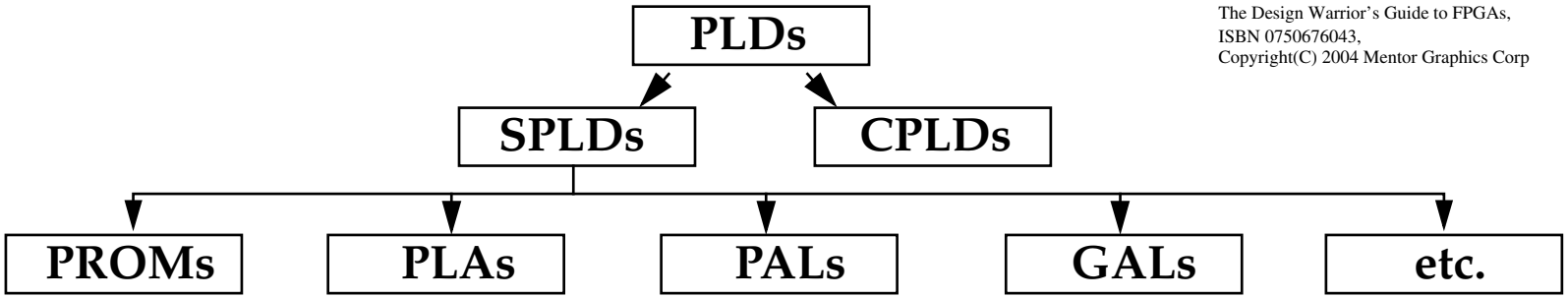
Origins of FPGAs

Xilinx introduced first FPGA in '84, but engineers didn't embrace them until early '90s.

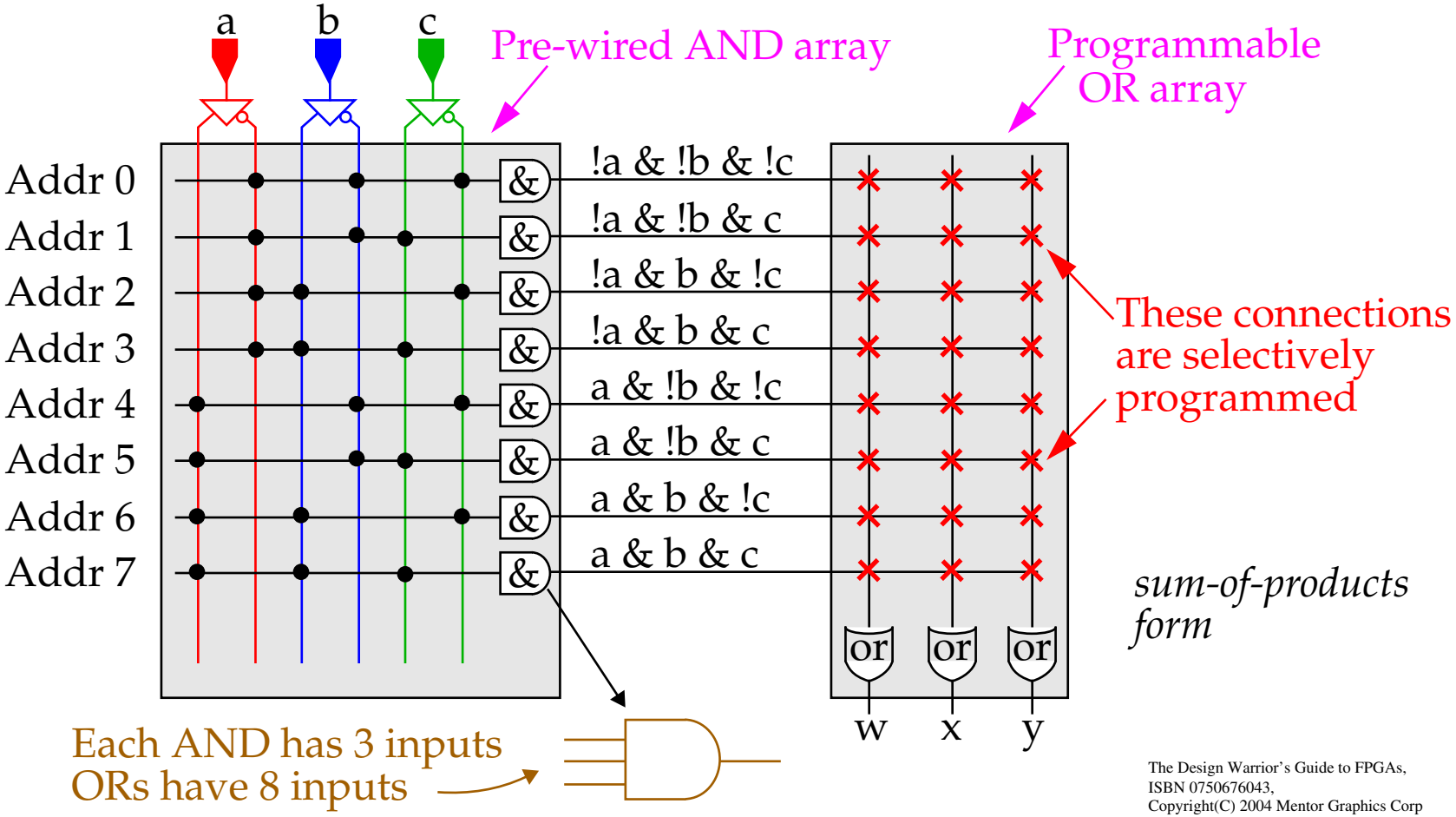
History:

