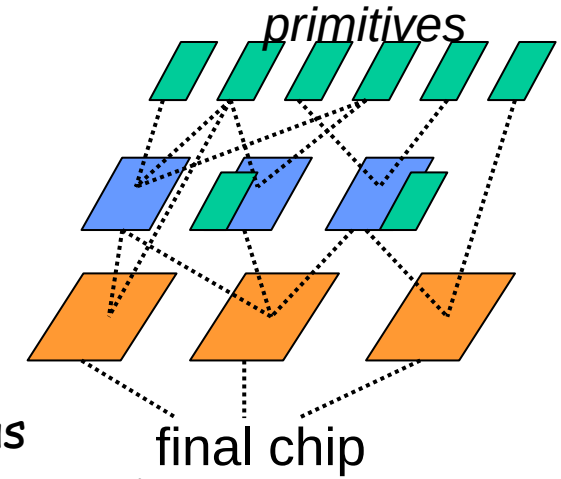
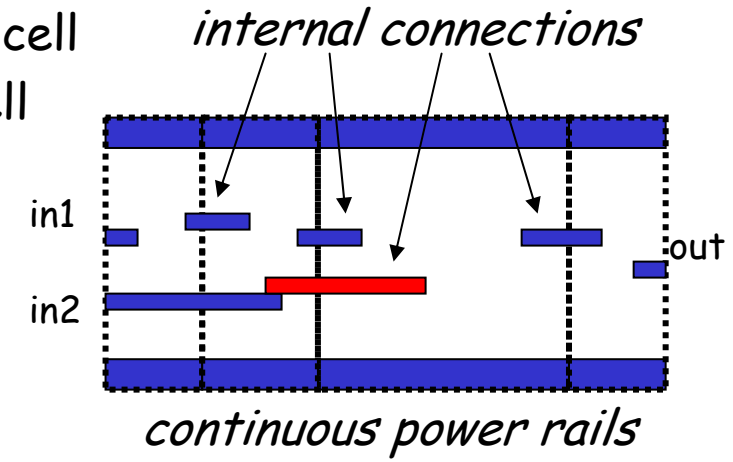
Layout of Multiple Cells

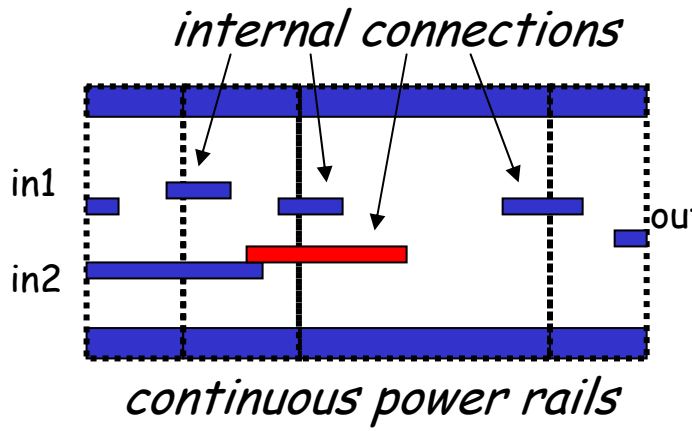
- Beyond the primitive tier
  - add instances of primitives
  - add additional transistors if necessary
    - add substrate/well contacts (plugs)
  - add additional polygons where needed
    - add metal-1 to make VDD/GND rail continuous
    - add n-well to avoid breaks in n-wells that violate rules
    - add interconnects and contacts to make signal interconnections
  - connect signals within cell boundary
    - if possible, keep internal signal within cell
    - ensure cell I/Os accessible outside cell
  - minimize layout area
    - avoid unnecessary gaps between cells
  - pass design rule check
    - ALWAYS, at every cell level



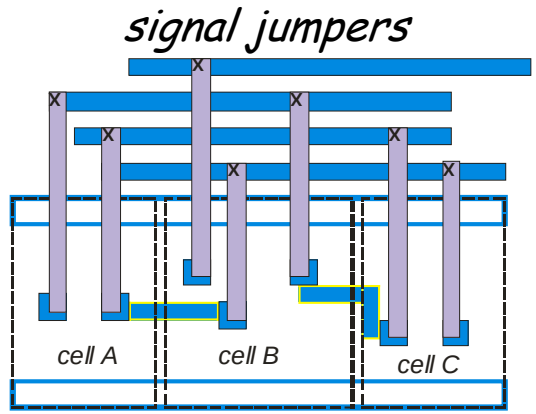


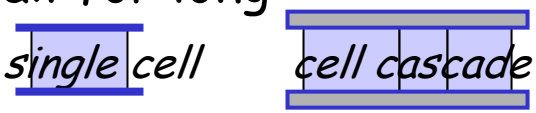
Multi-Instance Cells

- Cell Placement
  - pack cells side-by-side
    - abut cells and align power rails
  - avoid gaps between cells
    - unless needed for signal connections



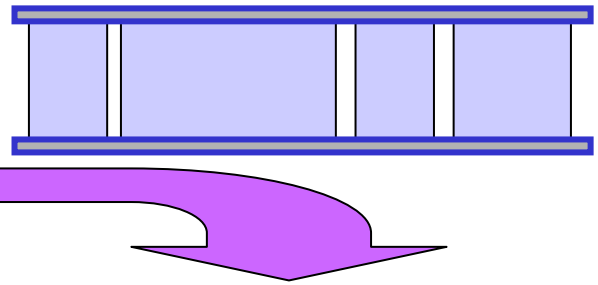
- Signal Routing
  - make internal connections using poly and metal-1, if possible
  - use jumpers outside rails only when necessary
    - jump up/down using poly (short trace) or metal-2 (if long trace)
      - poly for traces close to cell
      - metal-2 for traces far from cell
    - leave room for widened power rails
- Power Routing
  - more cells mean more supply current
  - widen power supply rail for long cascades of cells





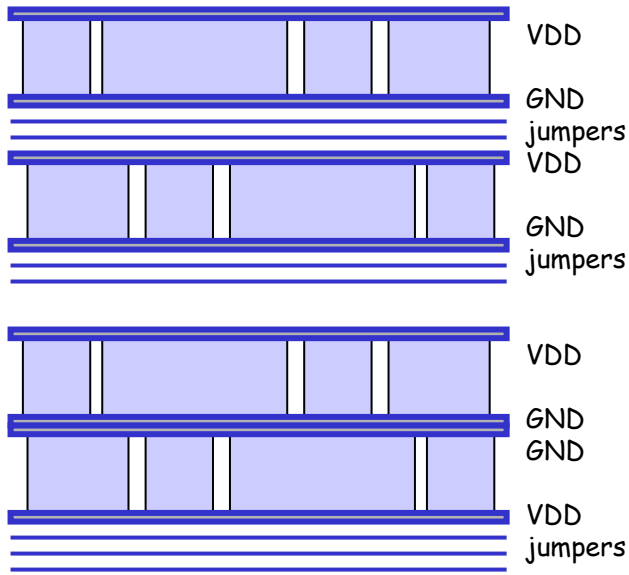
High-Level Layout

- Cell Placement
  - cascade cells with same pitch
  - stack cascaded cells

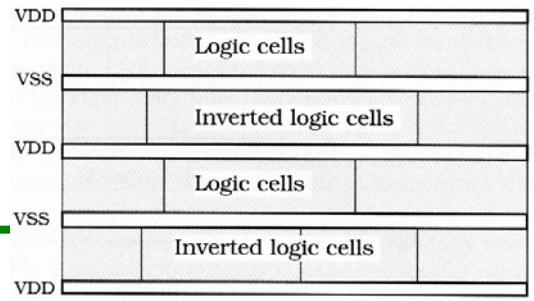


- Cell Orientation
  - maintain orientation when stacking
    - signal jumpers between stacks

  or
  - alternate orientation
    - signal jumpers on top and/or bottom



- Power Routing
  - widen supply rails for long cascades
  - connect rails outside cell cascades
    - example follows



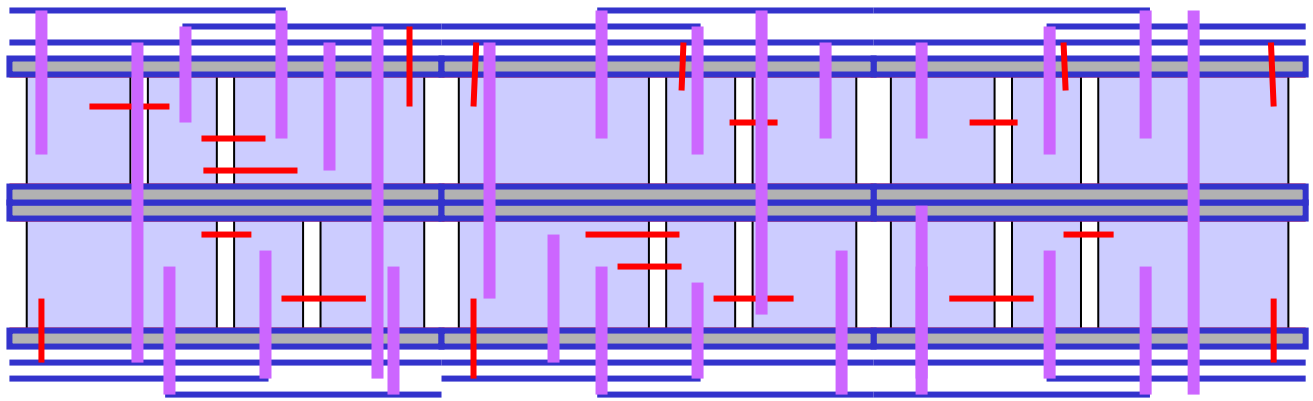
Metal Routing Strategy

- General Rules
  - use lowest level interconnects possible
    - if process has less than ~3 metal layers
      - try to route a cell cascade using only poly and metal-1
    - if process has more than ~3 metals
      - route cell cascade using metal-1 and metal-2, avoid using poly
  - alternate directions for each interconnect
    - e.g., metal1 horizontally, metal2 vertically, metal 3 horizontally, etc.
- Example

poly
- within primitives
- local interconnects
  - only if <3 metal layers

metal1
- within primitives
- power rails
- horizontal jumpers

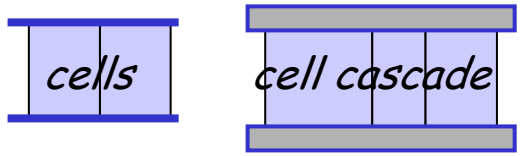
metal2
- vertical traces between stacked cascades

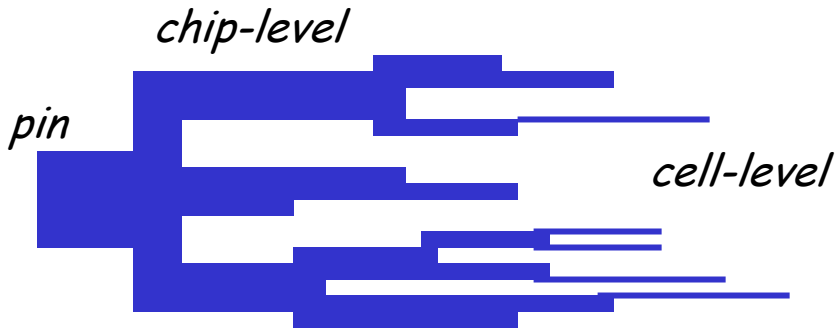


- Note: new process technologies have specially defined metal layers
  - e.g. metal_5 might be dedicated to VDD routing

Power Routing

- Power Rails for Combined Cells
  - join adjacent cells with continuous power rails
  - keep power rails wide enough for long power traces
    - more cells → more current → need traces with lower resistance
  - power tree concept
    - power enters chip on one pin
    - must "branch" across chip
    - traces should be thicker near pin and narrow into smaller cells

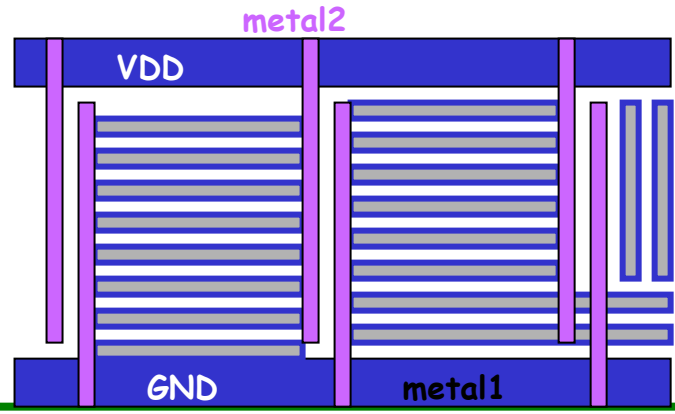




branching of power traces across a chip, from thick lines (chip) to thin lines (cell)

