Problem Definition

Build a Finite State Machine(FSM) based upon your Student ID number (SID).

FSM will have one external input called ODD.

If ODD is true, then the FSM will reset to display the LEFTMOST odd digit in your SID, and then the output will sequence over the odd digits in your SSN, skipping over the even digits.

If ODD is false, then the FSM will reset to display the LEFTMOST even digit in your SID, and then the output will sequence over the even digits in your SSN, skipping over the odd digits.

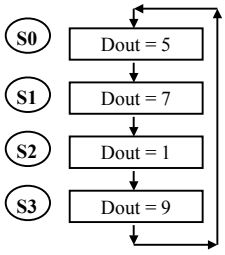
Example Output Sequences

For SID = 458 70 2198
if ODD is true:
$5 \rightarrow 7 \rightarrow 1 \rightarrow 9$, repeat ($5 \rightarrow 7 \rightarrow 1 \rightarrow 9 \rightarrow 5 \rightarrow 7 \rightarrow 1 \rightarrow 9$ etc)
if ODD is false:
$4 \rightarrow 8 \rightarrow 0 \rightarrow 2 \rightarrow 8$, repeat ($4 \rightarrow 8 \rightarrow 0 \rightarrow 2 \rightarrow 8 \rightarrow 4 \rightarrow 8 \rightarrow 0 \rightarrow 2 \rightarrow 8$ etc)

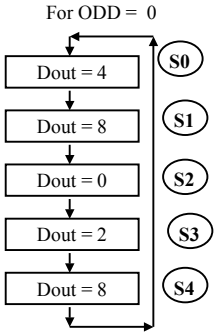
For SID = 688 99 1234
if ODD is true:
$9 \rightarrow 9 \rightarrow 1$, repeat ($9 \rightarrow 9 \rightarrow 1 \rightarrow 9 \rightarrow 9 \rightarrow 1$ etc)
if ODD is false:
$6 \rightarrow 8 \rightarrow 8 \rightarrow 2 \rightarrow 4$, repeat ($6 \rightarrow 8 \rightarrow 8 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 8 \rightarrow 8 \rightarrow 2 \rightarrow 4$ etc)

ASM Chart

For ODD = 1

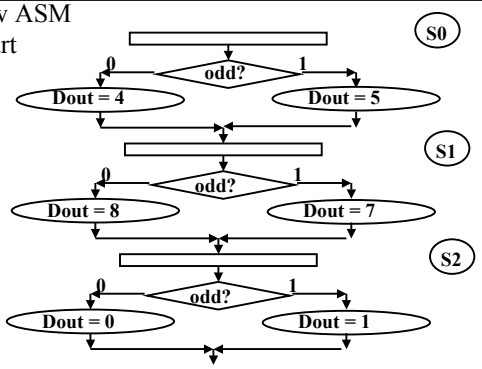


For ODD = 0

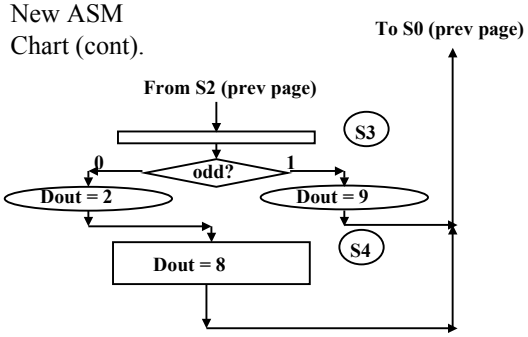


We can combine these states and use ODD as a conditional input.

New ASM Chart



New ASM Chart (cont).



