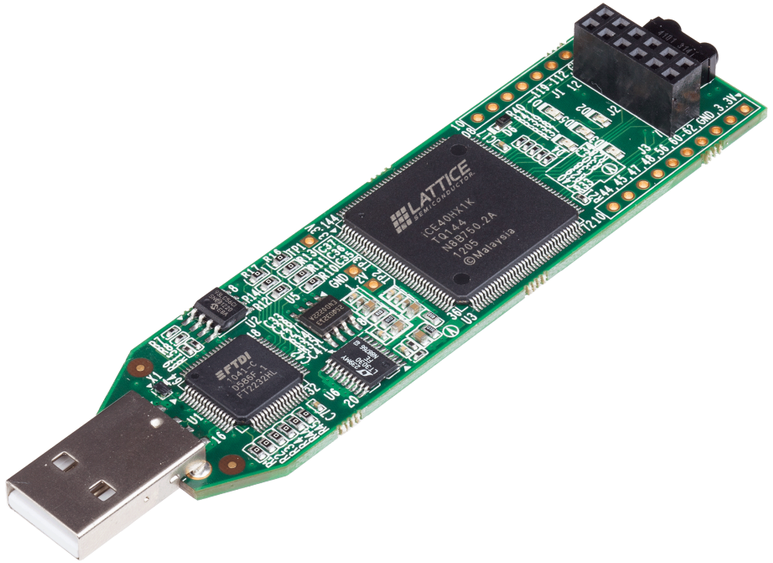
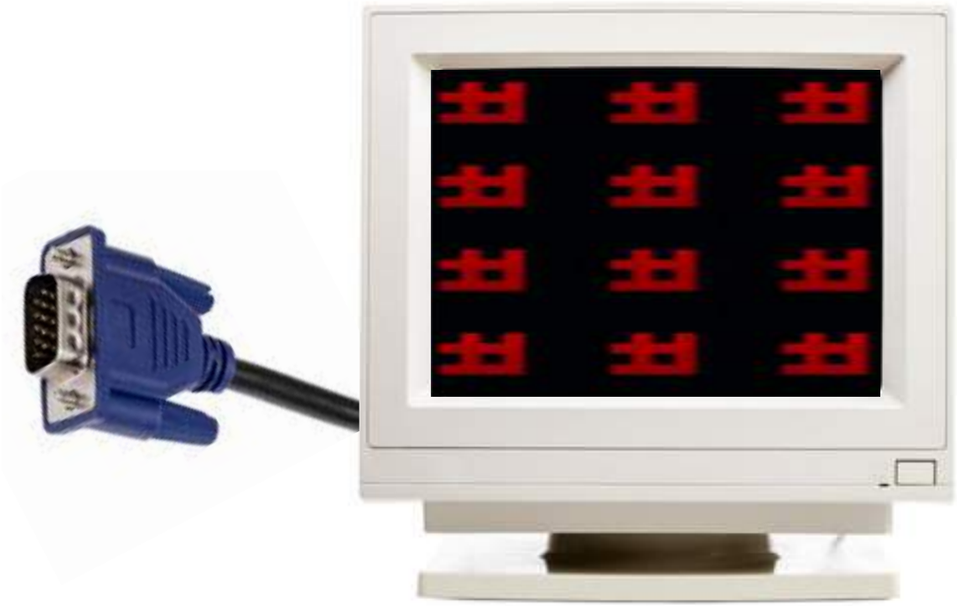
# HTMAA – FPGA

# HTMAA - FPGA




→



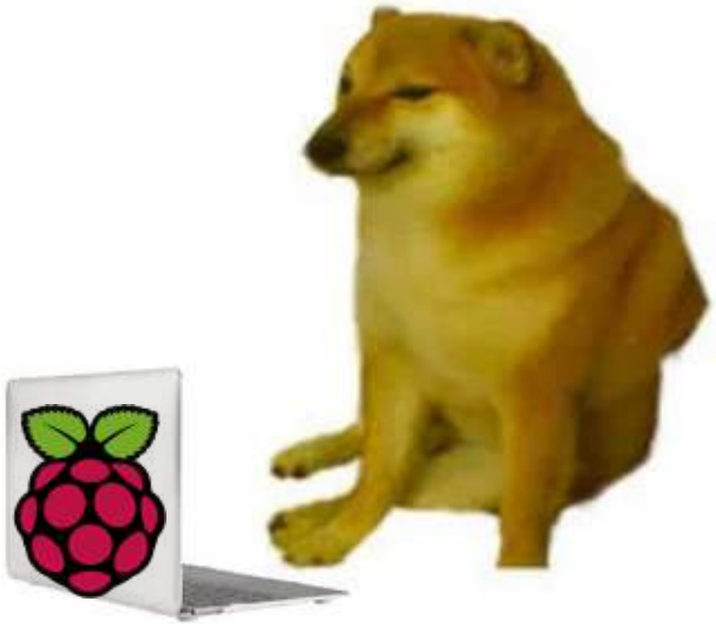
How to make your own simple, interactive VGA video patterns in verilog on the Lattice iCEstick

# why?

Have you started to hit up against the limits of **Arduino** for certain applications ?


# why?

Do you need more speed and parallelism for low level tasks not suited to the **rpi** ?



# why?

## FPGAs are used everywhere !


*resists space radiation

# why?

## employability

# FPGA Engineer (Field-Programmable Gate Array)

An FPGA Engineer designs and implements FPGA-based systems, ensuring they meet performance and functionality requirements. Learn about the technical skills needed, the tools used, and the various applications of FPGAs in industries such as telecommunications, aerospace, and consumer electronics.

Career Guides

| Demand | Salary | Education | Field |
|---|---|---|---|
| High | US $100,000+ | Undergraduate | Engineering |

# HTMAA - FPGA

## DAY 1

TP:
verilog
synthesis

*request
licence!

## DAY 2

TP:
Lattice
Toolchain

## DAY 3

TP:
VGA
animations

# DAY 1

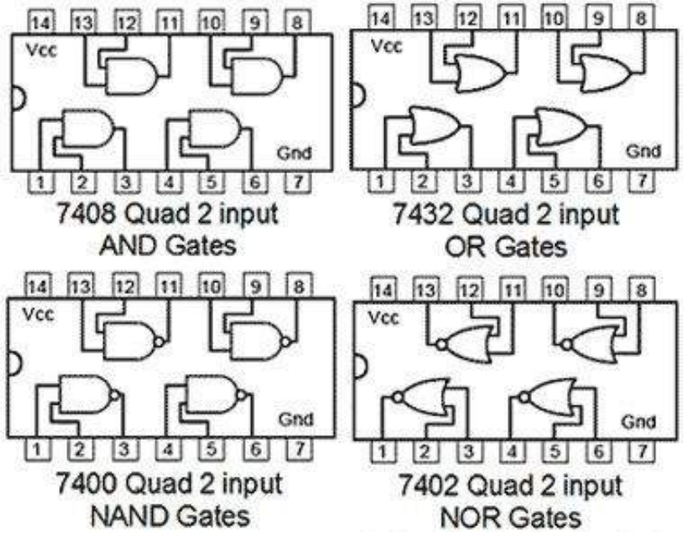
# FPGA?