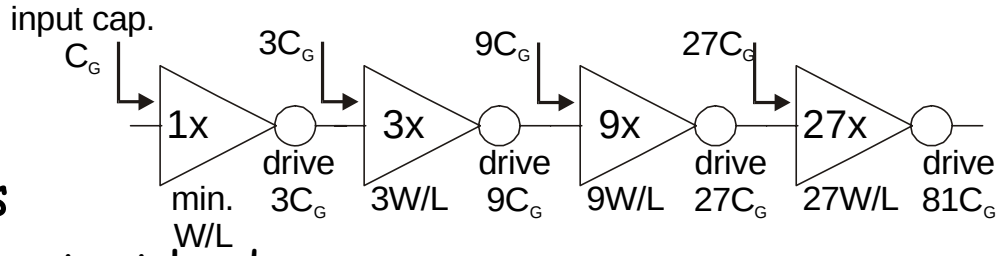
- Connecting rails in stacked cell cascades

zooming out…



Signal Buffers

- Loading and Fan-Out
  - gate input capacitance
    - C_G = 2CoxWL (1 for pMOS 1 for nMOS)
  - load capacitance
    - standard gate designed to drive a load of 3 gates → $C_L = 3C_G$
  - output drive capability
    - I ∝ W, increase W for more output signal drive
    - increasing W increase C_G

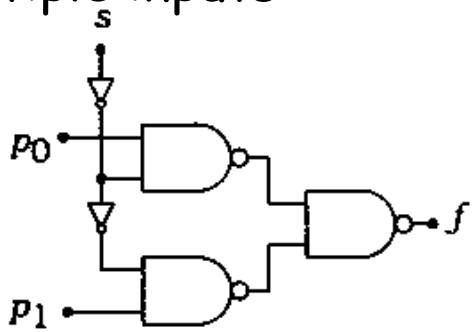


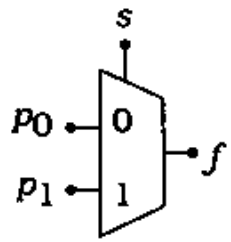
- Buffers
  - single stage inverter buffers
    - isolate internal signals from output load
  - scaled inverter buffers
    - add drive strength to a signal
    - inverters with larger than minimum tx
      - typically increase by 3x at with each stage

Transmission Gate Multiplexors

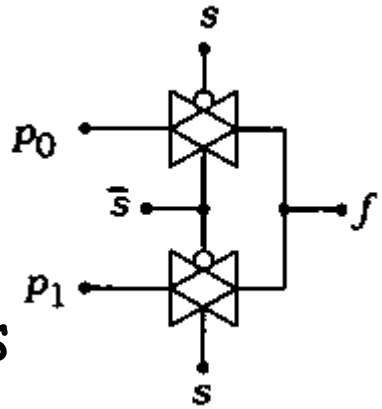
- Logical Function of a Multiplexor
  - select one output from multiple inputs
  - 2:1 MUX logic

$$f = p_0 \cdot \bar{s} + p_1 \cdot s$$

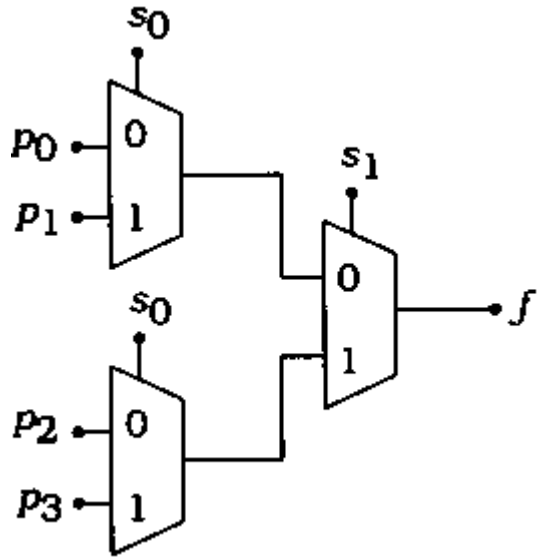




- CMOS Multiplexors
  - generally formed using switch logic rather than static
- 2:1 MUX using Transmission Gates

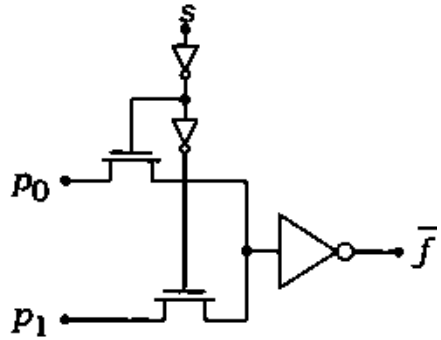


- 4:1 MUX using 2:1 MUXs

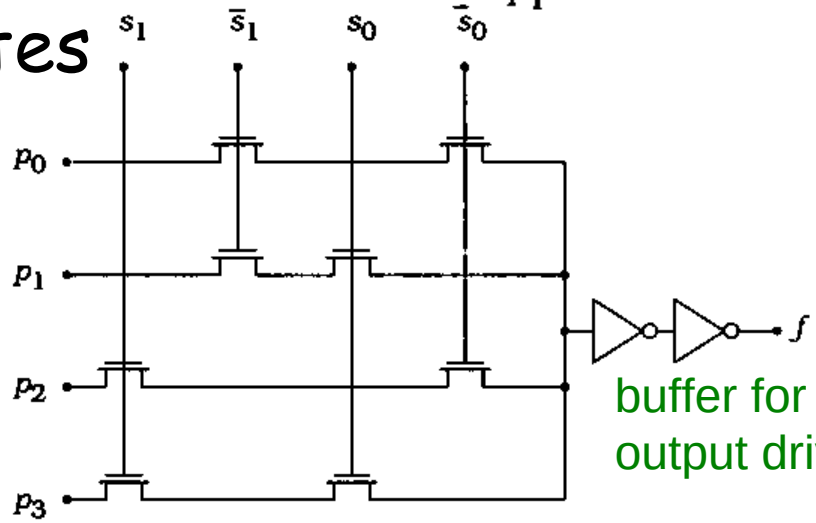


Pass-gate Multiplexors

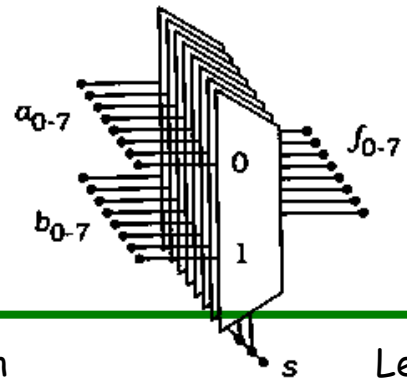
- 2:1 MUX using pass-gates
  - nMOS switch circuit



- 4:1 MUX using pass-gates



- Pass-gate MUX with rail-to-rail output
  - add full pMOS network
    - see Figure 11.7 in textbook
- Multi-bit MUXs
  - use parallel single-bit MUXs



Binary Decoders

- Decoder Basic Function
  - *n* bits can be decoded into *m* values
    - max *m* is 2^n
  - decoded values are active only one at a time
    - active high: only selected value is logic 1
    - active low: only selected value is logic 0

n select bits decode into 2^n outputs values

control inputs / active high decoded outputs

s_1	s_0	d_0	d_1	d_2	d_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

control inputs select one active output

- Example: 2/4 (*2-to-4*) Decoder
  - 2 control bits decoded into 4 values
    - truth table
    - equations

$$d_0 = \overline{s_1} \cdot \overline{s_0} = \overline{s_1 + s_0} \qquad d_2 = s_1 \cdot \overline{s_0} = \overline{\overline{s_1} + s_0}$$

$$d_1 = \overline{s_1} \cdot s_0 = \overline{s_1 + \overline{s_0}} \qquad d_3 = s_1 \cdot s_0 = \overline{\overline{s_1} + \overline{s_0}}$$

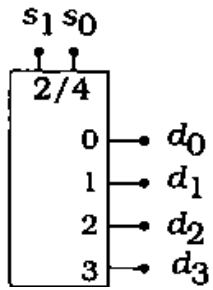
  - active high decoder equations require NOR operation

CMOS Decoder Circuits

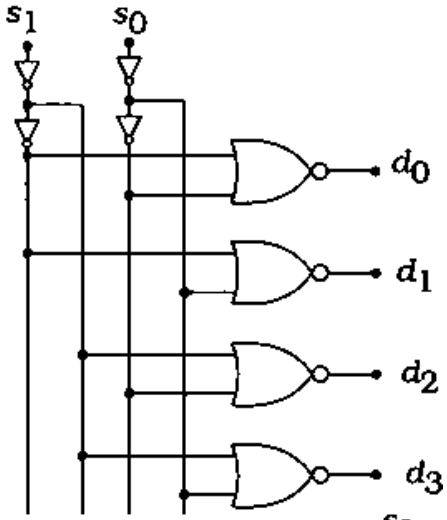
- 2/4 Active High Decoder

s_1	s_0	d_0	d_1	d_2	d_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Truth Table



Symbol

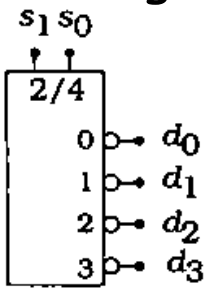


NOR2 Circuit
active high
2/4 decoder

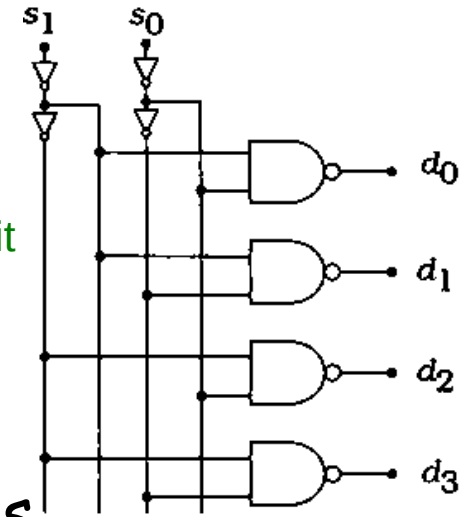
- 2/4 Active Low Decoder
  - implemented with NAND gates

s_1	s_0	d_0	d_1	d_2	d_3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Truth Table



Symbol



NAND2 Circuit
active low
2/4 decoder

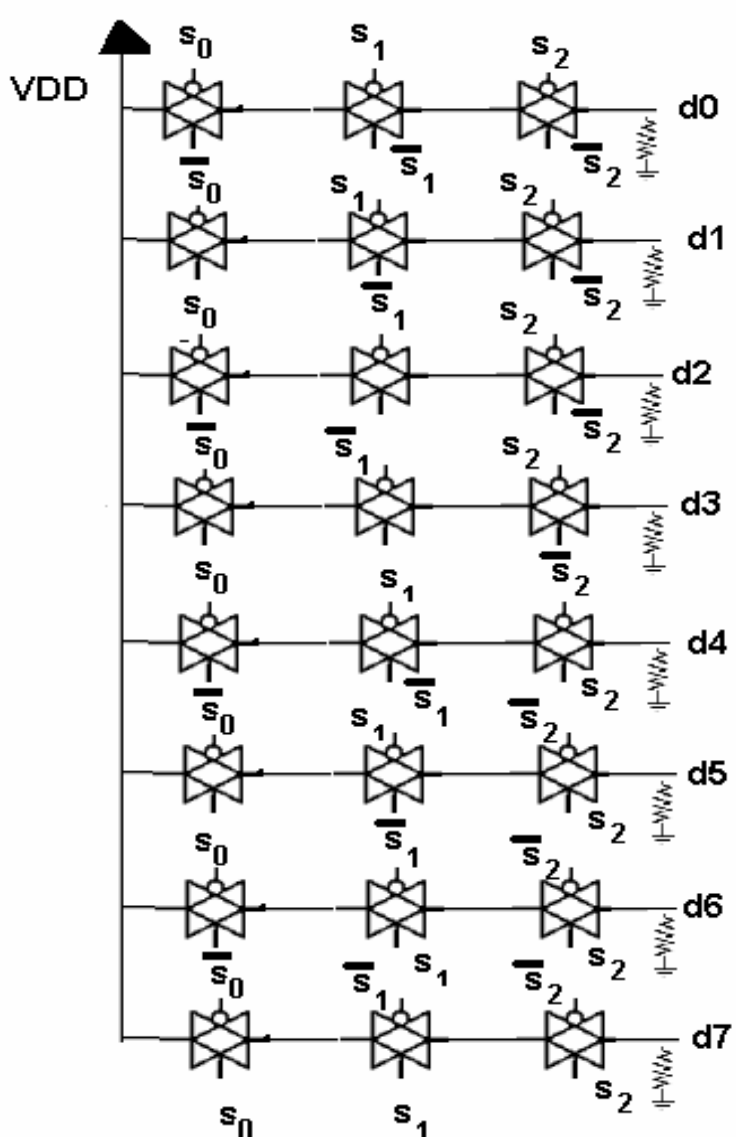
- Similar approach for higher-value decoders
  - 3/8 decoder requires 3-input gates, higher values get complex

Transmission Gate Decoders

- EXAMPLE: 3/8 Active-High Decoder
  - each output connected to VDD through 3 transmission gates
  - TG selects set to turn on only one of the 8 possible combinations of the 3-bit select

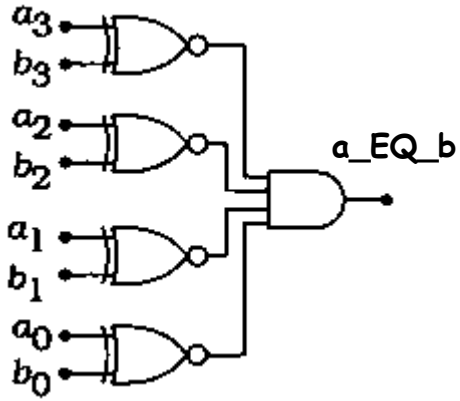
s2	s1	s0	d7	d6	d5	d4	d3	d2	d1	d0
0	0	0								1
0	0	1							1	
0	1	0						1		
0	1	1					1			
1	0	0				1				
1	0	1			1					
1	1	0		1						
1	1	1	1							

- What do the resistors at output do?
- What is the signal value at the unselected outputs?

