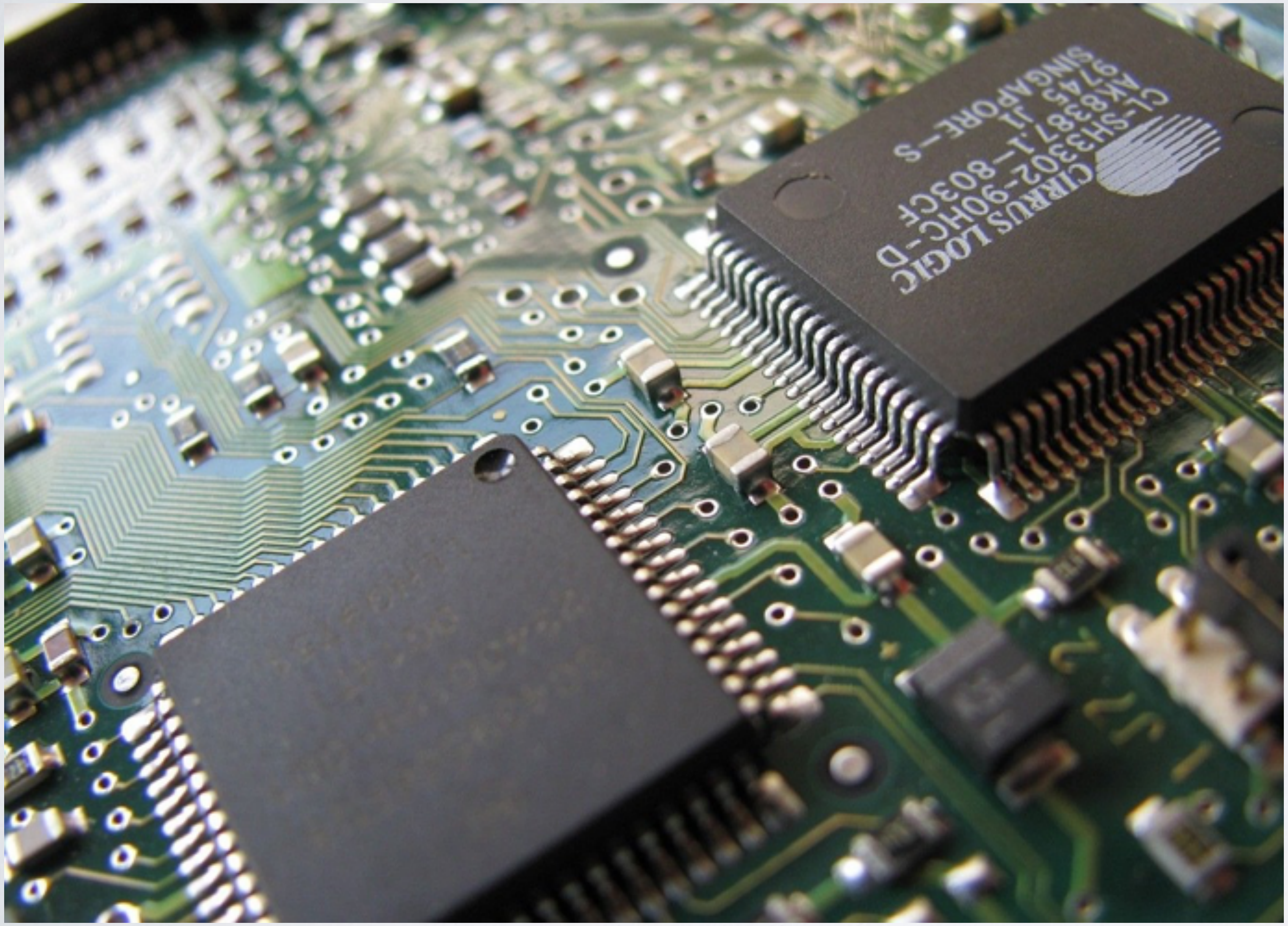
TUTORIAL
MY FIRST FPGA DESIGN

Tristan Gingold - tgingold@free.fr - FOSDEM'18

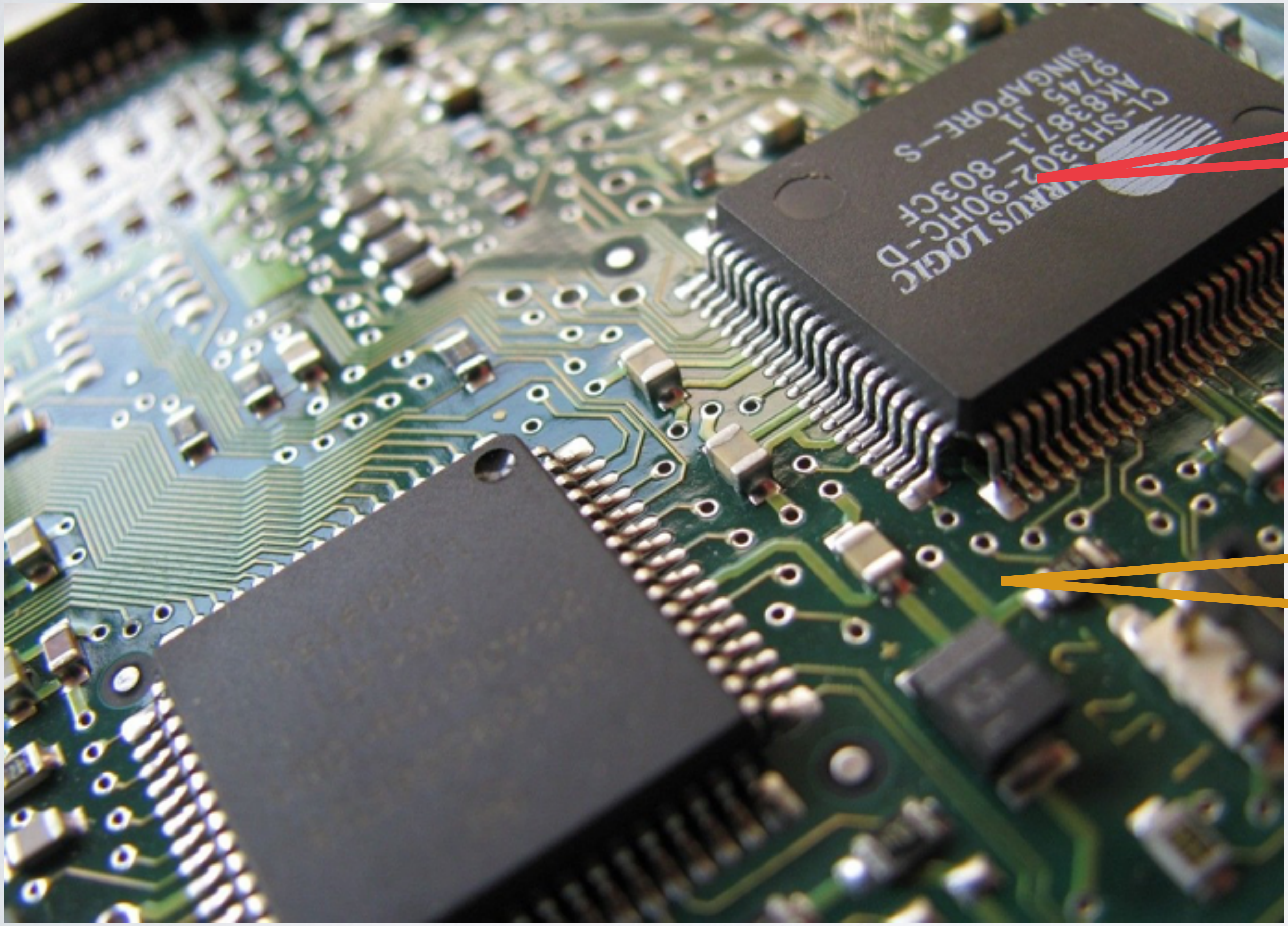
IT'S A TALK ABOUT HARDWARE!