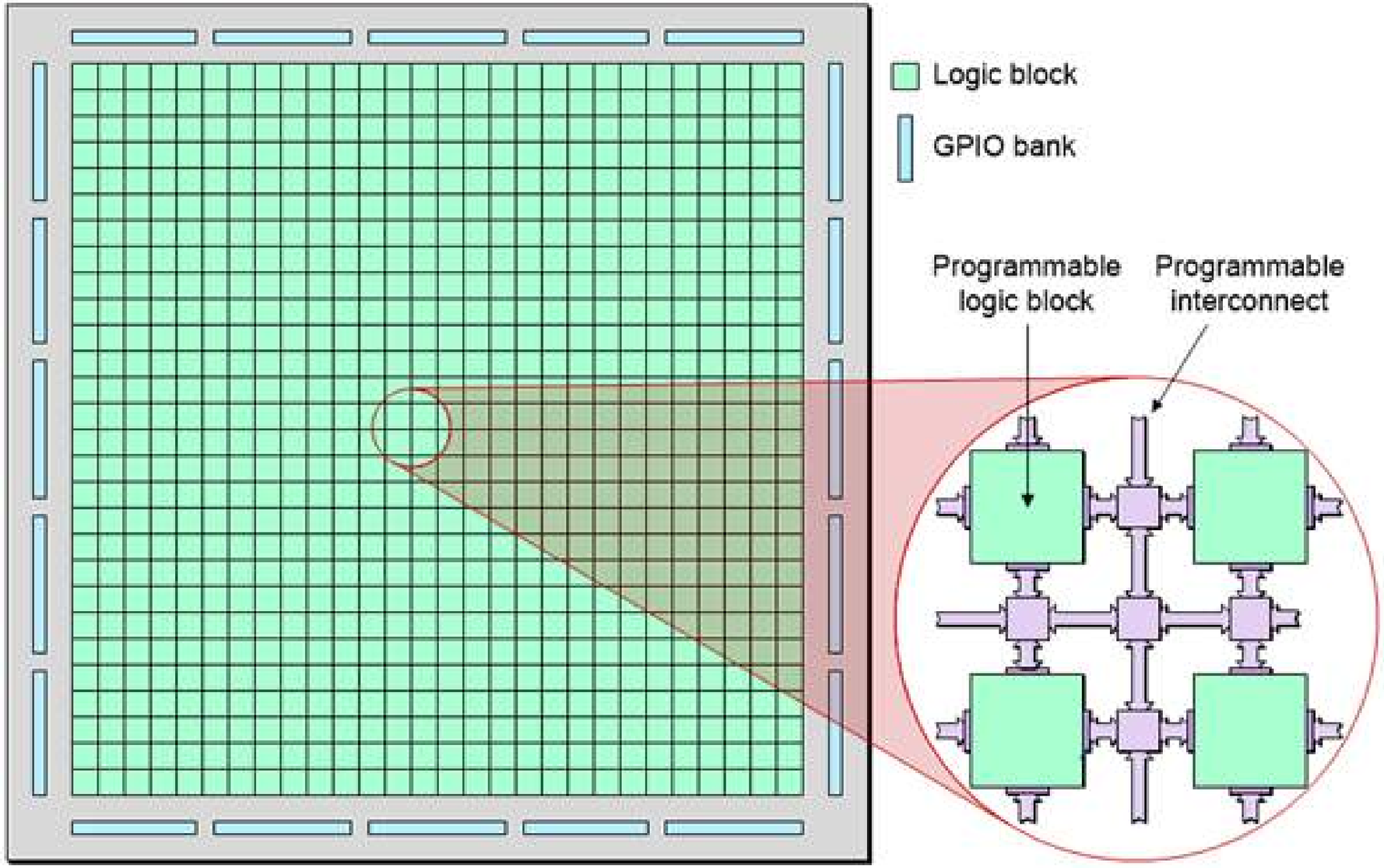
## FIELD PROGRAMMABLE :

You can (re)program this anywhere in the «field.»