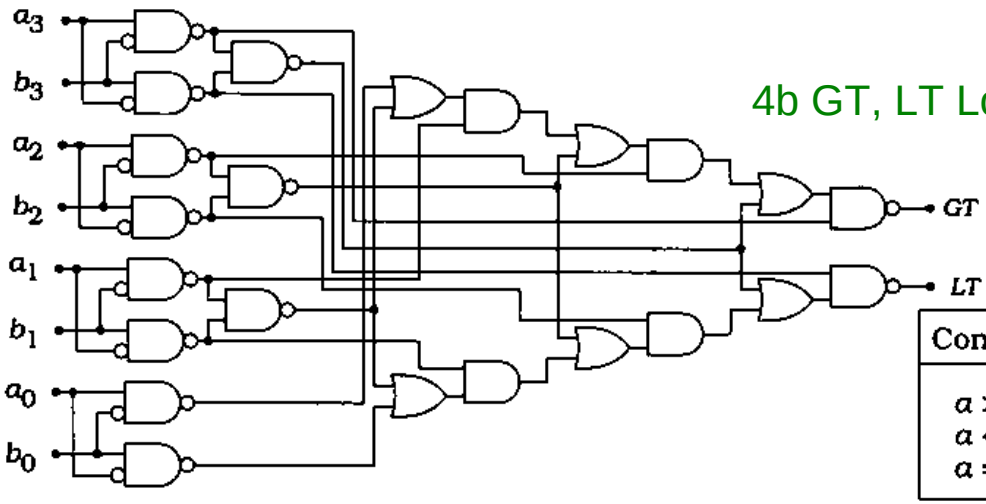


Magnitude Comparators

- Often need to compare the value of 2 n-bit numbers
  - EQUAL if values are the same
  - GREATER THAN if a is greater than b
  - LESS THAN if b is greater than a
- Equality: a_EQ_b, can be generated by XNOR operation
  - a = b iff aXNORb = 1 for each binary digit
    - example: 4b equality comparator using XNOR
  - also, a=b if a>b=0 and a<b=0 for ach binary digit
- Greater/Less Than, by bit-by-bit comparison



4b Equality Circuit



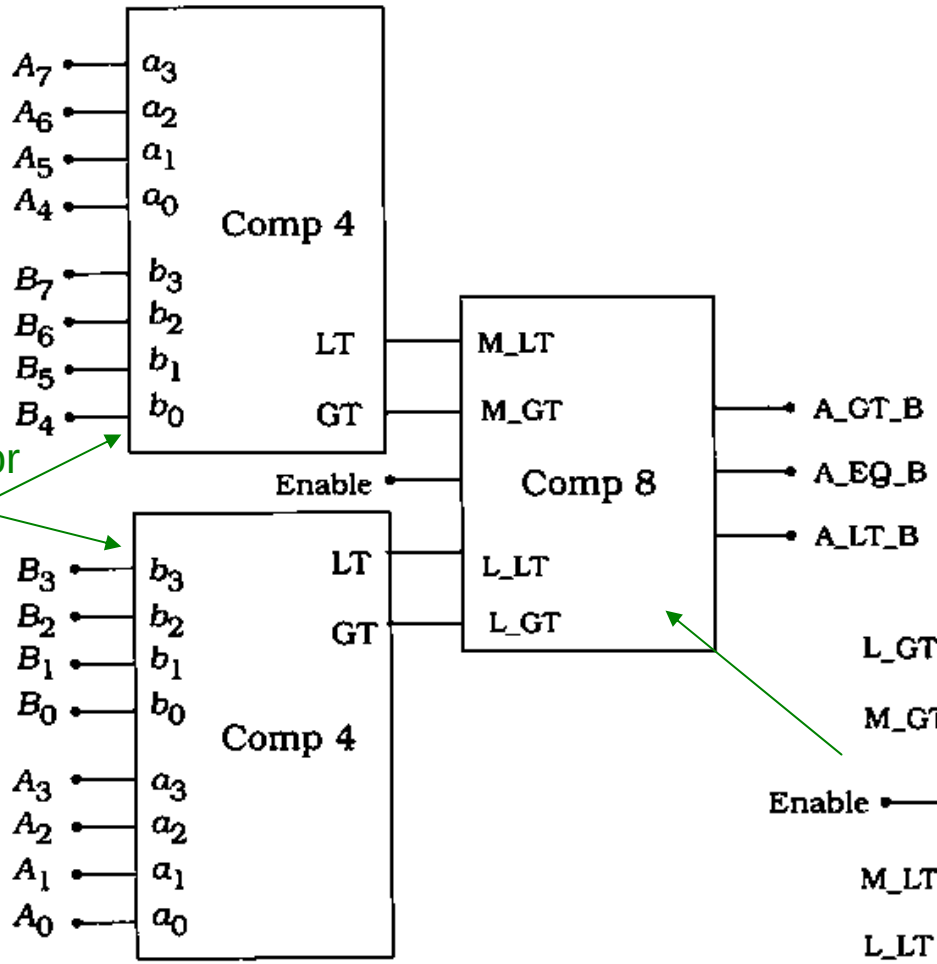
4b GT, LT Logic

Condition	GT	LT
$a > b$	1	0
$a < b$	0	1
$a = b$	0	0

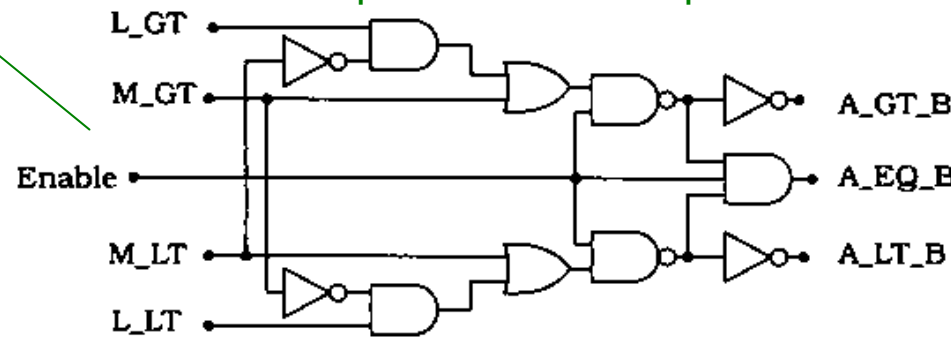
note: can get Equal from GT, LT circuit

Combined Comparator Circuits

- 8b Magnitude Comparator with Output Enable
  - generates, EQ (equal), GT (greater than), LT (less than)



4-bit comparator from previous page



compares outputs from 4b cells, implements Enable, produces 8b compare results

Priority Encoders

- Priority Encoders generates an encoded result showing
  - IF a binary number has a logic 1 in any bit
  - WHERE the most significant logic 1 occurs
- Output is an encoded value of the location of the most significant '1'
- Example: 8b priority encoder

d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	1	-	1	0	0	1
0	0	0	0	0	1	-	-	1	0	1	0
0	0	0	0	1	-	-	-	1	0	1	1
0	0	0	1	-	-	-	-	1	1	0	0
0	0	1	-	-	-	-	-	1	1	0	1
0	1	-	-	-	-	-	-	1	1	1	0
1	-	-	-	-	-	-	-	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0

assign d_7 highest priority, d_0 lowest

Q_2-Q_0 encode the value of the highest priority 1

Q_3 is high if any bit in d is logic 1

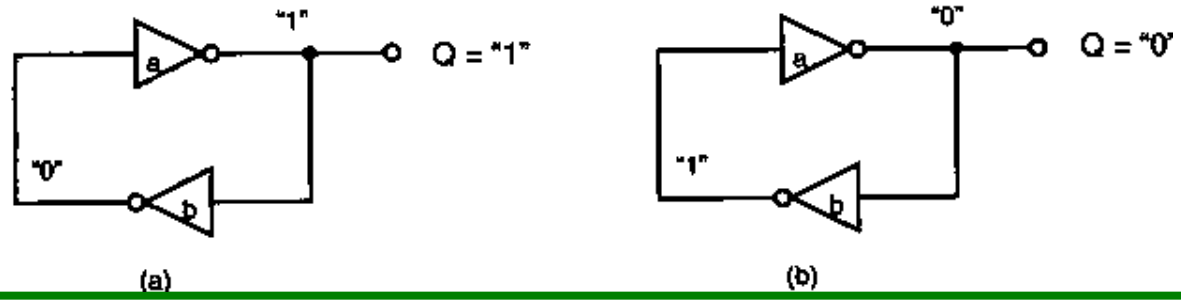
- Outputs can be constructed from the truth table
  - see textbook for illustrations of CMOS logic

Data Latches

- Latch Function
  - store a data value
    - non-volatile; will not lose value over time
  - often incorporated in static memory
  - building block for a master-slave flip flop
- Static CMOS Digital Latch
  - most common structure
    - cross-coupled inverters, in positive feedback arrangement
  - circuit forces itself to maintain data value
    - inverter ***a*** outputs a 1 causing inverter ***b*** to output a 0
    - or, inverter ***a*** outputs a 0 causing inverter ***b*** to output a 1

Latches also improve signal noise immunity; feedback forces signal to hold value and filters noise

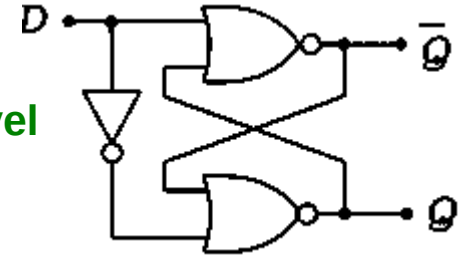
Bistable circuit



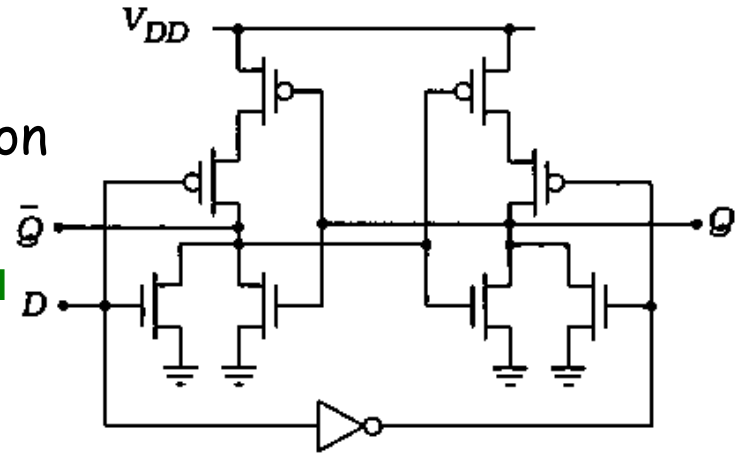
D-Latch Logic Circuit

- Accessing Latch to Set Value
  - apply input D to set latched value
- NOR D-Latch
  - uses NOR cells to create latch function