Things like that…

There are many talks
at FOSDEM about software.
Try a different room

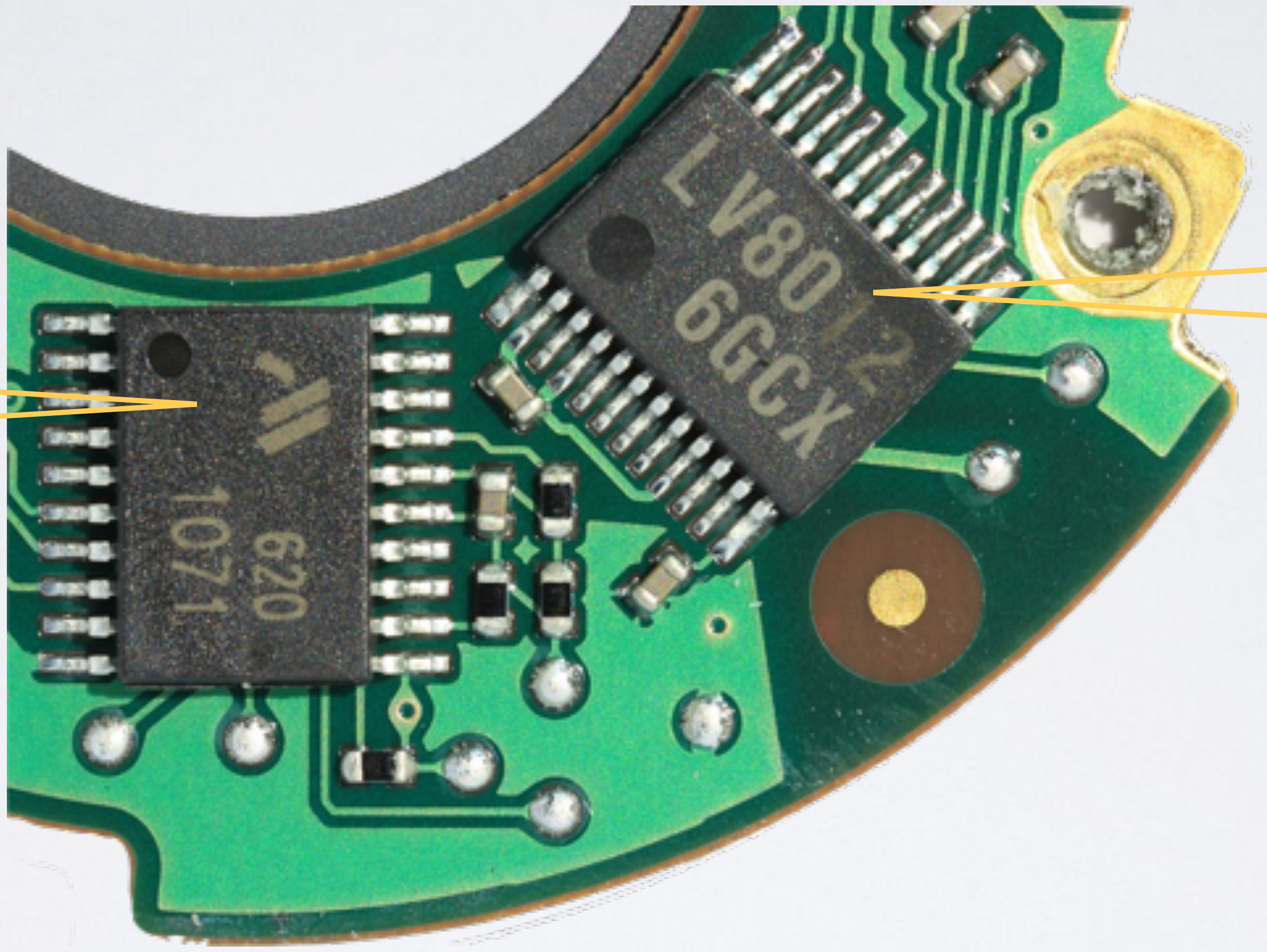
IT'S A TALK ABOUT CHIP DESIGN



This

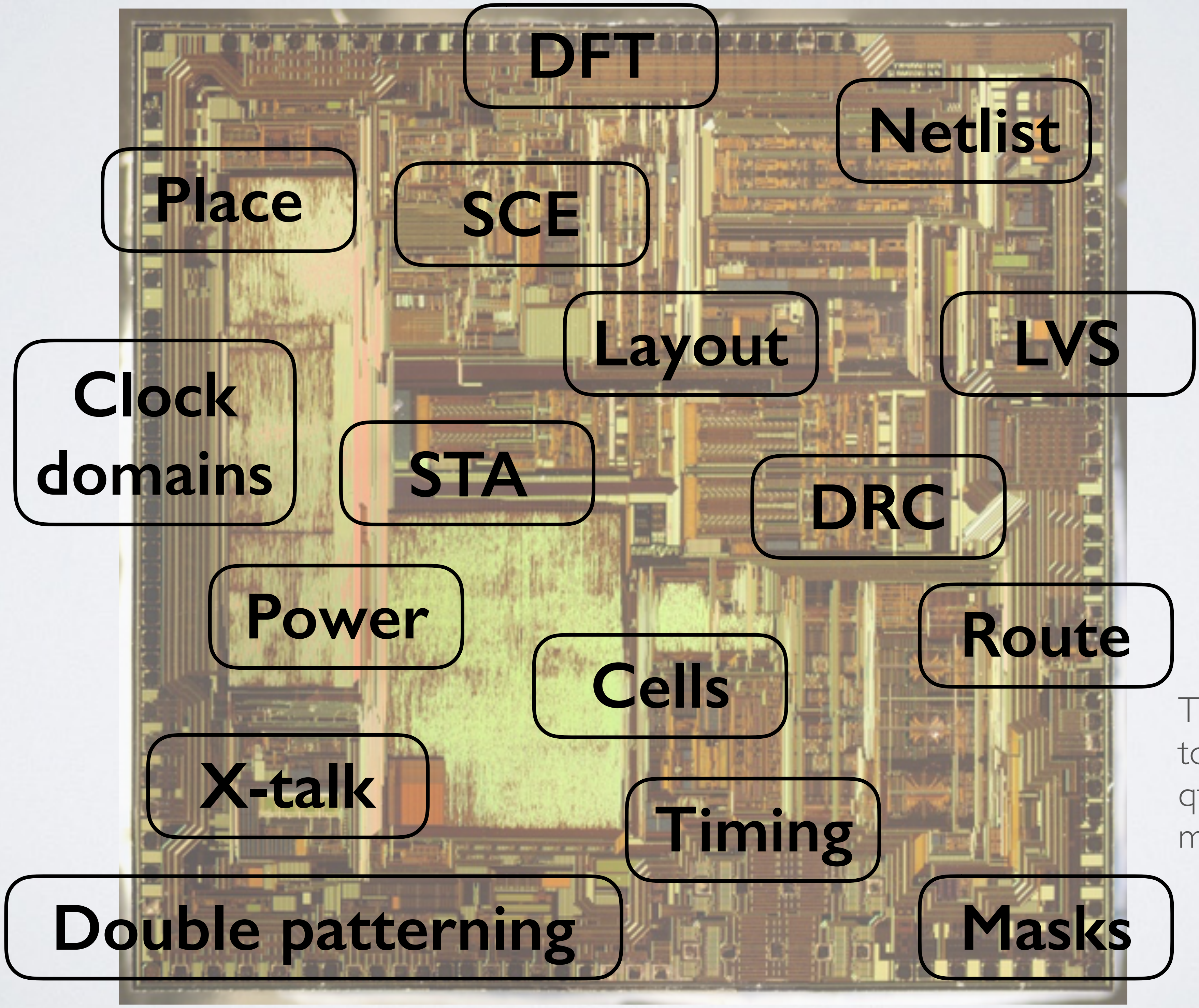
This is a PCB
(Printed Circuit Board)
KiCad is a tool to design boards, you also need electronic knowledge

MORE SPECIFICALLY, DIGITAL CHIPS



See the difference ?

DESIGNING AN IC IS COMPLEX…

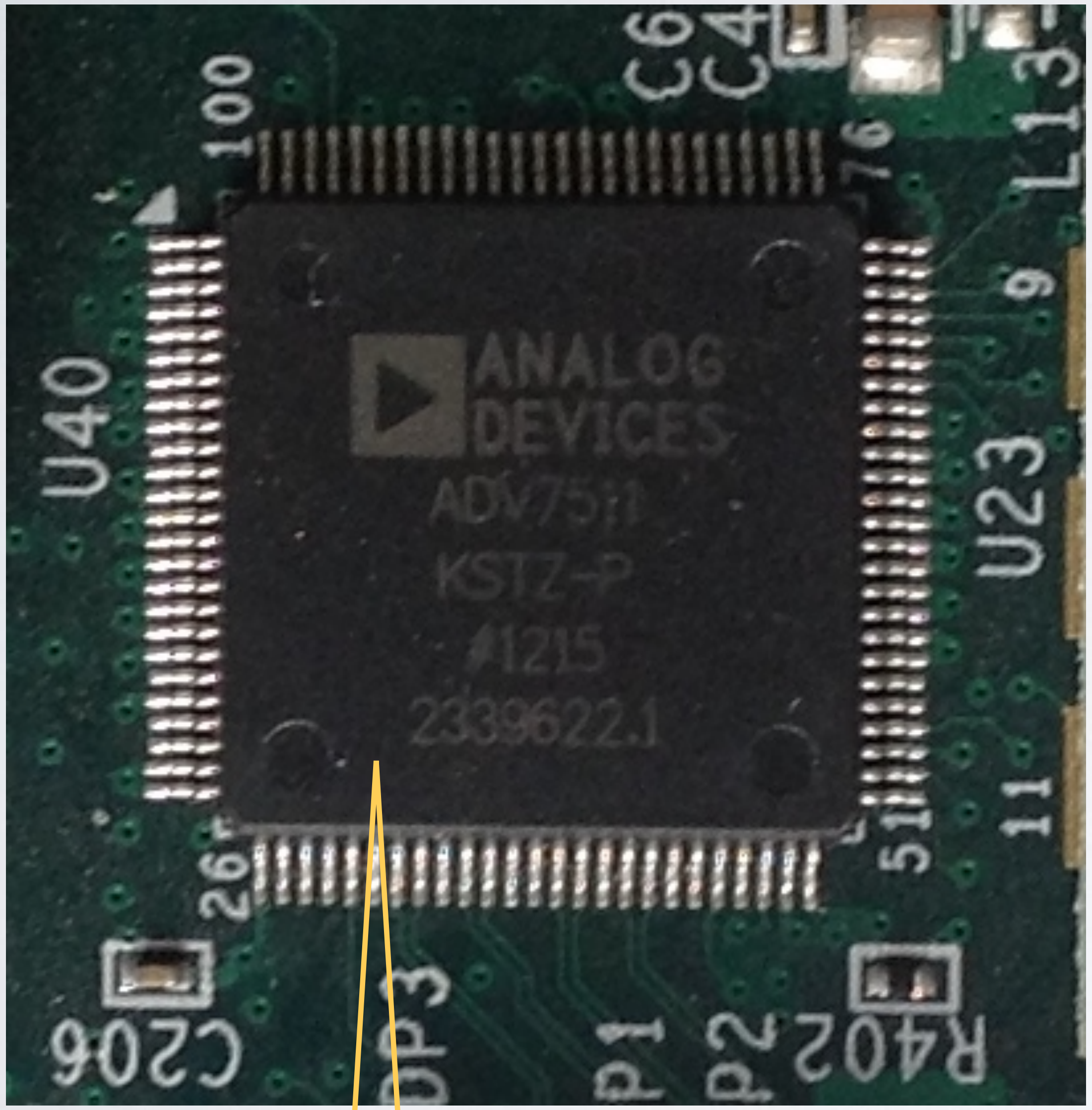


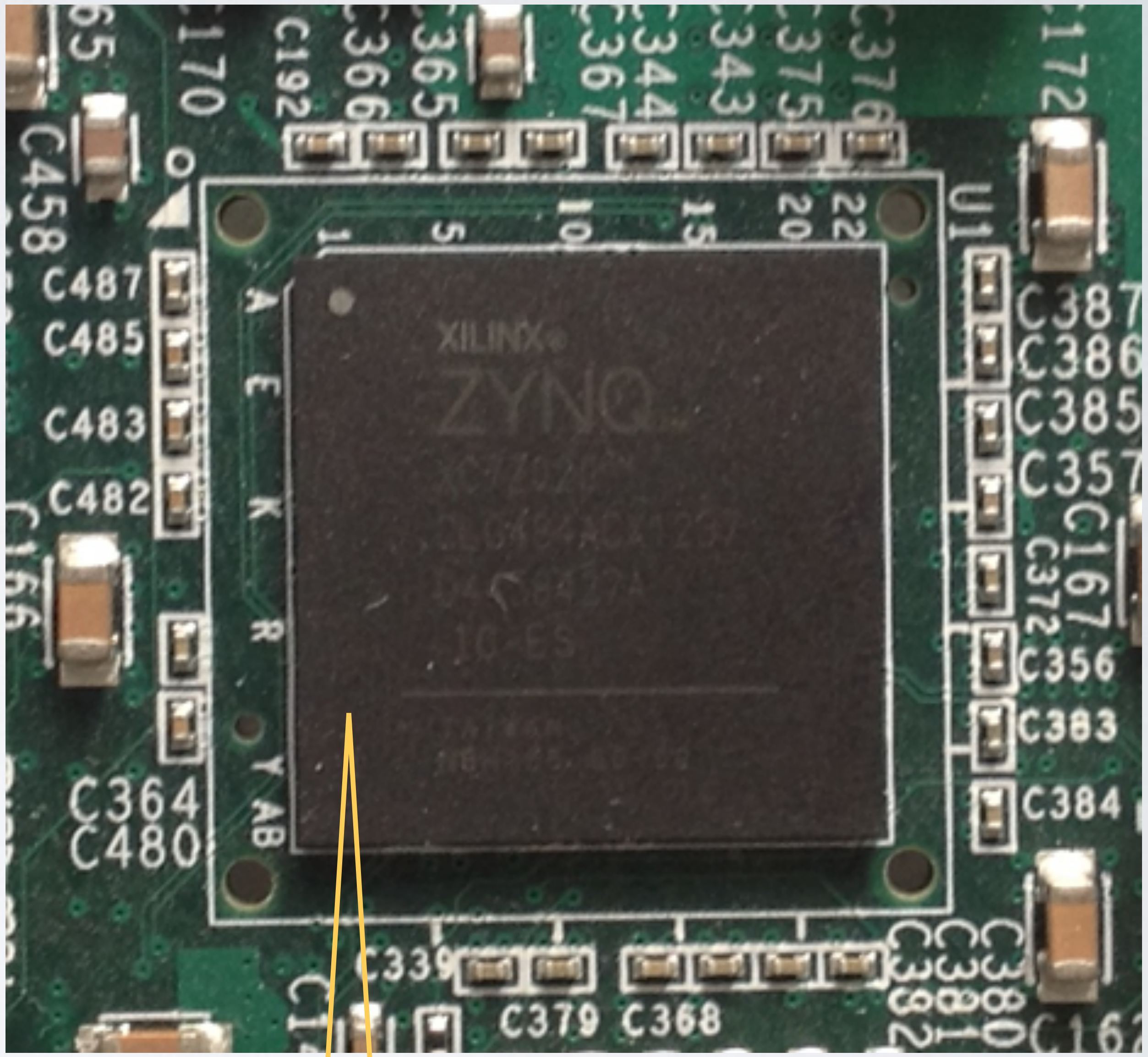
There aren't many OSS tools for ASICs.
qflow
magic VLSI

… AND VERY EXPENSIVE

ASML lithography machine
Expect $$$ for the first chip…

BUT SOME ARE PROGRAMMABLE!



