Programmable Logic

Antoine Lavault[1][2]

[1] Apeira Technologies

[2] UMR CNRS 9912 STMS, IRCAM, Sorbonne Université

January 19, 2024

Contents

Contents

ROM vs programmable logic

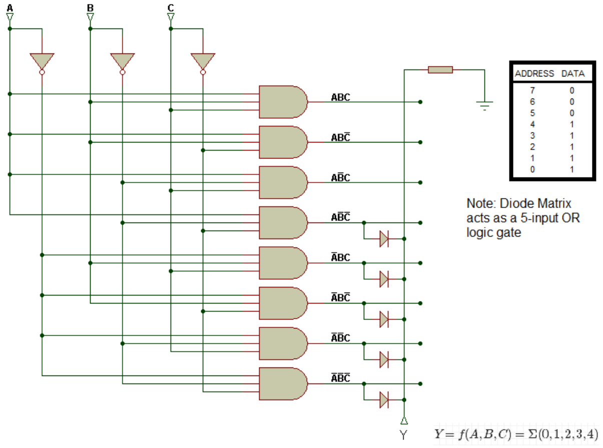
A long time ago...

- Discrete logic has its limitations when it comes to complex circuits
- ROMs (PROM in particular) could be useful for these functions, but they are, in general, too big to be useful
- Programmable Logic Arrays (1970) and Programmable Array Logic (1978) were a solution as they could be programmed at the manufacturing plant or directly on location.
- Their evolution was the GAL (Gate Array Logic), also programmable on location.

Rom vs PLA

- A ROM and PLA are very similar, BUT...
- A PLA does not implement all the products, unlike a ROM

ROM vs PLA

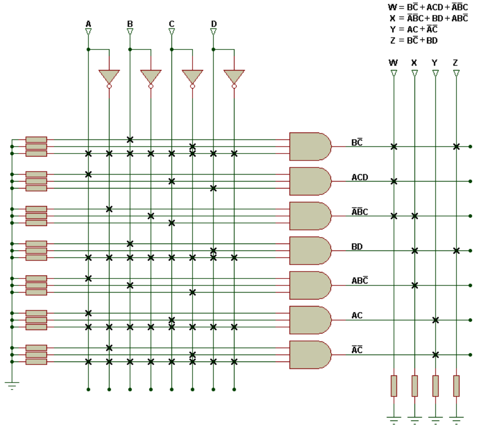


ADDRESS	DATA
7	0
6	0
5	0
4	1
3	1
2	1
1	1
0	1

$$Y = f(A,B,C) = \Sigma(0,1,2,3,4)$$

Figure: ROM

ROM vs PLA



$W = B\overline{C} + ACD + \overline{A}\overline{B}C$
$X = \overline{A}\overline{B}C + BD + AB\overline{C}$
$Y = AC + \overline{A}\overline{C}$
$Z = B\overline{C} + BD$
A
B
C
D
W
X
Y
Z
$B\overline{C}$
ACD
$\overline{A}\overline{B}C$
BD
$AB\overline{C}$
AC
$\overline{A}\overline{C}$

Figure: PLA

Like the good old days - PAL

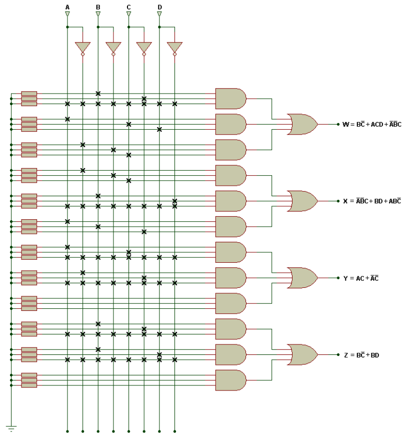
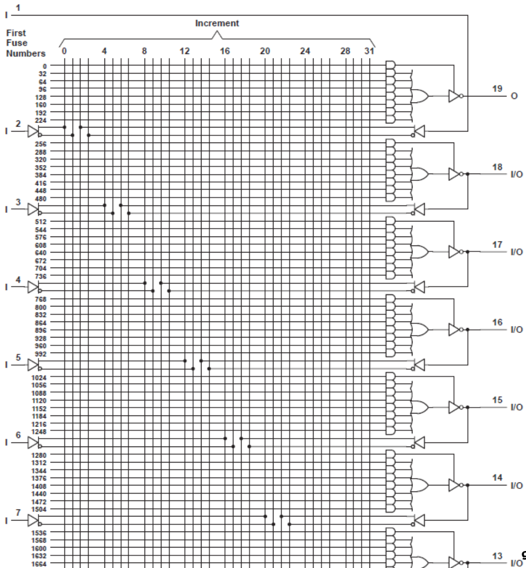

