Implementing Switching Functions

Consider a truth table for a multiple-output function with $k$ switching variables and $m$ output functions:

$$
\begin{align*}
    f_1(x_1, \ldots, x_k) &= \Sigma(list\ 1) \\
    f_2(x_1, \ldots, x_k) &= \Sigma(list\ 2) \\
    \quad \text{etc.} \\
    f_m(x_1, \ldots, x_k) &= \Sigma(list\ m)
\end{align*}
$$

For each of the $2^k$ assignments to $(x_1, \ldots, x_k)$ there is an $m$-bit vector that defines $(f_1, \ldots, f_m)$.

- Essentially, this is memory with a capacity of $2^k$ word each $m$ bits in length.

- $(x_1, \ldots, x_k)$ is the address

- $(f_1, \ldots, f_m)$ is the contents of the location.
• RAM is too costly and often inappropriate in function for this application.

• Programmable ROM (PROM) is ideally suited.

• high-density compared to writable RAM

• generic form can be manufactured cheaply

• field-programmable to specify switching functions needed for particular applications

• nonvolatile therefore usable for power on/off situations

• Common uses:
  – processor control – instruction in and resource control signals out
  – bootstrap data and code for power-up of systems

• FPGAs are a more sophisticated (and more expensive) version of this idea (see Chapter 3).
Programmable OR Gate

1. Decide which $A_i$ are included in each $F$

2. "burn" fuses to make the connection

3. If any "connected" horizontal line asserts a 1 then $F = 1$ also.

4. Horizontal lines that are not "connected" do not affect $F$.

$F = A_1 + A_k$
A $2^k \times m$ PROM is a $k$-to-$2^k$ decoder and $m$ programmable OR gates. The crossings are burned to define $F_i \ i = 0, \ldots, m - 1$.

8 X 2 PROM before burning fuses
Multiple output networks are then trivial to form. Example: Full adder,

\[ m = 2 \]
\[ k = 3 \]
\[ s_i(c_i, x_i, y_i) = \Sigma(1, 2, 4, 7) \]
\[ c_{i+1}(c_i, x_i, y_i) = \Sigma(3, 5, 6, 7) \]
Programmable Logic Arrays (PLA)

- PROMs must use the address bits to create the fundamental products on which $f$ evaluates to 1.

- This can be inefficient if the 1 to 0 ratio of $f$ is small.

- We would like to use higher-order cubes.

- This requires a more flexibility decoder structure.

- The PLA replaces the standard decoder with an AND array.

- It also includes a programmable output array that allows each output line to be complemented when necessary. (This allows efficiency independently of the 1 to 0 ratio.)
• An $k \times n \times m$ PLA has
  
  – $k$ input bits that are used to set the assignment on which $f$ is to be evaluated.
  
  – $n$ programmable AND gates that work with burning connections as the programmable OR gates.
  
  – $m$ programmable OR gates that produce the $f_i$ $i = 0, \ldots, m - 1$

• Note that $n$ is not necessarily $2^k$. It is simply the number of cubes that can be defined in the PLA.

• Since the output lines of the AND array are no longer tied to a particular fundamental product each AND gate can evaluate any of the cubes required to evaluate $f$
4 × 8 × 4 PLA before burning