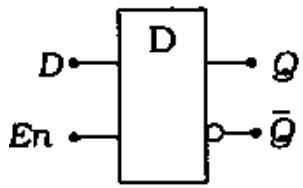
Logic-Level Circuit



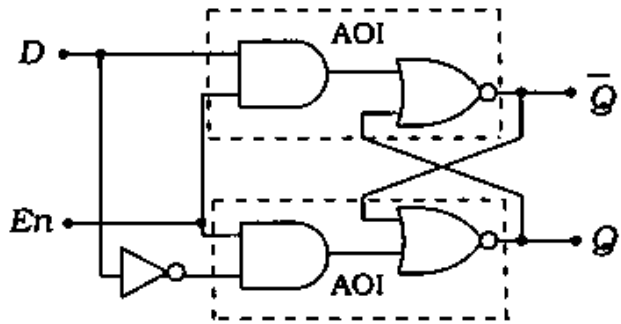
Transistor-Level Circuit



- D-Latch with Enable
  - En selects if output
    - set by input, D
    - or from internal feedback



(a) Symbol

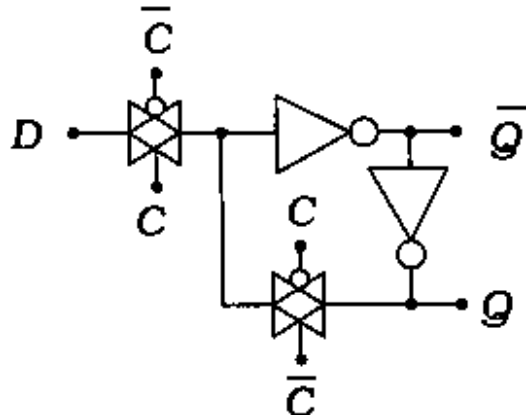


(b) Logic diagram

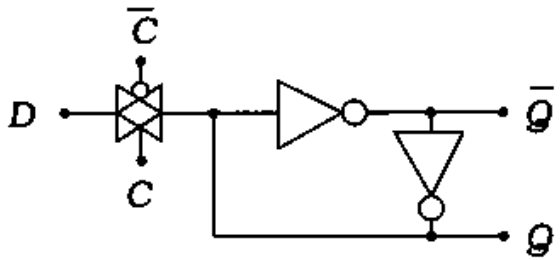
- Different structures used in VLSI

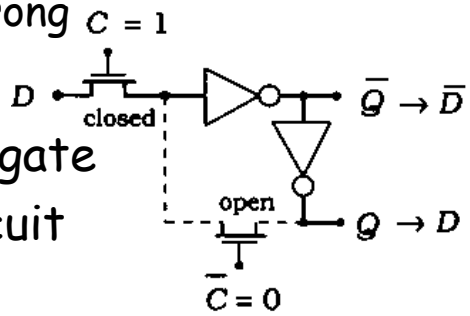
CMOS VLSI Clocked Latches

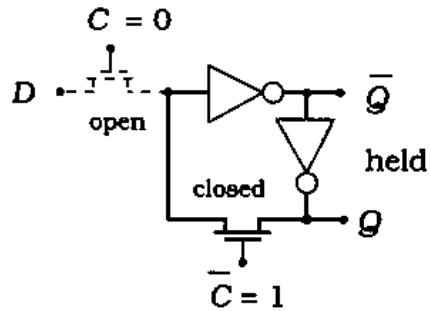
- Clocked (enable) Latch using TGs
  - can use TGs to determine
    - if latch sees D
      - $C = 1 \Rightarrow Q' = D'$, set data mode
    - or if positive feedback is applied
      - $C = 0 \Rightarrow Q' = Q'$, hold data mode



- Reducing Transistor Count
  - Single TG D-Latch
    - input must overdrive feedback signal
      - must use weak feedback inverter
    - useful when chip area is critical
      - but input signal must be strong
  - Pass-gate D-Latch
    - replace TG with nMOS Pass-gate
    - very common VLSI latch circuit

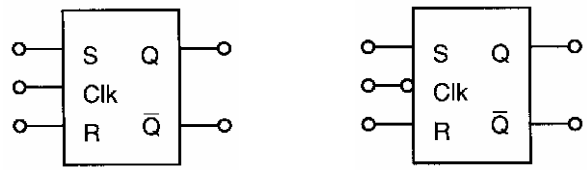






Flip Flop Basics

- storage element for synchronous circuits
  - save logic state at each clock cycle
- 1 or 2 signal inputs and a ***clock***
- differential outputs, Q and Q'
  - output changes on rising (or falling) clock edge
  - output held until next rising (or falling) clock edge
- optional asynchronous ***set*** and/or ***reset***
  - regardless of clock state, output set (1) or reset (0)
- typically *master-slave* circuit using 2 cascaded latches
- types include
  - JK
  - T (toggle)
  - SR (set-reset)
  - D -most common for ICs

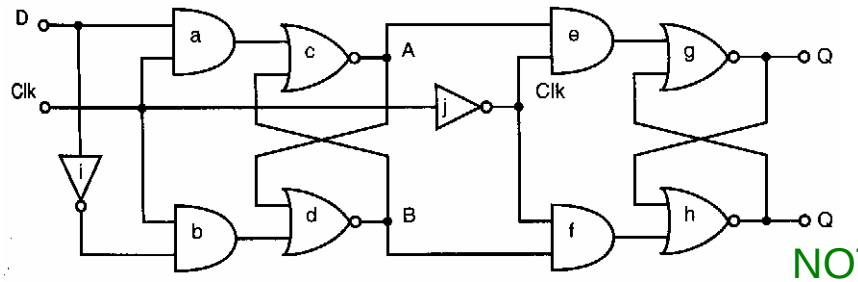


Flip-flop symbol (SR) for rising and falling edge clocks

Types of Flip Flops

- D-type (DFF)

D	Q_{n+1}
0	0
1	1

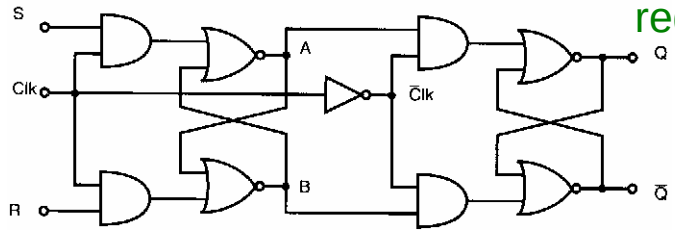


NOTE: Circuit based on standard logic gates is typically much larger than possible with a reduced CMOS circuit

- SR-type

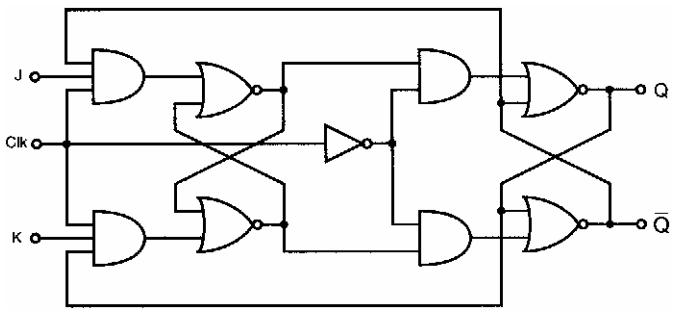
  same as D if S=D and R=D'

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	Indeterminate



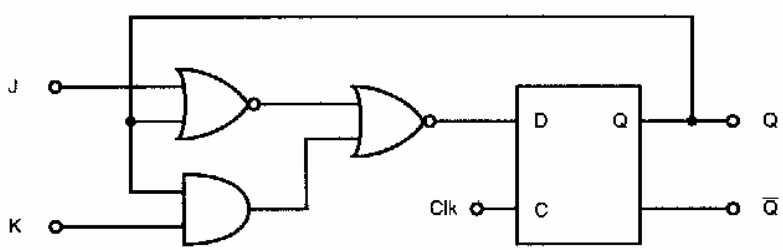
- JK-type

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\bar{Q}_n$



JK and T Flip Flops from DFF

- D-Flip Flop can be used to create most other FF types
- Can construct a JK FF from a DFF



J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\bar{Q}_n$

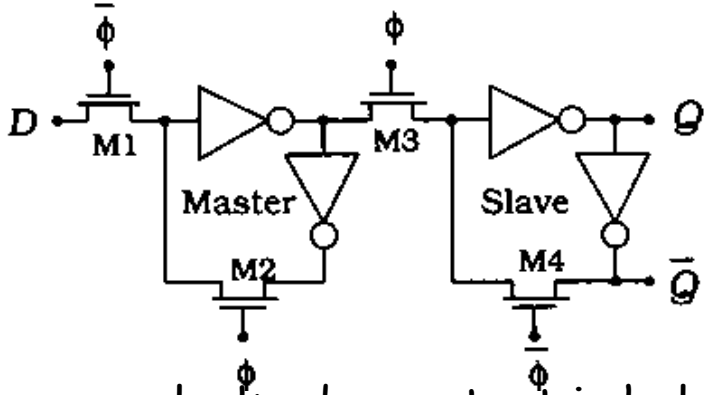
- T-type (toggle) FF can be constructed from a JK FF
  - T=1
    - output changes state on each clock cycle
  - T=0
    - hold output to previous value
  - form from JK by connecting J and K inputs together as T

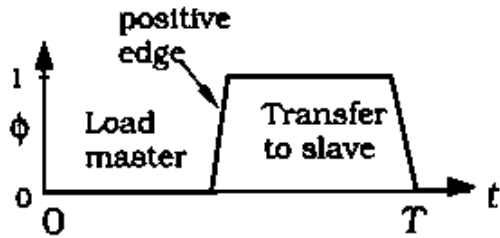
T	Q_{n+1}
0	Q_n
1	$\bar{Q}_n$

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\bar{Q}_n$

Master-Slave D Flip Flop

- D-type master-slave flip flop is the most common in VLSI
- Master-Slave Concept
  - cascade 2 latches clocked on opposite clock phases
    - $\phi = 1, \overline{\phi} = 0$: D passes to master, slave holds previous value
    - $\phi = 0, \overline{\phi} = 1$: D is blocked from master, master holds value and passes value to slave

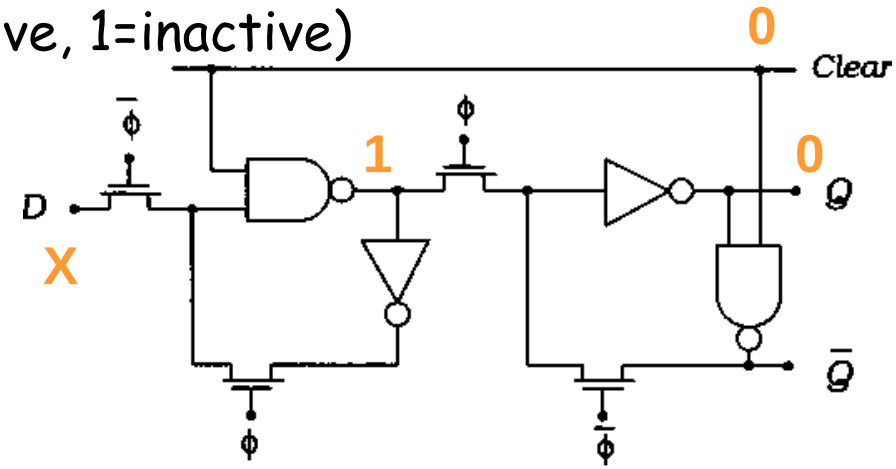


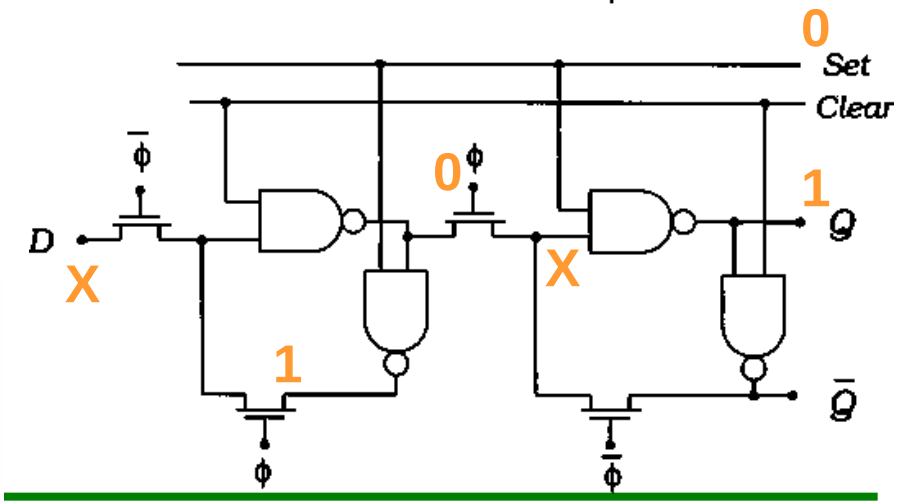


- Triggering
  - Output only changes on clock edge; output is held when clock is at a level value (0 or 1)
  - Positive Edge
    - output changes only on rising edge of clock
  - Negative Edge
    - output changes only on falling edge of clock