- '47: Shockley, et. al. introduce first transistor at Bell Labs.
- '50: *Bipolar junction transistor* (BJT) introduced.
- '62: Hofstein, et. al. introduce *metal-oxide semiconductor field-effect transistor* (MOSFET) at RCA.
- '58: Jack Kilby introduced the *integrated circuit*.
- '70: Intel introduced 1024-bit DRAM, Fairchild introduced 256-bit SRAM.
- '71: Intel introduced first microprocessor, 4004.
- '70: PLDs introduced, later CPLDs.



PROMs

The first of the simple PLDs were **PROMs** (around '70).



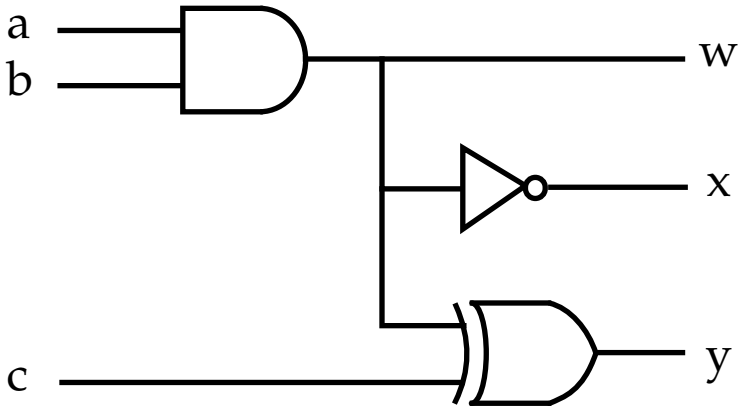
The programmable links in the OR array can be implemented as *fusible links* or as *EPROM/EEPROM* transistors.

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.



The Design Warrior's Guide to FPGAs,
ISBN 0750676043,

a	b	c	w	x	y
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	0	0

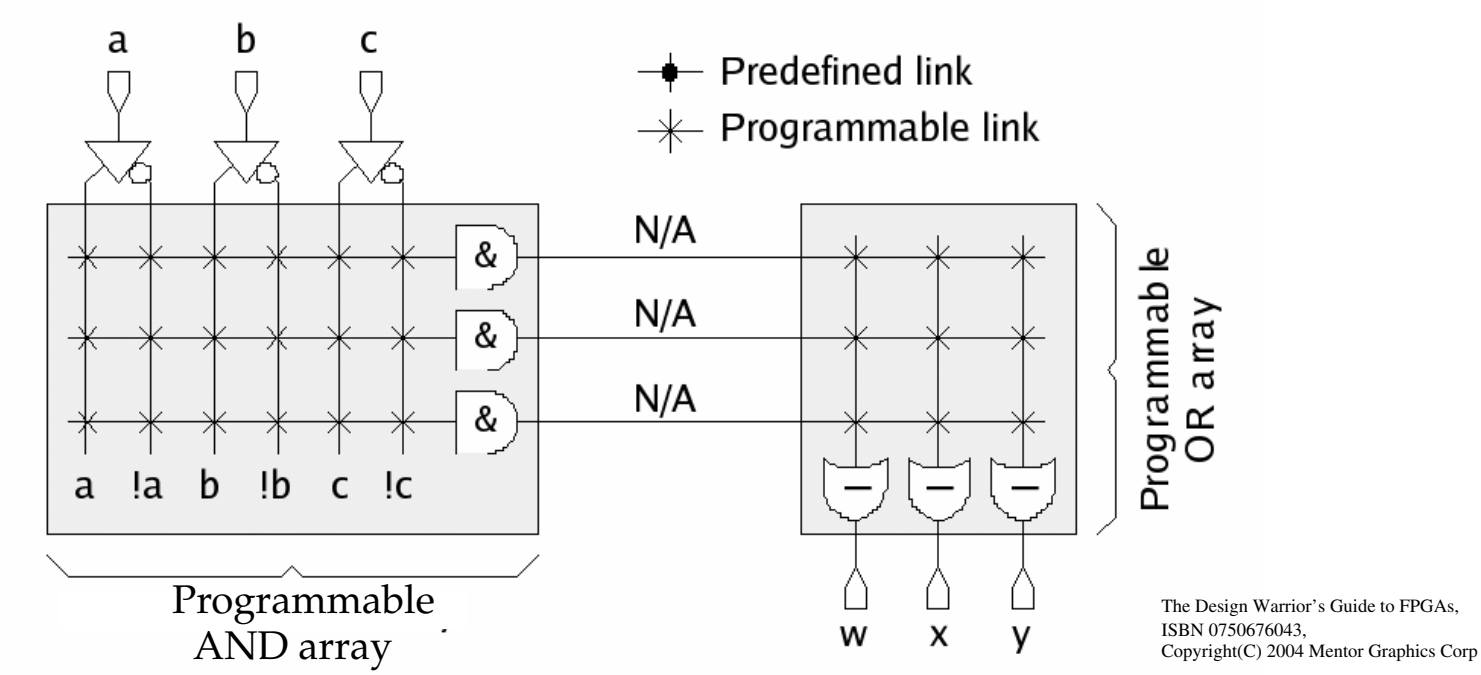
Programming these functions is a simple matter of choosing the correct links in the OR array.