A
B
C
D
$W = B\overline{C} + ACD + \overline{A}\overline{B}C$
$X = \overline{A}\overline{B}C + BD + AB\overline{C}$
$Y = AC + \overline{A}\overline{C}$
$Z = B\overline{C} + BD$

Figure: PAL

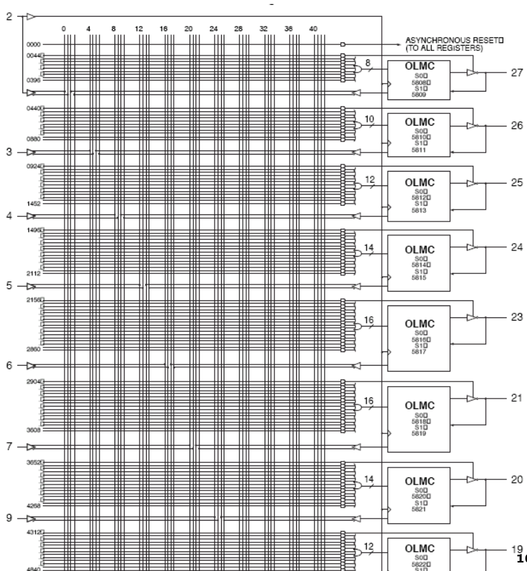
Programmable Array Logic

- In a PAL circuit, the AND network is programmable, but the OR network is fixed.
- Each intersection between a horizontal and vertical line is programmable
- AND gates have a default output of 0
- Output pins have a tristate buffer each
- Pop a fuse to make a connection
- Exemple : PAL16L8
- With output FF: PAL16L8R



Generic Array Logic

- By Lattice Semiconductors in 1985
- Can emulate a good amount of PAL
- Replaced a lot of discrete logic but is not manufactured anymore.



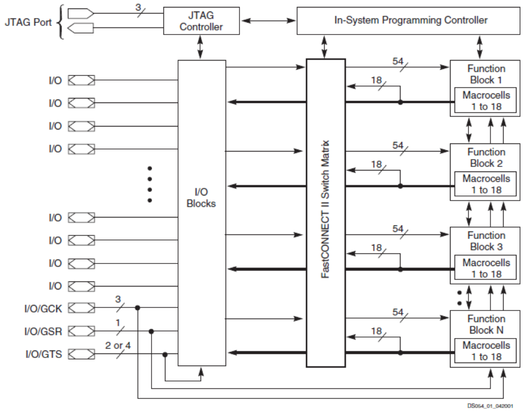
GAL Macro Cell

CPLD - Complex Programmable Logic Devices

- ROM, PLA, PAL, and GAL are often described as "simple programmable logic circuits."
- Complex Programmable Logic Devices are an extension of PAL
- A CPLD integrates several PAL on one die with an interconnection network
- This network allows connection of the physical input pins and different internal blocks

Xilink CPLD XC9500XL family

- Each block is a 54×18 PAL
- A macro-cell has a memory element
- A circuit holds from 2 to 16 of these blocks
- The interconnection network allows the connection of the physical input pins and different internal blocks



Contents

FPGA

- A FPGA is a user-programmable integrated circuit
- The first FPGAs were sold in the 80s. They are among the most complex integrated circuits when considering the number of transistors
- Two big manufacturers: Altera-Intel and Xilinx-AMD. Lattice and Achronix are other well known names.