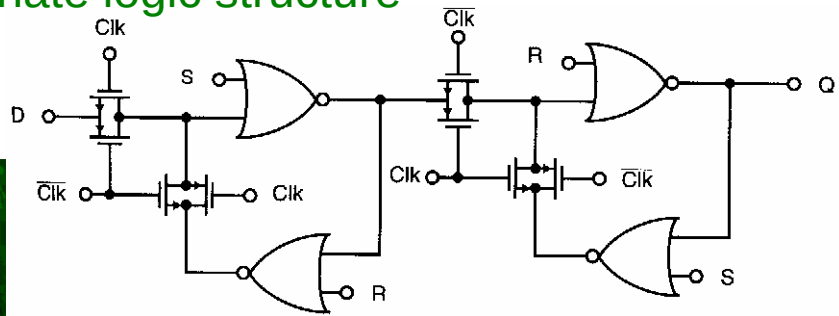
Set/Reset Flip Flops

- Asynchronous Set and Reset
  - Asynchronous = not based/linked to clock signal
  - Typically negative logic (0=active, 1=inactive)
  - Set: forces Q to logic 1
  - Reset: forces Q to logic 0
- Logic Diagrams
  - DFFR
    - with Reset (clear)
  - DFFRS
    - with Reset (clear) and Set



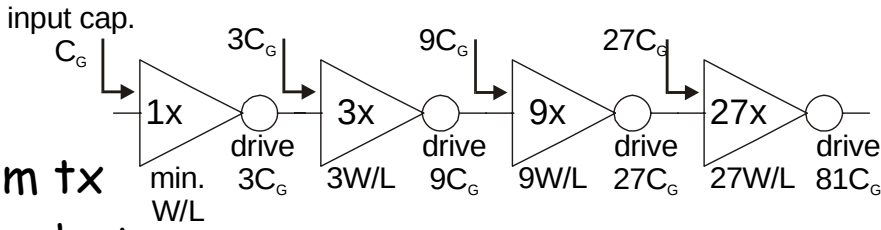


Alternate logic structure



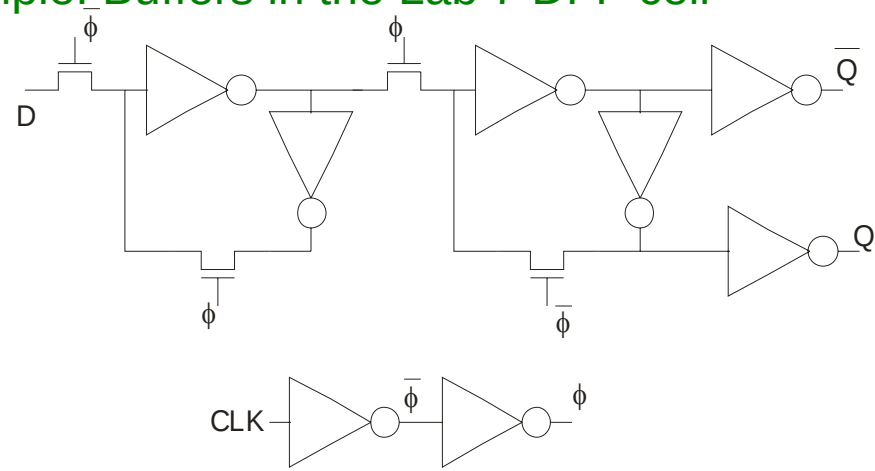
Buffering in Flip Flops

- What is a buffer?
  - inverter buffers
    - isolate output load from internal signals
  - scaled inverter buffers
    - add drive strength to a signal
    - inverters with larger than minimum tx
      - typically increase by 3x at with each stage



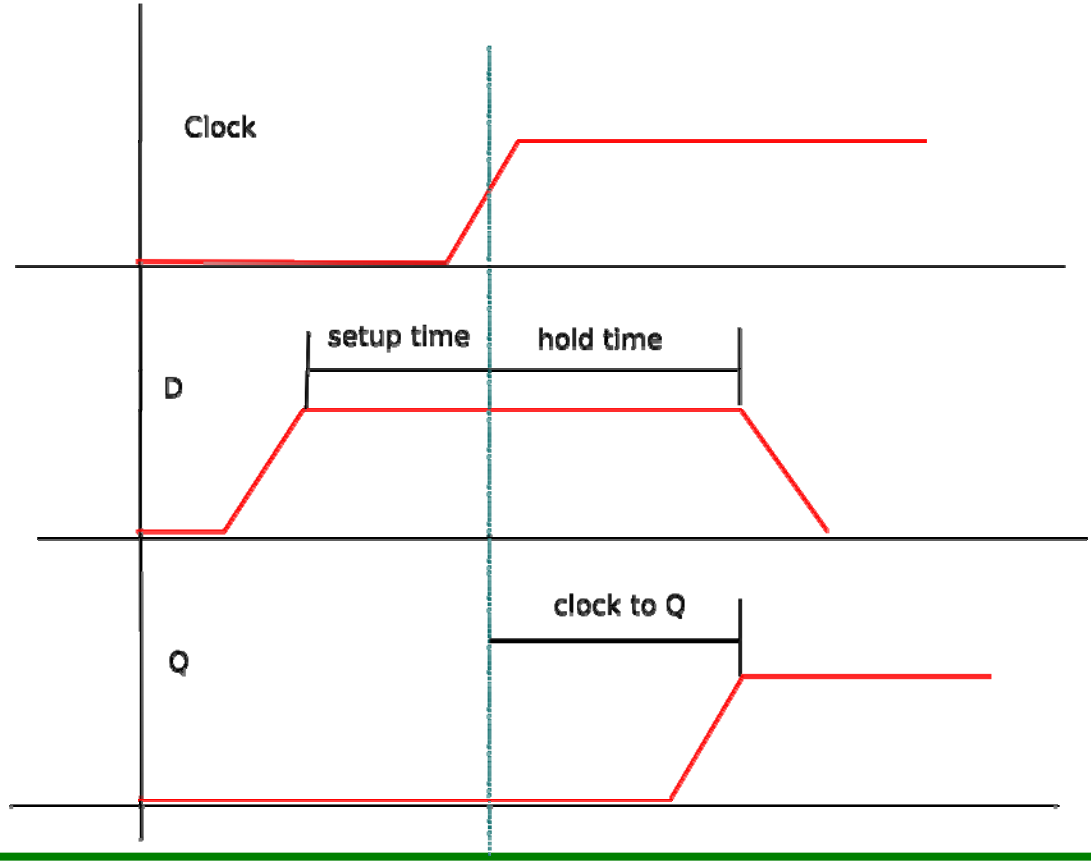
- Inter-cell Buffering
  - Clock
    - so each flip flop provide only 1 C_G load on input CLK
  - Output
    - so load at output won't affect internal operation of the cell

Example: Buffers in the Lab 7 DFF cell



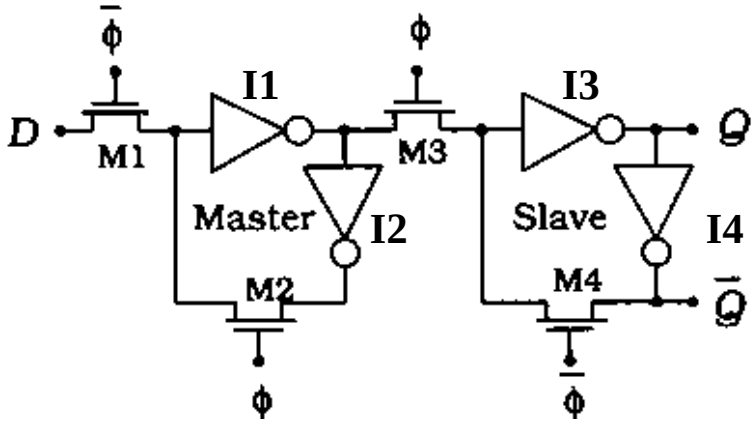
Characterizing Flip Flop Timing

- Setup Time: t_{su} – Time D must remain stable before the clock changes
- Hold Time: t_h – Time D must remain stable after the clock changes
- Clock to Q Time: t_{c2q} – Time from the clock edge until the correct value appears at Q



Analyzing DFF Timing

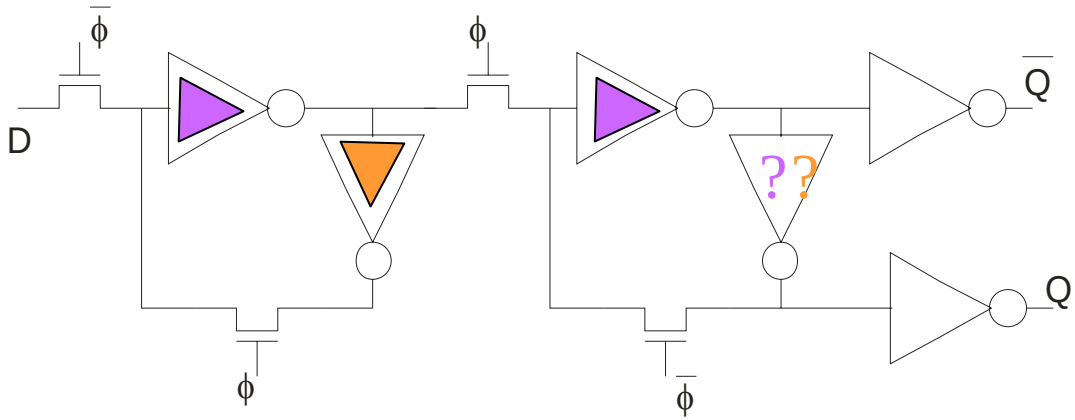
- Setup
  - When ϕ is low D must propagates through both master inverters, if clock changes before then the master may switch
  - $t_{su} = t_{M1} + t_{I1} + t_{I2}$
- Hold
  - As soon as ϕ is goes high, D is cutoff
  - $t_h = 0$
- Clock to Q
  - For both outputs to be valid must wait for both slave inverters to change.
  - $t_{c2Q} = t_{M3} + t_{I2} + t_{I3}$



Different types of flip flops exhibit different timing characteristics

Transistor Sizing in Flip Flops

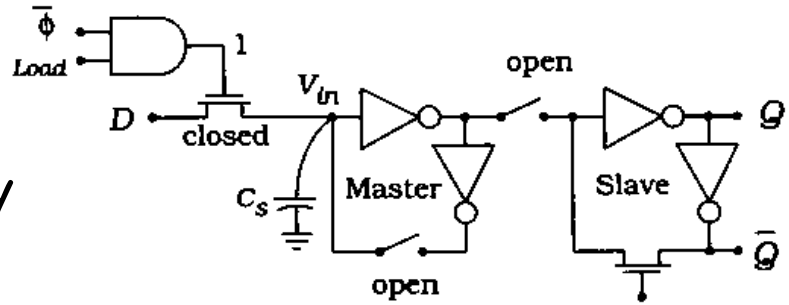
- All Minimum-Size Tx Flip Flops
  - will not be optimized for speed
  - might have some output glitches
  - but much more simple to lay out



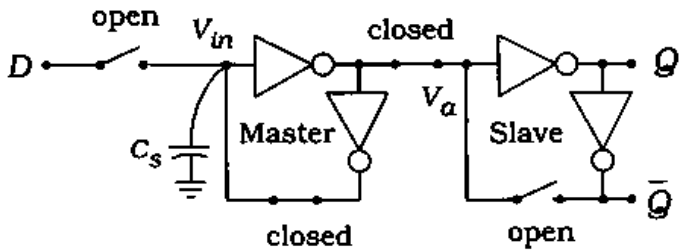
- Size Considerations
  - varies widely with chosen FF design
  - feedback INV can be weak
  - tx in direct path to signal output should be larger
  - switches -typically minimum sized to reduce noise

Load Control in Flip Flops

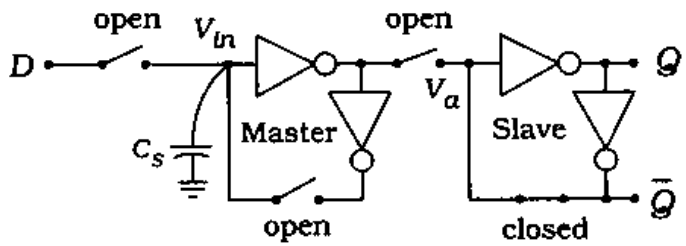
- To mask (block) clocking (loading) of the FF, a load control can be added
  - load control allows new data to be loaded or blocks the clock thereby stopping new data from loading
- Load Controlled FF
  - Load = 1, data passed
  - Load = 0, data blocked
- Alternative Design