NOTE: Real PROMs have significantly more inputs and outputs.

PLAs

An important limitation of PROM is that the AND plane produces all products whether they are used or not -- this limits the number of inputs.

Programmable Logic Arrays (PLAs) allowed both the AND and OR plane to be programmed.

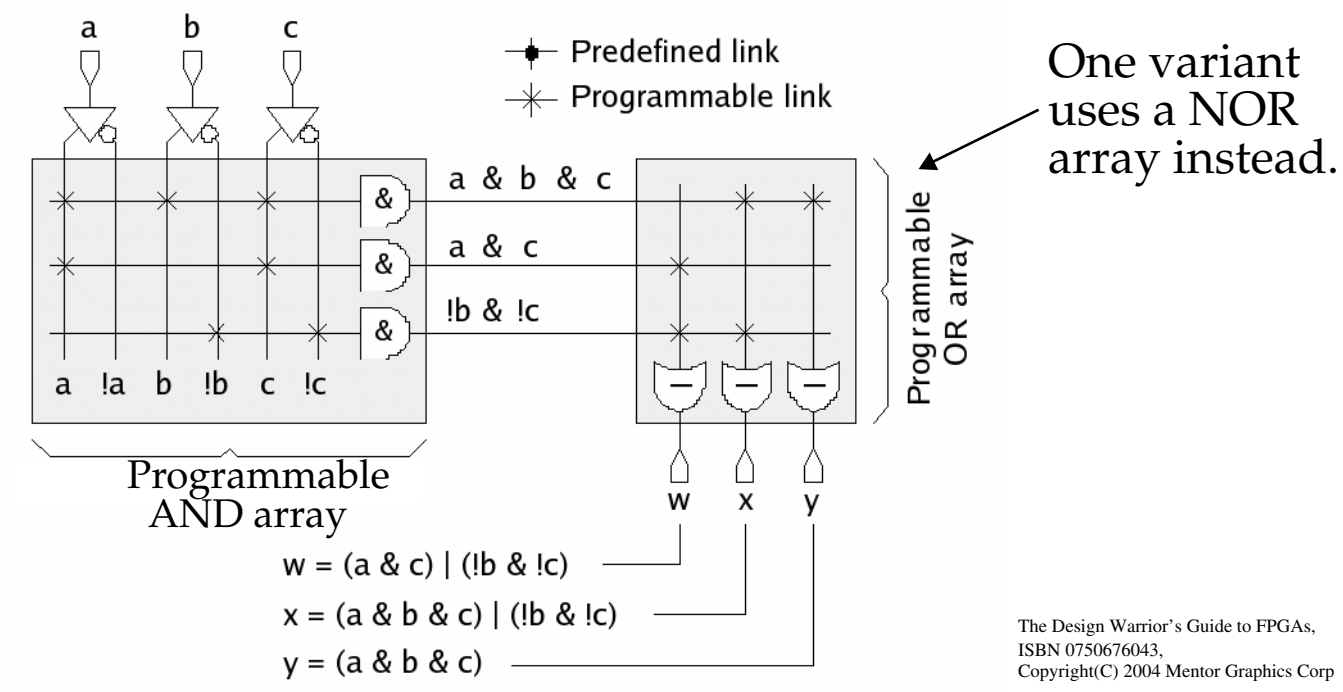


The Design Warrior's Guide to FPGAs,
ISBN 0750676043,
Copyright(C) 2004 Mentor Graphics Corp

Here the number of AND functions in the AND array in independent of the number of inputs to the device.

PLAs

The following example illustrates the implementation of 3 functions, *w*, *x* and *y*.