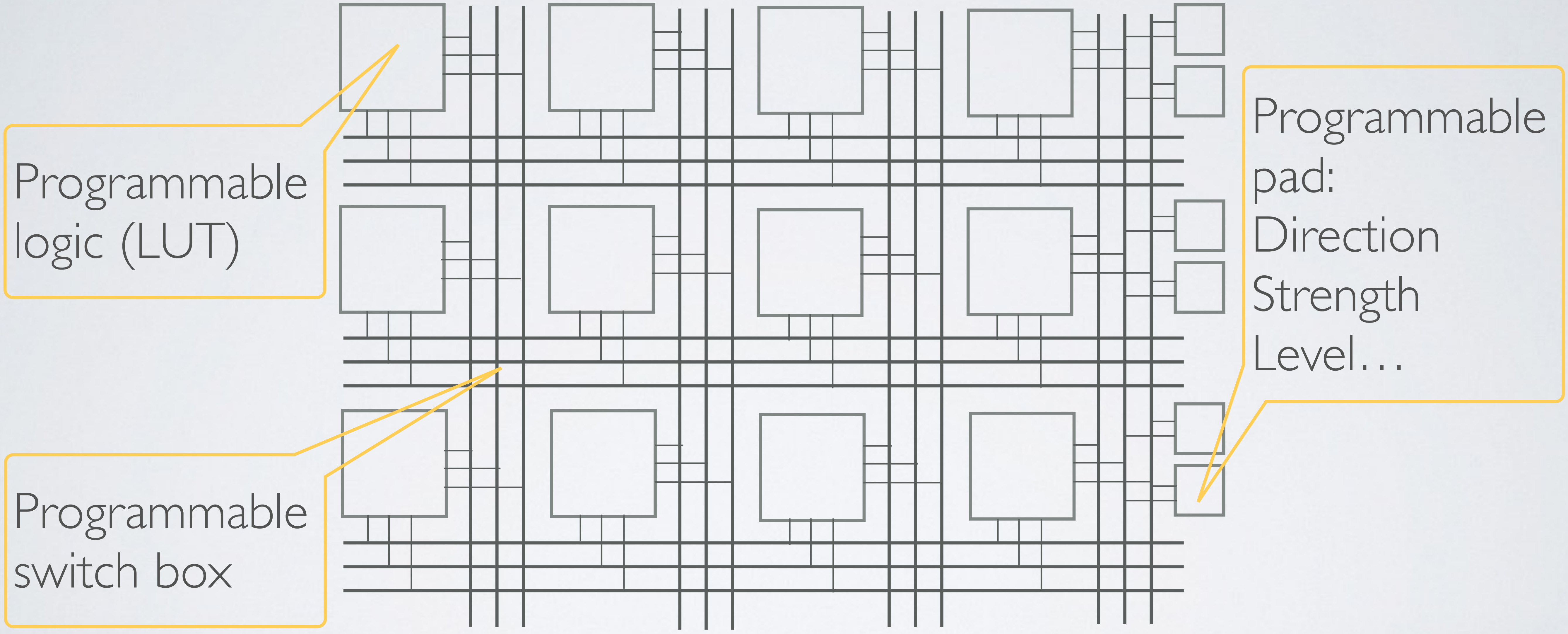


Normal chip

FPGA

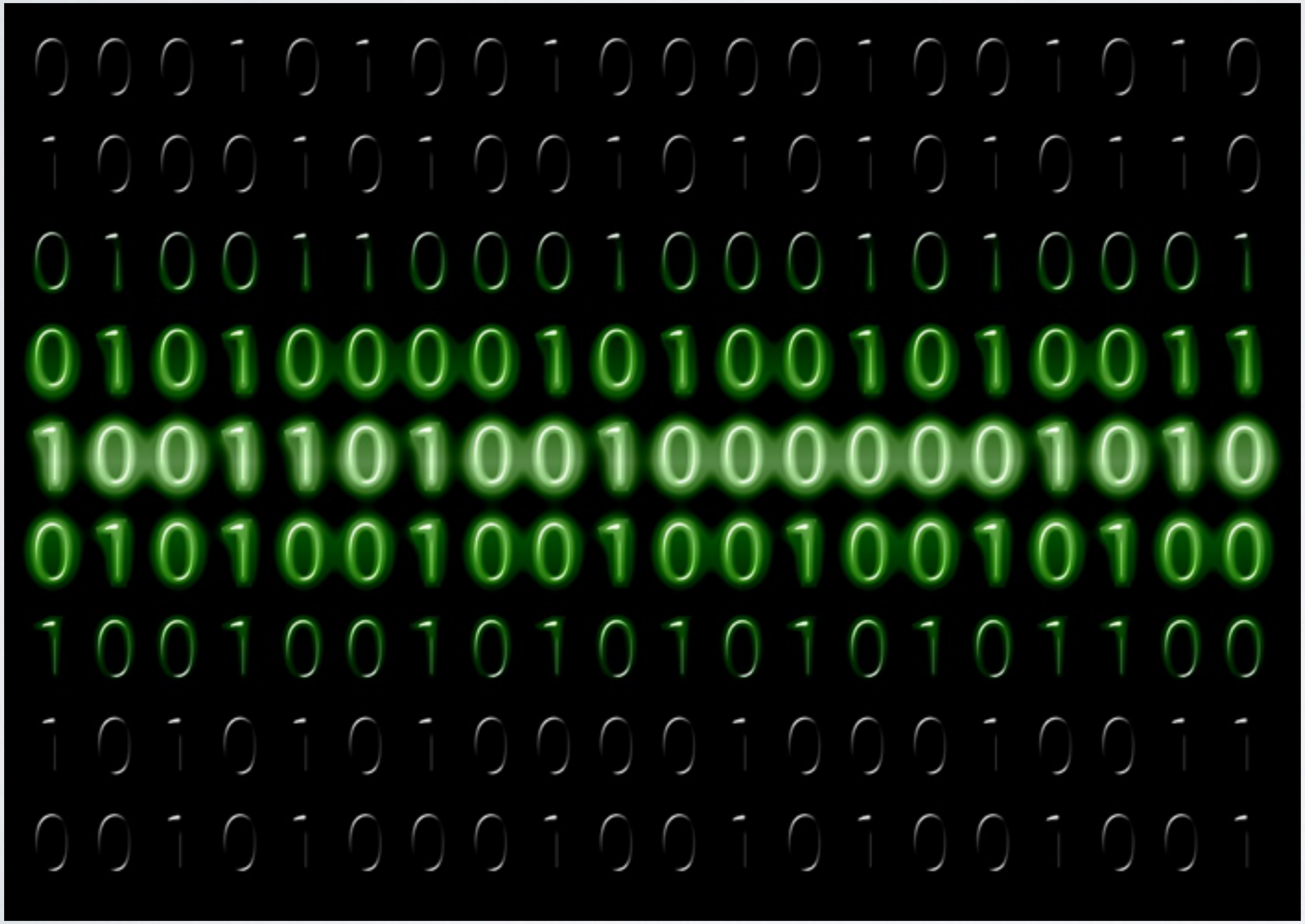
There are other kinds of programmable circuits:
Gate array
CPLD
…

FPGA ARCHITECTURE



That's a very simple view…
Most FPGAs also have PLL, memories, multipliers, or even SERDES/PCI-e blocks.
See FPGA databooks