State Table

Inputs Odd	Present State	Next State	Outputs Dout
0	S0	S1	4
0	S1	S2	8
0	S2	S3	0
0	S3	S4	2
0	S4	S0	8
1	S0	S1	5
1	S1	S2	7
1	S2	S3	1
1	S3	S0	9
1	S4	S0	8 (S4 only for even)

State Encodings

- Have 5 states, need at least three Flip Flops
- Binary Counting order: S0= 000, S1=001, S2=010, S3=011, S4 = 100
- Grey Code: S0=000, S1=001, S2=011, S3=010, S4 = 110
  - Grey Encoding oftens uses less logic than binary counting order
- One Hot Encoding (one FF per state): S0 = 00001, S1 = 00010, S2 = 00100, S3=01000, S4 = 1000

What type of FFs to use?

- D-FFs are most common in Programmable logic
  - If a PLD has a FF, it will at least be able to do the DFF function
- We will use DFFs for this example
- The NextState inputs are simply the D inputs of the FFs

State Table (Grey encoding for states)

Inputs Odd	Present State Q2Q1Q0	Next State D2D1D0	Outputs Dout
0	000	001	0100
0	001	011	1000
0	011	010	0000
0	010	110	0010
0	110	000	1000
1	000	001	0101
1	001	011	0111
1	011	010	0001
1	010	000	1001
1	110	000	1000

Unoptimized Next State equations

D2 = ODD' Q2' Q1 Q0'

D1 = ODD' Q2' Q1' Q0 + ODD' Q2' Q1 Q0
+ ODD' Q2' Q1 Q0' + ODD Q2' Q1' Q0
+ ODD Q2' Q1 Q0

D0 = ODD' Q2' Q1' Q0' + ODD' Q2' Q1' Q0
+ ODD Q2' Q1' Q0' + ODD Q2' Q1' Q0

Unoptimized Output Equations

DOUT(3) = ODD' Q2' Q1' Q0 + Q2 Q1 Q0' (state S4)
+ ODD Q2' Q1 Q0'

DOUT(2) = ODD' Q2' Q1' Q0' + ODD Q2' Q1' Q0'
+ ODD Q2' Q1' Q0

DOUT(1) = ODD' Q2' Q1 Q0' + ODD Q2' Q1' Q0

DOUT(0) = ODD Q2' Q1' Q0' + ODD Q2' Q1' Q0
+ ODD Q2' Q1 Q0 + ODD Q2' Q1 Q0'

For DOUT(3) a simple optimization was done -- if in State S4, DOUT(3) = 1, regardless of ODD input.

State Table (arranged in binary order)

Inputs Odd	Present State Q2Q1Q0	Next State D2D1D0	Outputs Dout	
0	000	001	0100	
0	001	011	1000	
0	010	110	0010	Illegal states in Red
0	011	010	0000	
0	100	xxx	xxxx	
0	101	xxx	xxxx	
0	110	000	1000	
0	111	xxx	xxxx	
1	000	001	0101	
1	001	011	0111	
1	010	000	1001	
1	011	010	0001	
1	100	xxx	xxxx	
1	101	xxx	xxxx	
1	110	000	1000	
1	111	xxx	xxxx	

Example Optimization

Q1,Q0 \ Odd,Q2	00	01	11	10
00	0	x	x	0
01	1	x	x	1
11	1	x	x	1
10	1	0	x	0

Optimize D1

D1 = Q2'Q0 + Odd'Q2'Q1

Q1,Q0 \ Odd,Q2	00	01	11	10
00	0	x	x	1
01	0	x	x	1
11	0	x	x	1
10	0	0	x	1

Optimize Dout(0)

Dout(0) = Odd

Try One Hot Encoding

S0 = 00001, S1 = 00010, S2 = 00100, S3=01000, S4 = 10000

Each FF represents a state. Each next state equation can be written as :

Dn = (how do I get to this state?) + (how do I stay in this state?)