## GATE ARRAY :

A bunch of logic gates dealing with 0s and 1s

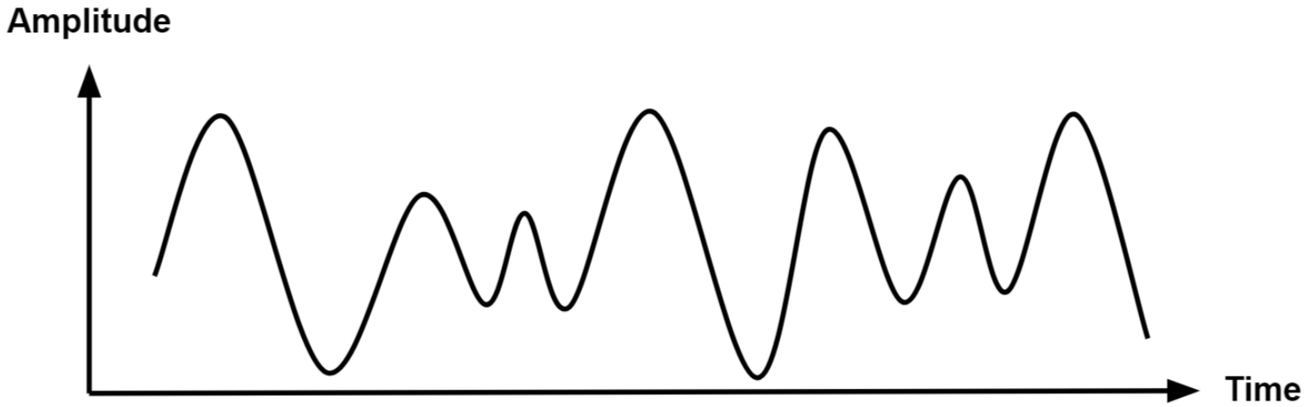


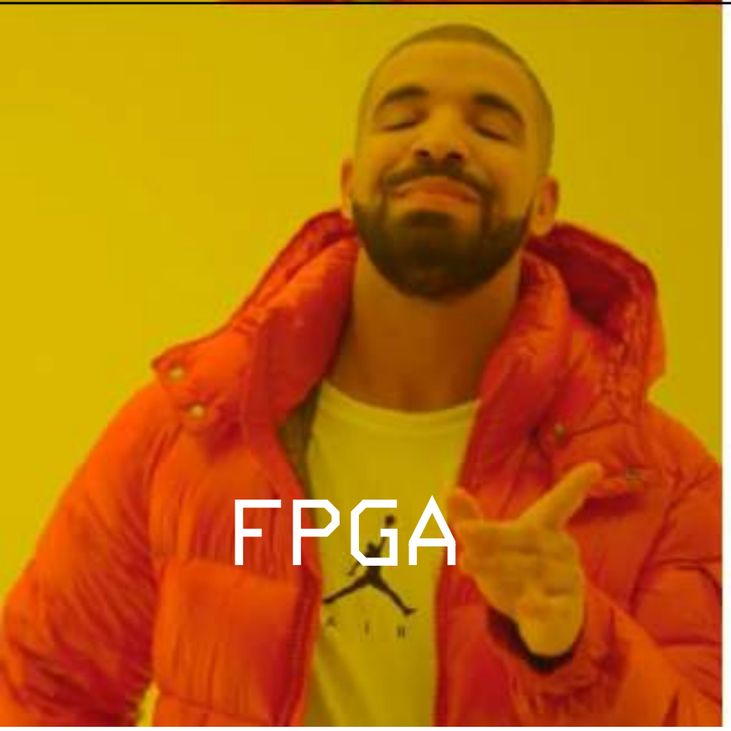



Bird's-eye view of FPGA

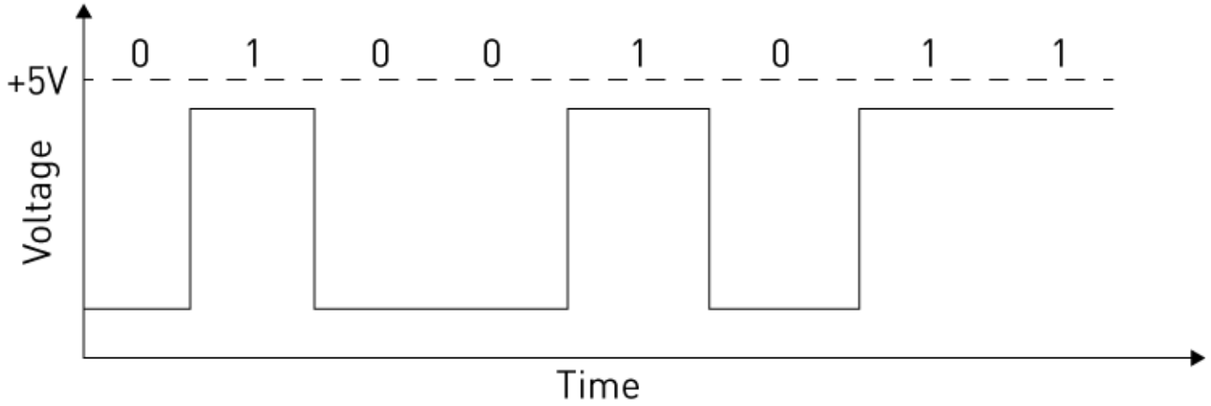
×There are also specialized modules for RAM, communication protocols, I/O etc.


