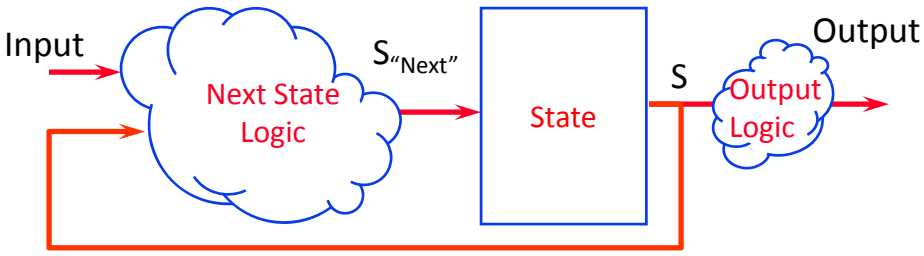
18-100 Lecture 24: Sequential Logic Design

James C. Hoe
Dept of ECE, CMU
April 21, 2015

Today's Goal:	Start thinking about stateful stuff
Announcements:	Read Rizzoni 12.6
	HW 9 due
	Exam 3 on April 30
	Final Exam, Fri., May 8, 8:30~11:30, GHC 4401 *Conflicts?*
Handouts:	HW 10 (on Blackboard)
	HW 9 solutions (on Blackboard later on)
	Lab 12 (on Blackboard later on)

Let's refresh our memory….

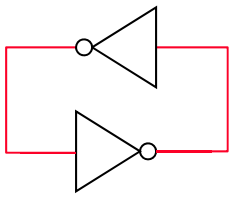
- All "sequential" digital systems comprise



  - input and output
  - state stuff that remembers
  - "combinational" stuff that computes a function (has no memory)
- In an execution, state is updated to a "next state" based on a function of the current state

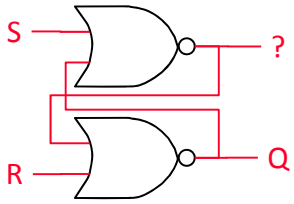
Memory 101

- Pure combinational logic always go from a current input to a current output without any looping back
  There is no notion of time, past or future
- To remember, must somehow incorporate previous values
- You mean like this?



Does it remember? What does it remember?
(It is easier to see if you associate a small propagation delay with wires and gates)

A Better Try: the SR Latch

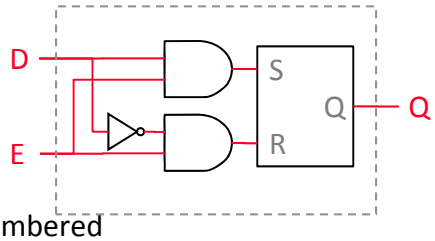


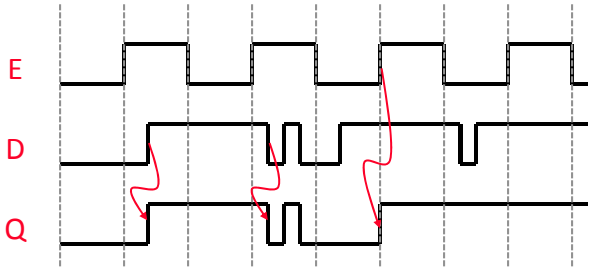
- Keep in mind that
  - X+0=X ⇒ (X+0)'=X'
  - X+1=1 ⇒ (X+1)'=0
- Hint: S stands for "set"; R for "reset"
- Consider
  - S=0 and R=0: the NORs simply act like inverters in the feedback loop; Q is remembered
  - S=1 and R=0: the top NOR's output is forced to 0; the bottom NOR inverts the feedback; Q is set to 1
  - S=0 and R=1: the bottom NOR's output is forced to 0; the top NOR inverts the feedback; Q is reset to 0
  - S=1 and R=1: **Just don't do it**
    After asserting S or R, the resulting Q is remembered when S and R are both deasserted again

If you feel brave try finding the "dual"