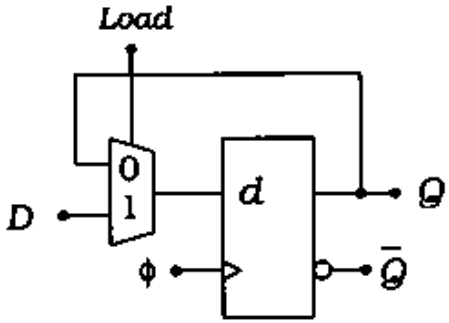
(a) Load with $\phi = 0$, *Load* = 1



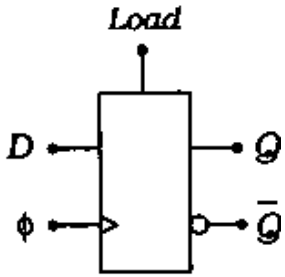
(b) Hold with $\phi = 1$, *Load* = 0



(c) Hold with $\phi = 0$, *Load* = 0



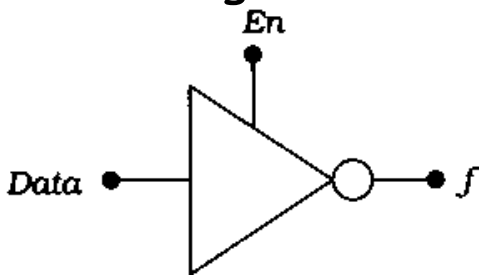
(a) Wiring diagram



(b) Symbol

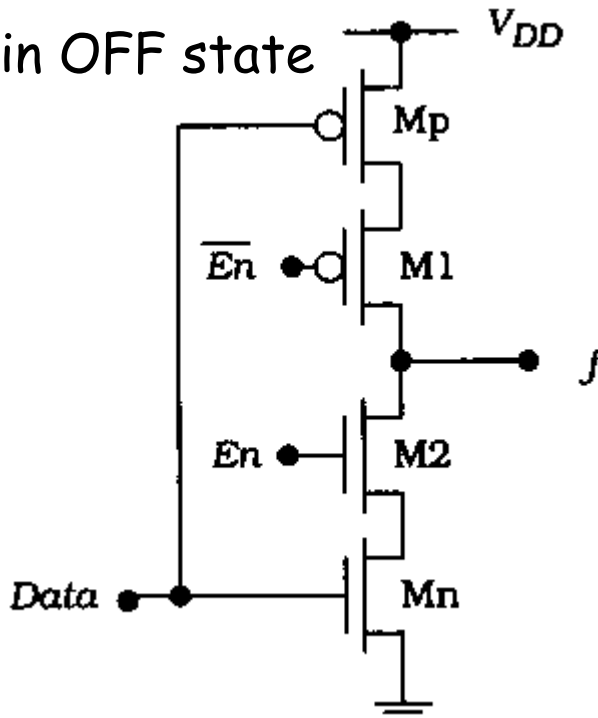
Tri-State Circuits

- covered in Section 9.3 in textbook
- Tri-State = circuit with 3 output states
  - high, low, high impedance (Z)
- High Impedance State
  - output disconnected from power or ground
  - open circuit, with impedance of a MOSFET in OFF state
- Tri-State Inverter
  - Enable signal, enable/disables output drive



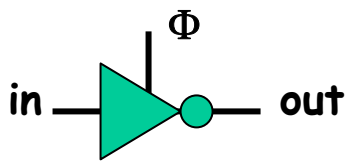
En	f
0	Z
1	$\overline{Data}$

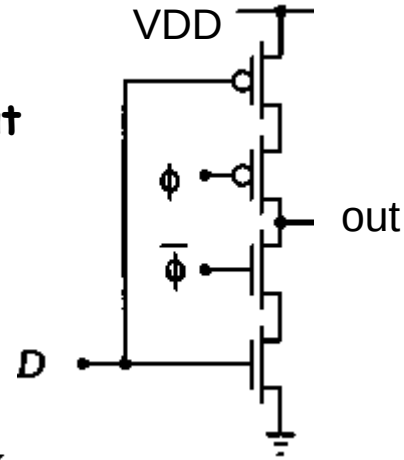
  - CMOS implementation



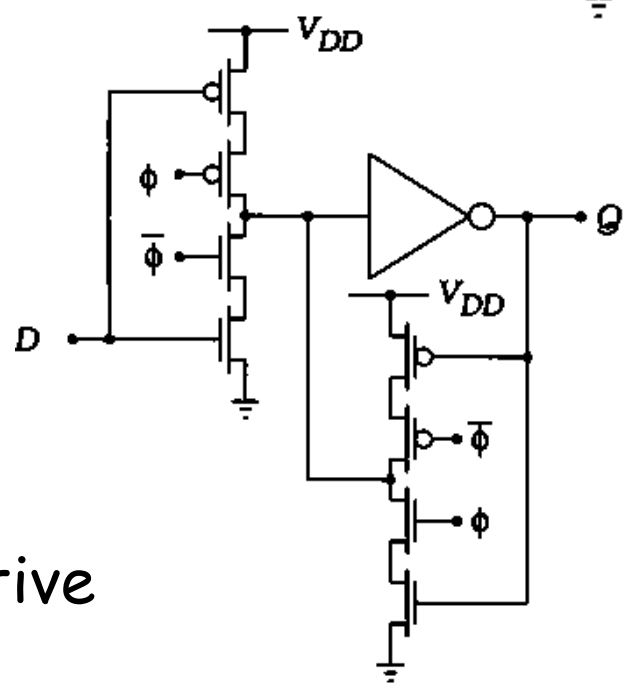
Advanced Latches and Flip Flops

- C^2MOS Inverter
  - C^2MOS = clocked CMOS
  - inverter where input can be enabled
    - Φ = 0, out = D'
    - Φ = 1, out = floating



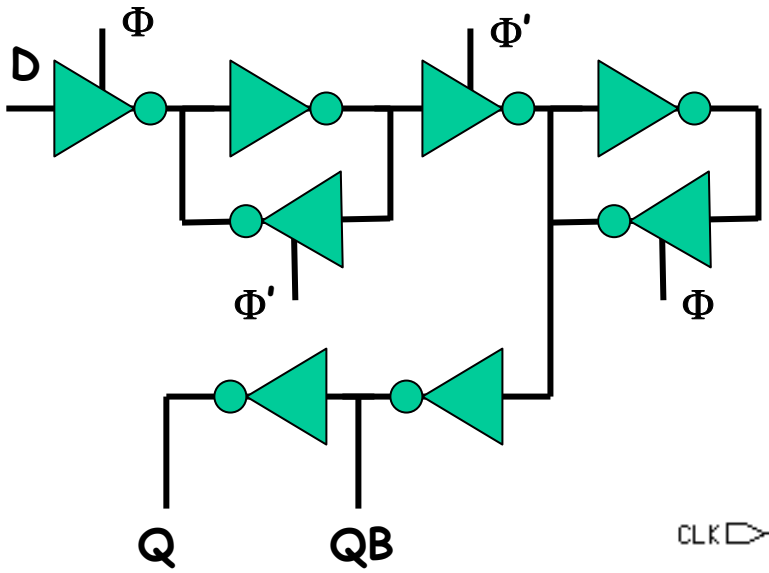


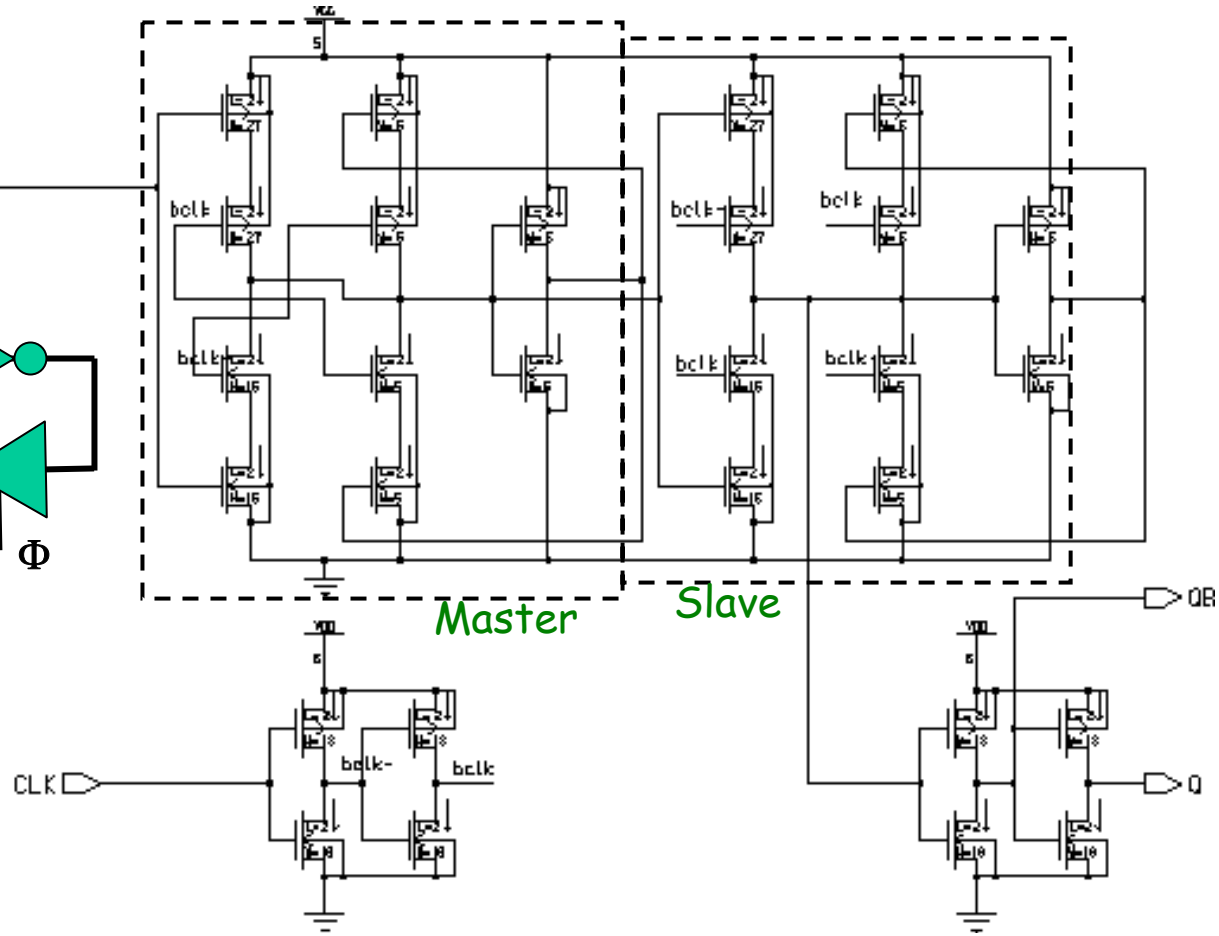
- C^2MOS Static Latch
  - merge TGs into latch design
  - C^2MOS inverter input stage
    - passes inverted input when Φ = 0
    - static inverter sets Q = D
  - C^2MOS inverter feedback
    - provides feedback when Φ = 1
  - Either input of feedback is active
    - not both at the same time



C²MOS D Flip Flop

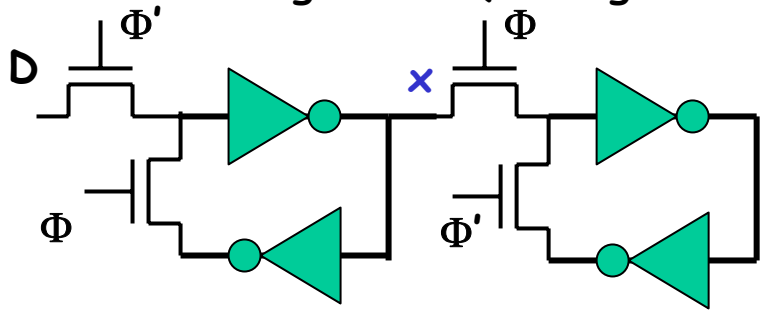
- Cascade 2 C²MOS Latches
  - switch clock phases of master and slave blocks





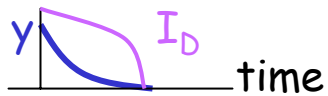
Discussion of DFF Timing

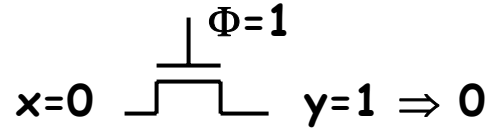
- Why is output propagation delay different for D=1 and D=0?
  - propagation delay in DFF = time between clock edge and Q change
  - delay set by transitions in the slave (second) stage
    - master stage can be ignored when output changes
  - output changes when Φ goes high



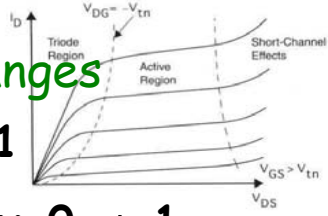
positive-edge triggered master-slave DFF using bistable circuits with pass gates