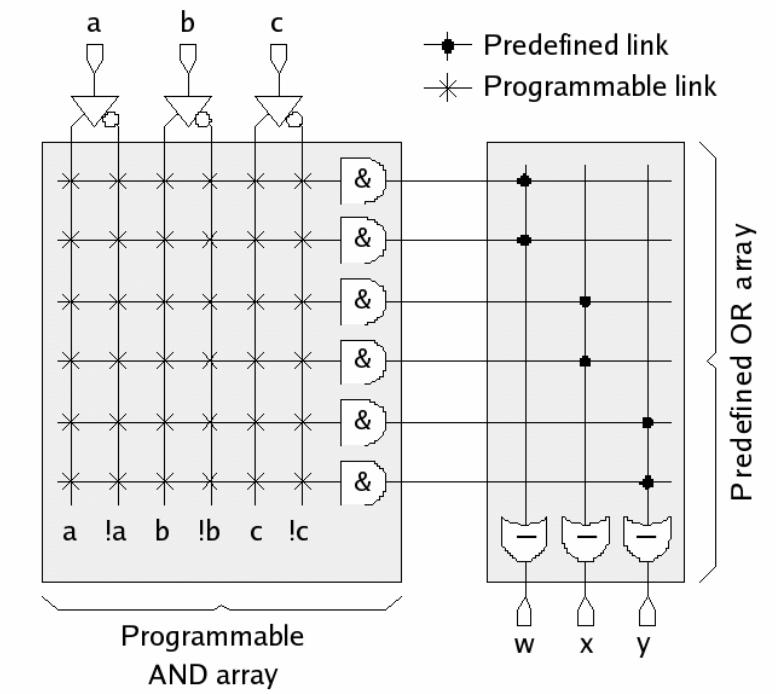
Note that *product terms* can be shared among output functions.

The programmable links slow signals -- thus PLAs are slower then PROMs

PLAs never achieved any significant level of market presence.

PALs

Programmable Array Logic (PALs) were introduced in late 70's to address speed problem of PLAs.



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

Here, the AND array is programmable and the OR array is predefined, therefore they are faster than PLAs.

However, PALs only allow a restricted number of product terms to be OR'ed, at least on chip.

PALs

Real devices have many more inputs and outputs plus a variety of options available including:

- The ability to invert the outputs
- The ability to *tristate* the outputs
- The ability to latch the outputs
- The ability to configure certain pins as input or output.

CPLDs

In '84, Altera introduced a CPLD based on a combination of CMOS and EPROM technologies.

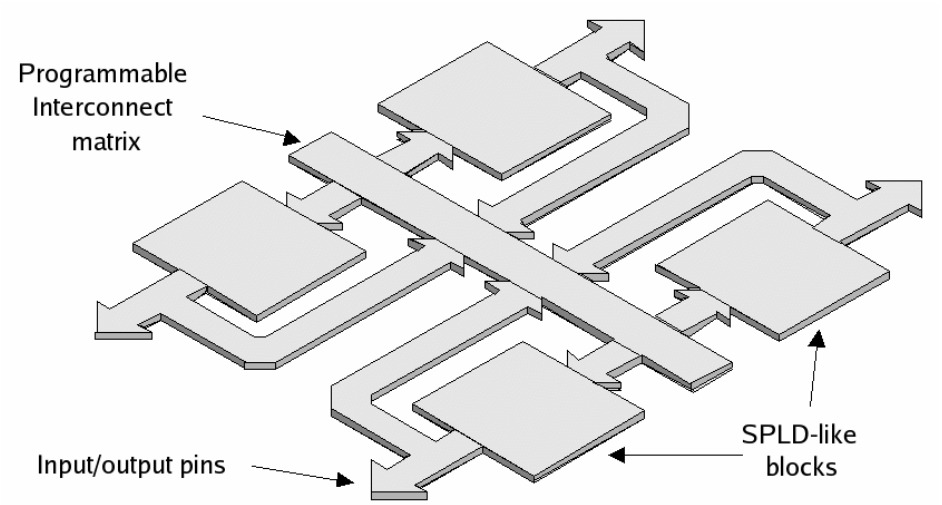
CMOS allowed low power and high density while EPROM enabled these devices to be used for development and prototyping.

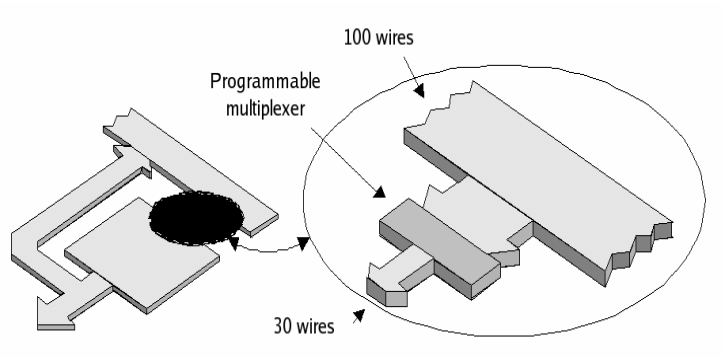
Altera's real contribution was to use an interconnection array **with less than 100% connectivity**.