DIGITAL IS ABOUT 0 AND 1

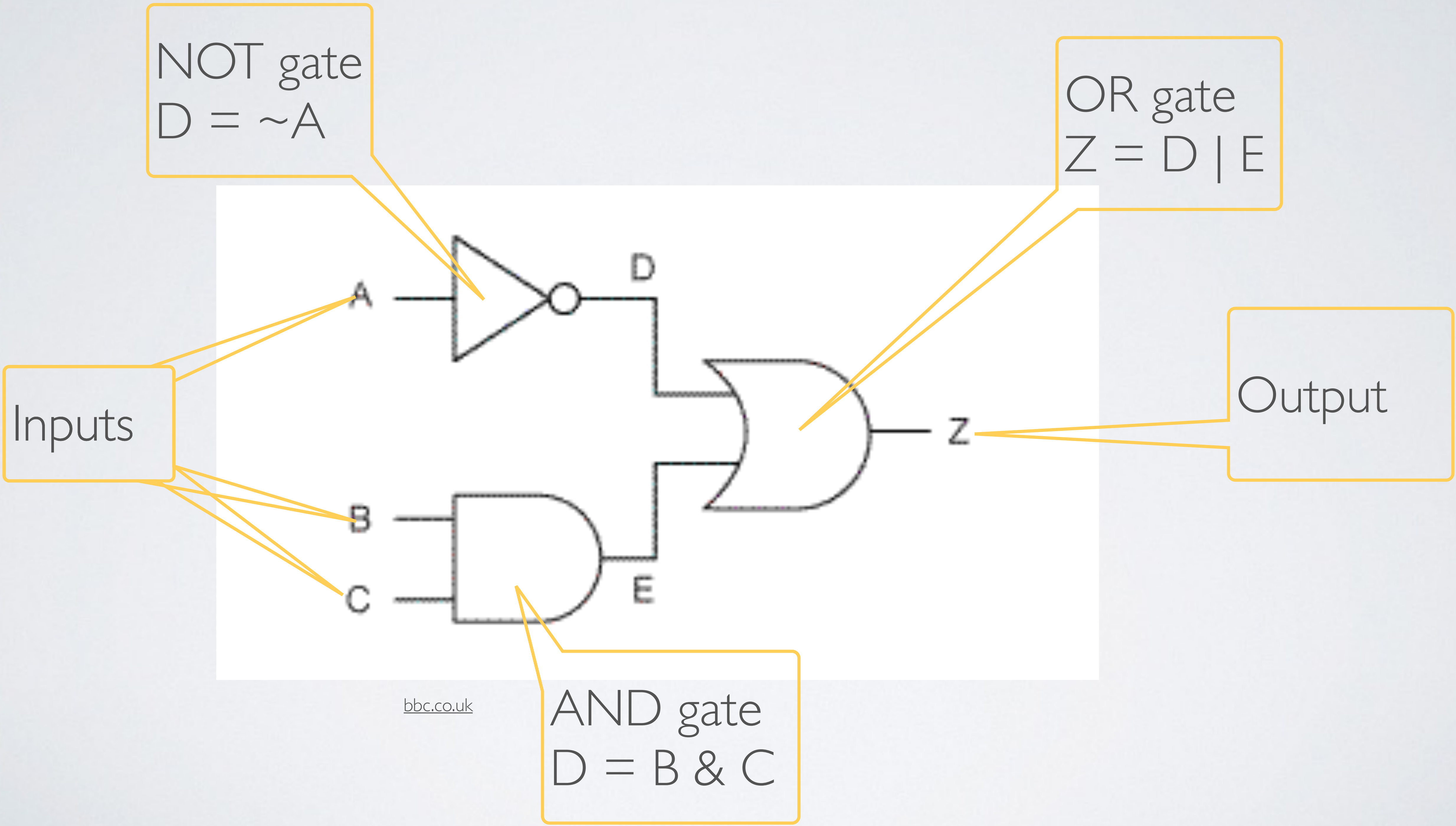


That's simple !
Assuming you know about binary computation

For analog design,
see gnucap, qucs, spice…

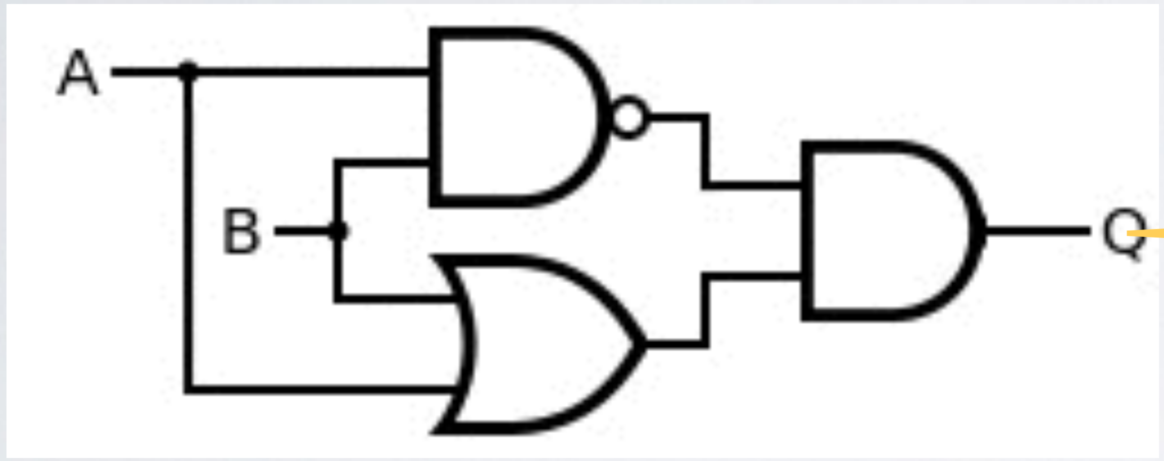
(There are always analog parts in a circuit)

DIGITAL IS ABOUT LOGIC BASIC OPERATIONS



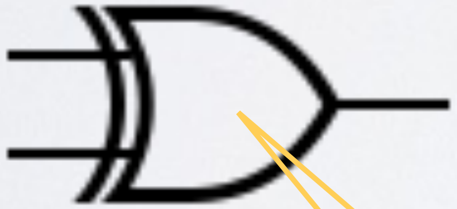
bbc.co.uk

COMBINE THEM!



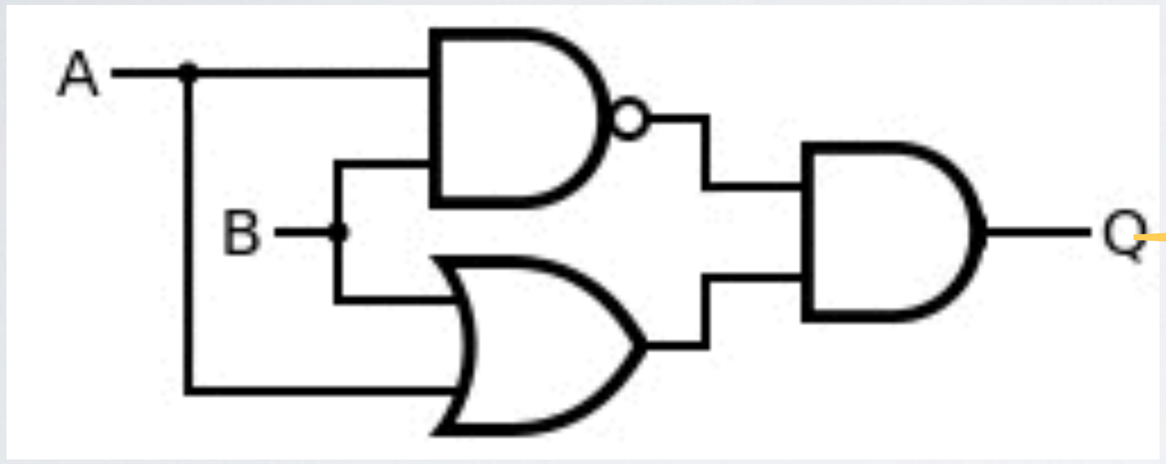
wikipedia.org

Q = A ^ B



Symbol for XOR gate

OR DO MATH (ONE BIT)

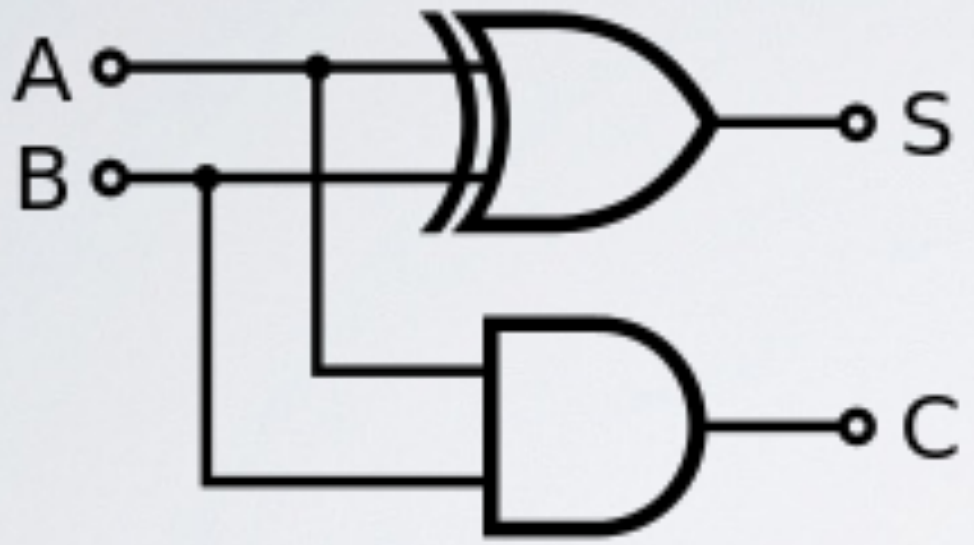


wikipedia.org

$Q = A \wedge B$
$= A + B$
$= A \sim= B$

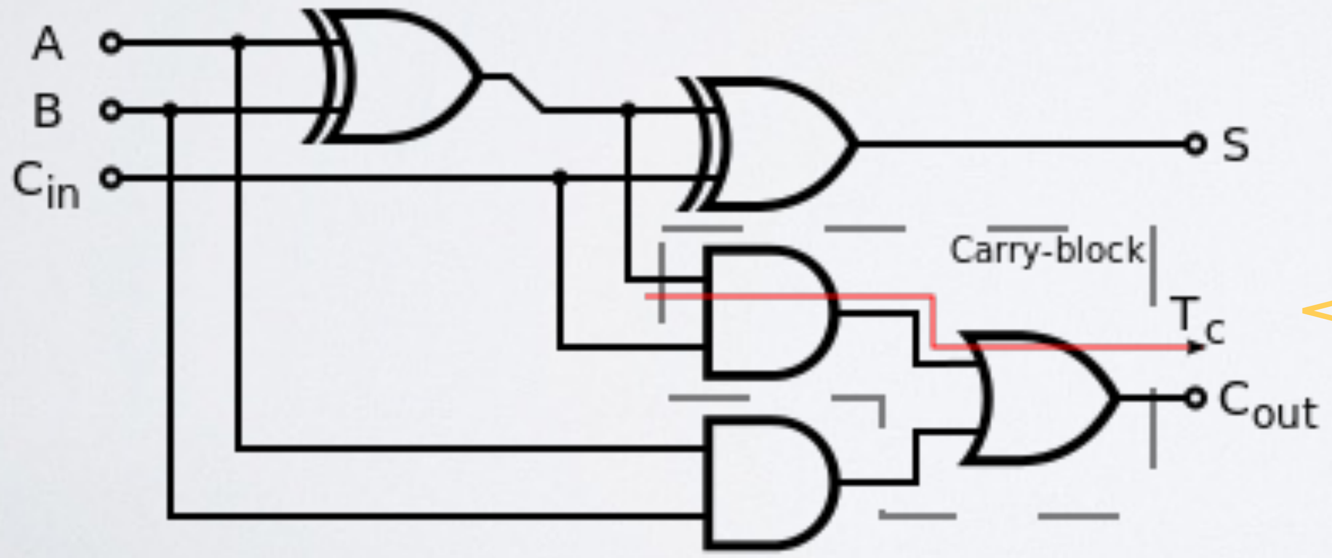
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

