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

---

final chip
primitives
in1
out

continuous power rails
internal connections
in1
in2
out
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

- Connecting rails in stacked cell cascades
  - use many contacts (vias)
  - GND VDD
  - VDD GND
  - GND VDD
  - GND
  - GND
  - GND
  - GND
  - zooming out…
  - branching of power traces across a chip, from thick lines (chip) to thin lines (cell)
Signal Buffers

• Loading and Fan-Out
  - gate input capacitance
    • $C_G = 2C_{ox}WL$ (1 for pMOS, 1 for nMOS)
  - load capacitance
    • standard gate designed to drive a load of 3 gates $\Rightarrow C_L = 3C_G$
  - output drive capability
    • $I \propto W$, 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 $W/L$
      - typically increase by 3x at each stage
Transmission Gate Multiplexors

- **Logical Function of a Multiplexor**
  - select one output from multiple inputs
  - 2:1 MUX logic
    \[ f = p_0 \cdot \overline{s} + p_1 \cdot s \]

- **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
    - \( m \) is at most \( 2^n \)
  - 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

- **Example: 2/4 (2-to-4) Decoder**
  - 2 control bits decoded into 4 values
    - Truth table
    - Equations
      \[
      \begin{align*}
      d_0 &= \overline{s_1} \cdot \overline{s_0} = \overline{s_1 + s_0} \\
      d_2 &= s_1 \cdot \overline{s_0} = \overline{s_1 + s_0} \\
      d_1 &= \overline{s_1} \cdot s_0 = \overline{s_1 + s_0} \\
      d_3 &= s_1 \cdot s_0 = \overline{s_1 + s_0}
      \end{align*}
      \]
    - Active high decoder equations require NOR operation

\[
\begin{array}{cccc|cccc}
 s_1 & s_0 & d_0 & d_1 & d_2 & d_3 \\
 0 & 0 & 1 & 0 & 0 & 0 \\
 0 & 1 & 0 & 1 & 0 & 0 \\
 1 & 0 & 0 & 0 & 1 & 0 \\
 1 & 1 & 0 & 0 & 0 & 1 \\
\end{array}
\]
**CMOS Decoder Circuits**

- **2/4 Active High Decoder**
  - Implemented with NAND gates
  - Truth Table
    | $s_1$ | $s_0$ | $d_0$ | $d_1$ | $d_2$ | $d_3$ |
    |------|------|------|------|------|------|
    | 0    | 0    | 1    | 0    | 0    | 0    |
    | 0    | 1    | 0    | 1    | 0    | 0    |
    | 1    | 0    | 0    | 0    | 1    | 0    |
    | 1    | 1    | 0    | 0    | 0    | 1    |
  - Symbol

- **2/4 Active Low Decoder**
  - Implemented with NAND gates
  - Truth Table
    | $s_1$ | $s_0$ | $d_0$ | $d_1$ | $d_2$ | $d_3$ |
    |------|------|------|------|------|------|
    | 0    | 0    | 0    | 1    | 1    | 1    |
    | 0    | 1    | 1    | 0    | 1    | 1    |
    | 1    | 0    | 1    | 1    | 0    | 1    |
    | 1    | 1    | 1    | 1    | 0    | 0    |
  - Symbol

- **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

<table>
<thead>
<tr>
<th>s2</th>
<th>s1</th>
<th>s0</th>
<th>d7</th>
<th>d6</th>
<th>d5</th>
<th>d4</th>
<th>d3</th>
<th>d2</th>
<th>d1</th>
<th>d0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- 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: \( a \oplus b \), can be generated by XNOR operation
  - \( a = b \) iff \( a \oplus b = 1 \) for each binary digit
    - example: 4b equality comparator using XNOR
  - also, \( a = b \) if \( a > b = 0 \) and \( a < b = 0 \) for each binary digit
- Greater/Less Than, by bit-by-bit comparison

<table>
<thead>
<tr>
<th>Condition</th>
<th>GT</th>
<th>LT</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a &gt; b )</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( a &lt; b )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( a = b )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>d_7</th>
<th>d_6</th>
<th>d_5</th>
<th>d_4</th>
<th>d_3</th>
<th>d_2</th>
<th>d_1</th>
<th>d_0</th>
<th>Q_3</th>
<th>Q_2</th>
<th>Q_1</th>
<th>Q_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

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
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
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](image1)

![Transistor-Level Circuit](image2)

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

- **Different structures used in VLSI**

![Symbol](image3)

![Logic diagram](image4)
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)**

  \[
  \begin{array}{c|c}
  D & Q_{n+1} \\
  \hline
  0 & 0 \\
  1 & 1 \\
  \end{array}
  \]

- **SR-type**

  same as D if S=D and R=D’

  \[
  \begin{array}{c|c|c}
  S & R & Q_{n+1} \\
  \hline
  0 & 0 & Q_n \\
  0 & 1 & 0 \\
  1 & 0 & 1 \\
  1 & 1 & \text{Indeterminate} \\
  \end{array}
  \]

- **JK-type**

  \[
  \begin{array}{c|c|c}
  J & K & Q_{n+1} \\
  \hline
  0 & 0 & Q_n \\
  0 & 1 & 0 \\
  1 & 0 & 1 \\
  1 & 1 & \bar{Q}_n \\
  \end{array}
  \]

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

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>Q_{n+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q_n</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>\bar{Q}_n</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>T</th>
<th>Q_{n+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Q_n</td>
</tr>
<tr>
<td>1</td>
<td>\bar{Q}_n</td>
</tr>
</tbody>
</table>
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
    - \( \phi = 1, \bar{\phi} = 0 \): D passes to master, slave holds previous value
    - \( \phi = 0, \bar{\phi} = 1 \): 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 $W/L$
      - typically increase by 3x at each stage

- **Inter-cell Buffering**
  - **Clock**
    - so each flip flop provides 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 $Q$. 