analog



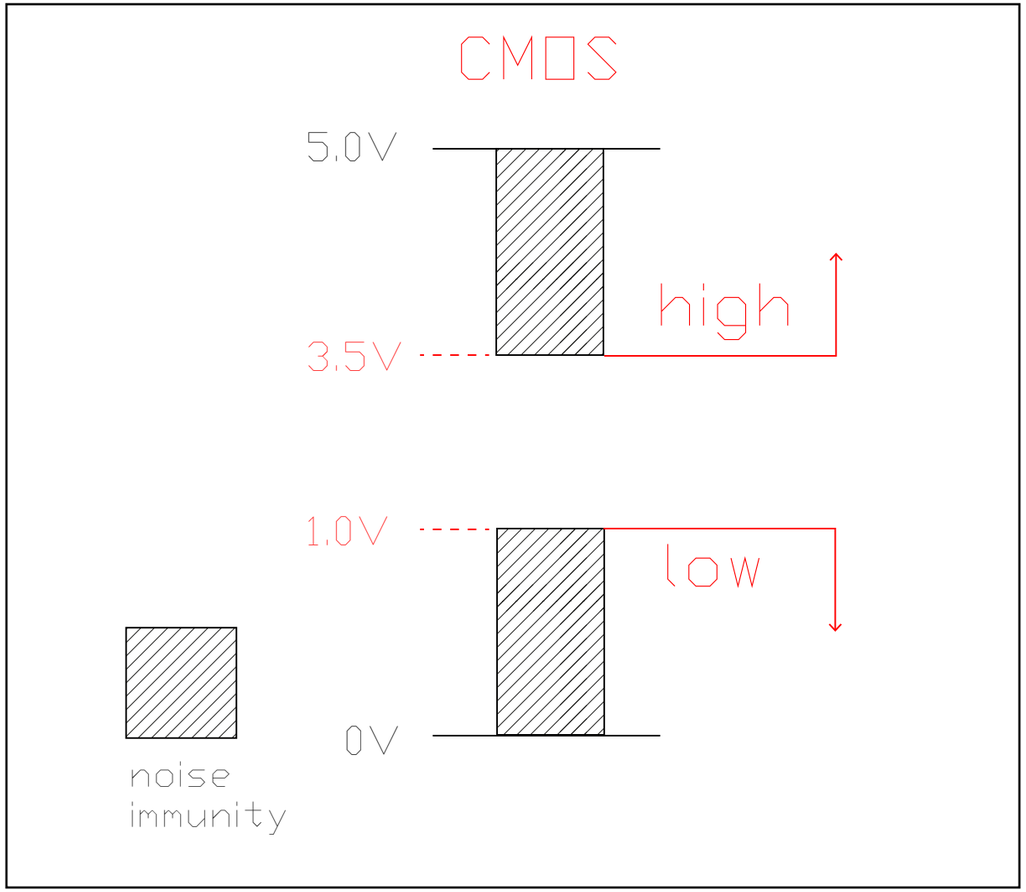

digital



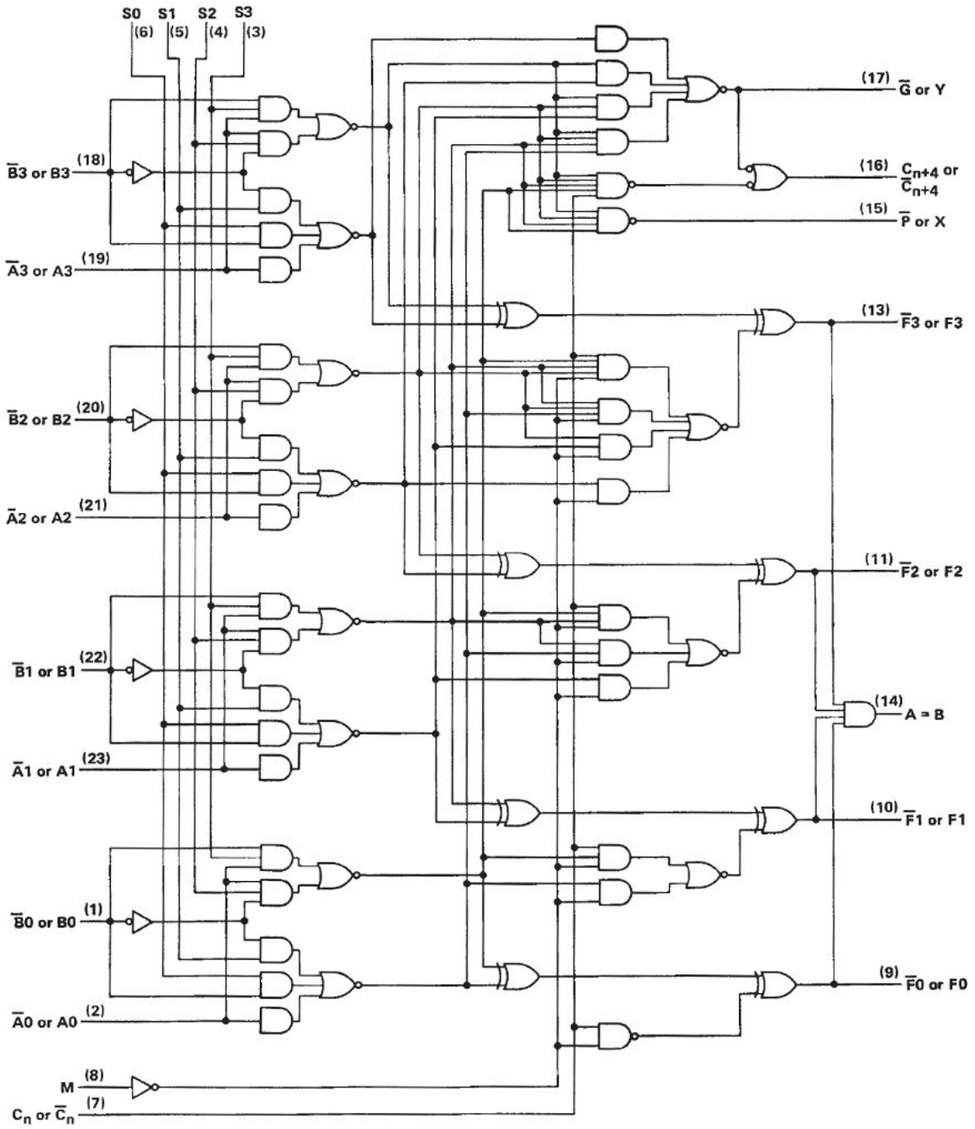
# WHY DIGITAL ?

1

high noise immunity !

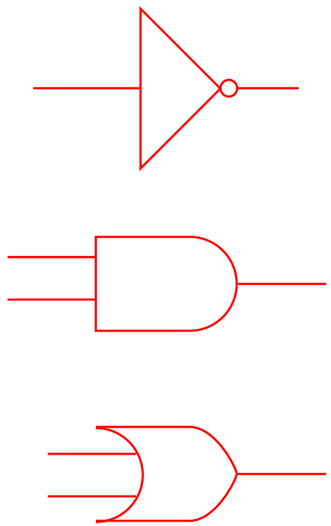



# WHY DIGITAL ?



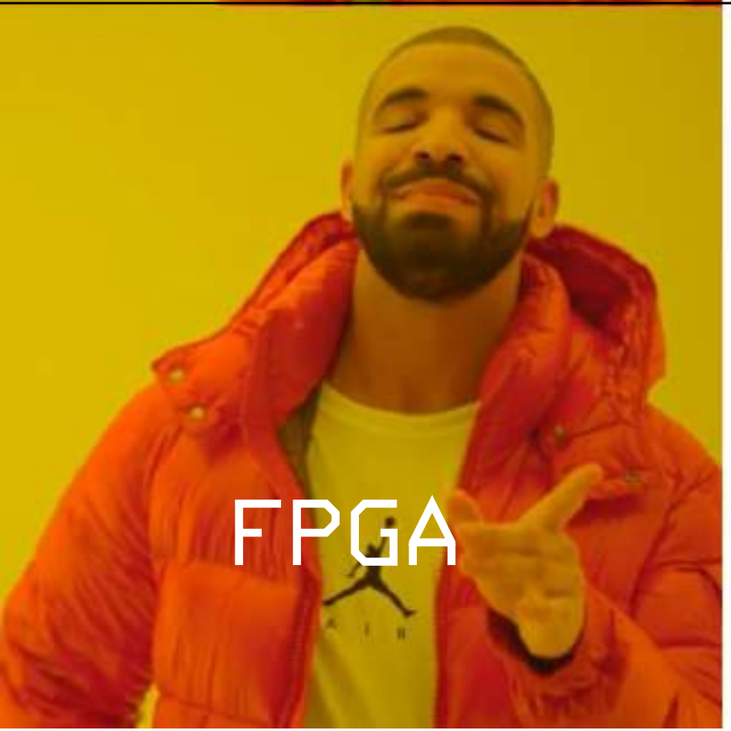

2

All digital devices (like computers, memory, etc.) can be made with only these logic gates !!! :