This increased complexity of software but keep the device scalable in terms of speed, power and cost.

CPLDs

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





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

Both the SPLDs 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.

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.

ASIC (gate array, etc.)

Four main classes exist today, in order of increasing complexity:

- Gate arrays
- Structured ASICs
- Standard cell devices
- Full-custom chips

In order of appearance.

Full-custom

In the early days, only two classes of chips existed.

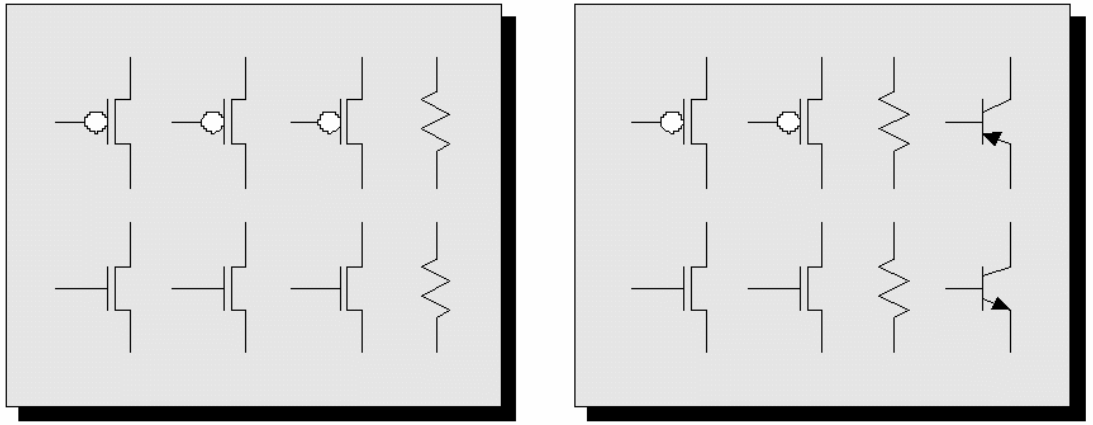
- Standard off-the-shelf components
- Full-custom ASICs (such as microprocessors)

For the full-custom class, nothing is predefined, not even standard logic gates.

All wires and gates are hand-crafted individually, and optimized for speed, area and power.

ASIC (gate array, etc.)

Gate arrays are based on the idea of a *basic cell* consisting of a collection of unconnected transistors and resistors.



(a) Pure CMOS basic cell (b) BiCMOS basic cell

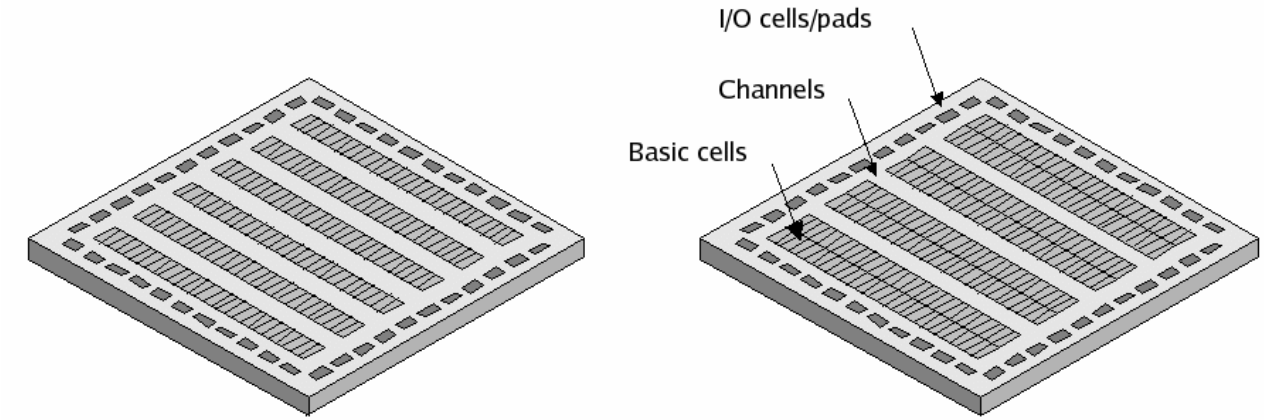
The ASIC vendor *prefabricates* silicon chips containing arrays of these **basic cells**.

Channels are typically provided between rows or columns of these arrays for routing.

Alternative, *sea-of-gates* arrays do not have routing channels.

The vendor also provides a *cell library* (a set of basic logic gates, MUXs, etc).

ASIC (gate array, etc.)



Designers use the latter and generate a *netlist*.

Special mapping, placement and routing CAD tools are used to assign the logic gates to *basic cells* and to design the interconnect.