THE ADDER

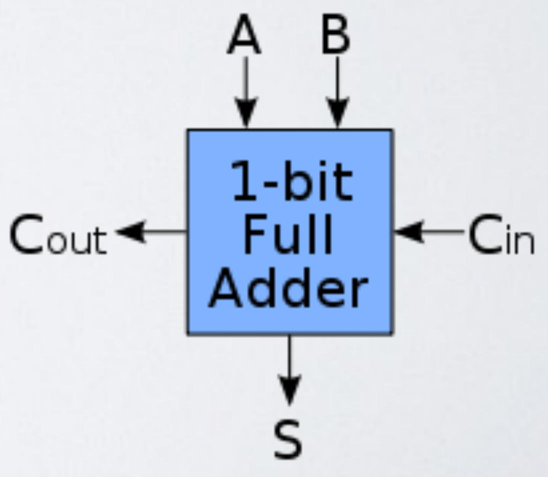


S : SUM
C : CARRY

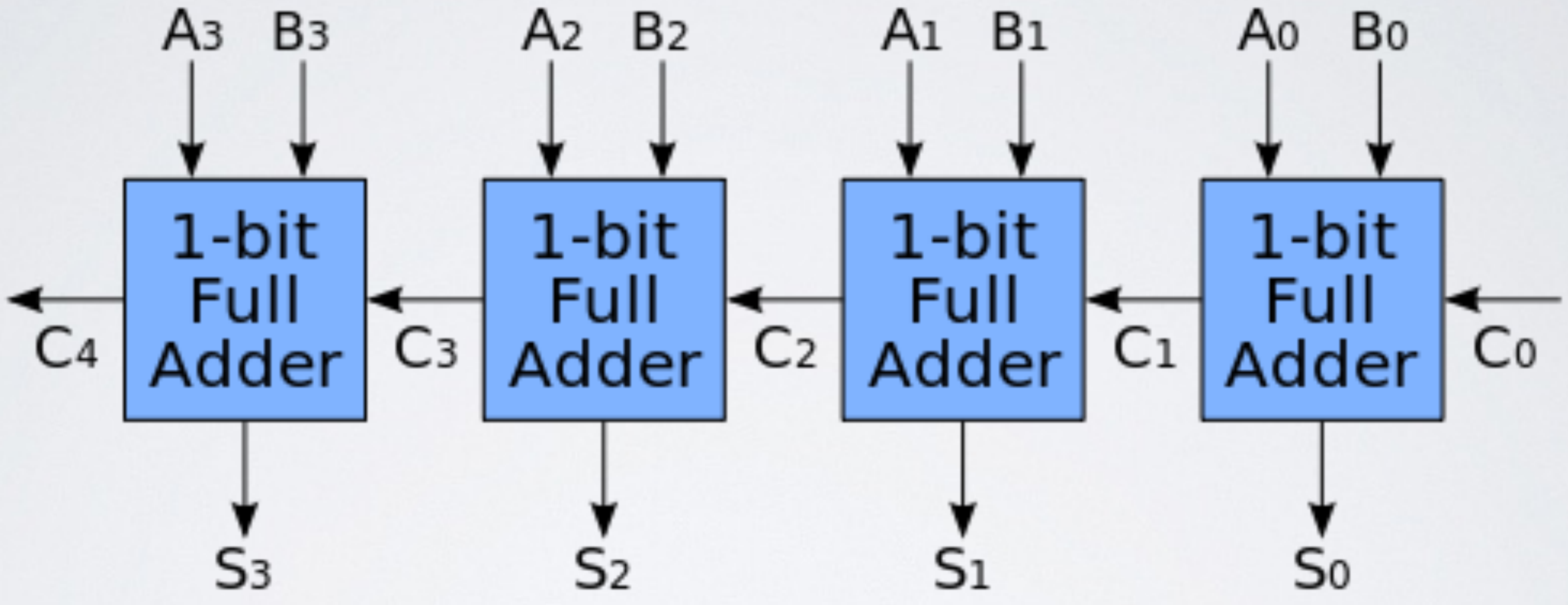
wikipedia.org



Full Adder



MULTIPLE BIT ADDER

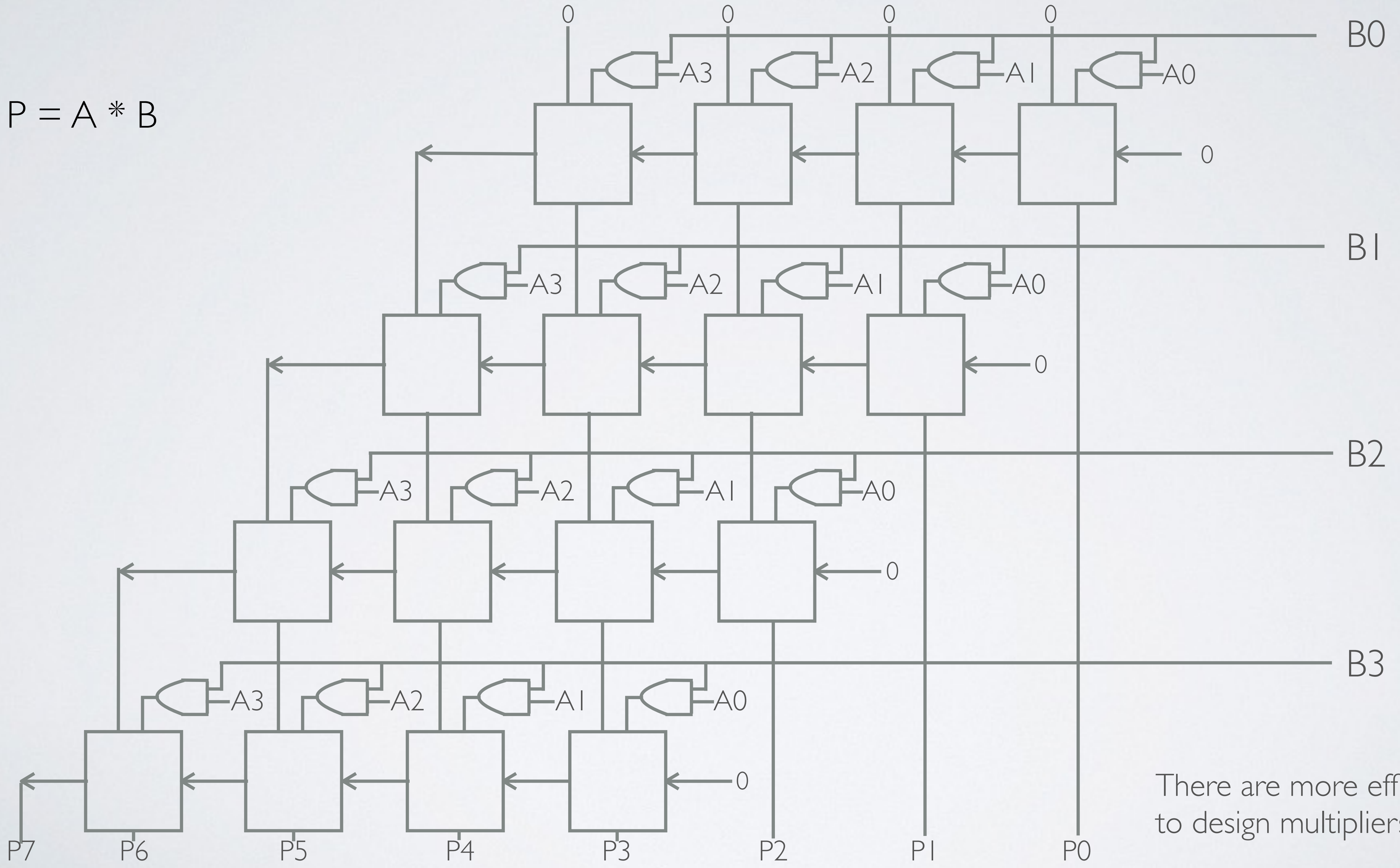


S = A + B

There are more efficient way to design large adders
Search for Digital Logic Architecture

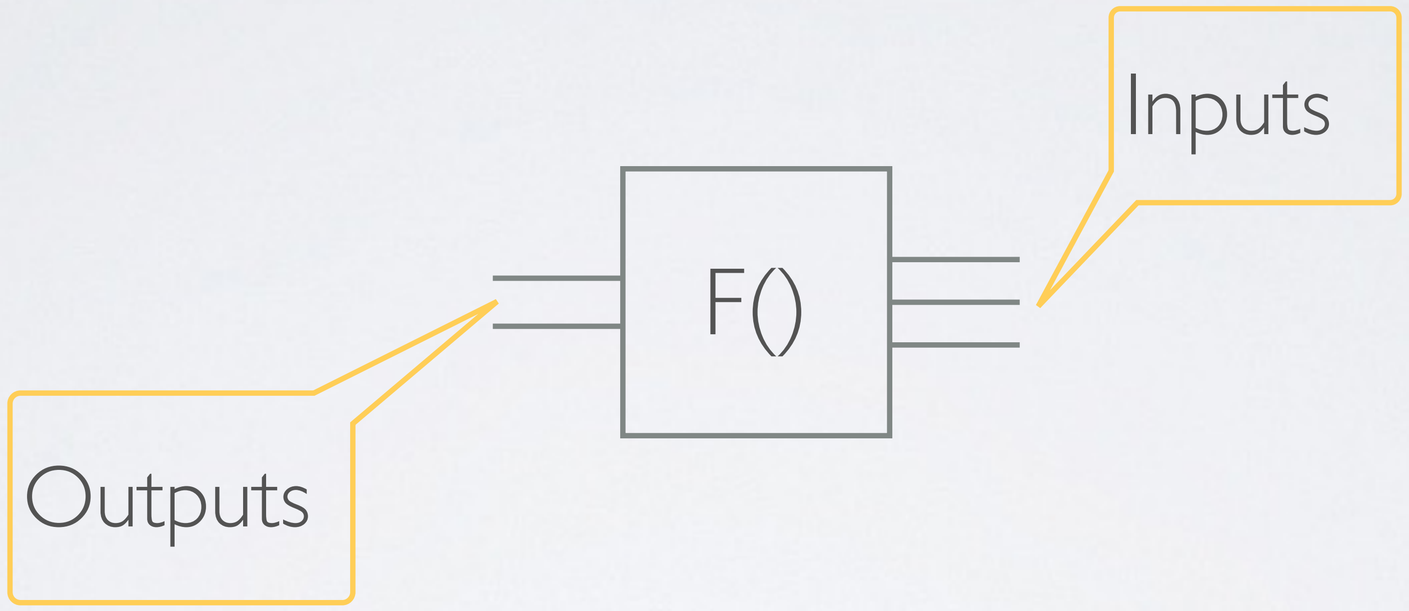
IF YOU CAN ADD, YOU CAN MULTIPLY!

P = A * B



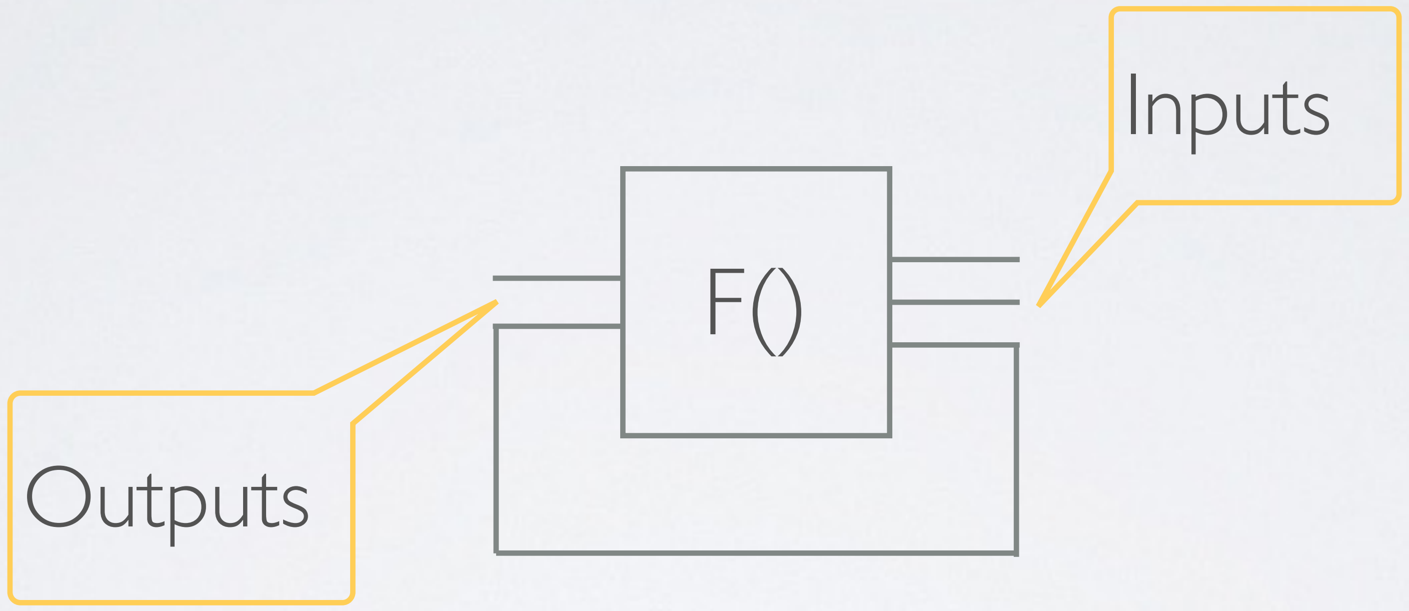
There are more efficient way to design multipliers

YOU CAN DESIGN ANY LOGICAL/ARITH FUNCTION



Well, many functions…
But this is not very efficient (can take a lot of gates)

MORE POWERFUL: RECURSION!

