History

• Origins of FPGAs
• Xilinx introduced first FPGA in ’84, but engineers didn’t embrace them until early ’90s.

• History:
  – ’47: Shockley, et. el. introduce first transistor at Bell Labs.
  – ’50: Bipolar junction transistor (BIT) introduced.
  – ’62: Hofstein, et. el. introduce metal-oxide semiconductor field-(fled transistor (MOSFET) at RCA.
  – ’58: Jack Kilby introduced the integrated circuit.
    • (Jack Kilby, Nobel Prize winner, 2000)
  – ’70: Intel introduced 1024-bit DRAM, Fairchild introduced 256-bit SRAM.
  – ’71: Intel introduced first microprocessor, 4004.
Family of Programmable Dev.

- Programmable Logic Devices PLDs arrived in 70’s and but only end of the 70’s did more complex variations emerge
- New complex variations were called complex PLDs (CPLDs) while the original line became known as SPLDs

Figure 3-2. A positive plethora of PLDs.
Programmable Logic

- Here, the data on input line may be used or not.
  - If it is not used it is pulled to "1", the AND identity, so it has no effect on the output.
  - If the line is used, the pull-up to "1" is overridden by the driven value.

Try to configure switches to implement not a and b.

We’ll now discuss technology for implementing the switches....
Fuses

- A fusible-link technology is required to implement links as fuses.
- All links are initially active and must be selectively removed.
- One-time programmable --- fusible-link-based devices are **one-time programmable**, unless redundant fuses or devices can be switched in to provide **twice-programmable** devices.
- Fuses are burned by selectively applying large voltages.

![Logic circuit diagrams](image)
All links are initially inactive and must be selectively added.

Links are implemented as insulators that are destroyed or changed such that a conducting path is realized.

There are one-time programmable unless redundant devices are provided.

No need to programmed unused sections of IC.
Most layers are predesigned and prefabricated, requiring a minimal number of custom layers and masks per new design.
Memory as Logic

- Mask programming was used to create memories known as a ROM (read only memory). This is shown on the next slide.
- It turns out a memory can mimic the behavior of an arbitrary circuit by effectively storing the circuit's truth table and recalling entries using the inputs as the address (index) into the table.

Description of a circuit:

\[ x = a \oplus b \oplus c \]
\[ y = ((a \land b) \lor (a \land c) \lor (b \land c)) \]
\[ z = a \land b \land c \]

<table>
<thead>
<tr>
<th>Address (Input)</th>
<th>Stored Data (Output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
</tr>
<tr>
<td>010</td>
<td>001</td>
</tr>
<tr>
<td>011</td>
<td>010</td>
</tr>
<tr>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td>101</td>
<td>110</td>
</tr>
<tr>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>111</td>
<td>111</td>
</tr>
</tbody>
</table>
Now for the piece needed to select a row based on the input....
Decoder

Functional schematic of decoder (actual circuitry may differ, e.g. wired and may be used):

3 bits binary code

8 bits one-hot code
Metal Mask LUT

- Transistors are same, but metal layer is added
- Can en mass prefab wafers except top metal and added it later.
Fuse/Anitfuse Link

- No need to re-manufacture mask and incur fab costs and time with every change. To change design, throw IC out and "burn" another.

- Even better case is being able to "erase" programming on IC and reuse it....
A positive potential at the gate input turns transistor on by using capacitive coupling to collect charge to form a channel.

Here, the input influences channel through a series capacitance, but a stored potential on the floating gate has an effect too.

Total effect is that of input through series cap + effect of stored potential.

- Negative charge storage prevents channel formation.
- Positive charge storage assists channel formation.
- PFET works opposite
Depending on design, and stored charge a floating-gate fet can be set to switch on and off with the input or it can be set to be always on or always off.

