

EECS150 - Digital Design

Lecture 23 - FSMs & Counters

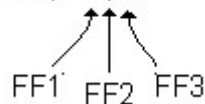
April 8, 2010
John Wawrzynek

State Encoding

- **One-hot encoding of states.**
- One FF per state.

Ex: 3 States

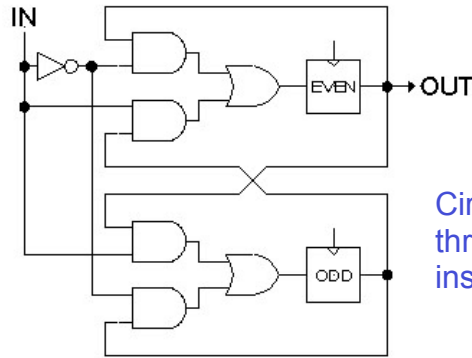
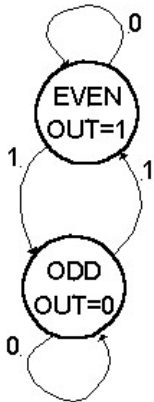
STATE1: 001
STATE2: 010
STATE3: 100



- Why one-hot encoding?
 - Simple design procedure.
 - Circuit matches state transition diagram (example next page).
 - Often can lead to simpler and faster “next state” and output logic.
- Why not do this?
 - Can be costly in terms of FFs for FSMs with large number of states.
- FPGAs are “FF rich”, therefore one-hot state machine encoding is often a good approach.

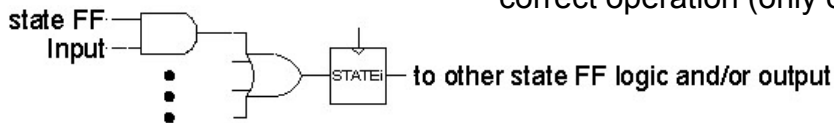
One-hot encoded FSM

- Even Parity Checker Circuit:



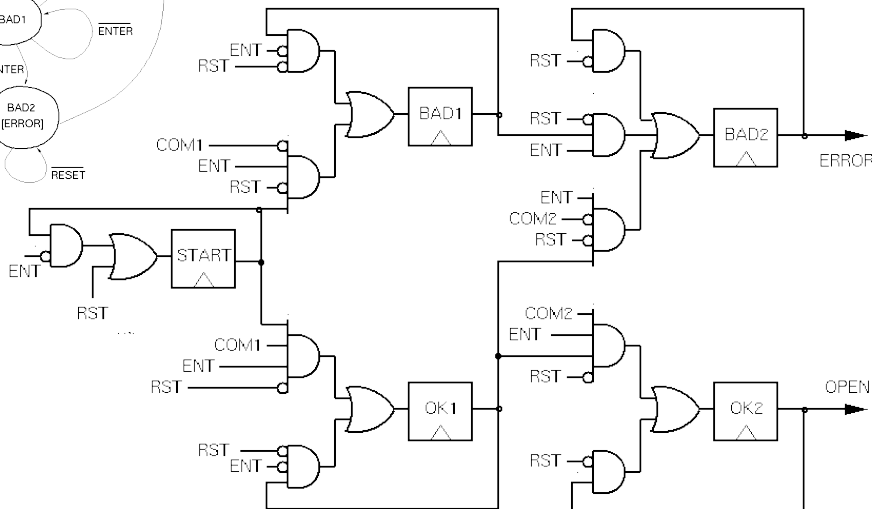
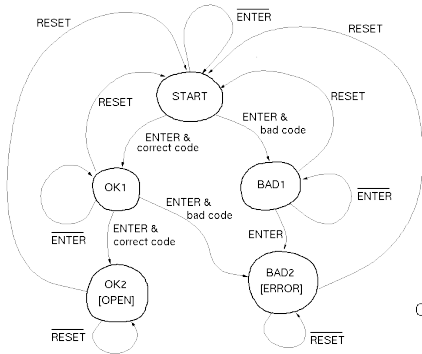
Circuit generated through direct inspection of the STD.