The output of this process are *photo-masks* that define the metalization layers.

Since the transistors and other components are **prefabricated**, there is a considerable cost savings.

The main drawback is *resource under-utilization* and *less-than-optimal routing* (because of routing constraints).

Standard Cell Devices

Became available in the early '80s to address the problems with gate arrays.

Similar to gate arrays, the ASIC vendor defines the *cell library*.

The vendor supplies *hard-macro* and *soft-macro* libraries, which include elements such as processors, comm. functions, RAM, ROM, etc.

Designers also have the option of purchasing blocks of **intellectual property** (IP).

Designers generate a *gate-level netlist* using CAD tools (similar to gate arrays) but none of the components are prefabricated.

P&R tools *place* and *route* each of the gates and optimize for area and delay.

The standard cells themselves are designed as *constant height cells*, which simplifies their placement (PWR and GND are connected via abutment).

The output of the process is a *complete* set of photomasks.

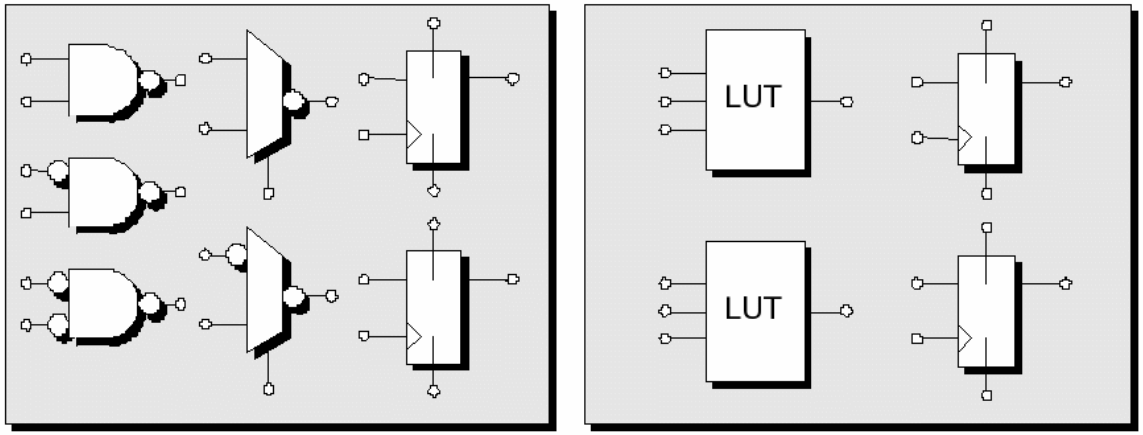
Structured ASICs

Entered the scene in the early '90s but were not accepted until '03.

ASIC manufacturers were looking for ways to reduce ASIC *design costs* and *development times*, without reverting to gate arrays.

The basic element is called a *module* or *tile*.

It contains a mixture of prefabricated generic logic (implemented as gates, MUXs or lookup tables), registers and some local RAM.



(a) Gate, mux, and flop-based (b) LUT and flop-based

An array (sea) of these elements are prefabricated.

Structured ASICs

Peripheral elements are also prefabricated, and include RAM blocks, clock generators, boundary scan logic, etc.

Similar to gate arrays, the chip can be customized using only metalization layers.

Since structured ASIC tiles are more complex than gate arrays, *most* of the metalization layers are predefined.

Most require only 2 or 3 layers to be customized, in the extreme, one requires the definition of only a single *via* layer.

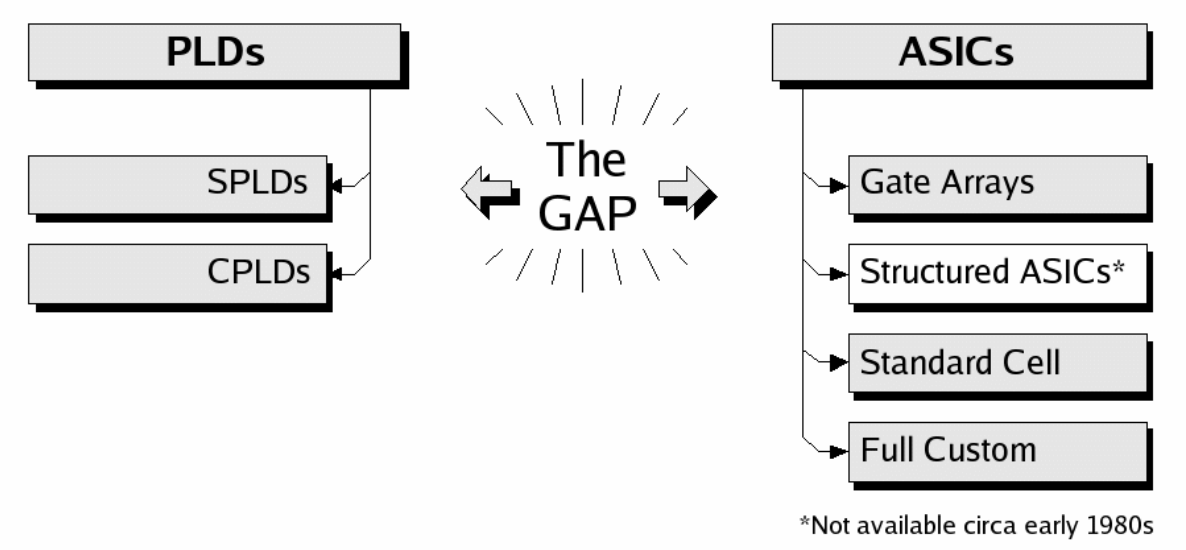
This saves considerable time and cost since only a couple photo-masks need to be designed.