Here are two ways to design a programmable switch cell. The second one uses two FETs...including the space between them it is about 2.5 times larger.
Floating Gate LUT

- Nothing in manufacturing sets function.
- Charge stored on "floating" node determines transistors function.
- Charge can be changed by electric fields and UV light. (next slide)
- No need to re-manufacture mask and use fab every time, or even get a new IC. To change design, just "erase" IC and reprogram it.
- Opportunity for in-system programming (ISP) and in-the-field updates (field programmable).
EPROM and EEPROM

- **EPROM** - Erasable Programmable ROM. This refers to the fact that with normal operating voltages it functions as a ROM, but UV light can be used to erase it (around 20 minutes). Cells typically implemented using single FET. Order of magnitude smaller than fusible links -> better density.

- **EEPROM (E2PROM)** - Electronically Erasable programmable ROM. This refers to the fact that with normal operating voltages it functions as a ROM, but special high voltages can be used to erase it. Cells typically implemented using two-FET design, thus it is typically larger than EPROM.

Fundamental structure and operation are same. Difference in details of material, dimensions, and spacings to allow for UV or electronic erasing and proper capacitive couplings.
SRAM

- Typically larger than EEPROM cell
- Electronically Erasable
- VERY fast reprogramming
- Volatile

1 or 0 Stored in each latch
Most devices use SRAM

Fast reprogramming

SRAM is an extremely common building block in IC design, this means the structure will be well tested and sure to be reliable in most any technology.

Can be implemented in standard CMOS fabrication technology without need for extra layers or processes for special materials that significantly increase cost.

Volatile...must be reprogrammed at powerup

- System needs may demand a non-volatile configuration memory on-board (not fuse and flash technologies store the configuration as non-volatile on-chip)

Programming is fast, it can support dynamic reconfiguration were hardware is reconfigured on-the-fly during operation to accelerate functions on-demand

* A security concern is that a design is transferred as a data stream at boot..it should be encrypted if IP theft is a concern
Antifuse

- One-time programmable (unless redundant fuses are provided to provide twice-programmable)
- Radation hard – not as susceptible to radiation induced “bit flips” that alter configuration in SRAM
  - In particular SRAM is most susceptible as data is being loaded, during programming
Flash

- About 2.5 times larger cells than EPROM, but still smaller than SRAM
  - Note area impacts logic delay and power
- Dedicated flash processes require ~5 additional process steps
- Flash technology integrated with logic is not quite as rapidly updated as SRAM on newer, smaller technology nodes
- Vulnerable to long term effects from Radiation
- “Hybrid flash/SRAM” can use local flash to store configuration and SRAM to implement switches
# Programmable Element Technology

<table>
<thead>
<tr>
<th>Feature</th>
<th>SRAM</th>
<th>Antifuse</th>
<th>E2PROM / FLASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology node</td>
<td>State-of-the-art</td>
<td>One or more generations behind</td>
<td>One or more generations behind</td>
</tr>
<tr>
<td>Reprogrammable</td>
<td>Yes (in system)</td>
<td>No</td>
<td>Yes (in-system or offline)</td>
</tr>
<tr>
<td>Reprogramming speed (inc. erasing)</td>
<td>Fast</td>
<td>----</td>
<td>3x slower than SRAM</td>
</tr>
<tr>
<td>Volatile (must be programmed on power-up)</td>
<td>Yes</td>
<td>No</td>
<td>No (but can be if required)</td>
</tr>
<tr>
<td>Requires external configuration file</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Good for prototyping</td>
<td>Yes (very good)</td>
<td>No</td>
<td>Yes (reasonable)</td>
</tr>
<tr>
<td>Instant-on</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IP Security</td>
<td>Acceptable (especially when using bitstream encryption)</td>
<td>Very Good</td>
<td>Very Good</td>
</tr>
<tr>
<td>Size of configuration cell</td>
<td>Large (six transistors)</td>
<td>Very small</td>
<td>Medium-small (two transistors)</td>
</tr>
<tr>
<td>Power consumption</td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>Rad Hard</td>
<td>No</td>
<td>Yes</td>
<td>Not really</td>
</tr>
</tbody>
</table>
PROM

- Implements sum of products
- Built from TWO arrays:
  - Fixed AND gates (example uses 3-input ANDs)
  - Programmable OR gates (variable 1 to 8 inputs)
- For the fixed AND array, all inputs outputs must be fabricated since you don't know which product terms are needed. (8 gates required to exhaustive cover possibility based on 3 inputs), whether they are needed or NOT
- For programmable OR portion, the programmer can decide which terms are needed. The number to fabricate in silicon is decided based on past experience of the manufacturer while the number to actually wire is decided by the programmer.