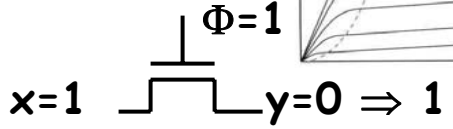
- D=1 ➡ x=0
  - V_{GS} = VDD, tx is ON with strong V_{GS}, V_{GS} constant as output changes
  - V_{DS}: VDD ⇒ 0, tx in Saturation then in Triode



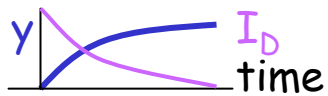


- D=0 ➡ x=1
  - V_{GS} = VDD ⇒ Vtn, tx is ON, but V_{GS} decreases as output changes
  - V_{DS}: VDD ⇒ Vtn, tx in Saturation then in Triode



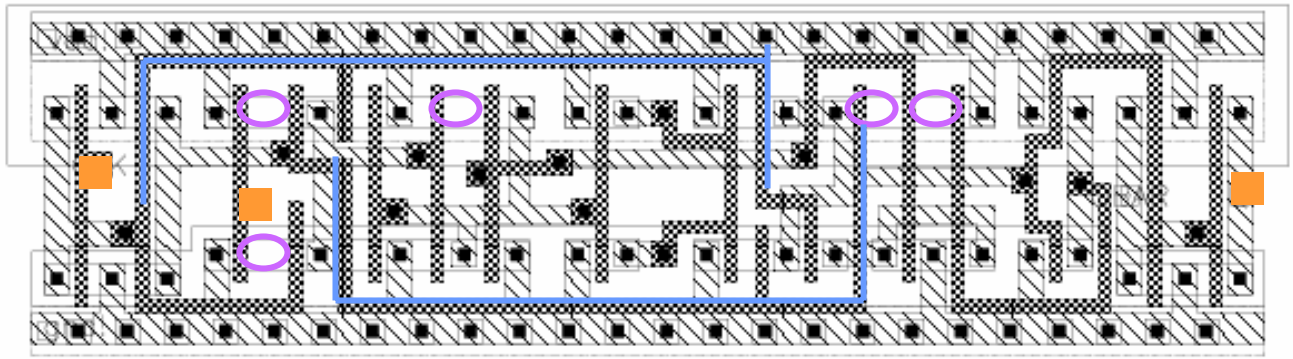


Output change is slower for D=0 since pass-gate has weak current



Flip Flop Layout

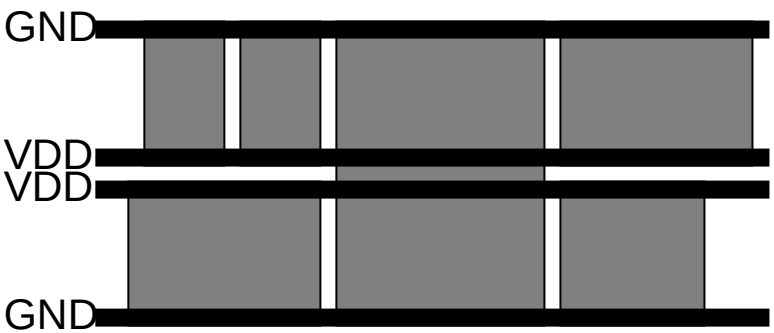
- A DFFR (with reset) cell with
  - all tx. min. size
  - no buffers



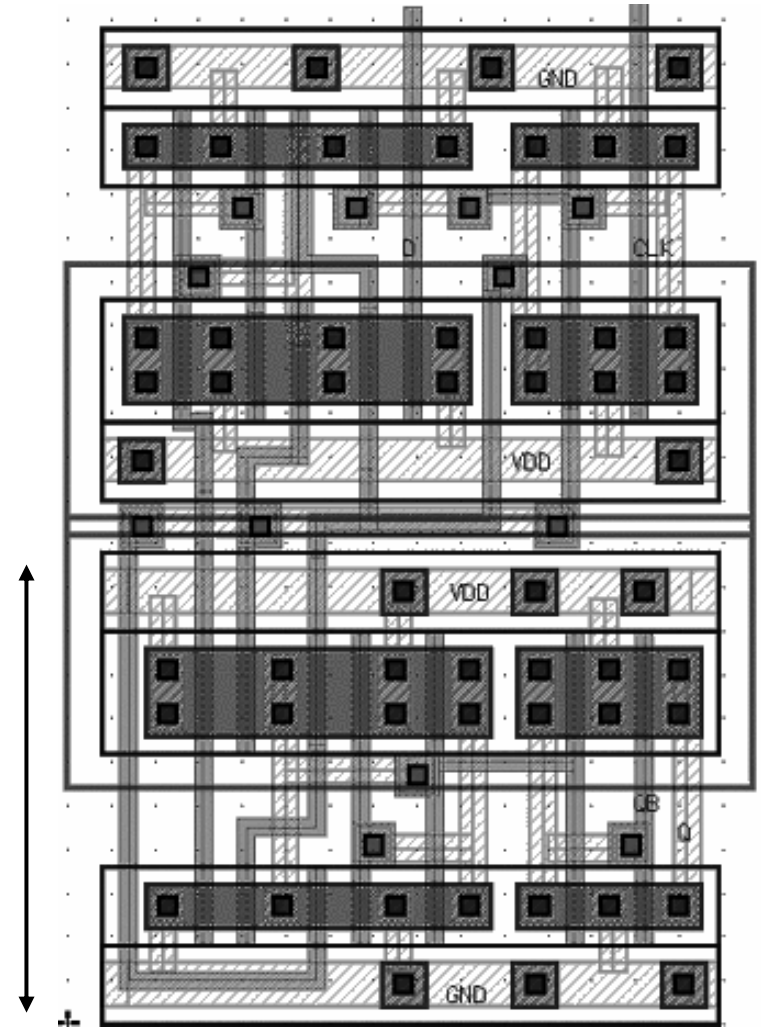
- Good features
  - compact layout, small area demand
  - very 'regular' physical structure
    - due to all minimum-sized transistors
  - pitch matched to other primitive cells
- Bad features
  - several S/D junctions larger than necessary
  - several long poly traces, might affect speed
  - access to inputs/outputs must be in metal2

Flip Flop Layout II

- Physical Design of C^2MOS Flip Flop
  - double-wide FF
    - pitch is 2x pitch of basic gates
- Using tall cells with standard height cells
  - match power rails

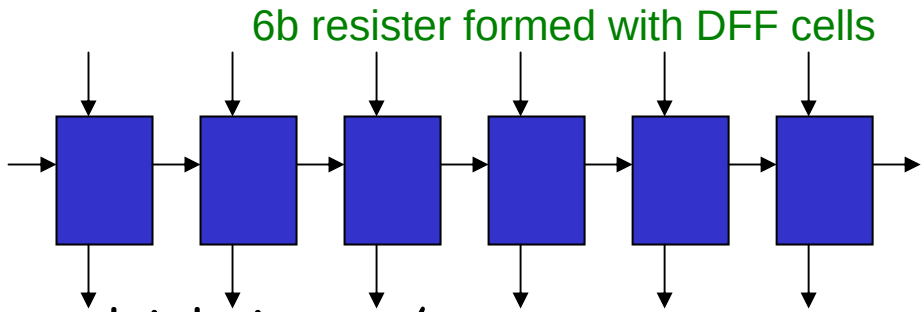


52λ pitch



Registers

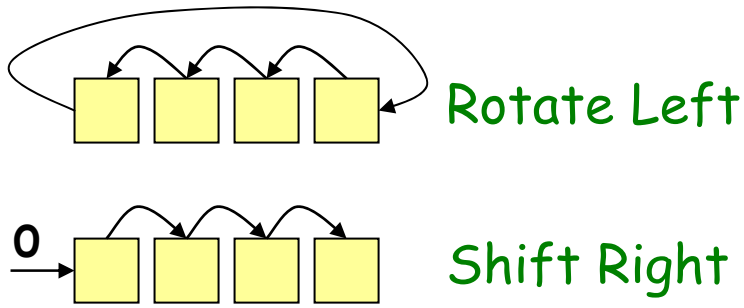
- Basic Register Function
  - store a byte of data
  - implement data movement functions such as
    - shift
    - rotate
  - basis for other functions
    - counter/timer
- Basic Register Circuit
  - cascade of DFF cells
  - additional logic to multiplex multiple inputs/outputs
  - typical I/O options
    - parallel load
    - load from left/right cell (shift)
    - parallel output



Shift and Rotate Operations

- Rotate
  - move each bit of data to an adjacent bit
  - roll end bit to other end
- Shift
  - move each bit of data to an adjacent bit
  - load '0' into the open end bit

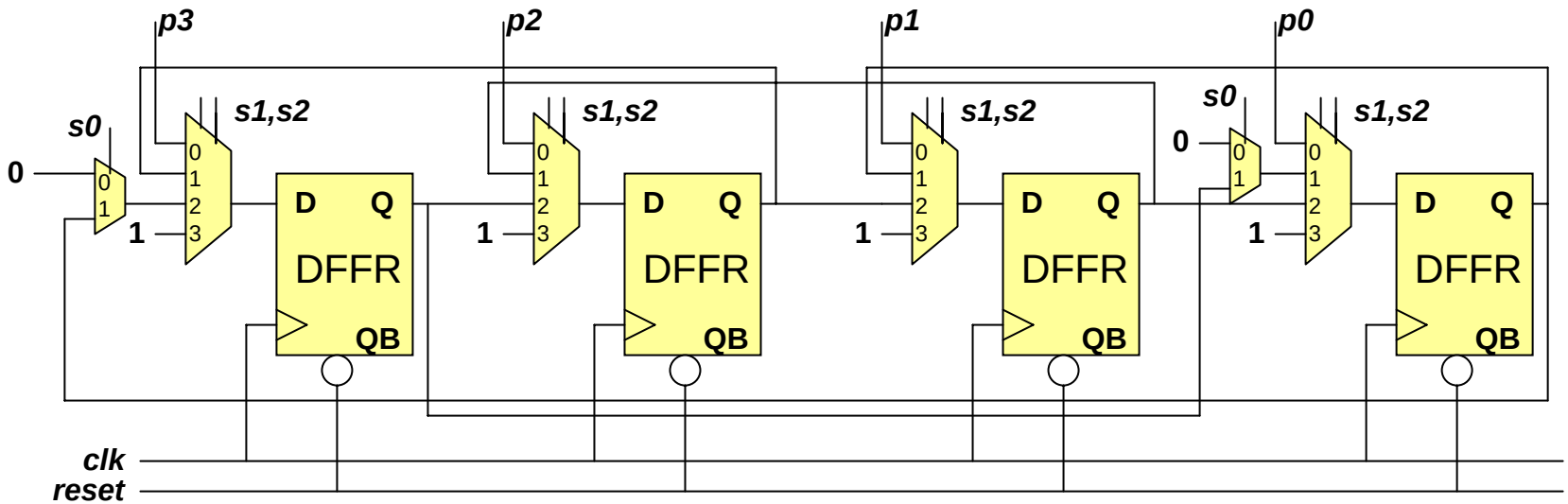
- Examples: 4b operations on data $a_3a_2a_1a_0$
  - Rotate Left: output = $a_2a_1a_0a_3$
  - Rotate Right: output = $a_0a_3a_2a_1$
  - Shift Left: output = $a_2a_1a_0$0
  - Shift Right: output = 0$a_3a_2a_1$



Shift Register

- Example: 4-bit register capable of
  - shift left/right
  - rotate left/right
  - parallel load
  - reset (all bits go to 0)
  - set (load all bits with 1)

s2	s1	s0	function
0	0	x	parallel load
0	1	0	shift right
0	1	1	rotate right
1	0	0	shift left
1	0	1	rotate left
1	1	x	load '1'