The disadvantages include about **3X** more area and **2-3X** more power, when compared with a standard cell chip.

FPGAs

In early '80s, a gap emerged in the digital IC continuum.

- At one end, SPLDs and CPLDs provided high configurability, fast design and modification times, but supported only small to moderate functions.
- At the other end, ASICs supported large complex designs but were immutable once fabricated, expensive and time-consuming to design.



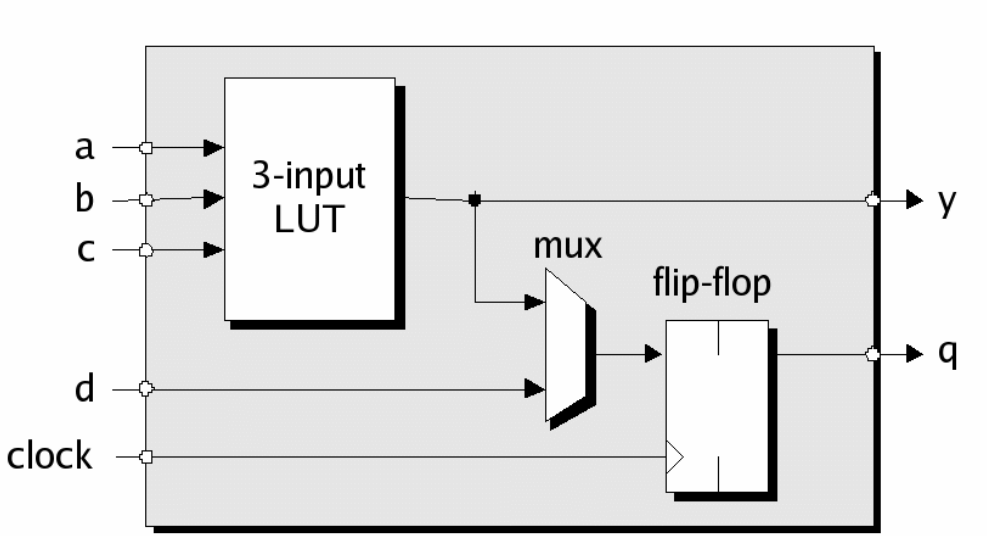
The Design Warrior's Guide to FPGAs,
ISBN 0750676043,
Copyright(C) 2004 Mentor Graphics Corp

Xilinx developed and made available in '84 a new class of IC called the **FPGA** to fill the gap.

FPGAs

The first FPGAs were based on CMOS and used SRAM cells for configuration.

The early chips used an array of **programmable logic blocks** (PLBs), which comprised a 3-input *lookup table* (LUT), a register and a MUX.



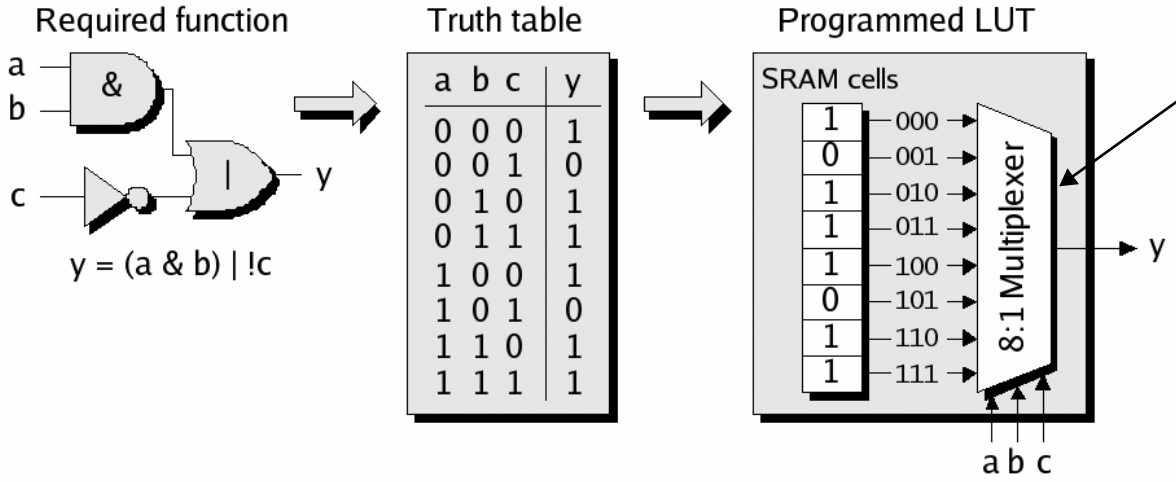
Each PLB can be programmed individually to perform an unique function.