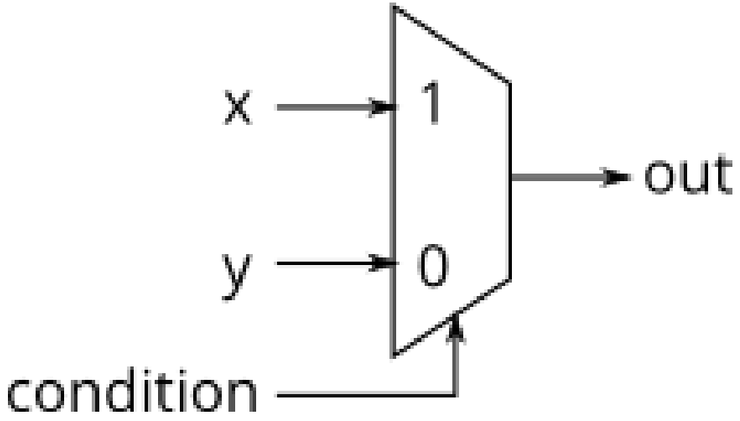



software

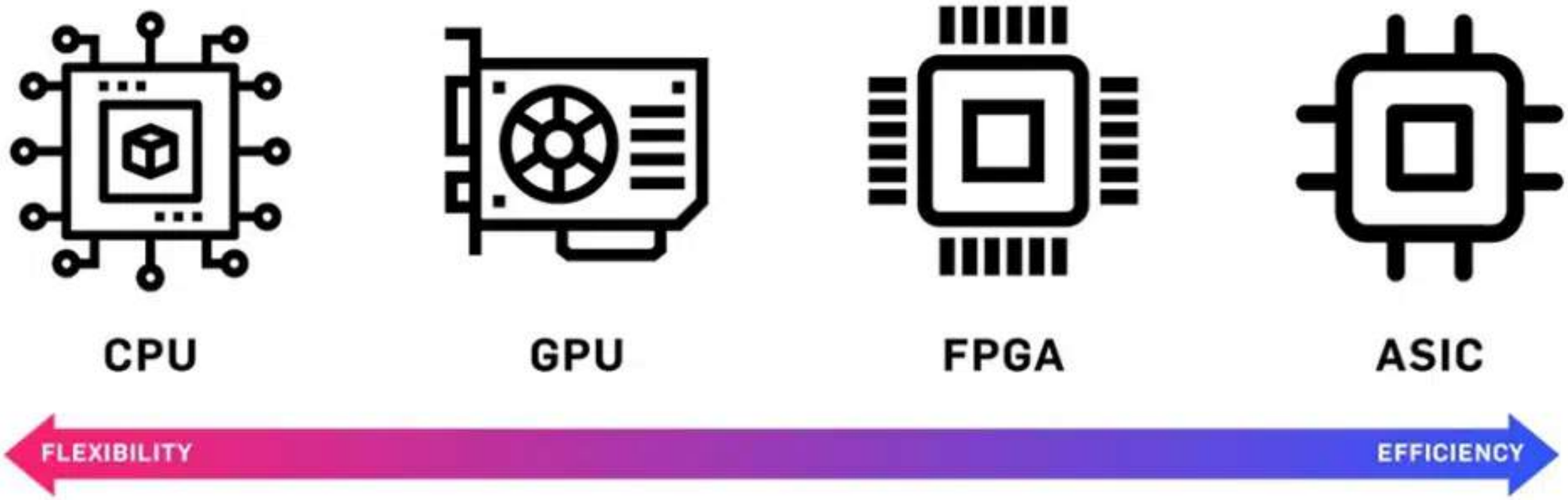
```
for(int i = 0; i < n; ++i)
{
    printf("Hello World");
}
```




hardware



# How do FPGAs compare ?



Build Custom Silicon with Google

Recherche /

Français

Créez des puces sur mesure pour tous, à grande échelle, comme les logiciels.

Accueil Notebooks Research

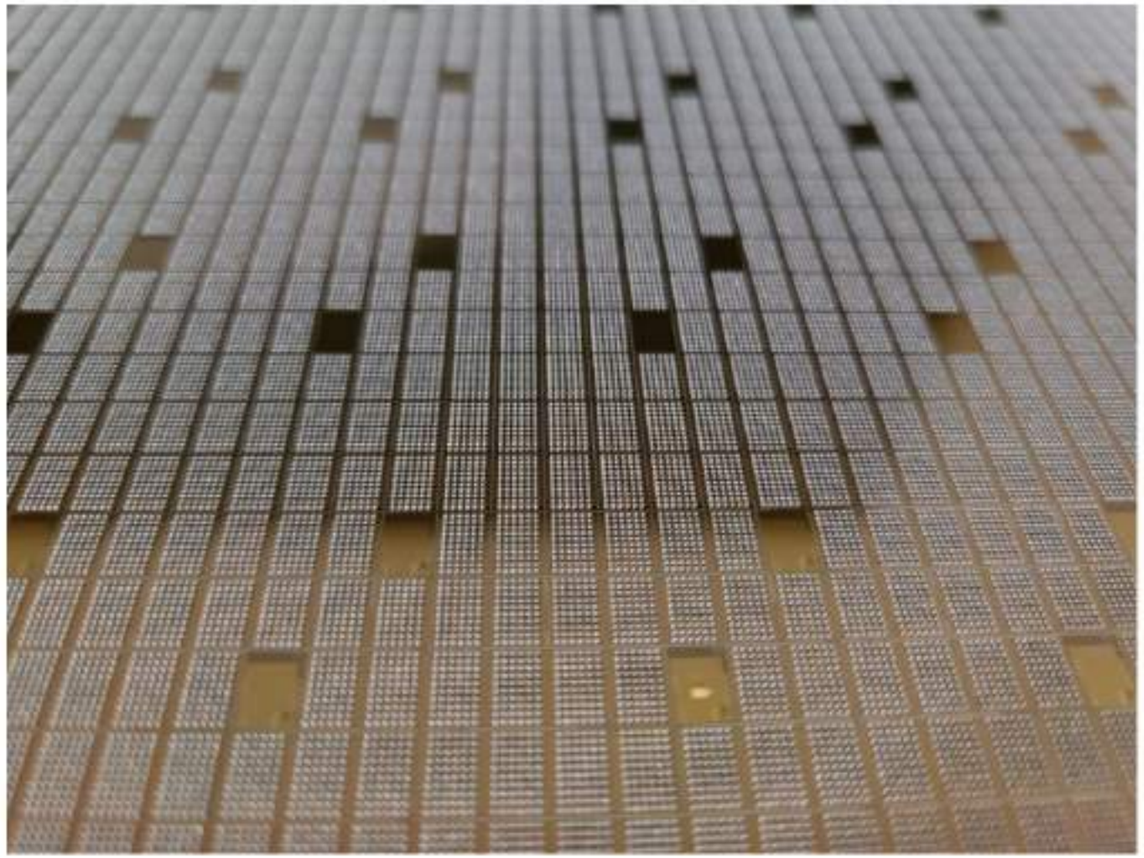
translated by Google Cette page a été traduite par l'API Cloud Translation.

Switch to English

# Créer votre propre puce

Google a collaboré avec GlobalFoundries, SkyWater Technology et Efabless pour fournir des kits de conception de processus (PDK) et des chaînes d'outils Open Source afin que n'importe quel développeur puisse créer des conceptions de silicium manufacturables.

Tous les deux mois, vous pouvez envoyer vos conceptions Open Source pour les inclure dans le programme de navette OpenMPW et avoir la possibilité de les faire fabriquer sans frais.