FPGA

- Three major blocks inside of an FPGA
  - Configurable Logic Block (CLB)
  - interconnection network between CLBs
  - Input/Output Block (IOB): to interface with the outside

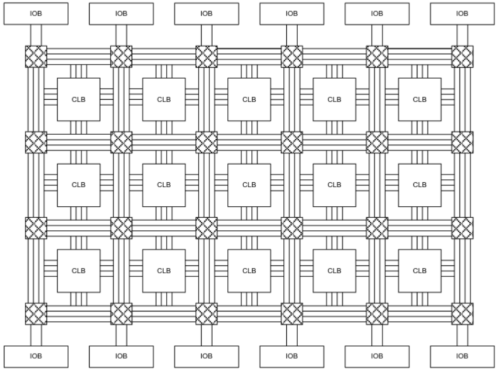
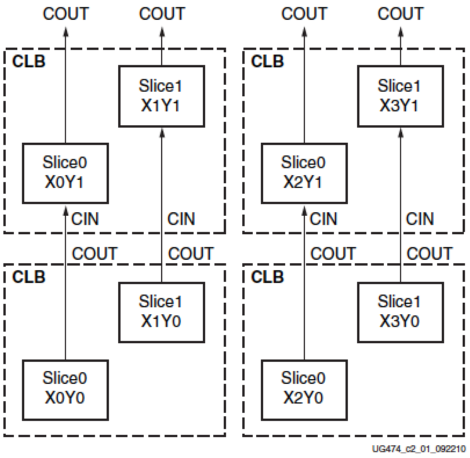


Figure: FPGA (general structure)

Xilink 7-series

- A CLB has two slices
- Each slice is connected to the interconnection net
- Slices are linked vertically



Logic Slice (SLICEL)

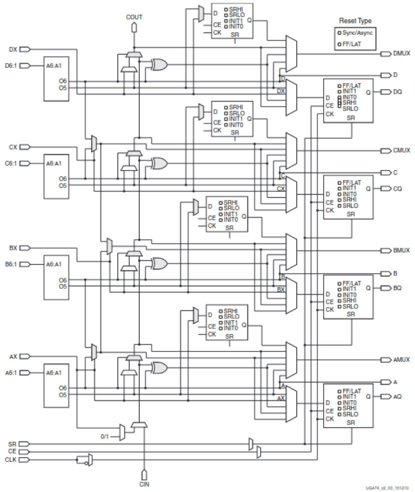


Figure: SLICEL Architecture

SLICEL

A slice is made of:

- 4 Look-up Tables (LUTs) with 6 inputs (A6:A1) and two outputs, O6 and O5. They can implement a logic function with 6 inputs or two 5-input logic functions.
- Three multiplexers (grayed out, vertical) to implement logic functions with 7 or 8 inputs.
- Logic gates for quick additions
- 8 memory elements:
  - 4 (in the center) are flip-flops
  - 4 (to the right) that can implement either flip-flops or latches
- Programable multiplexers (in white) for internal routing

There are also SLICEM slices that implement the same set of features but also add memory features like registers.

DSP Block (DSP48E1)

For fast and efficient arithmetic operations, FPGAs have embedded specialized structures for computation tasks. On the Xilinx 7-Series, it is called the DSP48.
Such specialized blocks are numbered (240 on the FPGA embedded on the Nexys 4).

