Advanced Digital Logic Design – EECS 303

http://ziyang.eecs.northwestern.edu/eecs303/

Teacher:	Robert Dick
Office:	L477 Tech
Email:	dickrp@northwestern.edu
Phone:	847–467–2298

NORTHWESTERN UNIVERSITY

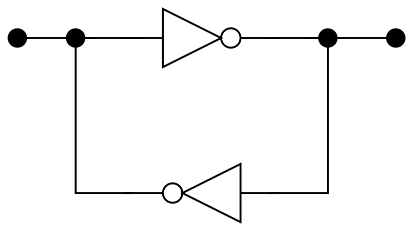
Flip-flop introduction

- Stores, and outputs, a value
- Puts a special clock signal in charge of timing
- Allows output to change in response to clock transition
- More on this later
  - Timing and sequential circuits

Introduction to sequential elements

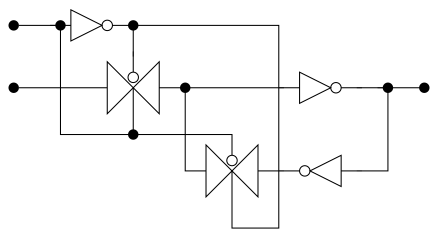
- Feedback and memory
- Memory
- Latches

Feedback and memory



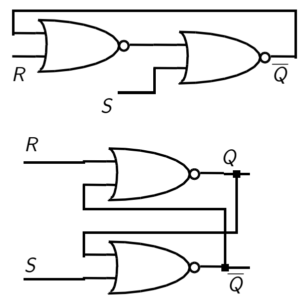
- Feedback is the root of memory
- Can compose a simple loop from NOT gates
- However, there is no way to switch the value

TG and NOT-based memory

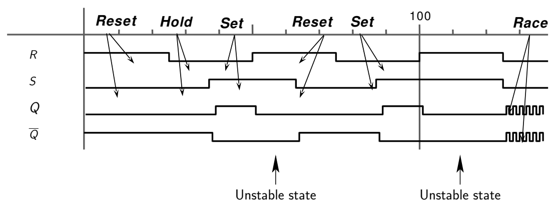


- Can break feedback path to load new value
- However, potential for timing problems

Reset/set latch

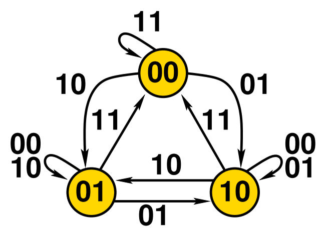


Reset/set timing

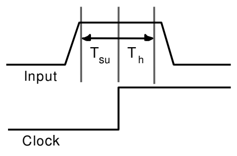


RS latch state diagram

output=$Q\ \overline{Q}$
input=$R\ S$

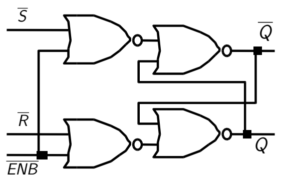


Clocking terms

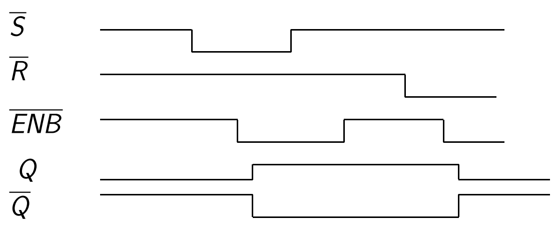


- Clock – Rising edge, falling edge, high level, low level, period
- Setup time: Minimum time before clocking event by which input must be stable (T_{SU})
- Hold time: Minimum time after clocking event for which input must remain stable (T_H)
- Window: From setup time to hold time