In this FSM, never stay in one state more than clock, so only need to know how to get to that state. By Inspection:

D0 = Q4 + Q3Odd
D1 = Q0
D2 = Q1
D3 = Q2
D4 = Q3 Odd'

State Table

Inputs Odd	Present State	Next State	Outputs Dout
0	S0	S1	4 (0100)
0	S1	S2	8 (1000)
0	S2	S3	0 (0000)
0	S3	S4	2 (0010)
0	S4	S0	8 (1000)
1	S0	S1	5 (0101)
1	S1	S2	7 (0111)
1	S2	S3	1 (0001)
1	S3	S0	9 (1001)
1	S4	S0	8 (1000)

Output Equations (one hot encoding)

DOUT(3) = ODD' Q1 + Q4 + ODD Q3

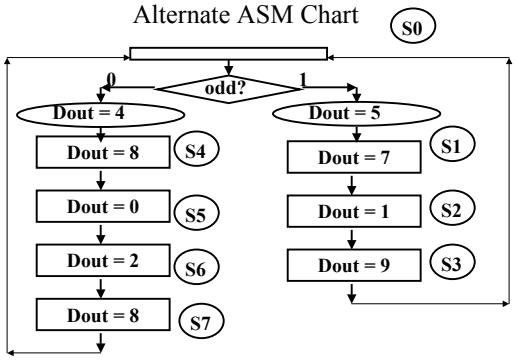
DOUT(2) = ODD' Q0 + ODD Q0 + ODD Q1

DOUT(1) = ODD' Q3 + ODD Q1

DOUT(0) = ODD S0 + ODD S1 + ODD S2 + ODD S4

Obviously, Dout(0) = Odd

Alternate ASM Chart

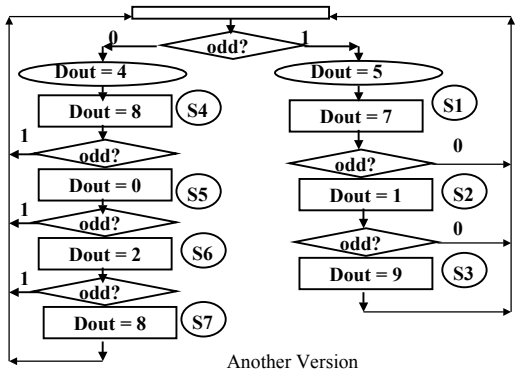


Odd only effects sequence in State S0 (Reset state)

Alternate ASM Chart

The previous ASM chart will take more states

- The 22V10 PLD has only 10 outputs; each output has a DFF. Four outputs are used to output SID value; this leaves only 6 FFs that can be used for state encoding!
- This means that one-hot encoding would not be an option for this implementation



Another Version

Which Version to use?

- The previous ASM chart checked the odd input and restarted the sequence if a different sequence was chosen
- All of these ASM charts are an interpretation of an English problem statement
  - English is imprecise
  - ASM charts are a precise definition of behavior
- You can use any of the three ASM chart interpretations you wish (the first is the easiest, the last two will take more states).

VHDL Simulation, JEDEC File

We will use the same approach as with the SSN Decoder lab for VHDL simulation, JEDEC File production.

See the link titled "SSN Finite State Machine Lab: Form for VHDL Compilation/Simulation, Jedec file production" on http://www.ece.msstate.edu/~reese/EE3714 under the "Other Links" heading.

This link has two sample VHDL solutions.

One solution uses 3 DFFs (the Grey code solution) and one uses 5 DFFs (the onehot encoding solution) . Both solutions are for the SID: 458 70 2198

DFFs in VHDL

A portion of the Grey code solution defines DFFs in VHDL:

```
-- State Flip Flops
  stateff: process (clk,reset)
   begin
   if (reset = '1') then
     q <= "000";   -- students, change this your initial state
   elsif (clk'event and clk='1') then
     q <= d;
   end if;
  end process stateff;
```

The ‘q’ signal is the output of the DFFs, the ‘d’ is the input of the DFF. The “q”, “d” are internal signals declared just above this.

DFFs in VHDL (cont)

```
-- State Flip Flops
  stateff: process (clk,reset)
   begin
   if (reset = '1') then
     q <= "000";
   elsif (clk'event and clk='1') then
     q <= d;
   end if;
  end process stateff;
```

Initial state when reset=‘1’

DFF changes state (q = d) on rising clock edge.