Level-Sensitive D Latch

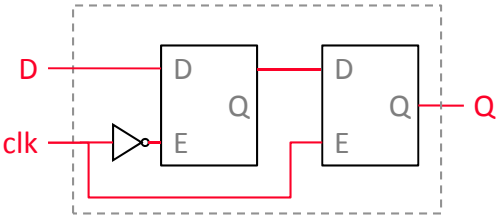
- E is "latch enable"
  - when E is asserted Q follows D combinationally
  - when E is de-asserted the last Q value is remembered
- Timing Diagram



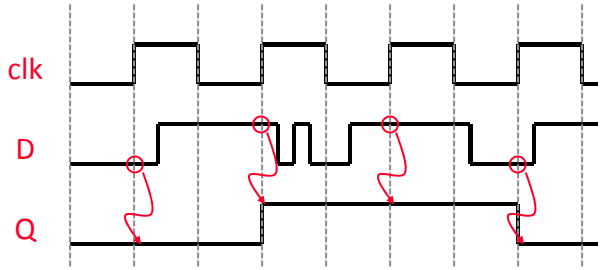


Edge-Triggered D Flip-Flop

- Q follows D only at the "instant" of rising edge of clock
- Q independent of D at all other time
- Timing Diagram

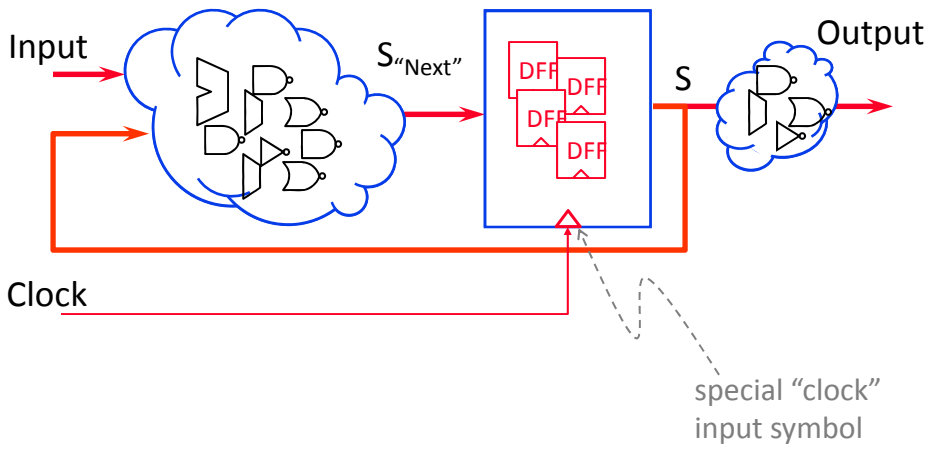


(master-slave)

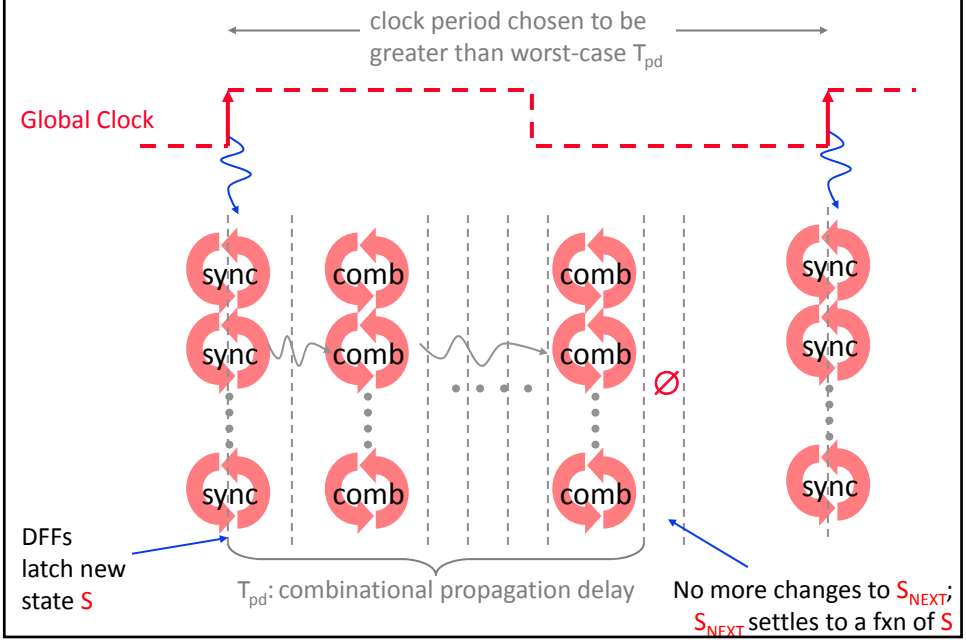


Q only make "synchronous" transitions

Synchronous Finite State Machines



Synchronous Timing (Simplified)