- In General:



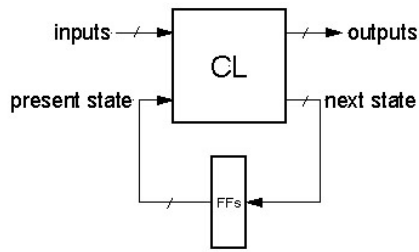
- FFs must be initialized for correct operation (only one 1)

One-hot encoded combination lock

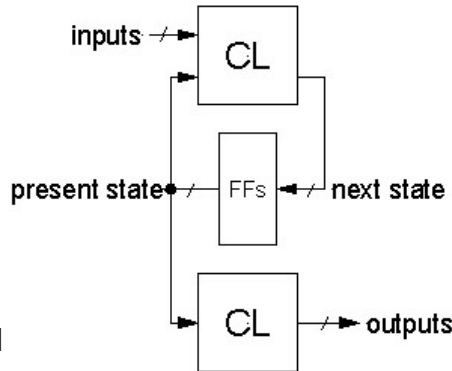


FSM Implementation Notes

- General FSM form:



- All examples so far generate output based only on the present state:



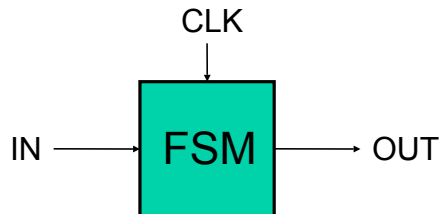
- Commonly name **Moore Machine**
(If output functions include both present state and input then called a **Mealy Machine**)

Finite State Machines

- **Example: Edge Detector**

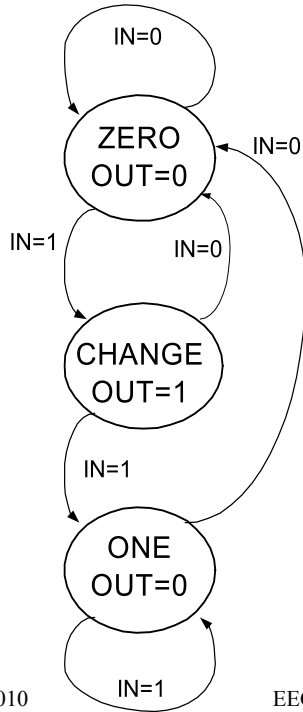
Bits are received one at a time (one per cycle),
such as: 000111010 \longrightarrow *time*

Design a circuit that asserts its output for one cycle when the input bit stream changes from 0 to 1.



Try two different solutions.

State Transition Diagram Solution A



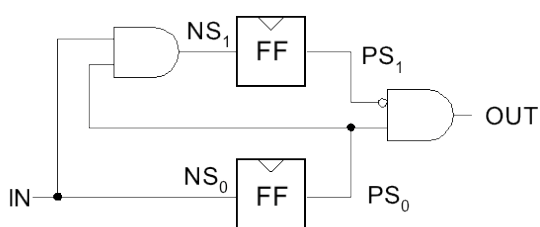
	IN	PS	NS	OUT
ZERO	0	00	00	0
	1	00	01	0
CHANGE	0	01	00	1
	1	01	11	1
ONE	0	11	00	0
	1	11	11	0

Solution A, circuit derivation

	IN	PS	NS	OUT
ZERO	0	00	00	0
	1	00	01	0
CHANGE	0	01	00	1
	1	01	11	1
ONE	0	11	00	0
	1	11	11	0

		PS				
		00	01	11	10	
IN	0	0	0	0	-	$NS_1 = IN PS_0$
	1	0	1	1	-	

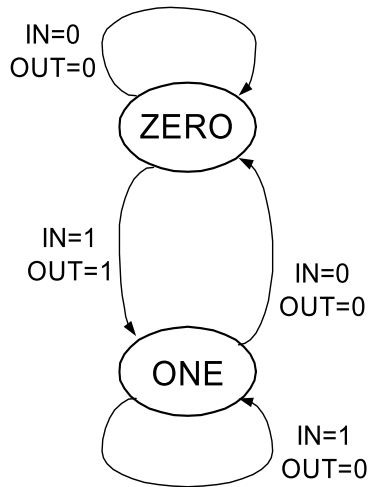
		PS				
		00	01	11	10	
IN	0	0	0	0	-	$NS_0 = IN$
	1	1	1	1	-	



		PS				
		00	01	11	10	
IN	0	0	1	0	-	$OUT = \overline{PS_1} PS_0$
	1	0	1	0	-	

Solution B

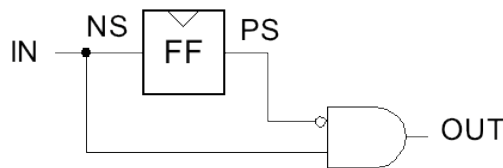
Output depends not only on PS but also on input, IN