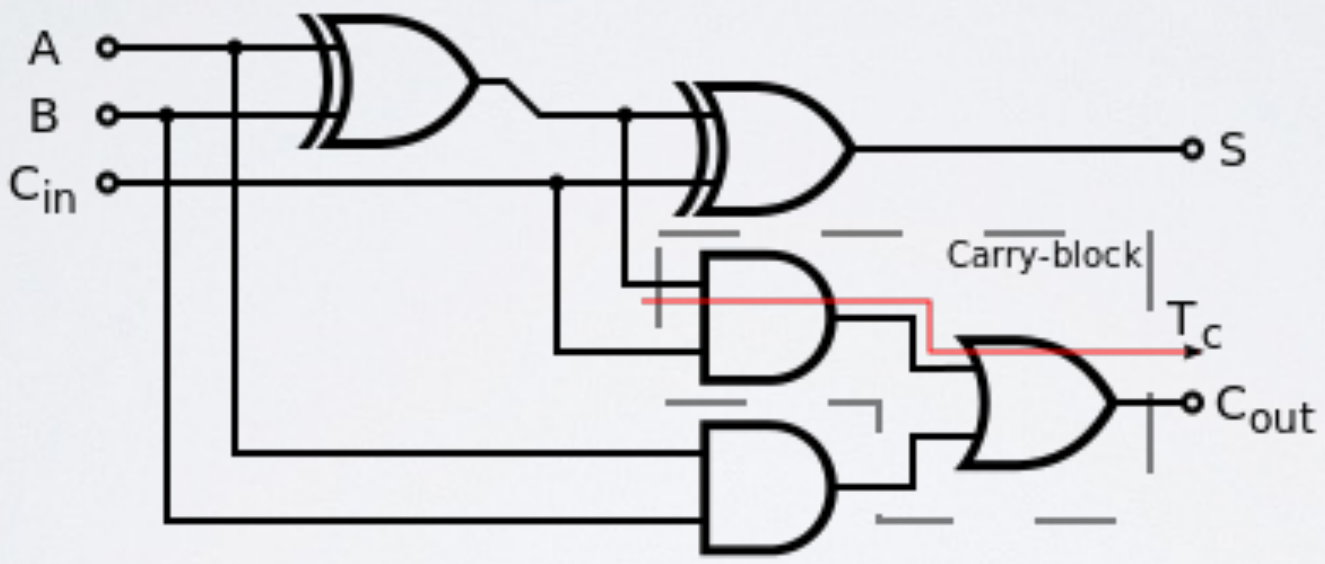


In math,
recursion is very powerful.

In digital design,
it doesn't work directly!

TIMING SYNCHRONISATION

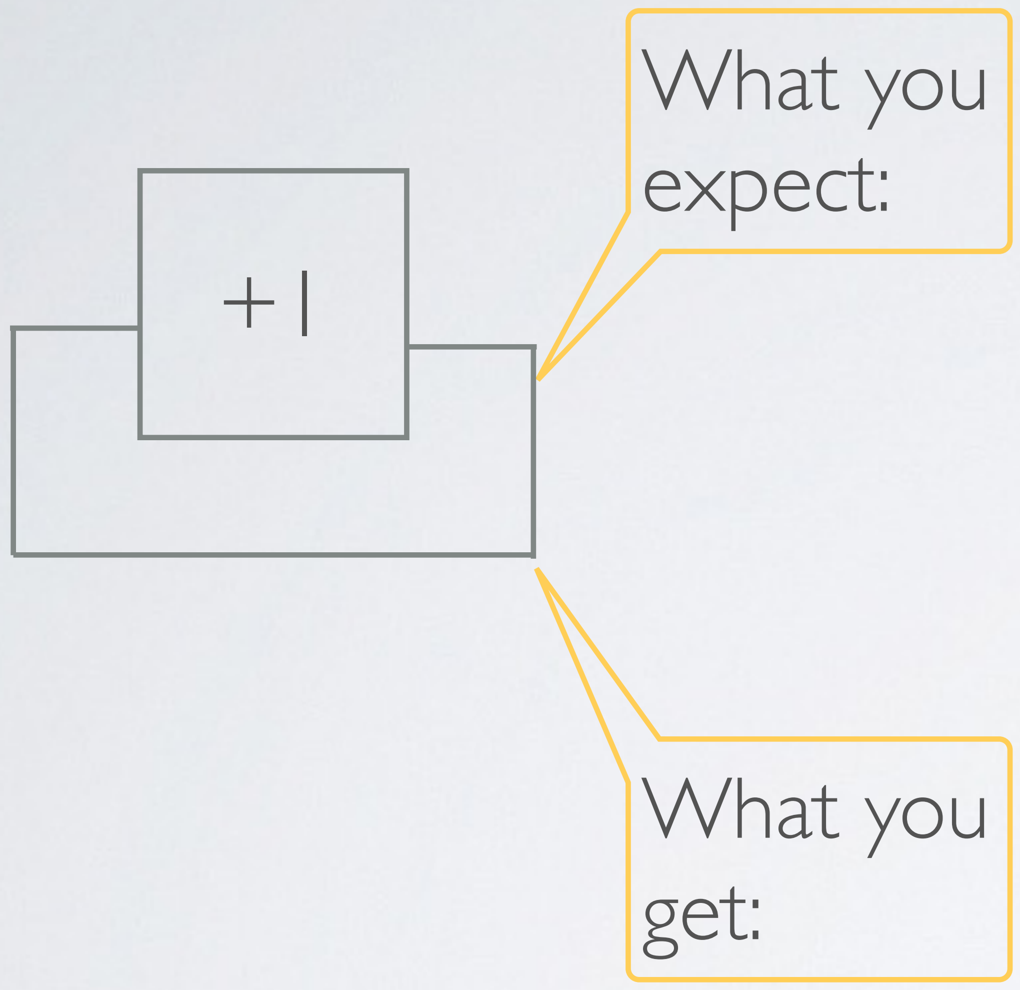
Do you remember the full adder ?

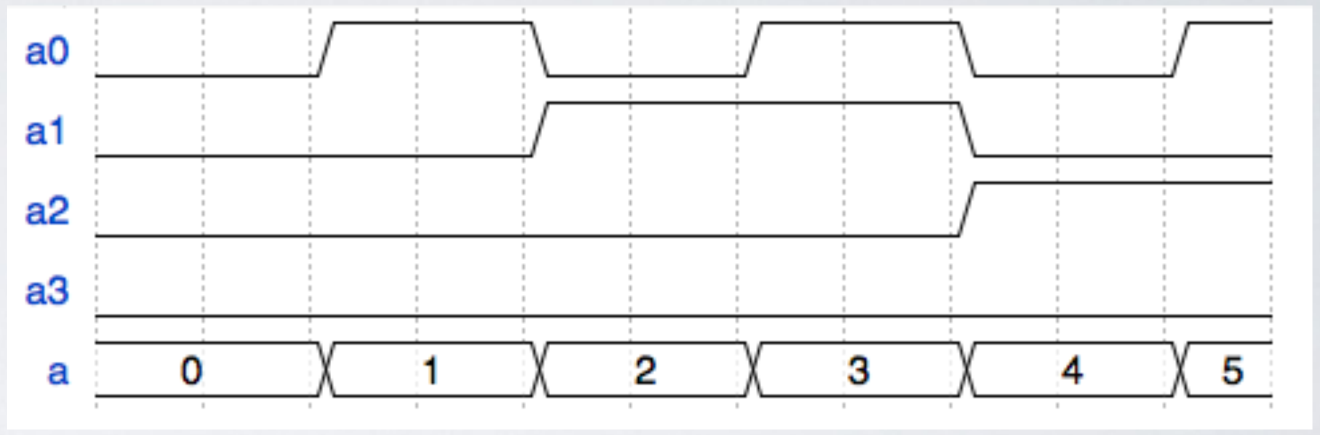


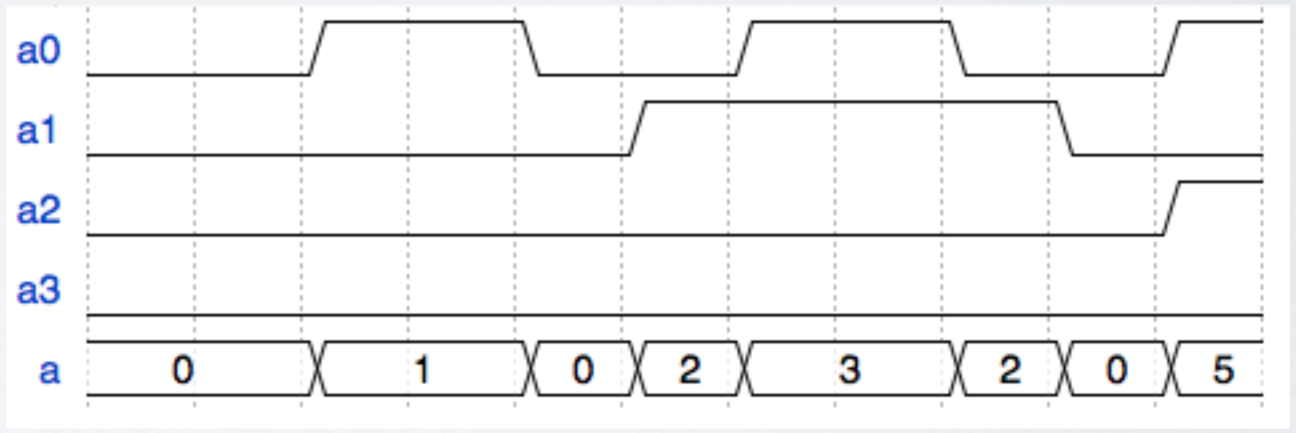
wikipedia.org

It takes time for a signal to propagate through gates.
(due to capacities).
So the arrival times at S and Cout differ.

