**Combination Logic Technologies**

- Standard gates
  - gate packages
  - cell libraries

- Regular logic
  - multiplexers
  - decoders

- Two-level programmable logic
  - PALs
  - PLAs
  - ROMs

---

**Random logic**

- Transistors quickly integrated into logic gates (1960s)

- Catalog of common gates (1970s)
  - Texas Instruments Logic Data Book – the yellow bible
  - all common packages listed and characterized (delays, power)
  - typical packages:
    - in 14-pin IC: 6-inverters, 4 NAND gates, 4 XOR gates

- Today, very few parts are still in use

- However, parts libraries exist for chip design
  - designers reuse already characterized logic gates on chips
  - same reasons as before
  - difference is that the parts don’t exist in physical inventory – created as needed
Random logic

- Too hard to figure out exactly what gates to use
  - map from logic to NAND/NOR networks
  - determine minimum number of packages
    - slight changes to logic function could decrease cost
- Changes to difficult to realize
  - need to rewire parts
  - may need new parts
  - design with spares (few extra inverters and gates on every board)

Regular logic

- Need to make design faster
- Need to make engineering changes easier to make
- Simpler for designers to understand and map to functionality
  - harder to think in terms of specific gates
  - better to think in terms of a large multi-purpose block
Making connections

- Direct point-to-point connections between gates
  - wires we’ve seen so far
- Route one of many inputs to a single output --- multiplexer
- Route a single input to one of many outputs --- demultiplexer

Mux and demux

- Switch implementation of multiplexers and demultiplexers
  - can be composed to make arbitrary size switching networks
  - used to implement multiple-source/multiple-destination interconnections
**Mux and demux (cont'd)**

**Uses of multiplexers/demultiplexers in multi-point connections**

![Diagram of MUX and DEMUX connections]

**Multiplexers/selectors**

**Multiplexers/selectors: general concept**
- \(2^n\) data inputs, \(n\) control inputs (called "selects"), 1 output
- Used to connect \(2^n\) points to a single point
- Control signal pattern forms binary index of input connected to output

\[ Z = A'I_o + A'I_1 \]

<table>
<thead>
<tr>
<th>A</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I_1</th>
<th>I_0</th>
<th>A</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Functional form

Logical form

Two alternative forms for a 2:1 Mux truth table
Multiplexers/selectors (cont'd)

- **2:1 mux:** \( Z = A' I_0 + A I_1 \)
- **4:1 mux:** \( Z = A' B' I_0 + A' B I_1 + A B' I_2 + A B I_3 \)
- **8:1 mux:** \( Z = A' B' C' I_0 + A' B' C I_1 + A' B C' I_2 + A' B C I_3 + A B' C' I_4 + A B' C I_5 + A B C' I_6 + A B C I_7 \)

- **In general:** \( Z = \sum_{k=0}^{n-1} (m_k I_k) \)

Diagram in minterm shorthand form for a 2^n:1 Mux

Gate level implementation of muxes

- **2:1 mux**

- **4:1 mux**
**Cascading multiplexers**

- Large multiplexers can be implemented by cascading smaller ones.

control signals B and C simultaneously choose one of I0, I1, I2, I3 and one of I4, I5, I6, I7

control signal A chooses which of the upper or lower mux's output to gate to Z

---

**Multiplexers as general-purpose logic**

- A $2^n:1$ multiplexer can implement any function of $n$ variables
  - with the variables used as control inputs and
  - the data inputs tied to 0 or 1
  - in essence, a lookup table

- **Example:**
  
  $F(A,B,C) = m_0 + m_2 + m_6 + m_7$
  
  $= A'B'C' + AB'C + ABC + ABC$
  
  $= A'B'(C') + AB(C') + AB(0) + AB(1)$

---
Multiplexers as general-purpose logic (cont'd)

- A $2^n:1$ multiplexer can implement any function of $n$ variables.
  - with $n-1$ variables used as control inputs and
  - the data inputs tied to the last variable or its complement.

**Example:**

- $F(A,B,C) = m_0 + m_2 + m_6 + m_7$
- $= A'B'C' + A'BC + ABC' + ABC$
- $= A'B'(C') + A'B(C') + AB(0) + AB(1)$

**Generalization**

- $n-1$ mux control variables
- single mux data variable

**Example: $G(A,B,C,D)$ can be implemented by a $8:1$ MUX**

- choose $A,B,C$ as control variables
- multiplexer implementation

- four possible configurations of truth table rows can be expressed as a function of $I_n$.
Activity

**Map the following equation to an 4:1 multiplexer using a minimum of external gates:**

\[ Z = B'C'(0) + B'C(D') + BC'(A) + BC(1) \]