Let ZERO=0,
ONE=1

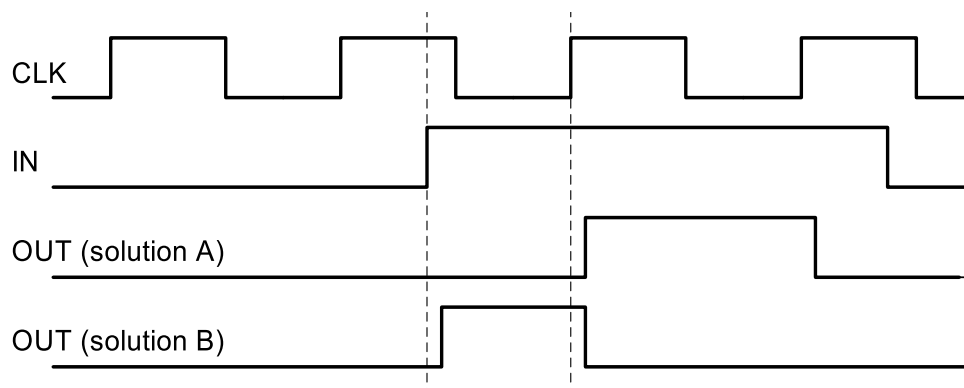
IN	PS	NS	OUT
0	0	0	0
0	1	0	0
1	0	1	1
1	1	1	0

$$NS = IN, \quad OUT = IN \oplus PS'$$



What's the *intuition* about this solution?

Edge detector timing diagrams



- Solution A: output follows the clock
- Solution B: output changes with input rising edge and is asynchronous wrt the clock.

FSM Comparison

Solution A

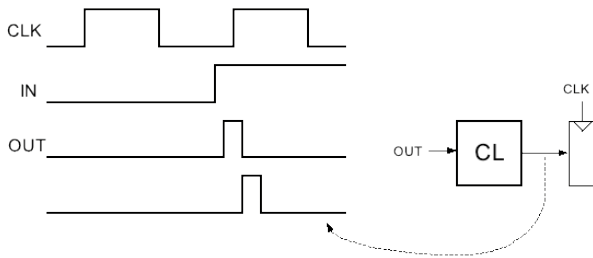
Moore Machine

- output function only of PS
- maybe more states (why?)
- synchronous outputs
 - no glitches
 - one cycle “delay”
 - full cycle of stable output

Solution B

Mealy Machine

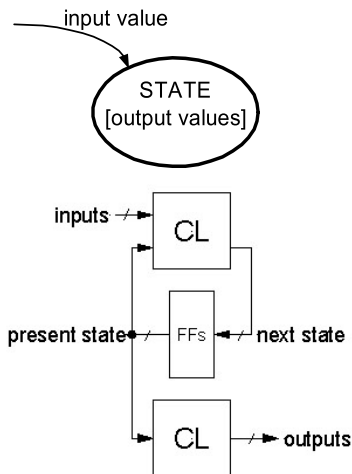
- output function of both PS & input
- maybe fewer states
- asynchronous outputs
 - if input glitches, so does output
 - output immediately available
 - output may not be stable long enough to be useful (below):



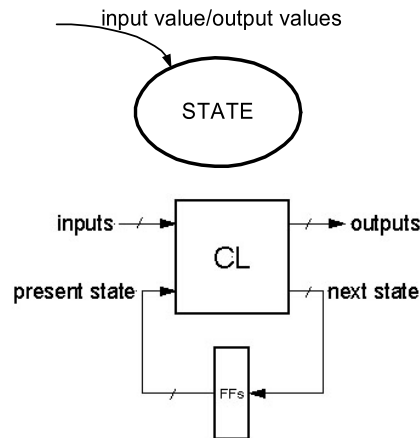
If output of Mealy FSM goes through combinational logic before being registered, the CL might delay the signal and it could be missed by the clock edge.