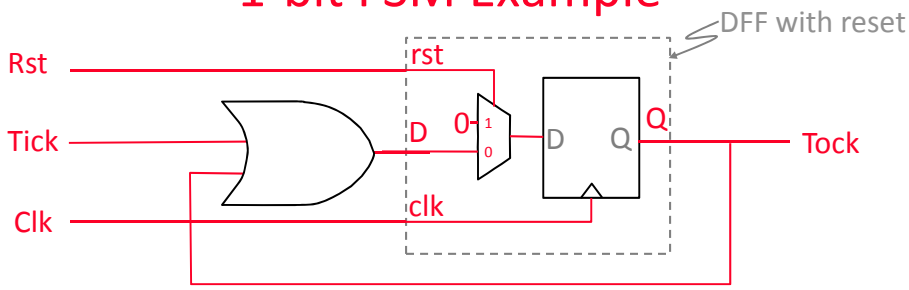


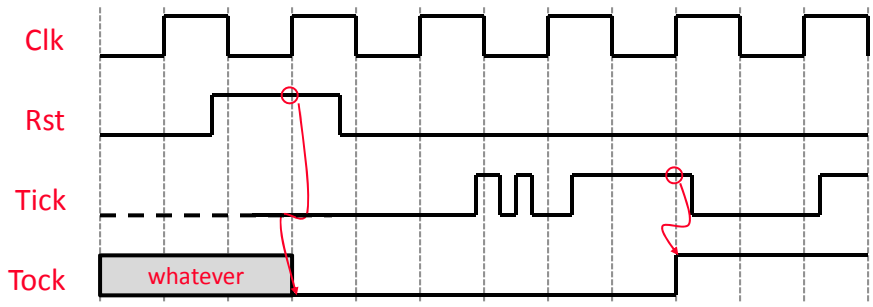
Let's play with this a bit first

1-bit FSM Example

- Implied input: Reset, Clk
- Input
  - 1-bit signal "Tick"
  - you can vary its value over time, anytime you like, but only the values at the rising clock edges matter
- Output
  - 1-bit signal "Tock"
  - Tock should be 0 after reset
  - Tock should become 1 and stay 1 after Tick has been 1 (as sampled on a rising clock edge)
- What are the two states?
  - Tick has never been 1 since reset
  - not the above

1-bit FSM Example

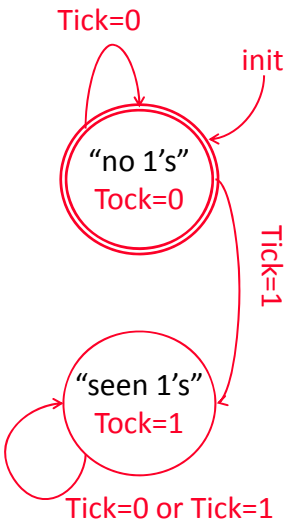




FSM as an Abstraction

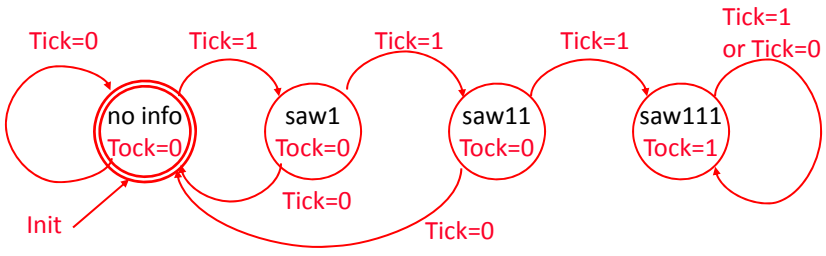
State Transition Diagram

- A convenient FSM abstraction
- States (bubbles)
  - best to give each state a meaningful "name"
  - output value associated with each state (Moore machines)
  - one state is designated as the initialization state
- Transitions (edges)
  - edges are predicated by input conditions (i.e., follow this edge if condition is met)
  - each state must have transitions for all possible input values