The FF can be triggered by a positive or negative-going clk.

The MUX allows selection of the LUT output or an external input.

FPGAs

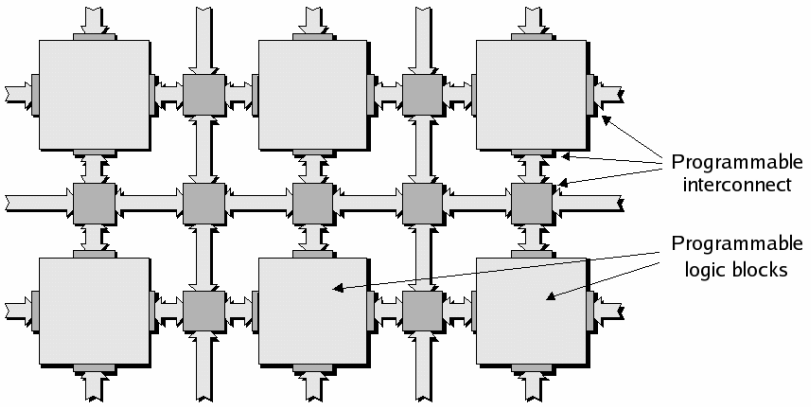
The LUT can implement any 3-input logic function.



(Actual implementation does not use a MUX here -- more on this later).

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

The FPGA architecture consisted of a 2-dimensional array of PLBs separated by a *sea* of programmable interconect.



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

FPGAs

Todays FPGAs are much more complex (to be discussed).

For example, in addition to the local interconnect, an FPGA typically has a **global** (high-speed) **interconnection** network.

This allows signals to cross the chip without having to pass through local switching elements.

FPGA-ASIC hybrids

Although it doesn't make sense to embed an ASIC inside an FPGA, it is meaningful in the other direction, i.e., embedded FPGA cores.

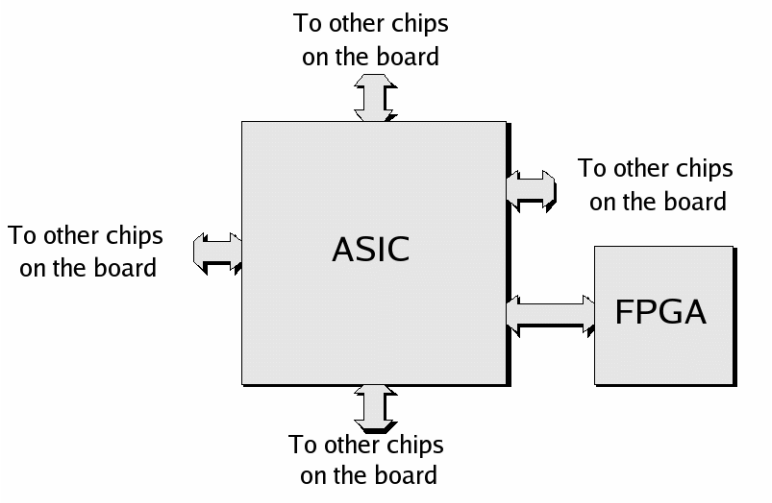
This is useful for *platform design*, a term used at the board level to refer to a design from which **multiple products** can be derived.

Here, the *platform* is the ASIC and the embedded FPGA allows it to be customized for a specific application.

FPGA-ASIC hybrids

Another driver is provided by the increasing number of incidences of FPGAs being used to augment ASIC designs.

This has traditionally been accommodated at the board level.



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

Here, the designers of the ASIC *off-loads* to the FPGA any part of the ASIC design that is subject to modifications or enhancements.

Board level realization incurs a performance penalty because signals must travel off-chip.

Embedding the FPGA solves this problem.

FPGA-ASIC hybrids

Designing these hybrids however is challenging because ASIC and FPGA design tools/flows are significantly different underneath.

ASIC are **fine-grained** because they are implemented at the *primitive logic gate* level.

FPGAs are **medium-grained** (or coarse-grained according to others) because they are realized using higher-level blocks (PLBs).

The CAD tools that do the synthesis and P&R need a consistent view (fine-grained or medium grained) of the world in order to do a good job.

Structured ASIC and FPGAs however are both medium-grained and therefore can enjoy a unified tool and design flow.

That is, the same block-based synthesis and P&R engines can be used for both the ASIC and FPGA portions.