FSM Recap

Moore Machine



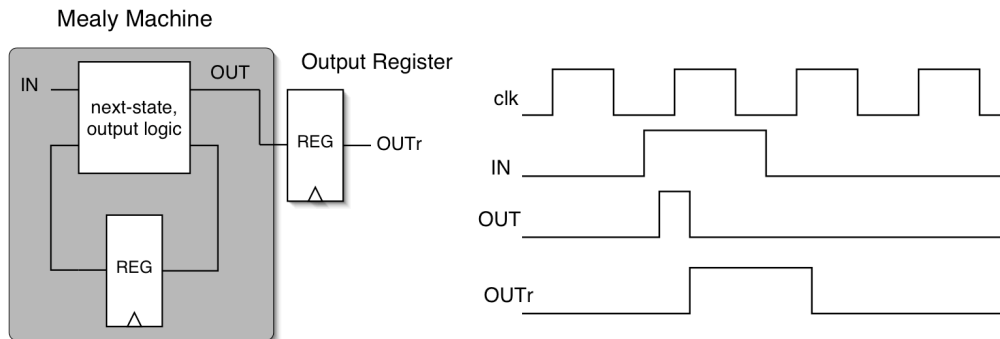
Mealy Machine



Both machine types allow one-hot implementations.

Final Notes on Moore versus Mealy

1. A given state machine *could* have *both* Moore and Mealy style outputs. Nothing wrong with this, but you need to be aware of the timing differences between the two types.
2. The output timing behavior of the Moore machine can be achieved in a Mealy machine by “registering” the Mealy output values:



Spring 2010

EECS150 - Lec22-counters

Page 13

General FSM Design Process with Verilog

Design Steps: Implementation

1. Specify **circuit function** (English)
2. Draw **state transition diagram**
3. Write down **symbolic state transition table**
4. Assign encodings (bit patterns) to symbolic states
5. Code as Verilog behavioral description
 - ✓ Use parameters to represent encoded states.
 - ✓ Use separate always blocks for register assignment and CL logic block.
 - ✓ Use case for CL block. Within each case section assign all outputs and next state value based on inputs. *Note: For Moore style machine make outputs dependent only on state not dependent on inputs.*

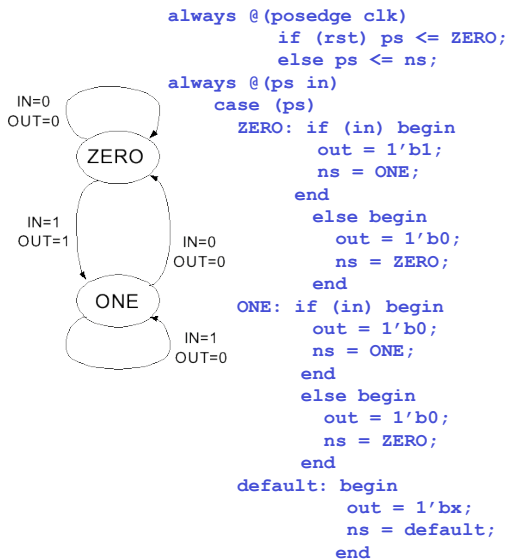
Spring 2010

EECS150 - Lec22-counters

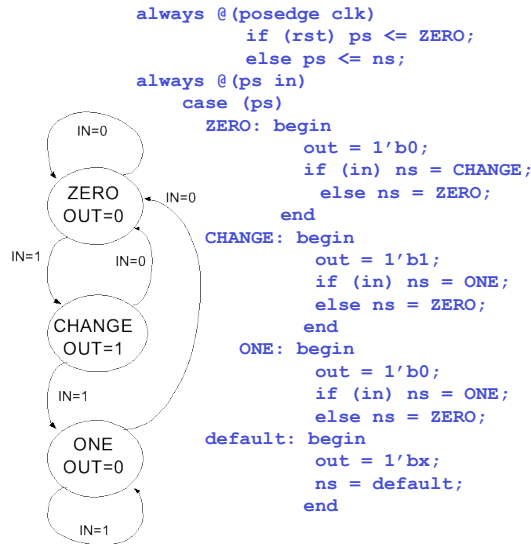
Page 14

FSMs in Verilog

Mealy Machine

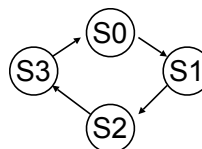


Moore Machine

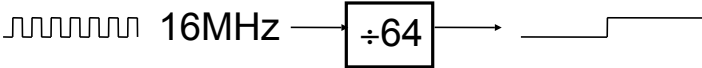


Counters

- Special sequential circuits (FSMs) that repeatedly sequence through a set of outputs.
- Examples:
 - binary counter: 000, 001, 010, 011, 100, 101, 110, 111, 000, ...
 - gray code counter: 000, 010, 110, 100, 101, 111, 011, 001, 000, 010, 110, ...
 - one-hot counter: 0001, 0010, 0100, 1000, 0001, 0010, ...
 - BCD counter: 0000, 0001, 0010, ..., 1001, 0000, 0001
 - pseudo-random sequence generators: 10, 01, 00, 11, 10, 01, 00, ...
- Moore machines with "ring" structure in State Transition Diagram:

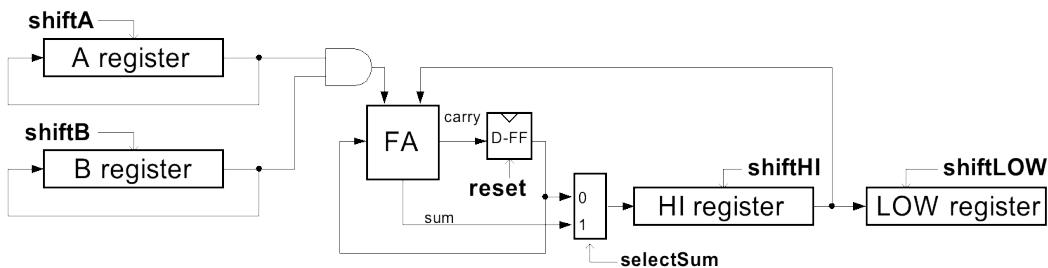


What are they used?

- Counters are commonly used in hardware designs because most (if not all) computations that we put into hardware include iteration (looping). Examples:
 - Shift-and-add multiplication scheme.
 - Bit serial communication circuits (must count one "words worth" of serial bits).
- Other uses for counter:
 - Clock divider circuits
 - 
 - Systematic inspection of data-structures
 - Example: Network packet parser/filter control.
- Counters simplify "controller" design by:
 - providing a specific number of cycles of action,
 - sometimes used with a decoder to generate a sequence of timed control signals.
 - Consider using a counter when many FSM states with few branches.

Controller using Counters

- Example, Bit-serial multiplier (n^2 cycles, one bit of result per n cycles):



- Control Algorithm:

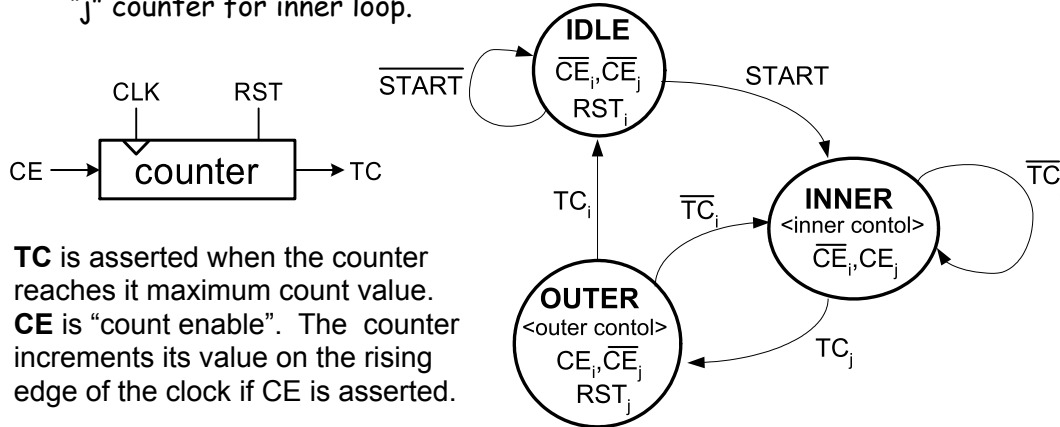

```
repeat n cycles { // outer (i) loop
  repeat n cycles{ // inner (j) loop
    shiftA, selectSum, shiftHI
  }
  shiftB, shiftHI, shiftLOW, reset
}
```

Note: The occurrence of a control signal x means $x=1$. The absence of x means $x=0$.

Controller using Counters

- State Transition Diagram:**

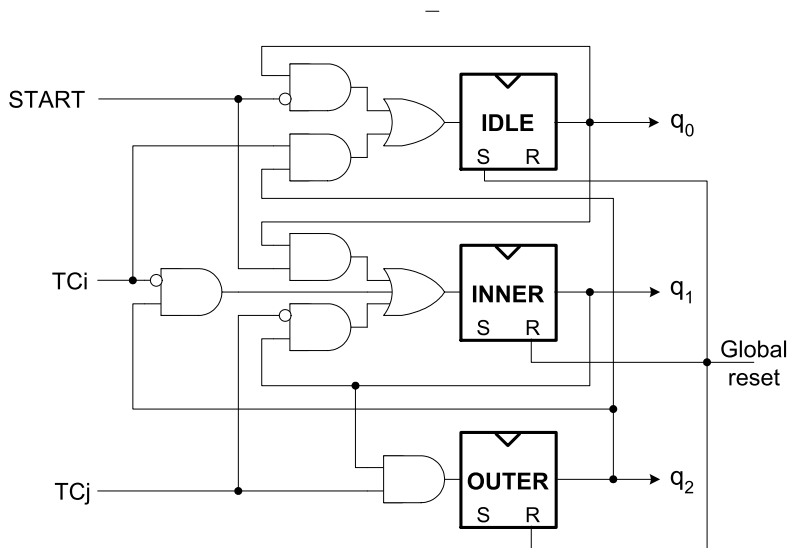
- Assume presence of two binary counters. An "i" counter for the outer loop and "j" counter for inner loop.