TIMING DIAGRAM







Thanks to http://wavedrom.com/editor.html

Outputs are not available at the same time.

SYNCHRONOUS DESIGN

You can try to balance paths, but:

- It's very hard
- propagation time depends on too many factors

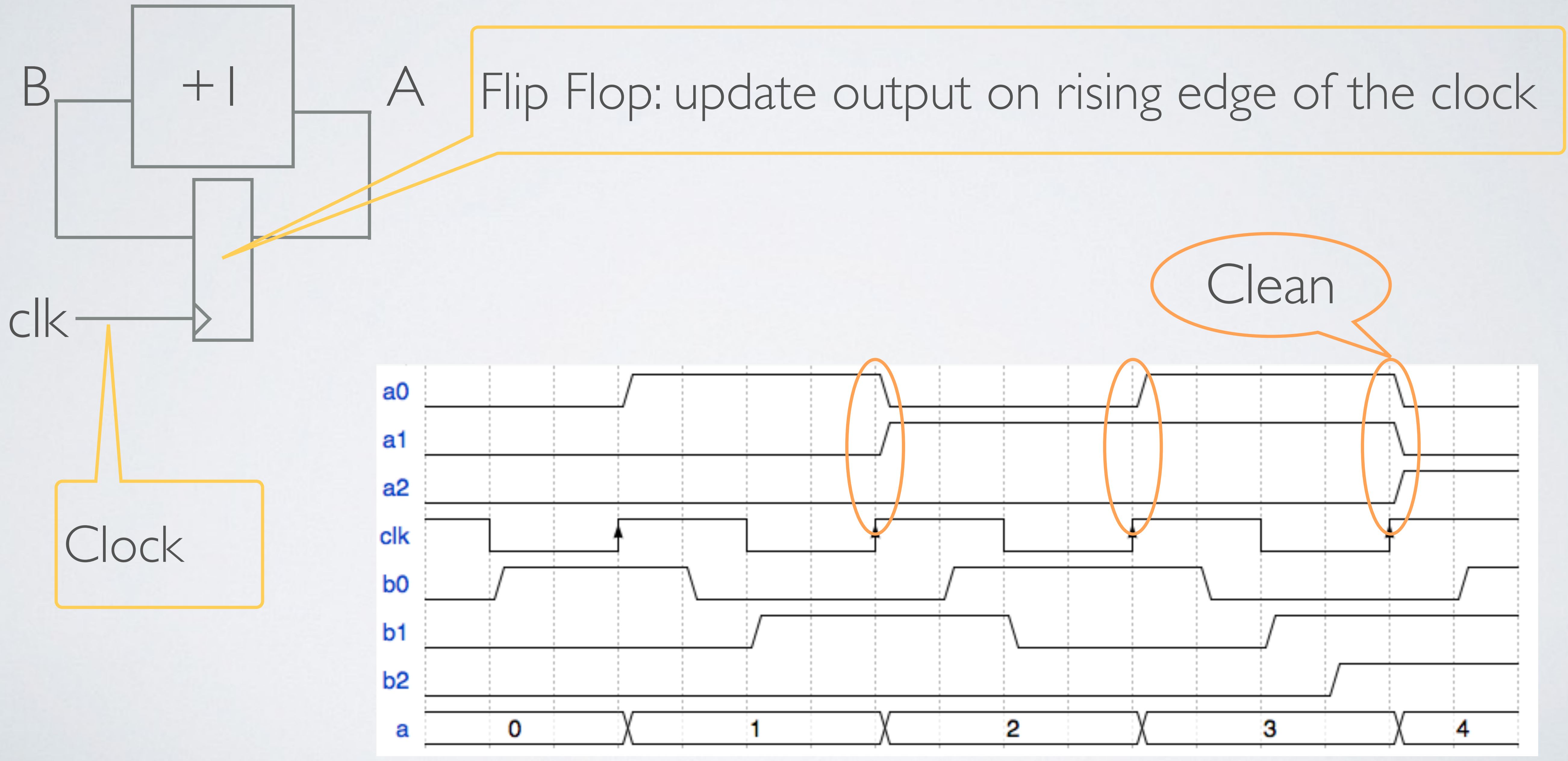
You can use a logic that is not affected by delay variation (like gray code), but:

- works only in some cases.

Rule #1:
no direct loop/feedback

So how to do ?

SYNCHRONOUS DESIGN



DIGITAL DESIGN

It's a mix of:

- logic gates
- flip flops

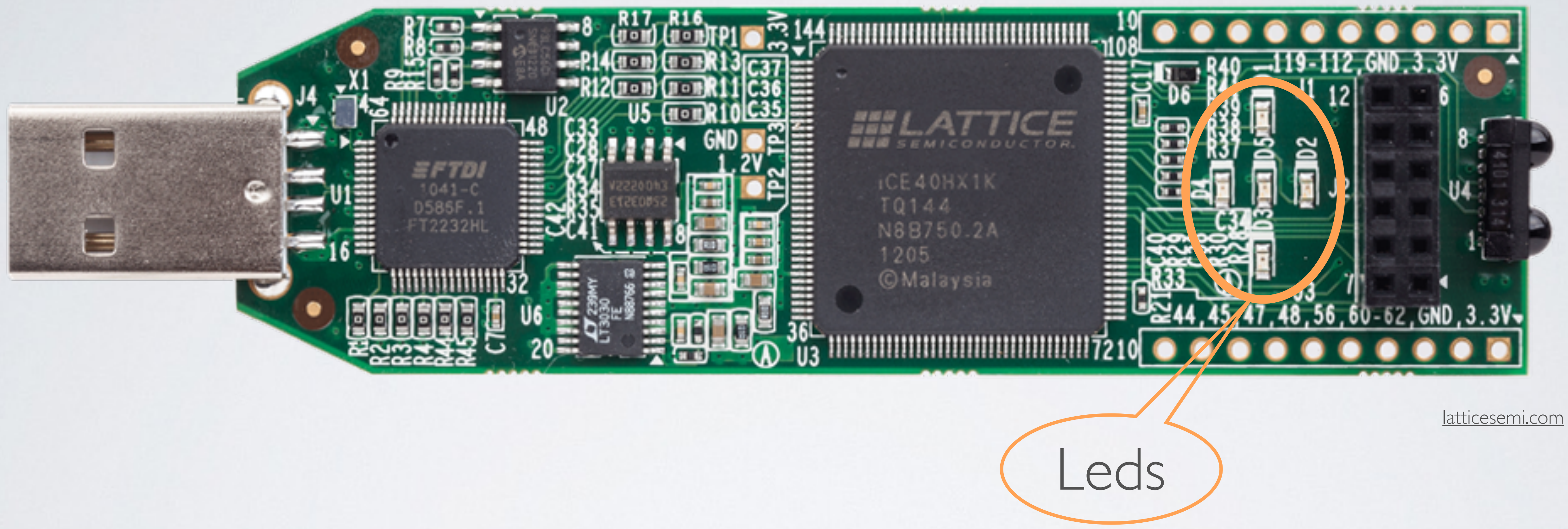
There are other way to synchronise
(latch, falling edge, double edge…)

It is possible to use schematic editors, but

- tedious
- doesn't scale well

Use an HDL
Hardware Description Language
I will use VHDL

MY FIRST DESIGN BLINKING LEDS



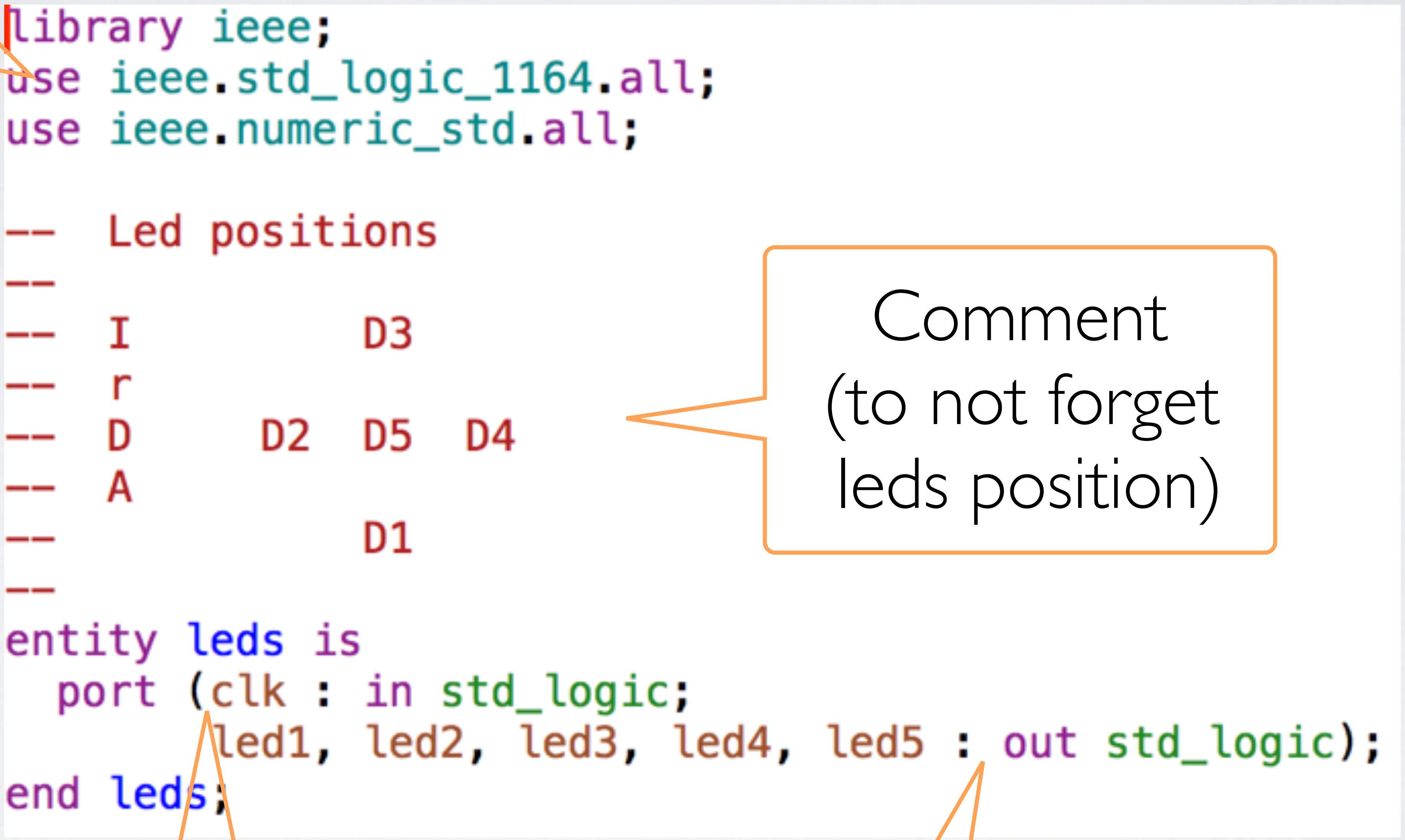
Using OSS tools:

- ghdl
- yosys
- arachne-pnr
- iceStorm

Target: Lattice iCEstick
~ 22 euros
Supported by OSS tools

VHDL: EXTERNAL INTERFACE

boilerplate



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--  Led positions
--
--  I         D3
--  r
--  D     D2  D5  D4
--  A
--            D1
--
entity leds is
  port (clk : in std_logic;
        led1, led2, led3, led4, led5 : out std_logic);
end leds;
```

Comment
(to not forget
leds position)

interface

Input: clock
(externally generated 3Mhz)

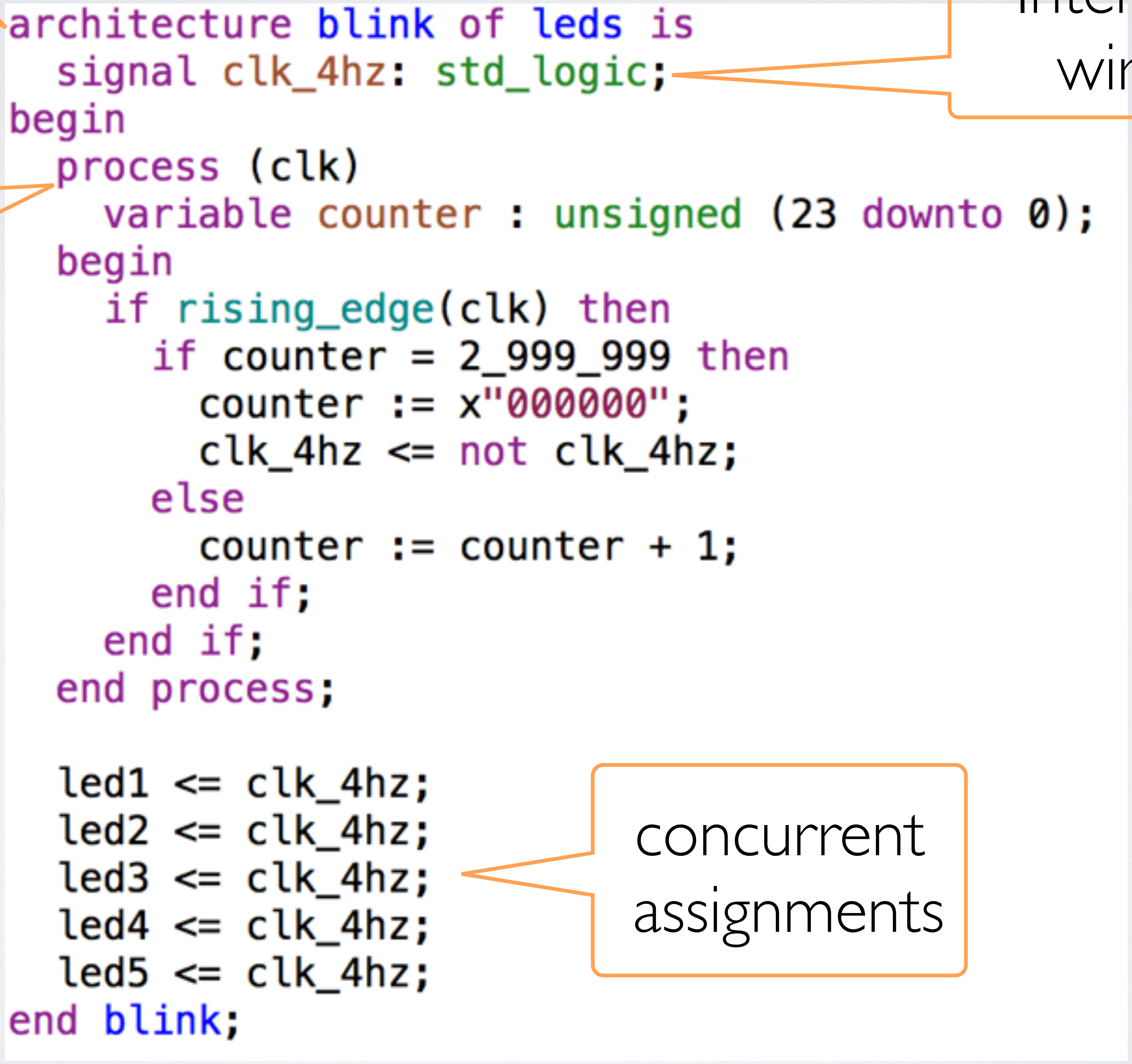
outputs: leds

INTERNALS

Internals

Internal wire

Process: concurrent execution, triggered on clk



```
architecture blink of leds is
  signal clk_4hz: std_logic;
begin
  process (clk)
    variable counter : unsigned (23 downto 0);
  begin
    if rising_edge(clk) then
      if counter = 2_999_999 then
        counter := x"000000";
        clk_4hz <= not clk_4hz;
      else
        counter := counter + 1;
      end if;
    end if;
  end process;

  led1 <= clk_4hz;
  led2 <= clk_4hz;
  led3 <= clk_4hz;
  led4 <= clk_4hz;
  led5 <= clk_4hz;
end blink;
```

concurrent assignments

There are many VHDL or Verilog tutorials on the web.

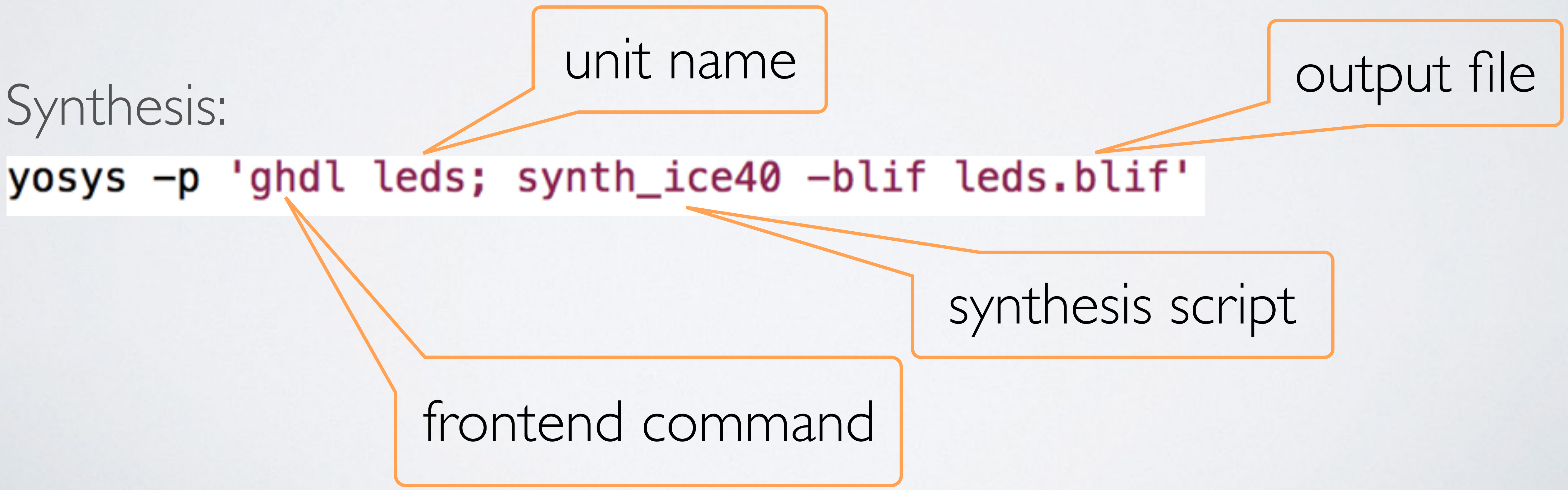
SYNTHESIS

Translating (or compiling) sources to gates (netlist)

First, analysing sources:

```
ghdl -a leds.vhdl
ghdl -a blink.vhdl
```

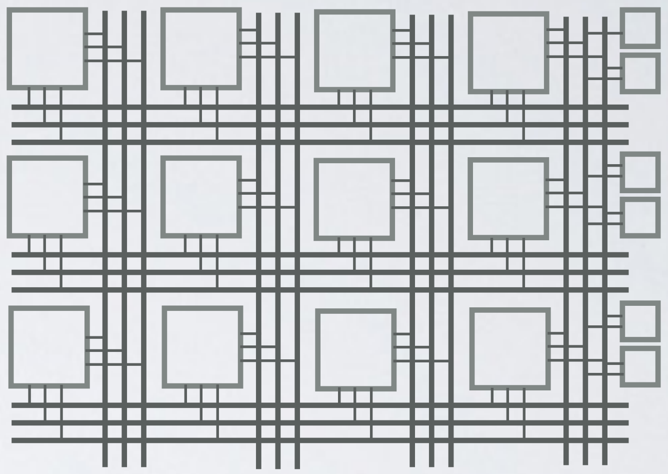
Synthesis:



```
yosys -p 'ghdl leds; synth_ice40 -blif leds.blif'
```

PLACE & ROUTE

Allocate resources on the FPGA



device

input

```
arachne-pnr -d 1k -o leds.asc -p leds.pcf leds.blif
```

output

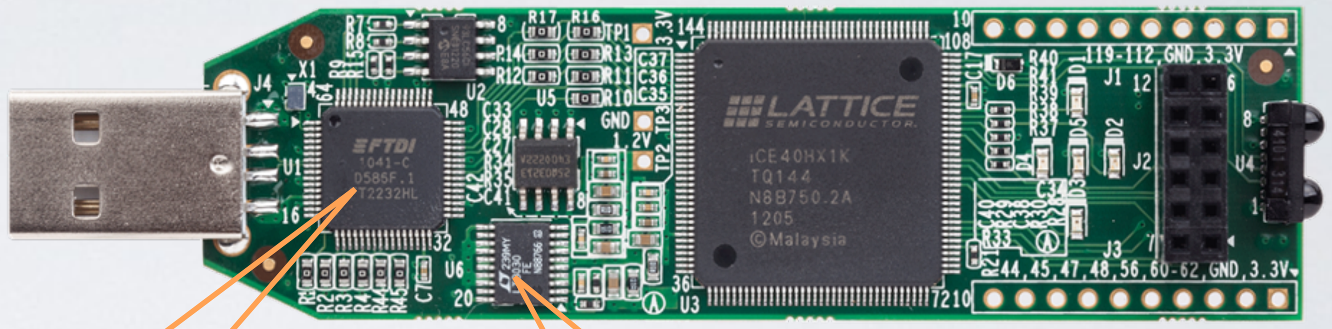
place file

```
set_io led1 99
set_io led2 98
set_io led3 97
set_io led4 96
set_io led5 95
set_io clk 21
```

IC pin #

PROGRAM

Write into the FPGA



Create the binary file:

```
icepack leds.asc leds.bin
```

Write to flash:

```
iceprog leds.bin
```

The FPGA is automatically reset and then load the new config

TOOLS USED

Synthesis:
http://www.clifford.at/yosys/

VHDL front-end:
https://github.com/tgingold/ghdlsynth-beta
https://github.com/tgingold/ghdl

Place and route:
https://github.com/cseed/arachne-pnr

iCE40 tools:
http://www.clifford.at/icestorm/

QUESTIONS ?