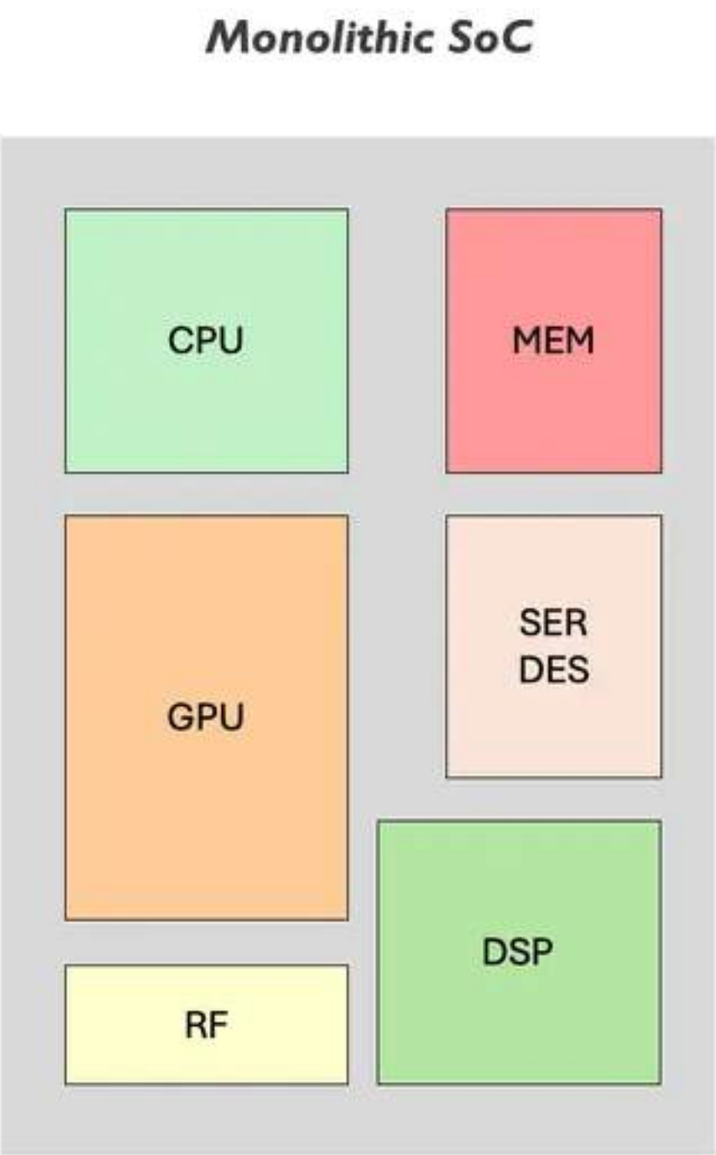


Google Silicon

# System on Chip (SoC)

When CPU and FPGA are both needed :

heterogenous computing (CPU, GPU, FPGA) on a single die




IP Softcores
Intellectual Property :