TC is asserted when the counter reaches its maximum count value. **CE** is "count enable". The counter increments its value on the rising edge of the clock if CE is asserted.

Controller using Counters

- Controller circuit implementation:**



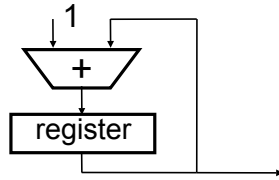
- Outputs:**

$$\begin{aligned} CE_i &= q_2 \\ CE_j &= q_1 \\ RST_i &= q_0 \\ RST_j &= q_2 \end{aligned}$$

$$\begin{aligned} \text{shiftA} &= q_1 \\ \text{shiftB} &= q_2 \\ \text{shiftLOW} &= q_2 \\ \text{shiftHI} &= q_1 + q_2 \\ \text{reset} &= q_2 \\ \text{selectSUM} &= q_1 \end{aligned}$$

How do we design counters?

- For binary counters (most common case) incrementer circuit would work:



- In Verilog, a counter is specified as: $x = x+1$;
 - This does *not* imply an adder
 - An incrementer is simpler than an adder
 - And a counter is simpler yet.
- In general, the best way to understand counter design is to think of them as FSMs, and follow general procedure, however some special cases can be optimized.

Synchronous Counters

All outputs change with clock edge.

- Binary Counter Design:
Start with 3-bit version and generalize:

c	b	a	c ⁺	b ⁺	a ⁺
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

$$a^+ = a'$$

$$b^+ = a \oplus b$$

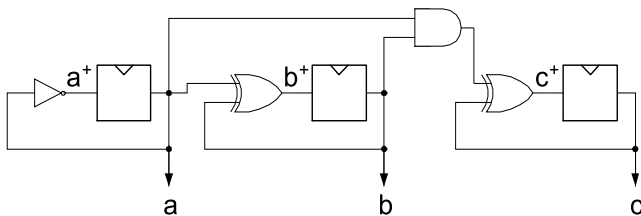
cb	00	01	11	10
a	0	0	1	1
0	0	0	1	1
1	0	1	0	1

$$c^+ = a'c + abc' + b'c$$

$$= c(a'+b') + c'(ab)$$

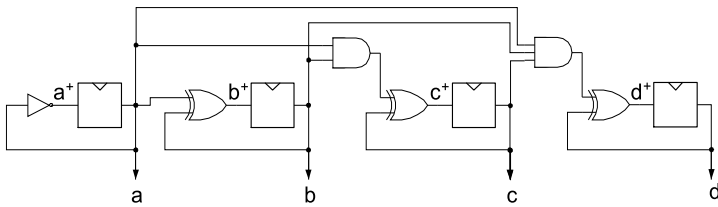
$$= c(ab)'+ c'(ab)$$

$$= c \oplus ab$$

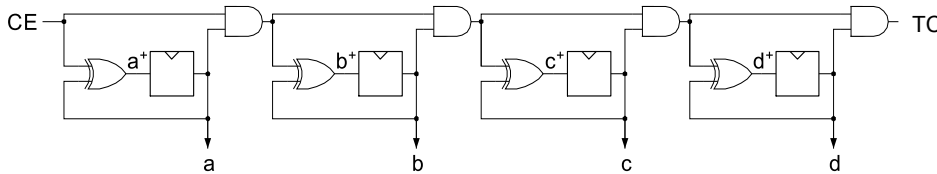


Synchronous Counters

- How do we extend to n-bits?
- Extrapolate c^+ : $d^+ = d \oplus abc$, $e^+ = e \oplus abcd$