![Diagram showing the timing parameters of a flip-flop](image)
Analyzing DFF Timing

• Setup
  - When $\phi$ 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 $\phi$ 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) Wiring diagram
(b) Symbol
(c) Hold with $\phi = 0$, Load = 0
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
  - CMOS implementation

<table>
<thead>
<tr>
<th>$En$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$Z$</td>
</tr>
<tr>
<td>1</td>
<td>$\overline{Data}$</td>
</tr>
</tbody>
</table>
Advanced Latches and Flip Flops

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

- **$C^2$MOS Static Latch**
  - merge TGs into latch design
  - $C^2$MOS inverter input stage
    - passes inverted input when $\Phi = 0$
    - static inverter sets $Q = D$
  - $C^2$MOS inverter feedback
    - provides feedback when $\Phi = 1$
  - Either input of feedback is active
    - not both at the same time
C\textsuperscript{2}MOS D Flip Flop

- Cascade 2 C\textsuperscript{2}MOS 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

• D=1 \(\rightarrow\) x=0
  - \(V_{GS} = VDD\), tx is ON with strong \(V_{GS}\), \(V_{GS}\) constant as output changes
  - \(V_{DS}: VDD \Rightarrow 0\), tx in Saturation then in Triode

• D=0 \(\rightarrow\) x=1
  - \(V_{GS} = VDD \Rightarrow Vtn\), tx is ON, but \(V_{GS}\) decreases as output changes
  - \(V_{DS}: VDD \Rightarrow Vtn\), 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 $C^2$MOS Flip Flop
  - double-wide FF
    - pitch is 2x pitch of basic gates

- Using tall cells with standard height cells
  - match power rails

GND

VBB

GND

52$\lambda$ 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

6b register formed with DFF cells
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 = $a_2a_1a_00$
  - Shift Right: output = $0a_3a_2a_1$
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)

<table>
<thead>
<tr>
<th>s2</th>
<th>s1</th>
<th>s0</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>x</td>
<td>parallel load</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>shift right</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>rotate right</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>shift left</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>rotate left</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>x</td>
<td>load ‘1’</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Rol0</th>
<th>Rol1</th>
<th>Rol2</th>
<th>Rol3</th>
</tr>
</thead>
<tbody>
<tr>
<td>f0</td>
<td>a0</td>
<td>a3</td>
<td>a2</td>
<td>a1</td>
</tr>
<tr>
<td>f1</td>
<td>a1</td>
<td>a0</td>
<td>a3</td>
<td>a2</td>
</tr>
<tr>
<td>f2</td>
<td>a2</td>
<td>a1</td>
<td>a0</td>
<td>a3</td>
</tr>
<tr>
<td>f3</td>
<td>a3</td>
<td>a2</td>
<td>a1</td>
<td>a0</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>shift</th>
<th>( b_0b_1b_2b_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( a_0a_1a_2a_3 )</td>
</tr>
<tr>
<td>1</td>
<td>( a_1a_2a_3a_4 )</td>
</tr>
<tr>
<td>2</td>
<td>( a_2a_3a_4a_5 )</td>
</tr>
<tr>
<td>3</td>
<td>( a_3a_4a_5a_6 )</td>
</tr>
<tr>
<td>4</td>
<td>( a_4a_5a_6a_7 )</td>
</tr>
</tbody>
</table>

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

\[
\begin{array}{cccccc}
\text{#clks} & q_3 & q_2 & q_1 & q_0 \\
0 & 0 & 0 & 0 & 0 \quad & \text{fclk/2} \\
1 & 0 & 0 & 1 & 0 \quad & \text{fclk/4} \\
3 & 0 & 0 & 1 & 1 \quad & \text{fclk/8} \\
5 & 0 & 1 & 0 & 1 \quad & \text{fclk/16} \\
7 & 0 & 1 & 0 & 0 \quad & \text{fclk/2} \\
9 & 0 & 1 & 0 & 0 \quad & \text{fclk/4} \\
11 & 0 & 1 & 1 & 0 \quad & \text{fclk/8} \\
13 & 0 & 1 & 1 & 1 \quad & \text{fclk/16} \\
15 & 1 & 0 & 0 & 0 \quad & \text{fclk/2} \\
31 & 1 & 1 & 1 & 1 \quad & \text{fclk/4}
\end{array}
\]

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 previous 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
   - $\rightarrow$ need 2 DFFs
State Machine Example Continued

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

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DFF_1</td>
</tr>
<tr>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>DFF_0_{old} = 0</th>
<th>DFF_0_{old} = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFF_1_{old} = 0</td>
<td>1</td>
</tr>
<tr>
<td>DFF_1_{old} = 1</td>
<td>1</td>
</tr>
</tbody>
</table>

Thus, DFF_0_{next} = \( \frac{DFF_0_{old}}{DFF_1_{old}} \)  

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