Semic Custom Logic Design

Dinesh Sharma

EE Department
IIT Bombay, Mumbai

August 27, 2015

Regularity in VLSI Design

If one looks at a complex VLSI design, it basically consists of active devices and interconnects.
Accordingly, we can classify these designs in 4 classes:

Devices $\rightarrow$ $\downarrow$ Interconnects	Regular	Irregular
Regular	Memories	PLAs
Irregular	Sea of Gates	Processors

- A completely irregular part like a processor requires a huge design effort. Our choices are therefore limited to the use of processor families which are already designed and available.
- The other combinations can be made field programmable. It is the designer who programs these devices to achieve the desired result.

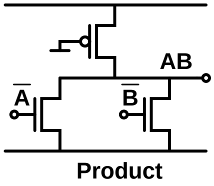
Re-configurable Logic with Pseudo NMOS gates

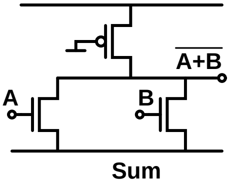
We want a regular circuit configuration which can be re-configured to implement a general sum of products.

- It is inconvenient to re-configure CMOS logic as configuration of the p channel pull up network as well as that of the n channel pull down network has to be changed.
- Pseudo NMOS gates are more suitable for re-configuration as the pull up is just a single grounded gate PMOS.
- But Pseudo NMOS circuits are ratioed and the geometry of series connected devices depends on the number of devices in series and hence, on the logic.
- How do we implement the product function then?

Pseudo NMOS NOR for Products as well as sums

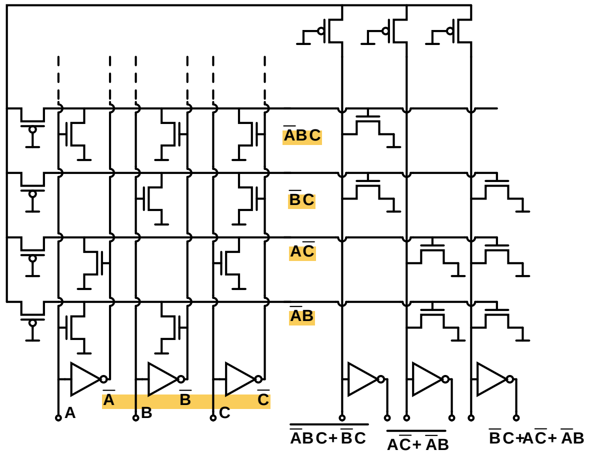
- We use the expression : $A \cdot B = \overline{\overline{A} + \overline{B}}$.
- Now the product of A and B can be implemented as the NOR of $\overline{A}$ and $\overline{B}$ which does not use series connected transistors.





By adding inverters at the input and output as required, we can implement products or sums using only NOR type of circuits, which use constant geometry NMOS transistors in parallel.

Programmable Logic Arrays


$\overline{A}BC$
$\overline{B}C$
$A\overline{C}$
$\overline{A}B$
A
$\overline{A}$
B
$\overline{B}$
C
$\overline{C}$
$\overline{\overline{A}BC+\overline{B}C}$
$\overline{A\overline{C}+\overline{A}B}$
$\overline{B}C+A\overline{C}+\overline{A}B$

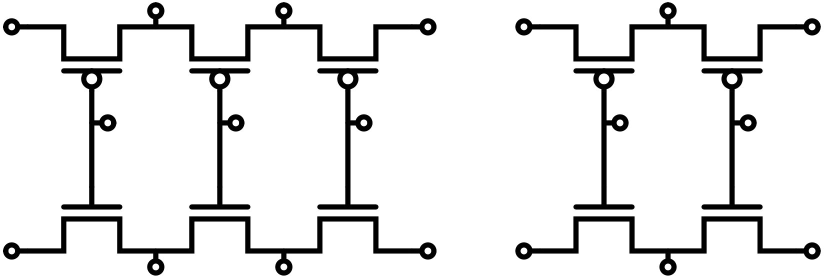
the first half of the circuit generates the min terms

second half i to add the min terms

all the nmos are in parallel pulled up by a single pmos

Sea of Gates

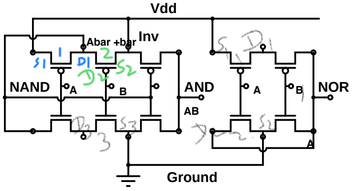
In this style of design, all transistors are pre-placed.
Interconnects determine what kind of logic will be implemented.