Let's play a bit more

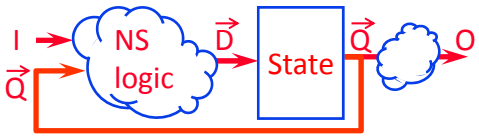
- New design: Tock should become 1 and stay 1 after Tick has been 1 three consecutive times!



What if I want Tock=1 only right after each time Tick has been 1 three consecutive times; Tock=0 at all other times

Realizing an FSM

- State assignment
  - require at least $n=\lceil \lg_2 N \rceil$ bits $Q_{n-1},\ldots,Q_0$ to encode an FSM with N states (each bit is a D flip-flop)
  - assign each state to an unique encoding
  - choice of encoding can affect the size of combinational next-state logic (don't worry about it in 18-100)
- Next-state logic
  - computes the next state value $D_{n-1},\ldots,D_0$ as a function of the FSM input and current state $Q_{n-1},\ldots,Q_0$
- "Moore-style" Output logic
  - computes the FSM output as a function of current state $Q_{n-1},\ldots,Q_0$



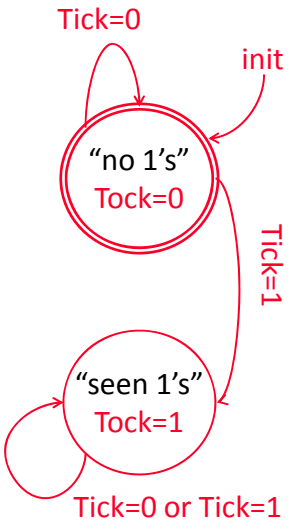
Tick-Tock Example

- 2 states, hence a 1-bit FSM
  - "no 1's" when Q_0=0
  - "seen 1's" when Q_0=1
- Next-state logic truth table

Q_0	Tick	D_0
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

- Output logic truth table

Q_0	Tock
0	**0**
1	**1**



Three 1's Example

state assignment

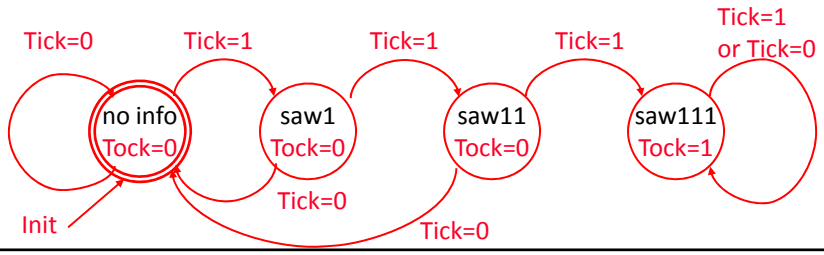
State	Q_1	Q_0
no info	0	0
saw 1	0	1
saw11	1	0
saw111	1	1

next-state logic

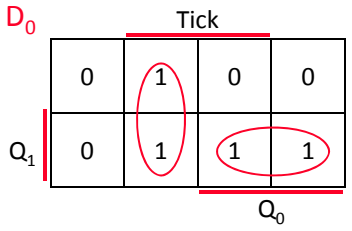
Q_1	Q_0	Tick	D_1	D_0
0	0	0	**0**	**0**
		1	**0**	**1**
0	1	0	**0**	**0**
		1	**1**	**0**
1	0	0	**0**	**0**
		1	**1**	**1**
1	1	0	**1**	**1**
		1	**1**	**1**

