

EECS150

Section 2

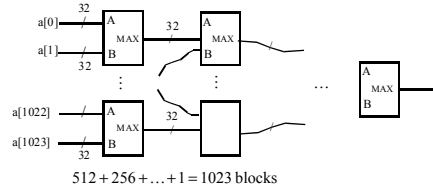
Introduction to Sequential Logic

Fall 2001



ALGORITHMS IN HW

Approach #2: Combinational – divide & conquer



Each MAX block has: 64 inputs; 32 outputs
 2^{64} entries in truth table
 Delay = ? $3 \times 10 = 30$ units

COMBINATIONAL vs. SEQUENTIAL

Combinational

$y = f(\text{inputs})$
 n inputs, m outputs

Examples:

Adder: $y[0:3] = a[0:3] + b[0:3]$
 memoryless systems

Sequential

$y = f(\text{inputs, time})$
 (most useful: clocked logic)

Example: (down) counter

Time (e.g. sec)	Y[0:3]
0	10 ₁₀
1	9
2	8
...	...
10 ₁₀	0



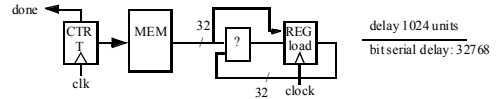
ALGORITHMS IN HW

Approach #3: Sequential

```
max = 0;
for (i=0; i<32768, i++)
    if (a[i]>max) max=a[i];
```

Hardware equivalence:

$a[i] \rightarrow 1024 \times 32$ bit memory
 for $(i=0; i<1024; i++) \rightarrow 15$ bit counter
 if $(a[i]>max) \rightarrow$ "comparator" (combinational)
 $max=a[i] \rightarrow$ register



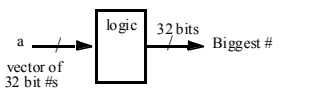
ALGORITHMS IN HW

Example: Finding the largest number in a list.

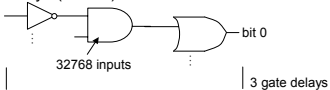
Unsigned int $a[1024]$; 32 bit int

Approach #1: Combinational

Number of input variables? 32768
 Number of lines in truth table? 2^{32768}



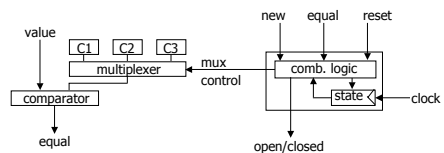
time delay? (S.O.P.)



Sequential Circuits

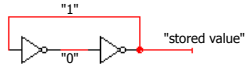
Circuits with Feedback

- Outputs = $f(\text{inputs, past inputs, past outputs})$
- Basis for building "memory" into logic circuits
- Door combination lock is an example of a sequential circuit
 - State is memory
 - State is an "output" and an "input" to combinational logic
- Combination storage elements are also memory

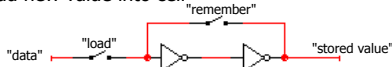


Simplest Circuits with Feedback

- Two inverters form a static memory cell
 - Will hold value as long as it has power applied



- How to get a new value into the memory cell?
 - Selectively break feedback path
 - Load new value into cell



EECS150 - Fall 2001

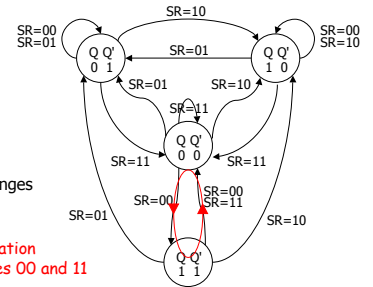
1-7

R-S Latch Behavior

S	R	Q
0	0	hold
0	1	0
1	0	1
1	1	unstable

- State Diagram
 - States: possible values
 - Transitions: changes based on inputs

possible oscillation between states 00 and 11

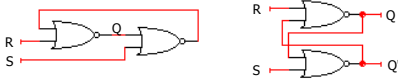


EECS150 - Fall 2001

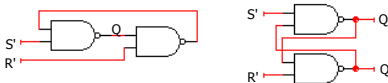
1-10

Memory with Cross-coupled Gates

- Cross-coupled NOR gates
 - Similar to inverter pair, with capability to force output to 0 (reset=1) or 1 (set=1)



- Cross-coupled NAND gates
 - Similar to inverter pair, with capability to force output to 0 (reset=0) or 1 (set=0)