Fixed, exhaustive prewired AND Array

Programmable, typically non-exhaustive OR array
As you know from previous courses, any truth table can be translated to a boolean sum of products or product of sums.

PROMs

PROMs were originally intended for use as computer memories to store programs and constant data.

However, engineers used them to implement lookup tables and state machines.

PROMs can be used to implement any block of combinational logic.

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>w</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<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>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</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>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>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Programming these functions is a simple matter of choosing the correct links in the OR array.

NOTE: Real PROMs have significantly more inputs and outputs.
PLA

• Implements sum of products
• Built from TWO **Programmable** arrays:
  – **Programmable** AND gates
    (example uses 3-input ANDs)
  – Programmable OR gates (variable 1 to 8 inputs)
• For the **programmable** AND array, not all inputs outputs must be fabricated. The number decided by the manufacturer sets how many product terms can be implemented
• Down side is that programmable technology tends to be slower...PLAs were never significant

**Programmable, non-exhaustive AND Array**

**Programmable, typically non-exhaustive OR/NOR array**

The Design Warrior’s Guide to FPGAs, ISBN 0750676043, Copyright(C) 2004 Mentor Graphics Corp
PALs

- Implements sum of products
- Built from TWO arrays in opposite approach of PROM:
  - **Programmable** AND gates (example uses 3-input ANDs)
  - **Predefined** OR gates
- Introduced in late 70s to address speed problems with PLAs. Since second array is not programmable it was designed to be faster.
- Downside compared to PLAs is the limited number of terms that can be OR’ed (illustrated is only two at a time)
More Complex Devices

• Real devices supported additional features
  – Programmable Output Inversion
  – Tristable Outputs (useful for buses and wired-or logic)
  – Ability to latch outputs
  – Configure pins as input or output (useful on smaller packages with limited physical pins)

• CPLDs replaced PLDs (which became known as SPLDs)
  – Were essentially multiple SPLDs on a chip but a key new enable technology was programmable interconnect (by Altera)
  – Introduced in ‘84 by Altera (now Intel) based on CMOS and EPROM technologies
Cost of Interconnect on Complex Programmable ICs

- As more blocks are added, exhaustive or fixed % global interconnect grows rapidly ( >O(n) ), in area, average delay, power...
- Less percentage of IC was “logic” and more was interconnect

A replacement for exhaustive interconnect was critical...
CPLDs

A generic CPLD structure typically consists of several SPLD blocks sharing a common programmable interconnection matrix.

Don't need (#outputs per block)^(#blocks) wires.

Both the SPLDs (usually PALs) and the interconnect can be programmed.

Interconnection matrix usually has more wires than the individual SPLD blocks

Therefore, a MUX is used to connect them.

The programmable switches may be EPROM, EEPROM, FLASH or SRAM based.
PALASM, JEDEC, etc.

In the early days, the design flow consisted of a hand-drawn schematic diagram that was later converted to tabular format and typed into a file.

The file (used by the *device programmer*) defined which fuses were to be blown (or which antifuses were to be grown).

Each PLD vendor developed its own file format, which made this task time consuming and error prone.

The *Joint Electron Device Engineering Council* (JEDEC) intervened and defined a standard language that everyone adopted.

*PAL Assembler* (PALASM) was also developed and allowed designers to specify the function in a sum-of-products form.

PALASM read the *HDL src file* and generated the text programming file.

PALASM and other early HDLs laid the foundation for Verilog and VHDL, and synthesis tools used today for ASIC and FPGA designs.

ABEL and CUPL are two other languages designed for programming CPLDs.