Gated RS latch



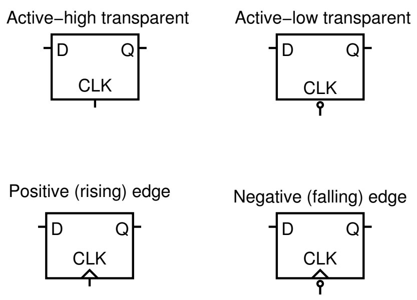
Gated RS latch



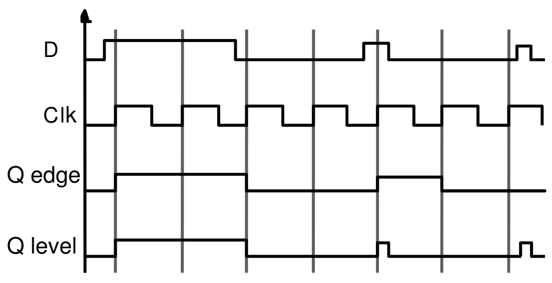
Memory element properties

Type	Inputs sampled	Outputs valid
Unclocked latch	Always	LFT
Level-sensitive latch	Clock high	LFT
	(T_{SU} to T_H) around falling clock edge	
Edge-triggered flip-flop	Clock low-to-high transition	Delay from rising edge
	(T_{SU} to T_H) around rising clock edge	

Clocking conventions



Timing for edge and level-sensitive latches

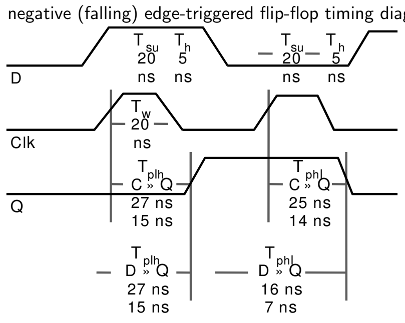


Latch timing specifications

- Minimum clock width, T_W
  - Usually period / 2
- Low to high propegation delay, P_{LH}
- High to low propegation delay, P_{HL}
- Worst-case and typical

Latch timing specifications

Example, negative (falling) edge-triggered flip-flop timing diagram

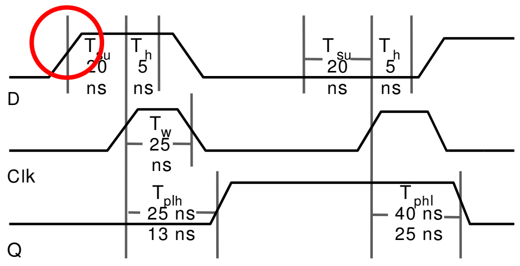


FF timing specifications

- Minimum clock width, T_W
  - Usually period / 2
- Low to high propagation delay, P_{LH}
- High to low propagation delay, P_{HL}

FF timing specifications

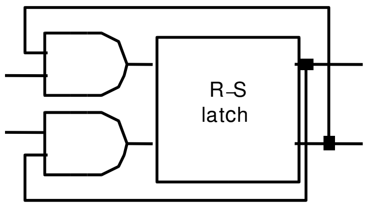
Example, positive (rising) edge-triggered flip-flop timing diagram



RS latch states

S	R	Q^+	$\overline{Q}^+$	Notes
0	0	Q	$\overline{Q}$	
0	1	0	1	
1	0	1	0	
1	1	1	1	unstable

JK latch



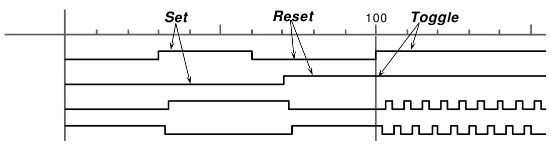
Use output feedback to ensure that $RS \neq 11$

$Q^+ = Q\ \overline{K} + \overline{Q}\ J$

JK latch

J	K	Q	Q^+	
0	0	0	0	hold
0	0	1	1	
0	1	0	0	reset
0	1	1	0	
1	0	0	1	set
1	0	1	1	
1	1	0	1	toggle
1	1	1	0	

JK race

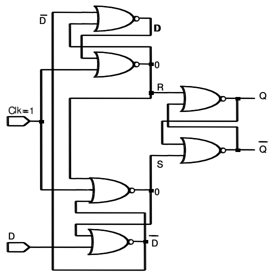


Race Condition

Falling edge-triggered D flip-flop

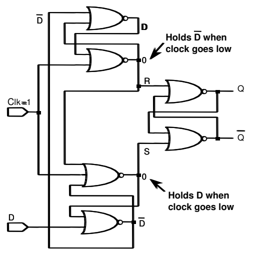
- Use two stages of latches
- When clock is high
  - First stage samples input w.o. changing second stage
  - Second stage holds value
- When clock goes low
  - First stage holds value and sets or resets second stage
  - Second stage transmits first stage
- $Q^+ = D$
- One of the most commonly used flip-flops