Both n and p channel transistors are in series here! How do we construct regular CMOS logic gates using these?

Logic from Sea of Gates

Actually, by wiring the apparently series connected devices, we can convert them to series or parallel as required.



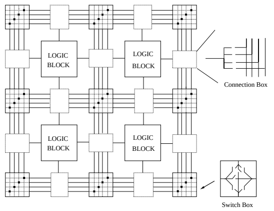
The example above shows a NAND gate, whose output is fed to an inverter to form the AND function. The structure on the right forms a NOR gate.

Using Memories as Logic

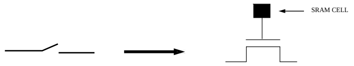
- A logic function can be represented by a truth table. This can be stored in memory. Inputs to the logic function act as address pins.
- Precomputed value of the logic function are stored at the address represented by its input values.
- For any input combination, the address represented by these is looked up and the stored value presented as the output.
- For example, any logic function of 5 inputs can be implemented by a 32×1 bit memory.
- If the same 32 bit memory is organized as 16×2, it can store two logic functions of up to 4 inputs.

FPGA Architectures

A field Programmable array allows the logic functions as well as the interconnect to be programmed.



(a) Construction of Connection Box and Switch Box using Switches

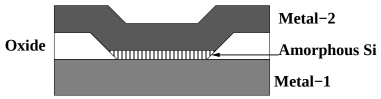


(b) Realization of Switches using NMOS Pass Transistors

Transistors driven by SRAM can be used to program interconnects.

Antifuse Technology

Interconnects may also be programmed by “antifuse” structures.



Initially, the amorphous silicon is insulating.

A pulse of high current will melt amorphous silicon and make it conducting, connecting Metal-1 to Metal-2.

This kind of programming is irreversible.

However, it offers a resistance much lower than the typical “on” resistance of a transistor.

Current FPGA Architecture

Apart from logic blocks and interconnect fabric, modern FPGA's contain other useful structures, such as

- Block RAM,
- Processor cores (Power PC on Xilinx Vertex family),
- Fast multipliers
- I-O Blocks

Reconfigurable logic

We have talked of four classes of electronic components based on the regularity and repetitiveness of placing devices and interconnects. How are these re-configured?

- Regular device placement, Regular interconnects: Memories:
  Programmed by storing pre-computed values.
- Semi regular structures: PLAs, CPLDs and FPGAs: Programmed by loading a memory at power on, or one time programmability through antifuse technology.
- Non repetitive devices and interconnects: Processors: Software programmed.

Types of Reconfigurable Logic

- A typical board, used for implementing a variety of applications will have a micro-controller and a few programmable devices for providing dedicated functions and glue logic.
- This is typically configured once at power on, and then left alone to perform its functions. This is called “static re-configurability”.
- But we may want to re-cofigure a structure while it is in operation! For example, one can imagine the design of digital filter, which adapts itself during operation itself depending on inputs. This kind of re-configurability is called “dynamic re-configurability”.
- Obviously, dynamic configurability requires that the dead time during re-configuration should be minimized.

Fine and Coarse Grain Reconfigurability

- There is a trade-off between flexibility of re-configuration and the time taken to re-configure.
- In fine grain re-configuration, we can re-program every gate of the design. This is the case for current FPGA and CPLD devices. This gives very good design flexibility, but takes a long time to re-configure.
- In coarse grain re-configuration, relatively large functional blocks are re-configured. For example, by re-connecting a collection of shifters and adders, we can generate any combination of multipliers and adders. The effort to re-configure is smaller, because we do not alter the inner design of shifters and adders. This is compatible with dynamic re-configuration.

Use of Dynamic Re-configurability

Using coarse grain re-configurability, it becomes possible to dynamically change a circuit, *during* its operation.
This has obvious advantages in adaptive circuits.
We normally do not want to re-configure the whole circuit.
Indeed the control signals for doing the re-configuration will be generated by a circuit which remains unchanged.
Therefore dynamic re-configurability requires circuits which can be partially re-programmed. Many FPGA are beginning to promise this feature.