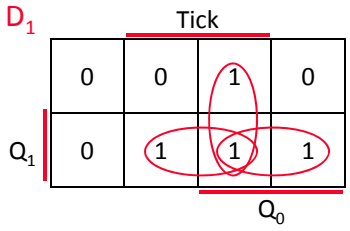
output logic

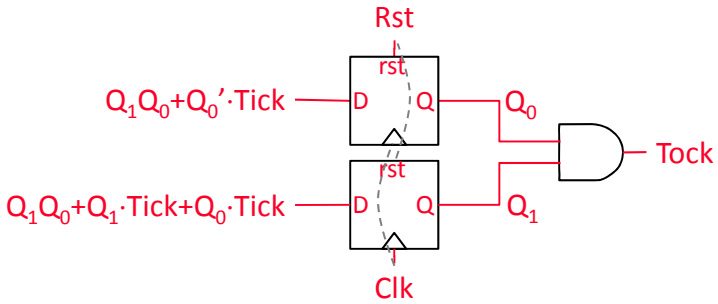
Q_1	Q_0	**Tock**
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**



Three 1's Example

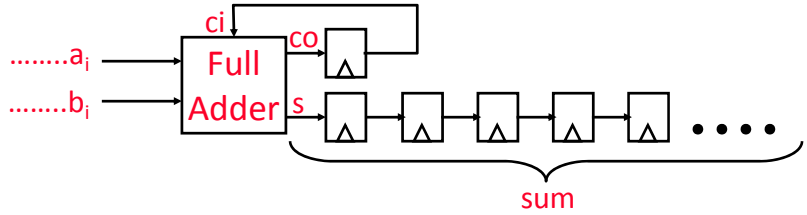






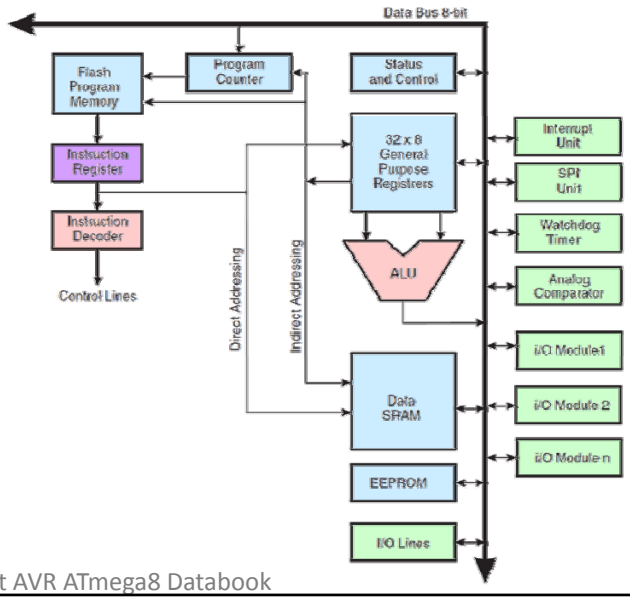
Why FSMs?

- Looking for 0s and 1s . . . . . (this is actually serious computer science; look up "regular expressions" on Wikipedia)
- Simple computation/calculations



- The first and foremost practical use is to sequence the control of a "datapath" or a "system"

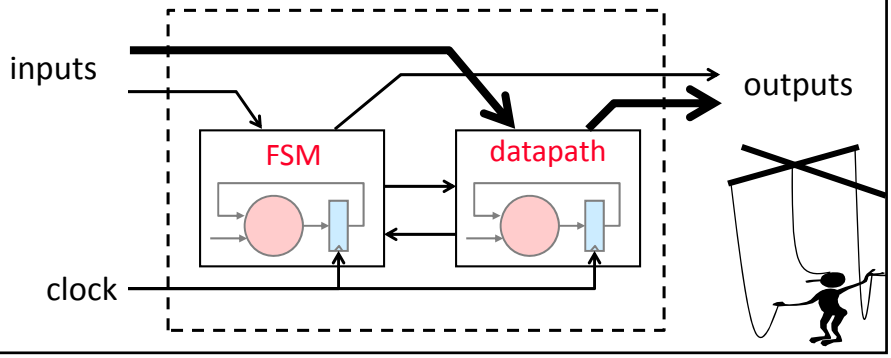
Atmel ATmega8 Datapath



Page 9 Atmel 8-bit AVR ATmega8 Databook

FSM-D

- datapath = combinational logic and registers to carry out computation (puppet)
- FSM = combinational logic and registers for control and sequencing (puppeteer)



SR Latch "Dual"