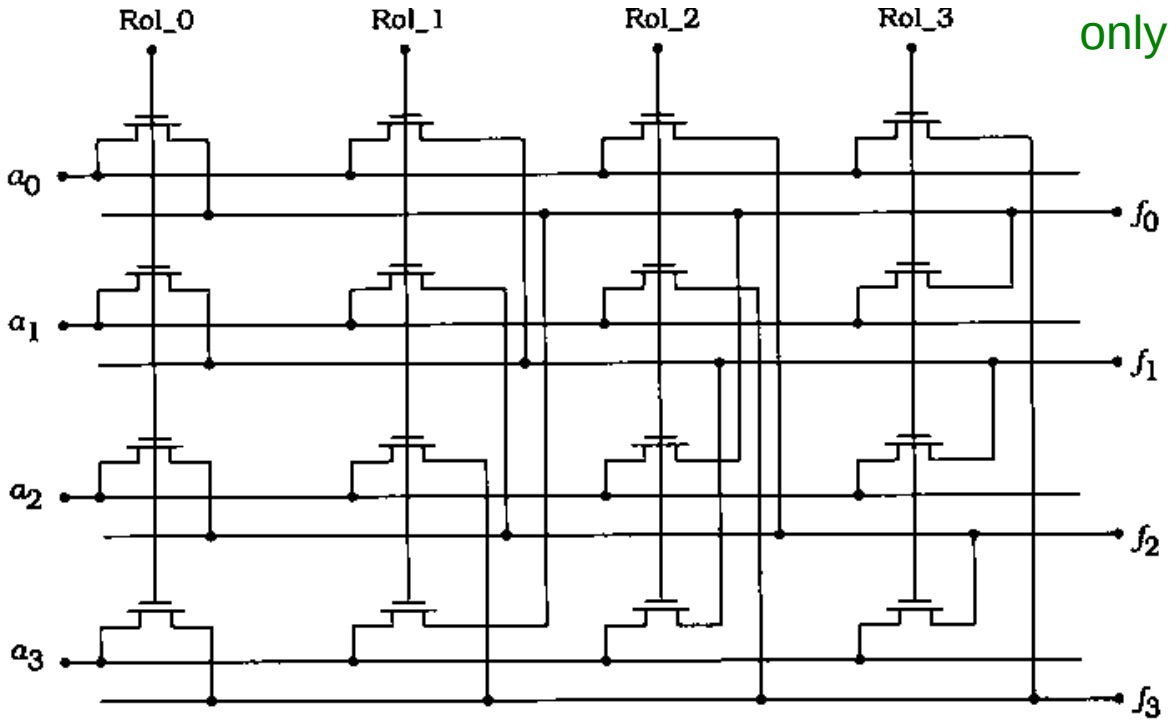


could add an Enable at the clock input to select between multiple bytes

Switch Shift/Rotate Circuits

- Can use switch circuits to implement fast multi-shift/rotate functions
  - will not store/hold data since no FF is used & is not synchronous
  - Example: 4-bit Left Rotate Switching Array

Rotate Left, moves lower bits to higher bits



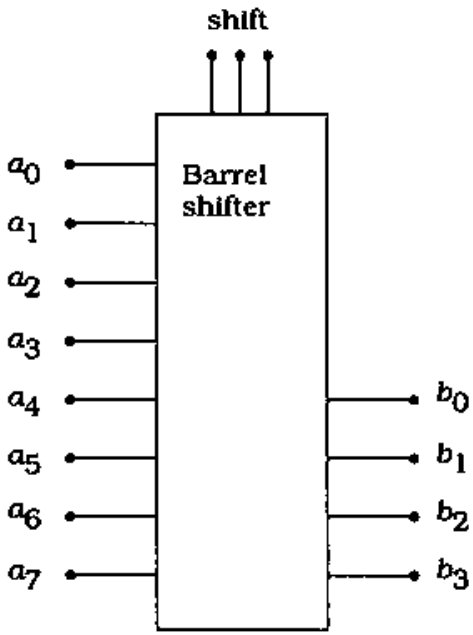
only one select (Rol_x) is active at a time

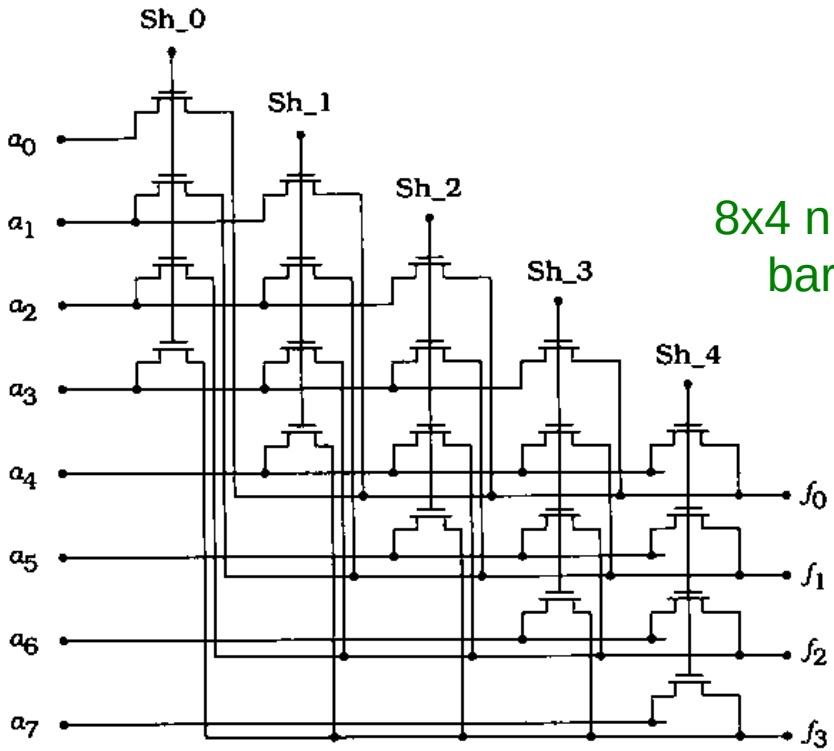
Rol0	Rol1	Rol2	Rol3
a0	a3	a2	a1
a1	a0	a3	a2
a2	a1	a0	a3
a3	a2	a1	a0

Barrel Shifter

- Shifts *m* inputs into *n* outputs
  - typically n = m or n = m/2
- Example 8x4 barrel shifter
  - outputs 1 of 4 combinations of 4-adjacent-bits

shift	$b_0 b_1 b_2 b_3$
0	$a_0 a_1 a_2 a_3$
1	$a_1 a_2 a_3 a_4$
2	$a_2 a_3 a_4 a_5$
3	$a_3 a_4 a_5 a_6$
4	$a_4 a_5 a_6 a_7$

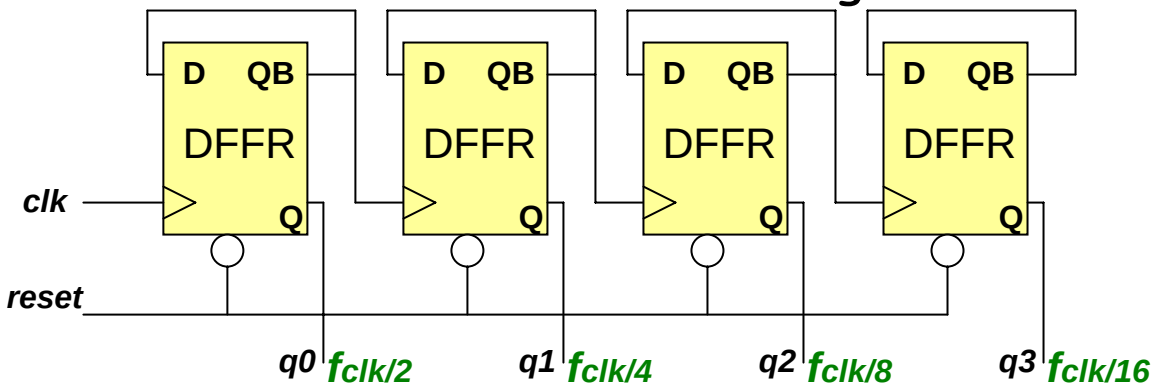




8x4 nMOS switch barrel shifter

Asynchronous Counter

- Counts the number of input clock edges (+ive or -ive)
- Output is a binary code of the number of clocks counted
- Example: 4-bit counter
  - output_bar of each bit provides clock to next bit
  - output is also fed back to input
  - frequency of each output is 1/2 the previous bit frequency
    - clock divider: divide by 2, by 4, by 8, by 16, etc.
  - reset used to start counting from Zero

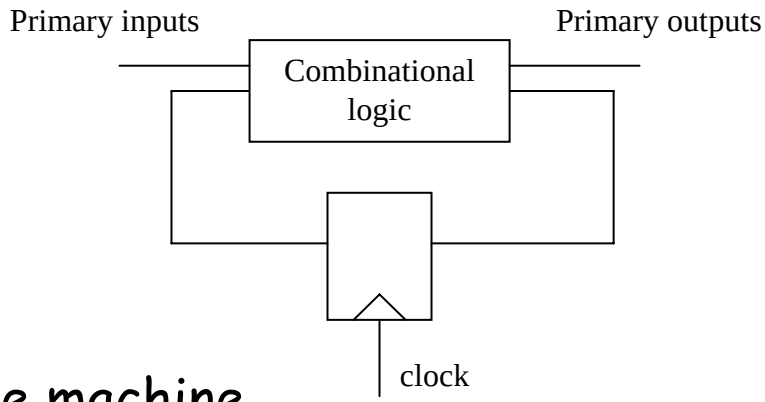


#clks	q3	q2	q1	q0
0	0	0	0	0
1	0	0	0	1
3	0	0	1	0
5	0	0	1	1
7	0	1	0	0
9	0	1	0	1
11	0	1	1	0
13	0	1	1	1
15	1	0	0	0
*				
31	1	1	1	1

can you design a counter that can count up and down and can parallel load a starting value?

Sequential Circuits

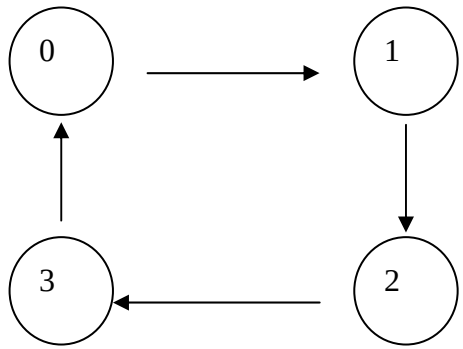
- A sequential circuit
  - outputs depend on current inputs
  - AND on pervious inputs (history)



- Finite State Machine
  - generic sequential circuit
  - a D-Flip-Flop holds the state of the machine
  - combinational logic generates the next state and output(s)
  - state machine inputs/outputs are called primary inputs/outputs
    - Mealy machine: primary outputs are a function of
      - current state
      - primary inputs
    - Moore machine: primary outputs depend only on
      - current state
- Sequential machines occur in nearly every chip design.

State Machine Example

- 2-bit synchronous counter
  - example of a sequential state machine
- 2-bit synchronous counter function
  - increments output from 0 to 3 at each clock
  - and then start from 0 again
  - counter has no inputs, only states (Moore machine)
- Design Steps



1. Specify the state transition graph.
- **Four states in the 2-bit counter: 0, 1, 2, 3.**
- **State transition graph for 2-bit counter machine changes to the next state on each clock**

2. Determine number of DFF in the state machine.
- **Number of FF needed for a state machine is given by $2^n=N$ N is the number of states and n is the number of flip flops**
- **2-bit counter has 4 states 00, 01, 10, and 11**
- **→ need 2 DFFs**

State Machine Example Continued

- Design Steps continued

3. Draw the state transition table for the state transition graph.

Present State		Next State	
DFF_1	***DFF_0***	***DFF_1***	***DFF_0***
0	0	0	1
0	**1**	**1**	**0**
1	0	1	1
1	1	0	0

Example: at present state 1 (binary "01"), next state will be 2 (binary "10").

4. Design the logic to compute the next state.

- **One K-map is used for each DFF**
- **Example: DFF_0 has the following K-map**

	$DFF\_0_{old} = 0$	$DFF\_0_{old} = 1$
$DFF\_1_{old} = 0$	1	0
$DFF\_1_{old} = 1$	1	0

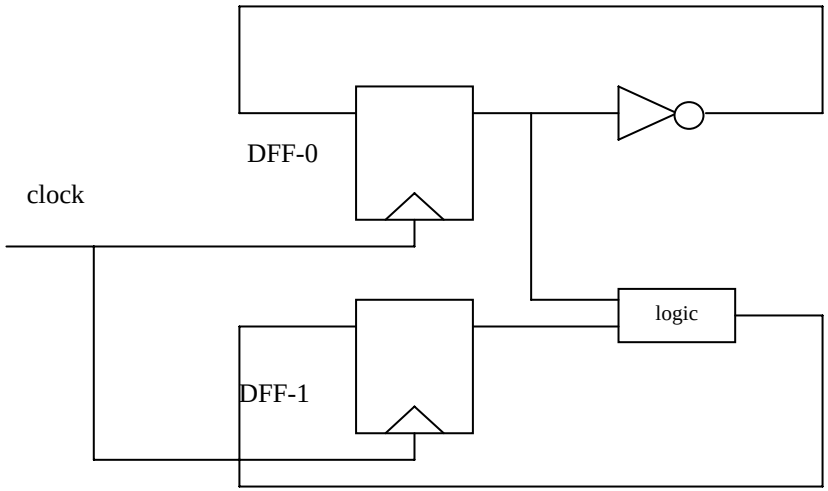
table shows **next** state value

Thus, $DFF\_0_{next} = \overline{DFF\_0_{old}}$ Similar approach to find DFF_1_{next}

Synchronous Counter

- Design Steps continued

5. **Connect combinational logic & the DFFs to construct 2-bit counter**



- A 2-bit, synchronous, 4-state counter