Falling edge-triggered D flip-flop



Clock high

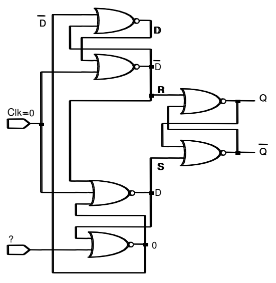
Falling edge-triggered D flip-flop


$\overline{D}$
D
Holds $\overline{D}$ when clock goes low
0
R
Q
Clk=1
$\overline{Q}$
S
0
Holds D when clock goes low
D
$\overline{D}$

Clock switching

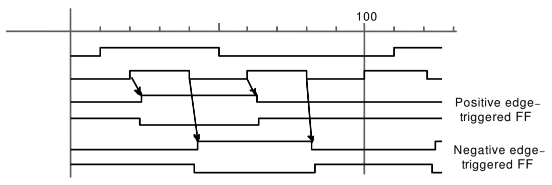
Inputs sampled on falling edge, outputs change after falling edge

Falling edge-triggered D flip-flop



Clock low

Edge triggered timing



RS clocked latch

- Storage element in narrow width clocked systems
- Dangerous
- Fundamental building block of many flip-flop types

JK flip-flop

- Versatile building block
- Building block for D and T flip-flops
- Has two inputs resulting in increased wiring complexity
- Don't use master/slave JK flip-flops
  - Ones or zeros catching
- Edge-triggered varieties exist

D flip-flop

- Minimizes input wiring
- Simple to use
- Common choice for basic memory elements in sequential circuits

Toggle (T) flip-flops

- State changes each clock tick
- Useful for building counters
- Can be implemented with other flip-flops
  - JK with inputs high
  - D with XOR feedback

Asynchronous inputs

- How can a circuit with numerous distributed edge-triggered flip-flops be put into a known state
- Could devise some sequence of input events to bring the machine into a known state
  - Complicated
  - Slow
  - Not necessarily possible, given trap states
- Can also use sequential elements with additional asynchronous reset and/or set inputs

Latch and flip-flop equations

RS

$$Q^+ = S + \overline{R}\ Q$$

D

$$Q^+ = D$$

Latch and flip-flop equations

JK

$$Q^+ = J\ \overline{Q} + \overline{K}\ Q$$

T

$$Q^+ = T \oplus Q$$

Sequential FSM design example

- We'll walk through the design of an example finite state machine (FSM)
- Some of the stages will be covered in more detail in later lectures
- I want you to have a high-level understanding of our overall goal before covering every detail of FSM synthesis

Regular expressions

- Naturally express control
- However, no simple direct HW implementation
- We want to get to sequential logic
- Need to go though other stages first

Regular expressions

Can be expressed with regular expressions, examples

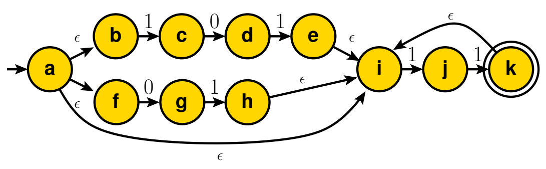
- Accept the empty string, ϵ
- Accept nothing, $\varnothing$
- Accept 0 or 11, $(0 + 11)$
- Accept anything starting with 1 and one or more 0 and ending with 0 or 10, $10^+(0 + 10)$
- Accept anything starting with zero or more 0010 or 1 and ending with 1, $(0010 + 1)^*1$

Nondeterministic finite state automata (NFA)

- State graph
- Multiple states can be active at the same time
- Some states ACCEPT
- The automata accepts if any accepting states are active

NFA

$$(101 + 01 + \epsilon)(11)^+$$



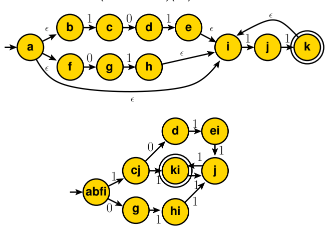
0 1 1 1

Deterministic finite state automata (DFA)

- NFAs require multiple states to be simultaneously active
- Can't represent this with conventional logic state variables
- Need to convert to deterministic representation

