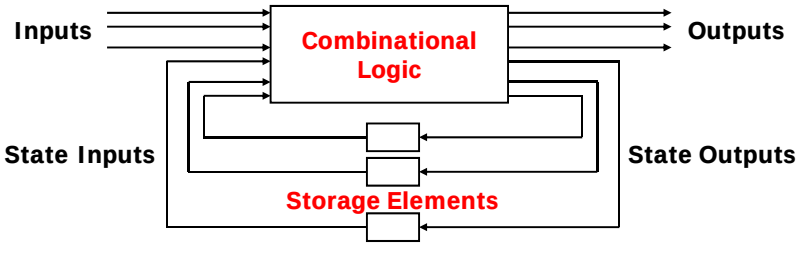
Lecture 16

- Logistics
  - HW6 due Wednesday
  - Midterm 2 creeping up (next week Wednesday 5/21)
  - Midterm 2 covers materials up to Friday lecture & HW7
  - Review next Tuesday 6:30pm?
- Last lecture
  - Timing issues for asynchronous inputs
  - Registers/counters
  - Wrapped up on sequential logic building blocks
- Today
  - Introduction to finite state machines
  - Counters as finite state machines

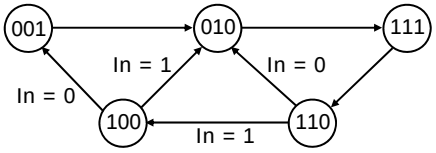
"States" for finite state machines are kept in the storage elements

- Combinational logic and storage elements
  - Localized feedback loops
  - Choice of storage elements alters the logic



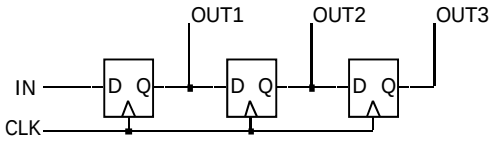
Finite-state machines (FSMs)

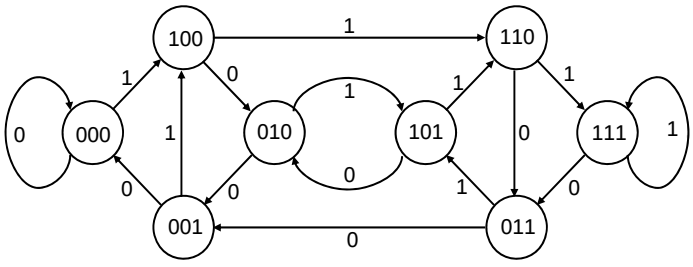
- States: Possible storage-element values
- Transitions: Changes in state
  - Clock synchronizes the state changes
- Sequential logic
  - Sequences through a series of states
  - Based on inputs and present state



Drawing state diagrams

- Show input values on transition arcs
- Show output values in state nodes





Counters revisited

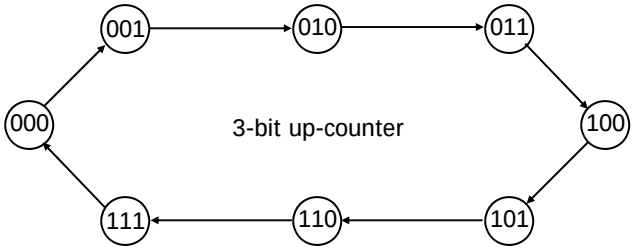
- Great simple examples of state machines
  - Output is the counter's state
- Next state is well defined
  - Does not depend on input (no inputs)

FSM design procedure (using counters)

1. Draw a state diagram
2. Draw a state-transition table
3. Encode the next-state functions
   - Minimize the logic using k-maps
4. Implement the design

We will use a '3-bit up counter' as an example in two different ways today

1. Draw a state diagram



2. Draw a state-transition table

- Like a truth-table
  - State encoding is easy for counters → Use count value

current state		next state	
0	000	001	1
1	001	010	2
2	010	011	3
3	011	100	4
4	100	101	5
5	101	110	6
6	110	111	7
7	111	000	0

3. Encode the next state functions

- Assume D flip-flops as state elements

C3	C2	C1	N3	N2	N1
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

N1

C1 \ C3	1	1	1	1
0	0	0	0	
C2				

N2

C1 \ C3	0	1	1	0
1	0	0	1	
C2				

N3

C1 \ C3	0	0	1	1
0	1	0	1	
C2				

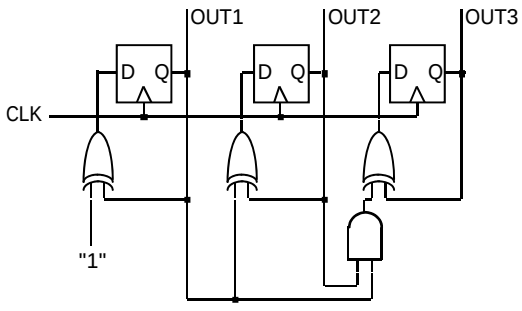
N1 := C1'

N2 := C1C2' + C1'C2
:= C1 xor C2

N3 := C1C2C3' + C1'C3 + C2'C3
:= C1C2C3' + (C1' + C2')C3
:= (C1C2) xor C3

4. Implement the design

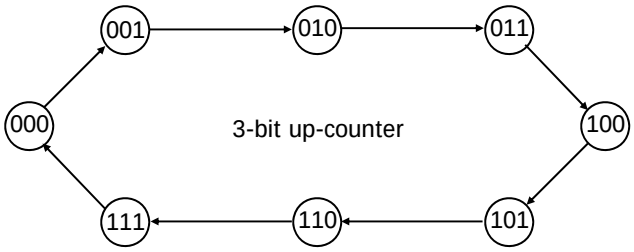
- 3 flip-flops hold state
  - Counter is synchronously clocked
- Minimized logic computes next state



Another example: 3-bit up counter with T flip flops

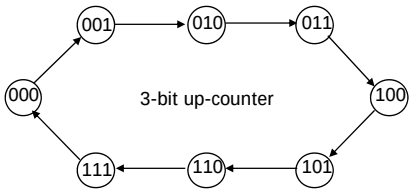
1. Draw a state diagram
2. Draw a state-transition table
3. Encode the next-state functions
   - Minimize the logic using k-maps
4. Implement the design

1. Draw a state diagram



2. Draw a state-transition table

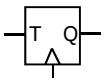
- Like a truth-table
  - State encoding is easy for counters → Use count value



current state		next state	
0	000	001	1
1	001	010	2
2	010	011	3
3	011	100	4
4	100	101	5
5	101	110	6
6	110	111	7
7	111	000	0

3. Encode the next state functions

T flip-flops



T1 := 1
T2 := C1
T3 := C1 C2

C3	C2	C1	N3	N2	N1	T3	T2	T1
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

T1

C1 \ C3	C3			C3
C1	1	1	1	1
	1	1	1	1
	C2			

T2

C1 \ C3	C3			C3
C1	0	0	0	0
	1	1	1	1
	C2			

T3

C1 \ C3	C3			C3
C1	0	0	0	0
	0	1	1	0
	C2			

4. Implement the design