a predesigned subcircuit for use in larger designs

open-source v proprietary

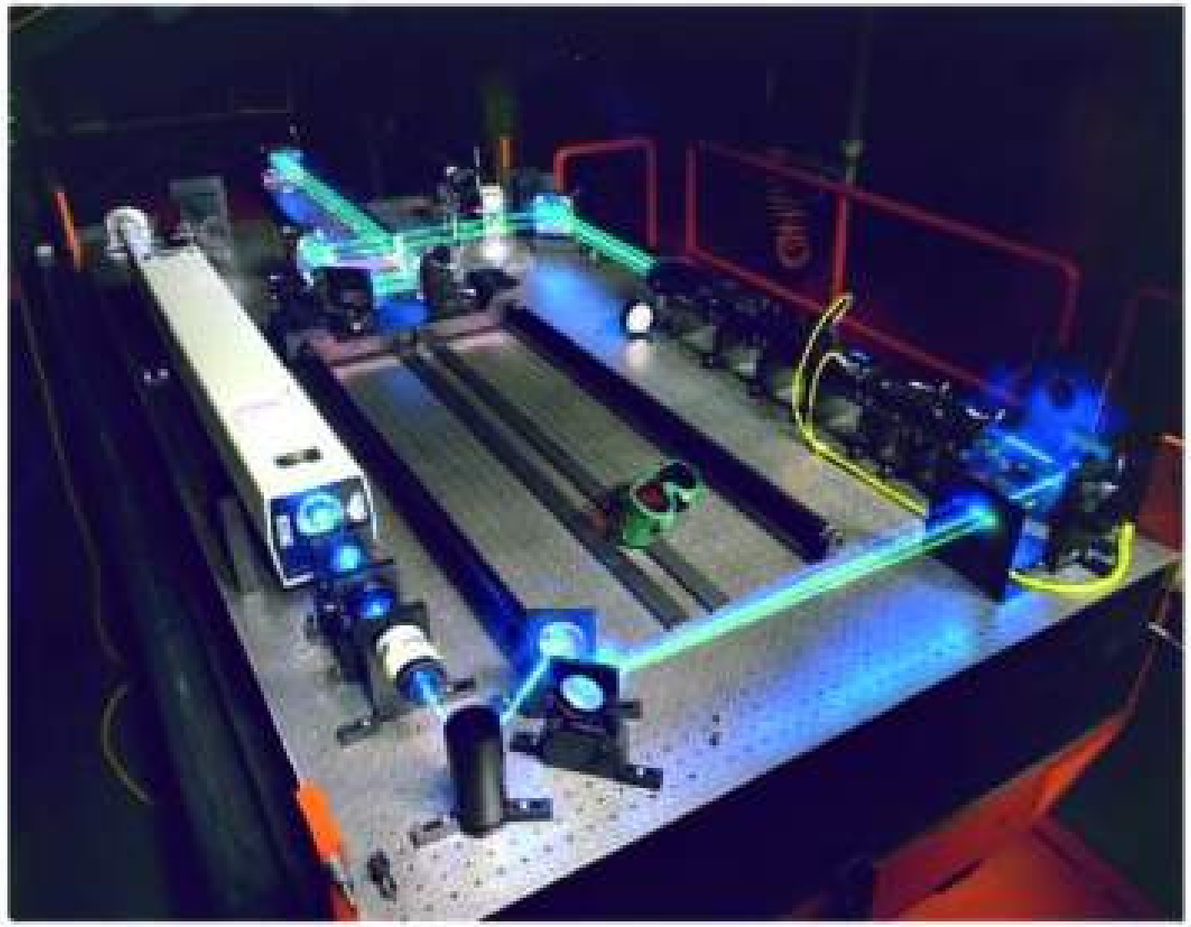
2D and 3D chiplets

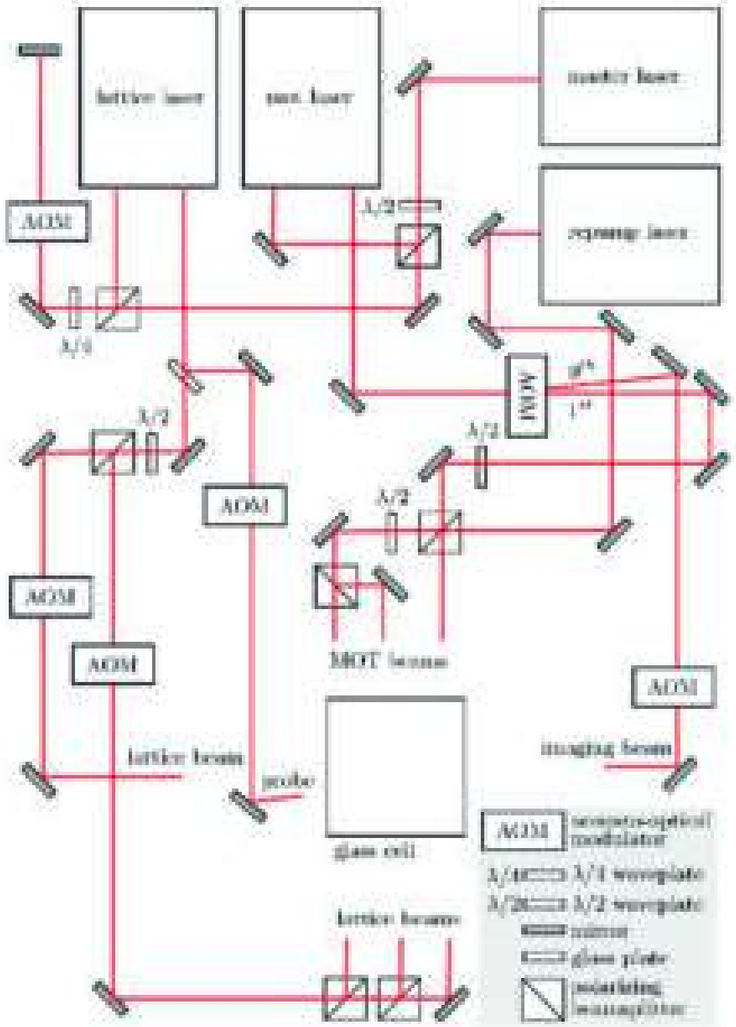
# FPGA superpower: parallelism

144 PINS ON THE HX1K !!

https://www.youtube.com/watch?v=IvUU8joBb1Q

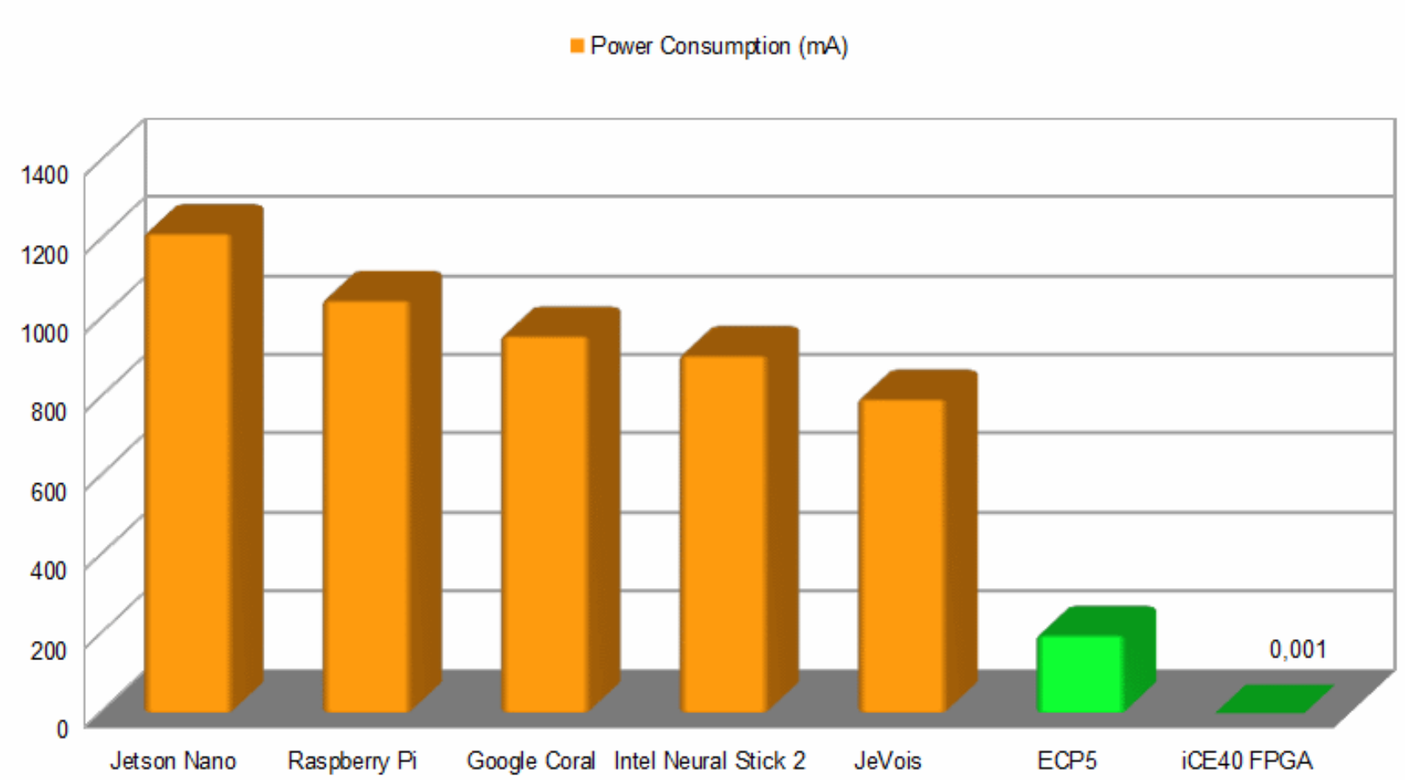
# FPGA superpower: hi thruput lo latency