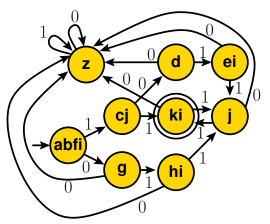
DFA

$(101 + 01 + \epsilon)(11)^+$



DFA to more explicit FSM

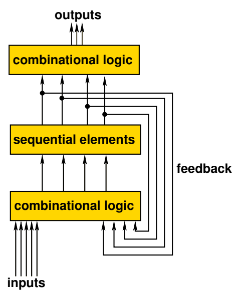
$(101 + 01 + \epsilon)(11)^+$



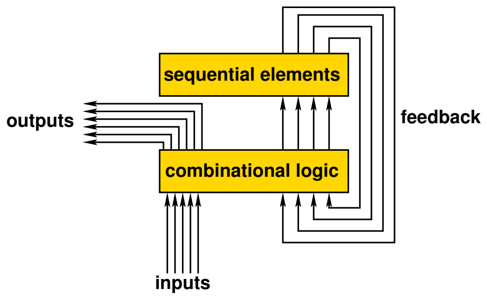
DFA to FSM

- DFA may only accept or reject
- Simple to convert Moore FSM
- Add explicit output values to states

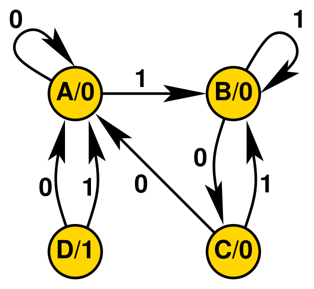
Moore block diagram



Mealy block diagram



Moore FSMs



Introduction to state reduction

	s^+		
s	0	1	Q
A	A	B	0
B	C	B	0
C	A	B	0
D	A	A	1

Introduction to state reduction

	s^+		
s	0	1	q
AC	AC	B	0
B	AC	B	0
D	AC	AC	1

State assignment

	s^+		
s	0	1	q
ABC	ABC	ABC	0
D	ABC	ABC	1

Only two adjacent states, state assignment is trivial

However, good to consider output, q

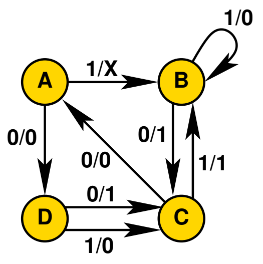
State variable functions

	s^+		
s	0	1	q
0	0	0	0
1	0	0	1

$s^+(s) = 0$

$q(s) = s$

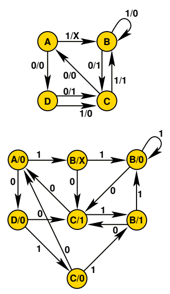
Mealy FSMs



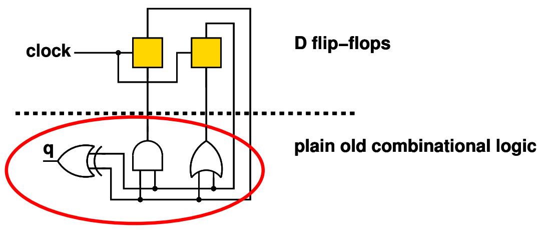
Mealy tabular form

	s^+/q	
s	0	1
A	D/0	B/X
B	C/1	B/0
C	A/0	B/1
D	C/1	C/0

Mealy to Moore conversion



State variable combinational synthesis



- Separate sequential and combinational portions of circuit
- Conduct standard logic synthesis

FSM design summary

- Specify requirements in natural form – regular expression or NFA
- Converting from NFA to DFA is straightforward
- Converting from DFA to FSM is straightforward
- Minimize the number of states using compatible states, class sets, and binate covering
- Assign values to states to minimize logic complexity
- Allow only adjacent or path transitions for asynchronous machines
- Optimize implementation of state and output functions

Recommended reading

- M. Morris Mano and Charles R. Kime. *Logic and Computer Design Fundamentals*. Prentice-Hall, NJ, third edition, 2004
- Chapter 7
- Chapter 6

Video controller repair lab

- Design a finite state machine based on an English problem specification
- The design problem isn't very difficult
- Going from a real-world problem to a formal representation may be difficult
- Be careful not to use too many state variables!!!
  - Could easily turn it from a 6–hour lab to a 12–hour lab

Next lecture

- More detail and examples on FSM design and optimization