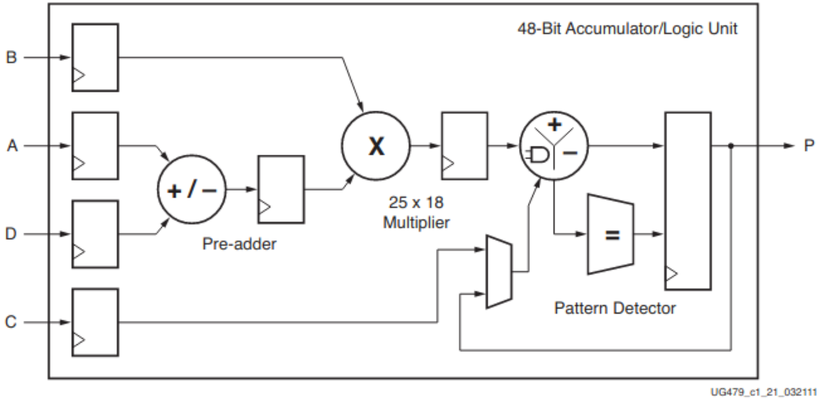


Figure: DSP48

DSP48

Multiplier

- A two's complement 25×18 multiplier. N
- For small numbers (e.g., 4×4), the multiplication is, in general, implemented through LUTs
- For bigger numbers, it is possible to combine several DSP48 multipliers!

Accumulator

- Multiplication-Accumulation (MAC): $S = S + A * B$
- Very common in DSP: convolution, dot product...

Contents

What is synthesis?

- Synthesis is converting HDL code into an implementable circuit.
- Done in several to optimize the structure
- The output of the synthesis process is called a netlist: a list of basic components and their connections.
- Follows a standard: 1076.6-2004 - IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis

Some guidelines for Vivado:
`https://docs.xilinx.com/v/u/2019.2-English/ug901-vivado-synthesis`

Types in VHDL cont.

VHDL is a strongly typed language, but not all types are synthesizable. Also, some "good" practices.

- Logic values and vector should use `std_logic` and `std_logic_vector`
- For numeric values, import the `numeric_std` package and use `signed` and `unsigned`.
- For signals, types like `integer`, `string`, or `char` are easier for the programmer to interpret but are, in general, optimized during the synthesis process, making them harder to debug.
- For constant and parameters, `integer`, `string` or `real` can be used to infer some values. Remember that `real` is not synthesizable, which will reduce its use for simulation (more on floating point numbers and FPGAs later).

Loops and conditions

Loops:

- Inside a process, loops, and conditions are useful to describe the behavior of a circuit.
- A loop is a compact way to represent several linked instructions
- To be synthesizable, the loop parameters must be static

Conditions:

- the `case` statement: useful to describe mutually exclusive choices. It corresponds (almost exactly) to the behavior of a multiplexer.
- the `if-elsif-else` statement: prioritize certain conditions. The synthesized circuit can become more complex than anticipated because the restrictions could be stronger than what the engineer had in mind
- In general: unless you need priority, use case statements.

An example with loops and priority

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY unsigned2dec IS
    PORT (
        number : IN unsigned(9 downto 0);
        hundred_BCD, ten_BCD, one_BCD : OUT unsigned(3 downto 0);
        error: OUT STD_LOGIC
    );
END unsigned2dec;
ARCHITECTURE arch OF unsigned2dec IS
BEGIN
    error <= '1' WHEN number >= 1000 ELSE
        '0';
    PROCESS (number)
        VARIABLE n, c, d, u : NATURAL := 0;
    BEGIN
        n := to_integer(number);
        c := 0;
        FOR hundreds IN 9 DOWNTO 1 LOOP
            IF n >= hundreds * 100 THEN
                c := hundreds;
                EXIT;
            END IF;
        END LOOP;
        n := n - c * 100;
        d := 0;
        FOR tenth IN 9 DOWNTO 1 LOOP
            IF n >= tenth * 10 THEN
                d := tenth;
                EXIT;
            END IF;
        END LOOP;
        u := n - d * 10;
        hundred_BCD <= to_unsigned(c, 4);
        ten_BCD <= to_unsigned(d, 4);
        one_BCD <= to_unsigned(u, 4);
```

Ambiguities

This is not synthesizable (dynamic loop)