“q”, “d” are internal signals. Declaration is after “architecture”:

```
architecture a of ssnseq is
  signal q,d:  std_logic_vector(2 downto 0);
```

The “2 downto 0” declares a bus that is 3 bits wide.

One Hot solution

The one hot solution needed 5 flip-flops, see what changed:

```
architecture a of ssnseq is
signal q,d:  std_logic_vector(4 downto 0);
```

q,d 5 bits wide

```
stateff: process (clk,reset)
   begin
   if (reset = '1') then
     q <= "00001";
   elsif (clk'event and clk='1') then
     q <= d;
   end if;
  end process stateff;
```

Note that one-hot solution requires initial state of “00001”.

Next State, Output Equations in VHDL

The onehot solution next state, output equations in VHDL are:

```
d(0) <= (q(4)) or (q(3) and odd);
 d(1) <= q(0);
d(2) <= q(1);
 d(3) <= q(2);
 d(4) <= q(3) and (not odd);
```

```
dout(0) <= odd;
 dout(1) <= ((not odd) and q(3)) or (odd and q(1));

 dout(2) <= ((not odd) and q(0)) or (odd and q(0)) or
          (odd and q(1));

 dout(3) <= ((not odd) and q(1)) or ((not odd) and q(4)) or
          (odd and q(3));
```


Input/Output

The input/output for both solutions are the same:

```
entity ssnseq is
  port (  clk,reset:  in std_logic;
              odd: in std_logic;
              qstate : out std_logic_vector(5 downto 0);
              dout: out std_logic_vector(3 downto 0)
      );
```

The "qstate" is the internal "q" signals that represents the state of your FSM. You can't debug a FSM without knowing the state. Assignments of "q" to "qstate" is in the code. I provided 6 bits for "qstate" because the maximum number of DFFs you can use is 6. DON'T CHANGE QSTATE to a fewer number of bits! Just assign unused bits to "0"s.

Simulation Results

Each line of the simulation results represents a clock cycle.

```
# Reset = 1, Odd = 0, qstate = 000001, dout= 0100
# Reset = 1, Odd = 0, qstate = 000001, dout= 0100
# Reset = 0, Odd = 0, qstate = 000010, dout= 1000
# Reset = 0, Odd = 0, qstate = 000100, dout= 0000
# Reset = 0, Odd = 0, qstate = 001000, dout= 0010
# Reset = 0, Odd = 0, qstate = 010000, dout= 1000
# Reset = 0, Odd = 0, qstate = 000001, dout= 0100
# Reset = 0, Odd = 0, qstate = 000010, dout= 1000
# Reset = 0, Odd = 0, qstate = 000100, dout= 0000
# Reset = 0, Odd = 0, qstate = 001000, dout= 0010
# Reset = 0, Odd = 0, qstate = 010000, dout= 1000
# Reset = 0, Odd = 0, qstate = 000001, dout= 0100
# Reset = 0, Odd = 0, qstate = 000010, dout= 1000
# Reset = 1, Odd = 1, qstate = 000001, dout= 0101
```

Partial results for one-hot implementation. Note that for Odd=0, sequence is 4,8,0,2,8,4,8,etc…

The "qstate" is the current state of the finite state machine.

Summary of Lab Requirements

- This is a ONE week lab.
- For prelab, you must design the ASM chart, logic equations and simulate in Altera Maxplus.
  - The FSM will be implemented in a single 22V10 PLD. Follow the instructions in these notes for creating a VHDL file and producing a JEDEC file needed for PLD programming.
  - Your equations DO NOT HAVE TO BE MINIMAL.
  - There is no particular State encoding required.
- When you walk into lab, you must have a printout of the correct VHDL simulation returned from the WWW interface.
  - If your simulation is incorrect, then the TA will not program you PLD.
  - Do the PLD implementation demonstrate the working design to the TA.