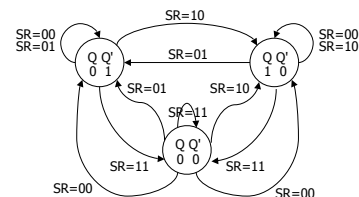


EECS150 - Fall 2001

1-8

Observed R-S Latch Behavior

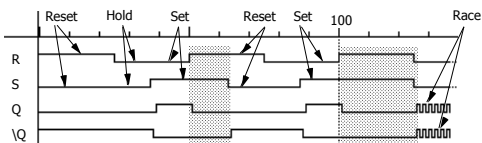
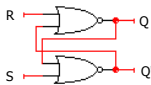
- Very difficult to observe R-S latch in the 1-1 state
 - One of R or S usually changes first
- Ambiguously returns to state 0-1 or 1-0
 - A so-called "race condition"
 - Or non-deterministic transition



EECS150 - Fall 2001

1-11

Timing Behavior

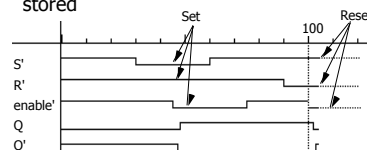
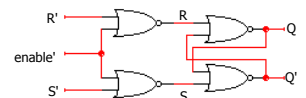


EECS150 - Fall 2001

1-9

Gated R-S Latch

- Control when R and S inputs matter
 - Otherwise, the slightest glitch on R or S while enable is low could cause change in value stored

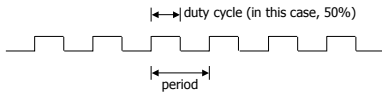


EECS150 - Fall 2001

1-12

Clocks

- Used to keep time
 - Wait long enough for inputs (R' and S') to settle
 - Then allow to have effect on value stored
- Clocks are regular periodic signals
 - Period (time between ticks)
 - Duty-cycle (time clock is high between ticks - expressed as % of period)

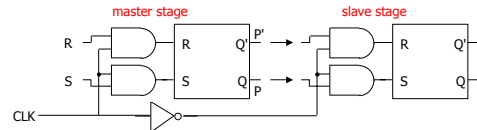


EECS150 - Fall 2001

1-13

Master-Slave Structure

- Break flow by alternating clocks (like an air-lock)
 - Use positive clock to latch inputs into one R-S latch
 - Use negative clock to change outputs with another R-S latch
- View pair as one basic unit
 - master-slave flip-flop
 - twice as much logic
 - output changes a few gate delays after the falling edge of clock but does not affect any cascaded flip-flops

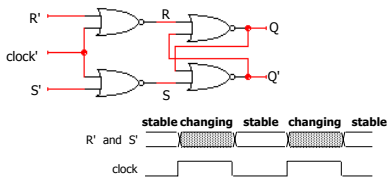


EECS150 - Fall 2001

1-16

Clocks (cont'd)

- Controlling an R-S latch with a clock
 - Can't let R and S change while clock is active (allowing R and S to pass)
 - Only have half of clock period for signal changes to propagate
 - Signals must be stable for the other half of clock period

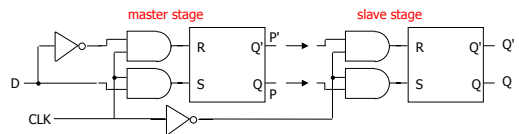


EECS150 - Fall 2001

1-14

D Flip-Flop

- Make S and R complements of each other
 - Eliminates 1s catching problem
 - Can't just hold previous value (must have new value ready every clock period)
 - Value of D just before clock goes low is what is stored in flip-flop
 - Can make R-S flip-flop by adding logic to make $D = S + R'Q$



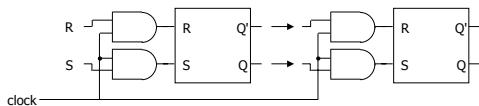
10 gates

EECS150 - Fall 2001

1-17

Cascading Latches

- Connect output of one latch to input of another
- How to stop changes from racing through chain?
 - Need to control flow of data from one latch to the next
 - Advance from one latch per clock period
 - Worry about logic between latches (arrows) that is too fast

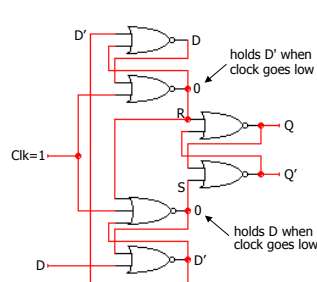


EECS150 - Fall 2001

1-15

Edge-Triggered Flip-Flops

- More efficient solution: only 6 gates
 - sensitive to inputs only near edge of clock signal (not while high)



negative edge-triggered D flip-flop (D-FF)

4-5 gate delays
must respect setup and hold time constraints to successfully capture input