```
ENTITY dyn_loop IS
    PORT (
        x0, xmax : IN INTEGER RANGE 0 TO 255;
        sum : OUT INTEGER RANGE 0 TO 65535
    );
END dyn_loop;
ARCHITECTURE arch OF dyn_loop IS
BEGIN
    PROCESS (x0, xmax)
        VARIABLE maxi : INTEGER;
    BEGIN
        maxi := 0;
        FOR k IN x0 TO xmax LOOP -- xmax is not static
            maxi := maxi + k;
        END LOOP;
        sum <= maxi;
    END PROCESS;
END arch;
```

Live Demo!

What will happen?!

Ambiguities

This is somewhat synthesizable (two sources for F) but not implementable

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY two_sources IS
    PORT (
        A, B, C, D, CLK: IN STD_LOGIC;
        F: OUT STD_LOGIC
    );
END two_sources;
ARCHITECTURE arch OF two_sources IS
BEGIN
    F <= CLK AND (C OR D); -- concurent domain
    -- also, it uses the clock signal in an assignment.
    -- It can be synthesized, but just don't.
    PROCESS (CLK)
    BEGIN
        IF (rising_edge(CLK)) THEN
            F <= A OR B; -- sequential domain
        END IF;
    END PROCESS;
```

Ambiguities

This is not synthesizable (two clocking conditions)

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY not_dff IS
    PORT (
        clk, D: IN STD_LOGIC;
        Q: OUT STD_LOGIC
    );
END not_dff;
ARCHITECTURE arch OF not_dff IS
BEGIN
    PROCESS (clk)
    BEGIN
        IF (rising_edge(CLK) OR falling_edge(CLK)) THEN
            Q <= D;
        END IF;
    END PROCESS;
END arch;
```

Ambiguities

This is synthesizable but is not a D flip-flop.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY not_dff IS
    PORT (
        clk, reset, enable, D: IN STD_LOGIC;
        Q: OUT STD_LOGIC
    );
END not_dff;
ARCHITECTURE arch OF not_dff IS
BEGIN
    PROCESS (clk, reset)
    BEGIN
        IF (rising_edge(CLK) AND reset = '0' AND enable = '1') THEN
            Q <= D;
        END IF;
    END PROCESS;
END arch;
```

Synthesizable DFF

This is synthesizable and is a D flip-flop (with synchronous clear and clock-enable signals). The component synthesized will be called FDRE.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY dff IS
    PORT (
        clr, ce, clk: IN STD_LOGIC;
        d : IN STD_LOGIC;
        q : OUT STD_LOGIC
    );
END ENTITY dff;
ARCHITECTURE rtl OF dff IS
BEGIN
    PROCESS (clk) IS
    BEGIN
        IF rising_edge(clk) THEN
            IF clr = '1' THEN
                q <= '0';
            ELSIF ce = '1' THEN
                q <= d;
            END IF;
        END IF;
    END PROCESS;
END ARCHITECTURE rtl;
```

How to avoid these ambiguities?

- It is possible for your HDL description to be simulated but not synthesized.
- What to look for?
    - Look for warnings from the synthesizer, especially "critical warnings." Consider them as errors.
    - For processes describing flip-flops, keep the sensitivity list of the process to `clk` and `rest`
    - For more complex processes, place all the signals part of an expression in the sensitivity list.

Contents

Conclusion

- Synthesis standard: IEEE 1076.6-2004
- Synthesis guide: `https://docs.xilinx.com/v/u/2019.2-English/ug901-vivado-synthesis`.
- HDL description can be ambiguous: while the synthesizer can throw errors quickly, sometimes, the results are far from what you expected.
- To write "good" (i.e., synthesizable without ambiguities), you should know the following:
  - The standard the synthesis tool is using
  - The idiosyncrasies of the synthesis process.
- As per usual, decomposing a complex system into basic interconnected blocks is a good way to avoid problems (i.e., playing with Lego bricks)