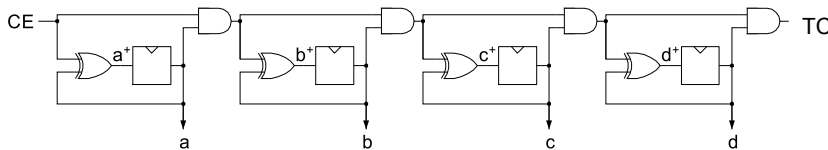


- Has difficulty scaling (AND gate inputs grow with n)

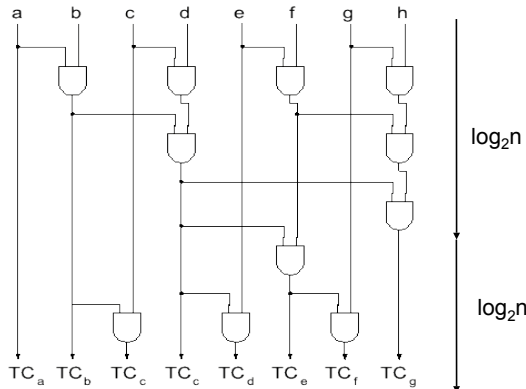


- CE is "count enable", allows external control of counting,
- TC is "terminal count", is asserted on highest value, allows cascading, external sensing of occurrence of max value.

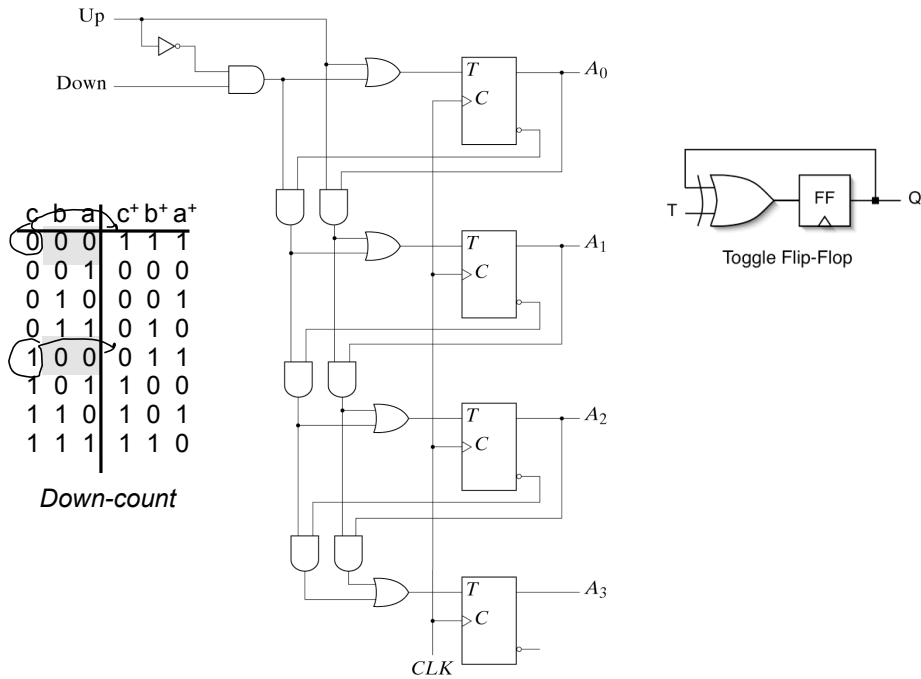
Synchronous Counters



- How does this one scale?
- ☹ Delay grows $\propto n$
- Generation of TC signals very similar to generation of carry signals in adder.
- "Parallel Prefix" circuit reduces delay:



Up-Down Counter



Spring 2010

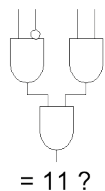
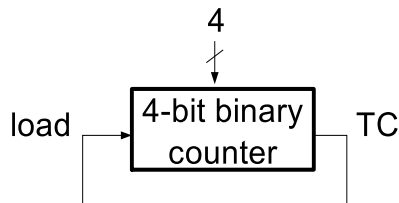
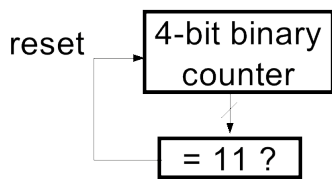
Fig. 6-13 4-Bit Up-Down Binary Counter

Page 25

Odd Counts

- Extra combinational logic can be added to terminate count before max value is reached:
- Example: **count to 12**

- Alternative:



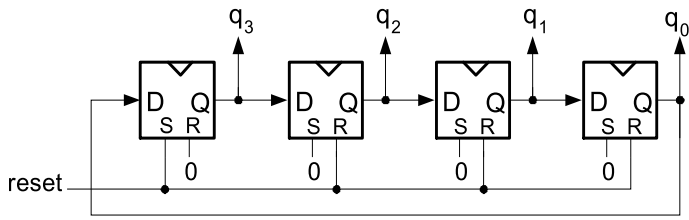
Spring 2010

EECS150 - Lec22-counters

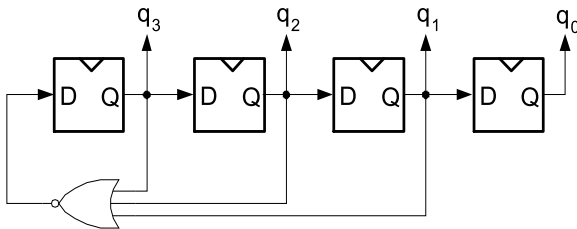
Page 26

Ring Counters

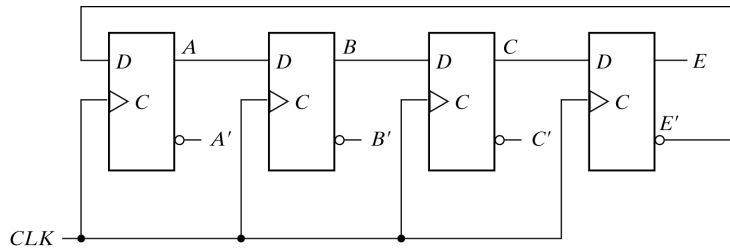
- "one-hot" counters
 - What are these good for?
- 0001, 0010, 0100, 1000, 0001, ...



"Self-starting" version:



Johnson Counter



(a) Four-stage switch-tail ring counter

Sequence number	Flip-flop outputs				AND gate required for output
	A	B	C	E	
1	0	0	0	0	$A'E'$
2	1	0	0	0	AB'
3	1	1	0	0	BC'
4	1	1	1	0	CE'
5	1	1	1	1	AE
6	0	1	1	1	$A'B$
7	0	0	1	1	$B'C$
8	0	0	0	1	$C'E$

(b) Count sequence and required decoding

Asynchronous "Ripple" counters

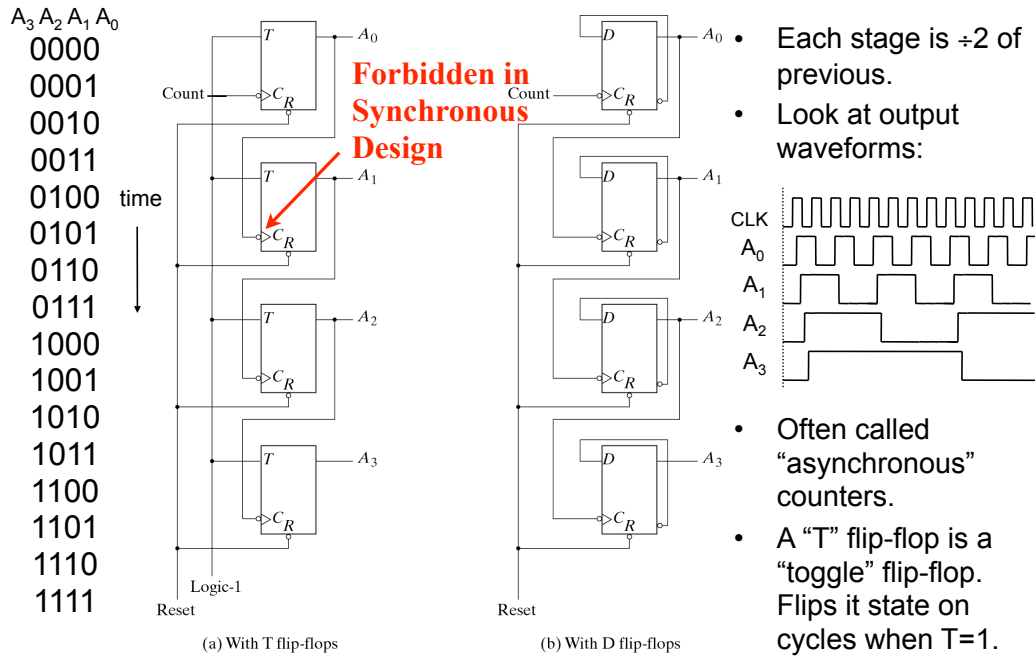


Fig. 6-8 4-Bit Binary Ripple Counter