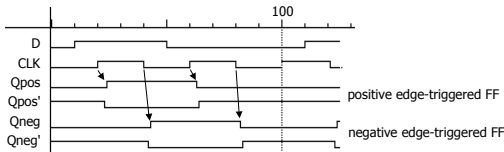
characteristic equation
 $Q(t+1) = D$

EECS150 - Fall 2001

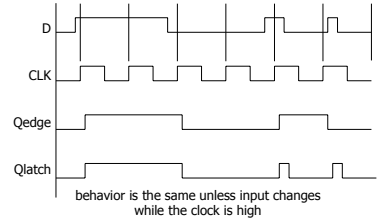
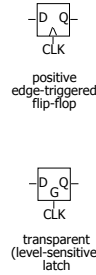
1-18

Edge-Triggered Flip-Flops

- Positive edge-triggered
 - Inputs sampled on rising edge; outputs change after rising edge
- Negative edge-triggered flip-flops
 - Inputs sampled on falling edge; outputs change after falling edge



Comparison of Latches & Flip-Flops



Timing Methodologies

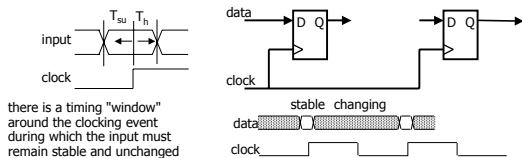
- Rules for interconnecting components and clocks
 - Guarantee proper operation of system when strictly followed
- Approach depends on building blocks used for memory elements
 - Focus on systems with edge-triggered flip-flops
 - Found in programmable logic devices
 - Many custom integrated circuits focus on level-sensitive latches
- Basic rules for correct timing:
 - (1) Correct inputs, with respect to time, are provided to the flip-flops
 - (2) No flip-flop changes state more than once per clocking event

Comparison of Latches & Flip-Flops

Type	When inputs are sampled	When output is valid
unlocked latch	always	propagation delay from input change
level-sensitive latch	clock high (Tsu/Th around falling edge of clock)	propagation delay from input change or clock edge (whichever is later)
master-slave flip-flop	clock high (Tsu/Th around falling edge of clock)	propagation delay from falling edge of clock
negative edge-triggered flip-flop	clock hi-to-lo transition (Tsu/Th around falling edge of clock)	propagation delay from falling edge of clock

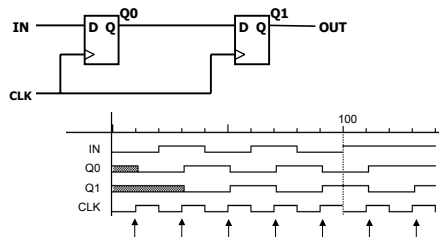
Timing Methodologies (cont'd)

- Definition of terms
 - clock: periodic event, causes state of memory element to change; can be rising or falling edge, or high or low level
 - setup time: minimum time before the clocking event by which the input must be stable (Tsu)
 - hold time: minimum time after the clocking event until which the input must remain stable (Th)



Cascading Edge-triggered FFs

- Shift register
 - New value goes into first stage
 - While previous value of first stage goes into second stage
 - Consider setup/hold/propagation delays (prop must be > hold)



Metastability: Asynchronous inputs

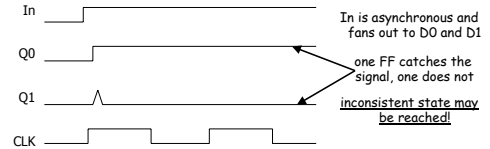
- Clocked synchronous circuits
 - Inputs, state, and outputs sampled or changed in relation to a common reference signal (called the clock)
 - E.g., master/slave, edge-triggered
- Asynchronous circuits
 - Inputs, state, and outputs sampled or changed independently of a common reference signal (glitches/hazards a major concern)
 - E.g., R-S latch
- Asynchronous inputs to synchronous circuits
 - Inputs can change at any time, will not meet setup/hold times
 - Dangerous, synchronous inputs are greatly preferred
 - Cannot be avoided (e.g., reset signal, memory wait, user input)

EECS150 - Fall 2001

1-25

Handling Asynchronous Inputs

- What can go wrong?
 - Input changes too close to clock edge (violating setup time constraint)

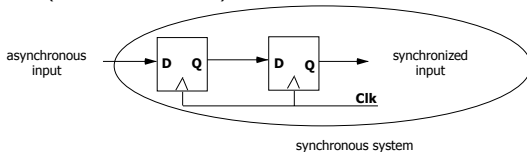


EECS150 - Fall 2001

1-28

Correcting Synchronization Failure

- Probability of failure can never be reduced to 0, but it can be reduced
 - (1) slow down the system clock: this gives the synchronizer more time to decay into a steady state; synchronizer failure becomes a big problem for very high speed systems
 - (2) use fastest possible logic technology in the synchronizer: this makes for a very sharp "peak" upon which to balance
 - (3) cascade two synchronizers: this effectively synchronizes twice (both would have to fail)



