state). This value should also be programmable and passed to the circuit using a parameter.
- Standby mode: if set (by a sensor accusing malfunctioning, for example, or a manual switch) the system should activate the yellow lights in both directions and remain so while the standby signal is active.
- Assume that a 60 Hz clock (obtained from the power line itself) is available.



State	Operation Mode		
	REGULAR	TEST	STANDBY
	Time	Time	Time
RG	timeRG (30s)	timeTEST (1s)	---
RY	timeRY (5s)	timeTEST (1s)	---
GR	timeGR (45s)	timeTEST (1s)	---
YR	timeYR (5s)	timeTEST (1s)	---
YY	---	---	Indefinite

Lecture 6: Encoding Styles

- Sequential (*)
- Gray
- Johnson
- One-hot
- User-defined

Lecture 6: Encoding Styles

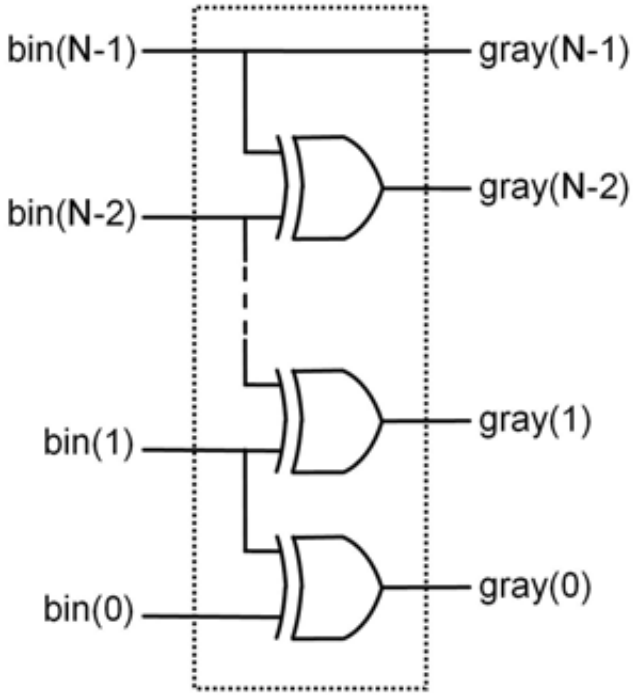
- Gray
  - Two successive values differ in only one bit

Decimal	Gray	Binary
0	000	000
1	001	001
2	011	010
3	010	011
4	110	100
5	111	101
6	101	110
7	100	111

Lecture 6: Encoding Styles

- Gray
  - Two successive values differ in only one bit

Decimal	Gray	Binary
0	000	000
1	001	001
2	011	010
3	010	011
4	110	100
5	111	101
6	101	110
7	100	111



Lecture 6: Encoding Styles

- Johnson
  - It is created using a simple rotation, the rotating bit is inverted.

Johnson

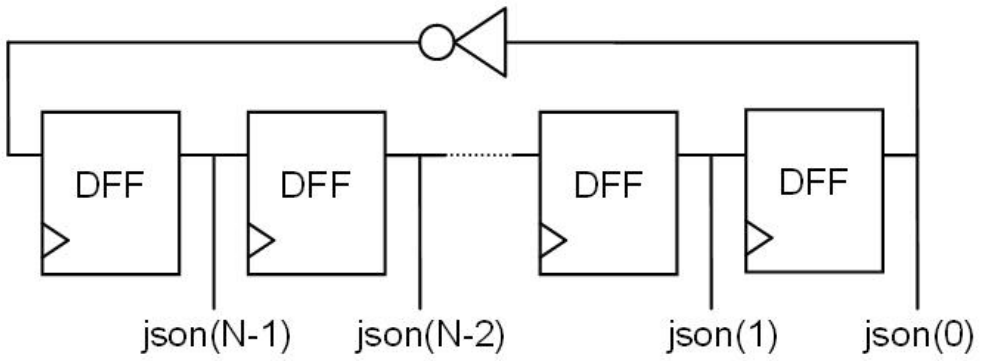
```
000
100
110
111
011
001
---
---
```

Lecture 6: Encoding Styles

- Johnson
  - It is created using a simple rotation, the rotating bit is inverted.

Johnson

```
000
100
110
111
011
001
---
---
```



Lecture 6: Encoding Styles

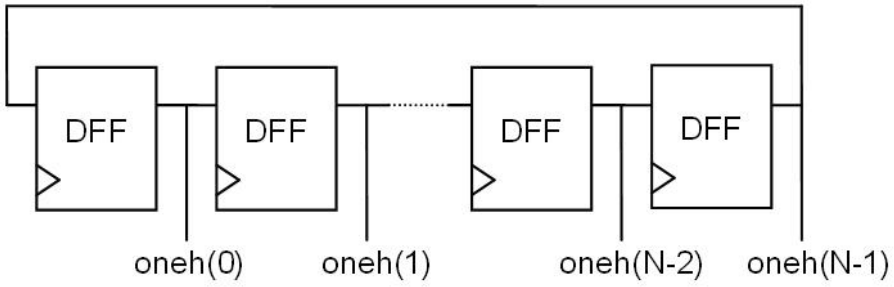
- One-hot
  - Only a single bit is high (‘1’) and all others are low (‘0’)

Binary	One-hot
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000
101	00100000
110	01000000
111	10000000

Lecture 6: Encoding Styles

- One-hot
  - Only a single bit is high (‘1’) and all others are low (‘0’)

Binary	One-hot
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000
101	00100000
110	01000000
111	10000000



Lecture 6: Encoding Styles

- How many states can be encoded with N=4 flip-flops using:
  - Sequential

Lecture 6: Encoding Styles

- How many states can be encoded with N=4 flip-flops using:
  - Sequential = 16 = 2^N
  - Gray

Lecture 6: Encoding Styles

- How many states can be encoded with N=4 flip-flops using:
  - Sequential = 16 = 2^N
  - Gray = 16 = 2^N
  - Johnson

Lecture 6: Encoding Styles

- How many states can be encoded with N=4 flip-flops using:
  - Sequential = 16 = 2^N
  - Gray = 16 = 2^N
  - Johnson = 8 = 2*N
  - One-hot

Lecture 6: Encoding Styles

- How many states can be encoded with N=4 flip-flops using:
  - Sequential = 16 = 2^N
  - Gray = 16 = 2^N
  - Johnson = 8 = 2*N
  - One-hot = 4 = N

Lecture 6: Encoding Styles

- When to use Which

  Sequential (or Gray) code?

Lecture 6: Encoding Styles

- When to use Which

  Sequential (or Gray) code?

  Least number of flip-flops, but largest combinational logic (slowest)

  One-hot code?

Lecture 6: Encoding Styles

- When to use Which

  Sequential (or Gray) code?

  Least number of flip-flops, but largest combinational logic (slowest)

  One-hot code?

  Largest number of flip-flops, but smallest combinational logic (fastest)

  Johnson code?

  In between