Demultiplexers/decoders

**Decoders/demultiplexers: general concept**
- single data input, \( n \) control inputs, \( 2^n \) outputs
- control inputs (called "selects" (S)) represent binary index of output to which the input is connected
- data input usually called "enable" (G)

<table>
<thead>
<tr>
<th>1:2 Decoder</th>
<th>3:8 Decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 = G * S'</td>
<td>00 = G * S2' * S1' * S0'</td>
</tr>
<tr>
<td>01 = G * S</td>
<td>01 = G * S2' * S1' * S0</td>
</tr>
<tr>
<td>02 = G * S'</td>
<td>02 = G * S2' * S1 * S0'</td>
</tr>
<tr>
<td>03 = G * S</td>
<td>03 = G * S2' * S1 * S0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2:4 Decoder</th>
<th>3:8 Decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 = G * S1' * S0'</td>
<td>00 = G * S2' * S1' * S0'</td>
</tr>
<tr>
<td>01 = G * S1' * S0</td>
<td>01 = G * S2' * S1 * S0</td>
</tr>
<tr>
<td>02 = G * S1 * S0'</td>
<td>02 = G * S2 * S1' * S0'</td>
</tr>
<tr>
<td>03 = G * S1 * S0</td>
<td>03 = G * S2 * S1 * S0</td>
</tr>
</tbody>
</table>
### Gate level implementation of demultiplexers

<table>
<thead>
<tr>
<th><strong>1:2 decoders</strong></th>
<th><strong>active-high enable</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>2:4 decoders</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3" alt="Diagram" /></td>
</tr>
</tbody>
</table>

### Demultiplexers as general-purpose logic

- **A**: Any 2^n decoder can implement any function of n variables
  - with the variables used as control inputs
  - the enable inputs tied to 1 and
  - the appropriate minterms summed to form the function

```
<table>
<thead>
<tr>
<th>Control Inputs</th>
<th>Output Minterms</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2, S1, S0</td>
<td>A'B'C', A'B'C, A'BC', ABC'</td>
</tr>
</tbody>
</table>
```

- demultiplexer generates appropriate minterm based on control signals (it "decodes" control signals)
Demultiplexers as general-purpose logic (cont’d)

\[ F_1 = A'BCD + A'B'CD + ABCD \]
\[ F_2 = ABCD' + ABC \]
\[ F_3 = (A' + B' + C' + D') \]

Cascading decoders

\[ 5:32 \text{ decoder} \]
- 1x2:4 decoder
- 4x3:8 decoders
Programmable logic arrays

- **Pre-fabricated building block of many AND/OR gates**
  - actually NOR or NAND
  - "personalized" by making or breaking connections among the gates
  - programmable array block diagram for sum of products form

```
AND array

inputs

product terms

outputs
```

Enabling concept

- **Shared product terms among outputs**
  - example:
    - \( F_0 = A + B'C' \)
    - \( F_1 = A'C' + A B \)
    - \( F_2 = B'C' + A B \)
    - \( F_3 = B'C + A \)

```

<table>
<thead>
<tr>
<th>product term</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F0</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B'C</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A'B</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B'C'</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

```

- input side:
  - 1 = uncomplemented in term
  - 0 = complemented in term
  - - = does not participate

- output side:
  - 1 = term connected to output
  - 0 = no connection to output

```

reuse of terms
```
Before programming

- All possible connections are available before "programming"
- In reality, all AND and OR gates are NANDs

After programming

- Unwanted connections are "blown"
  - Fuse (normally connected, break unwanted ones)
  - Anti-fuse (normally disconnected, make wanted connections)
**Alternate representation for high fan-in structures**

Short-hand notation so we don't have to draw all the wires.
- × signifies a connection is present and perpendicular signal is an input to gate.

```
F0 = A B + A' B'
F1 = C D' + C' D
```

Programmable logic array example

- Multiple functions of A, B, C
  - F1 = A B C
  - F2 = A + B + C
  - F3 = A' B' C'
  - F4 = A' + B' + C'
  - F5 = A xor B xor C
  - F6 = A xor B xor C

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
**PALs and PLAs**

- Programmable logic array (PLA)
  - what we’ve seen so far
  - unconstrained fully-general AND and OR arrays

- Programmable array logic (PAL)
  - constrained topology of the OR array
  - innovation by Monolithic Memories
  - faster and smaller OR plane

A given column of the OR array has access to only a subset of the possible product terms.