EECS150 - Fall 2001

1-26

Flip-Flop Features

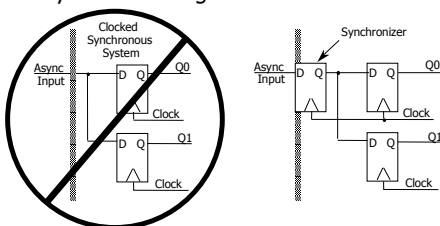
- Reset (set state to 0) – R
 - Synchronous: $D_{new} = R' \cdot D_{old}$ (when next clock edge arrives)
 - Asynchronous: doesn't wait for clock, quick but dangerous
- Preset or set (set state to 1) – S (or sometimes P)
 - Synchronous: $D_{new} = D_{old} + S$ (when next clock edge arrives)
 - Asynchronous: doesn't wait for clock, quick but dangerous
- Both reset and preset
 - $D_{new} = R' \cdot D_{old} + S$ (set-dominant)
 - $D_{new} = R' \cdot D_{old} + R'S$ (reset-dominant)
- Selective input capability (input enable/load) – LD or EN
 - Multiplexer at input: $D_{new} = LD' \cdot Q + LD \cdot D_{old}$
 - Load may/may not override reset/set (usually R/S have priority)
- Complementary outputs – Q and Q'

EECS150 - Fall 2001

1-29

Handling Asynchronous Inputs

- Never allow asynchronous inputs to fan-out to more than one flip-flop
 - Synchronize as soon as possible and then treat as synchronous signal

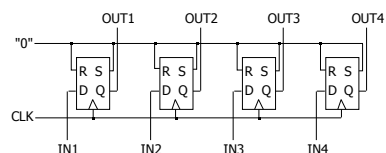


EECS150 - Fall 2001

1-27

Registers

- Collections of flip-flops with similar controls and logic
 - Stored values somehow related (e.g., form binary value)
 - Share clock, reset, and set lines
 - Similar logic at each stage
- Examples
 - Shift registers
 - Counters

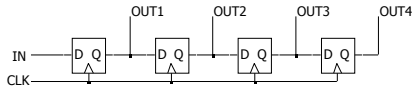


EECS150 - Fall 2001

1-30

Shift Register

- Holds samples of input
 - Store last 4 input values in sequence
 - 4-bit shift register:

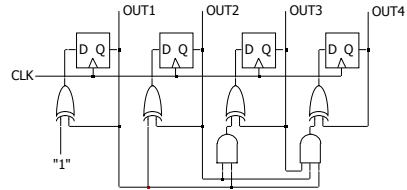


EECS150 - Fall 2001

1-31

Binary Counter

- Logic between registers (not just multiplexer)
 - XOR decides when bit should be toggled
 - Always for low-order bit, only when first bit is true for second bit, and so on

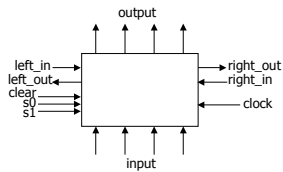


EECS150 - Fall 2001

1-34

Universal Shift Register

- Holds 4 values
 - Serial or parallel inputs
 - Serial or parallel outputs
 - Permits shift left or right
 - Shift in new values from left or right



clear sets the register contents and output to 0

s1 and s0 determine the shift function

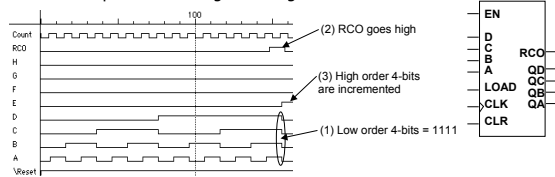
s0	s1	function
0	0	hold state
0	1	shift right
1	0	shift left
1	1	load new input

EECS150 - Fall 2001

1-32

4-bit Synchronous Up-Counter

- Standard component with many applications
 - Positive edge-triggered FFs w/ sync load and clear inputs
 - Parallel load data from D, C, B, A
 - Enable inputs: must be asserted to enable counting
 - RCO: ripple-carry out used for cascading counters
 - high when counter is in its highest state 1111
 - implemented using an AND gate

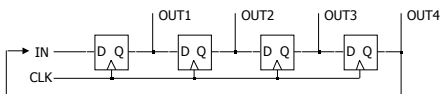


EECS150 - Fall 2001

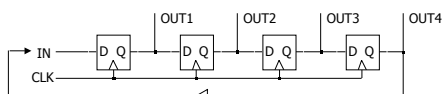
1-35

Counters

- Sequences through a fixed set of patterns
 - In this case, 1000, 0100, 0010, 0001
 - If one of the patterns is its initial state (by loading or set/reset)



- Mobius (or Johnson) counter
 - In this case, 1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000



EECS150 - Fall 2001

1-33