---

**PALs and PLAs: design example**

- BCD to Gray code converter

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>W</th>
<th>X</th>
<th>Y</th>
<th>Z</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>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</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>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</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>1</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Minimized functions:

- \( W = A + B \overline{D} + B C \)
- \( X = B \overline{C} \)
- \( Y = B + C \)
- \( Z = A'B'C'D + B'C'D + A'D' + B'C'D' \)
not a particularly good candidate for PAL/PLA implementation since no terms are shared among outputs however, much more compact and regular implementation when compared with discrete AND and OR gates

minimized functions:
\[ W = A + B D + B C \]
\[ X = B C' \]
\[ Y = B + C \]
\[ Z = A'B'C'D + B C D + A D' + B' C D' \]

4 product terms per each OR gate

Code converter: programmed PLA

Code converter: programmed PAL
**PALs and PLAs: design example (cont’d)**

- **Code converter: NAND gate implementation**
  - Loss or regularity, harder to understand
  - Harder to make changes

**PALs and PLAs: another design example**

- **Magnitude comparator**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>EQ</th>
<th>NE</th>
<th>LT</th>
<th>GT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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>1</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

Minimized functions:

- **EQ** = \( A'B'C'D' + A'B'C'D + ABCD + AB'CD' \)
- **NE** = \( A'C' + A + C' + B'D + BD' \)
- **LT** = \( AC + A'B + BCD \)
- **GT** = \( AC' + ABC + BC'D' \)

Winter 2001 CSE370 - III - Combinational Logic Technologies
Activity

Map the following functions to the PLA below:
- \( W = AB + AC' + BC' \)
- \( X = ABC + AB' + A'B \)
- \( Y = ABC' + BC + B'C' \)

Activity (cont'd)

9 terms won't fit in a 7 term PLA. Can apply consensus theorem to \( W \) to simplify to:
- \( W = AB + AC' \)

8 terms won't fit in a 7 term PLA. Observe that \( AB = ABC + ABC' \). Can rewrite \( W \) to reuse terms:
- \( W = ABC + ABC' + AC' \)

Now it fits:
- \( W = ABC + ABC' + AC' \)
- \( X = ABC + AB' + A'B \)
- \( Y = ABC' + BC + B'C' \)

This is called technology mapping - manipulating logic functions so that they can use available resources.
Read-only memories

- Two dimensional array of 1s and 0s
- Entry (row) is called a "word"
- Width of row = word-size
- Index is called an "address"
- Address is input
- Selected word is output

ROMs and combinational logic

- Combinational logic implementation (two-level canonical form) using a ROM

\[
\begin{align*}
F_0 &= A' B' C' + A B' C + A B C \\
F_1 &= A' B' C + A' B C' + A B C \\
F_2 &= A' B' C + A' B' C + A B' C' \\
F_3 &= A' B C + A B' C' + A B C'
\end{align*}
\]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F0</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</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>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

ROM block diagram

[Block diagram of ROM with truth table and outputs]
**ROM structure**

- Similar to a PLA structure but with a fully decoded AND array
  - completely flexible OR array (unlike PAL)

![Diagram of ROM structure]

**ROM vs. PLA**

- **ROM approach advantageous when**
  - design time is short (no need to minimize output functions)
  - most input combinations are needed (e.g., code converters)
  - little sharing of product terms among output functions

- **ROM problems**
  - size doubles for each additional input
  - can't exploit don't cares

- **PLA approach advantageous when**
  - design tools are available for multi-output minimization
  - there are relatively few unique minterm combinations
  - many minterms are shared among the output functions

- **PAL problems**
  - constrained fan-ins on OR plane
Regular logic structures for two-level logic

- **ROM** – full AND plane, general OR plane
  - cheap (high-volume component)
  - can implement any function of n inputs
  - medium speed
- **PAL** – programmable AND plane, fixed OR plane
  - intermediate cost
  - can implement functions limited by number of terms
  - high speed (only one programmable plane that is much smaller than ROM's decoder)
- **PLA** – programmable AND and OR planes
  - most expensive (most complex in design, need more sophisticated tools)
  - can implement any function up to a product term limit
  - slow (two programmable planes)

Regular logic structures for multi-level logic

- **Difficult to devise a regular structure for arbitrary connections between a large set of different types of gates**
  - efficiency/speed concerns for such a structure
  - in 467 you’ll learn about field programmable gate arrays (FPGAs) that are just such programmable multi-level structures
    - programmable multiplexers for wiring
    - lookup tables for logic functions (programming fills in the table)
    - multi-purpose cells (utilization is the big issue)
- **Use multiple levels of PALs/PLAs/ROMs**
  - output intermediate result
  - make it an input to be used in further logic
Combination logic technology summary

- **Random logic**
  - Single gates or in groups
  - Conversion to NAND-NAND and NOR-NOR networks
  - Transition from simple gates to more complex gate building blocks
  - Reduced gate count, fan-ins, potentially faster
  - More levels, harder to design

- **Time response in combinational networks**
  - Gate delays and timing waveforms
  - Hazards/glitches (what they are and why they happen)

- **Regular logic**
  - Multiplexers/decoders
  - ROMs
  - PLAs/PALs
  - Advantages/disadvantages of each