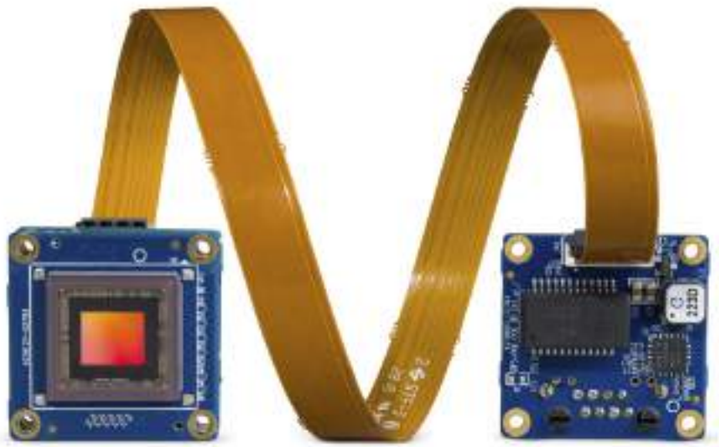
# FPGA superpower: Low power

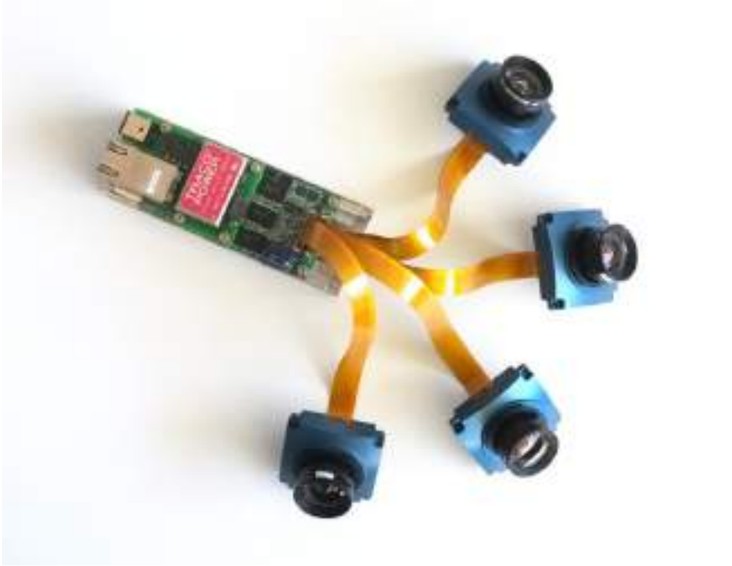
## Binary Neural Networks (BNNs) w FPGA

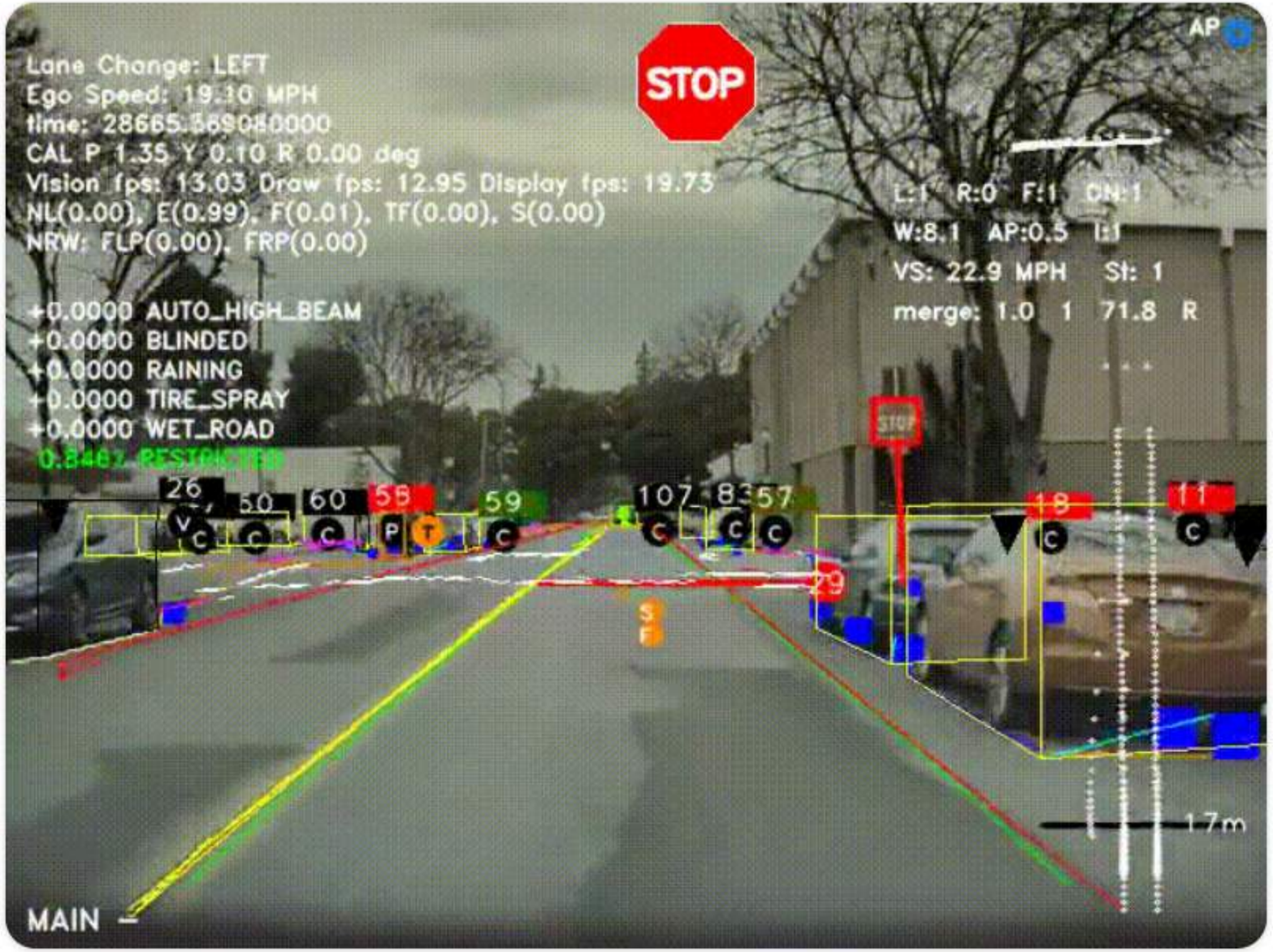



https://qengineering.eu/deep-learning-with-fpga-aka-bnn.html

# FPGA superpower:
# Machine Vision






https://tinyurl.com/59fzj8rx

# FPGA toolchain

## 1. SYNTHESIS

Converting from a high-level description of hardware to a more detailed, lower-level description. Does the specific FPGA have the resources?

## 2. PLACE AND ROUTE

Which specific Logic Blocks are connected? Which interconnects used?

## 3. SIMULATE W TESTBENCH

Does it generate the expected signals ?

## 4. FPGA CONFIGURATION

Generate the binary file, flash it to the on board memory.

# 1. SYNTHESIS

## Hardware Description Languages (HDLs)

VHDL has a verbose syntax and strong typing

constrains you to write good code

```
library IEEE;
use IEEE.STD_Logic_1164, all;

entity LATCH_IF_ELSEIF is
  port (En1, En2, En3, A1, A2, A3: in std_logic;
        Y: out std_logic);
end entity LATCH_IF_ELSEIF;

architecture RTL of LATCH_IF_ELSEIF is
begin
   process (En1, En2, En3, A1, A2, A3)
   begin
      if (En1 = '1') then
        Y <= A1;
      elseif (En2 = '1') then
        Y <= A2;
      elseif (En3 = '1') then
        Y <= A3;
      end if;
   end process;

end architecture RTL;
```

VHDL

VS

```
module LATCH_IF_ELSEIF (En1, En2, En3, A1, A2, A3, Y);
    input En1, En2, En3, A1, A2, A3;
    output Y;

    reg Y;

    always @(En1 or En2 or En3 or A1 or A2 or A3)
      if (En1 == 1)
        Y =A1;
     else if (En2 == 1)
        Y = A2;
     else if (En3 == 1)
        Y = A3;

end module
```

Verilog

verilog looks similar to C code

beware : lets you write code which will not work

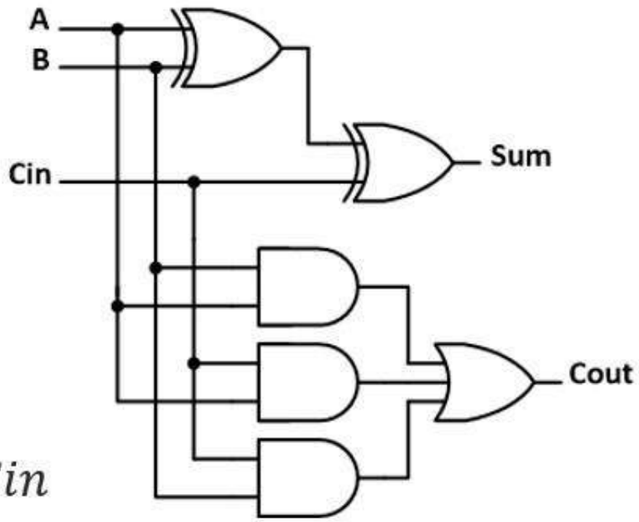
*SystemVerilog ≠ Verilog

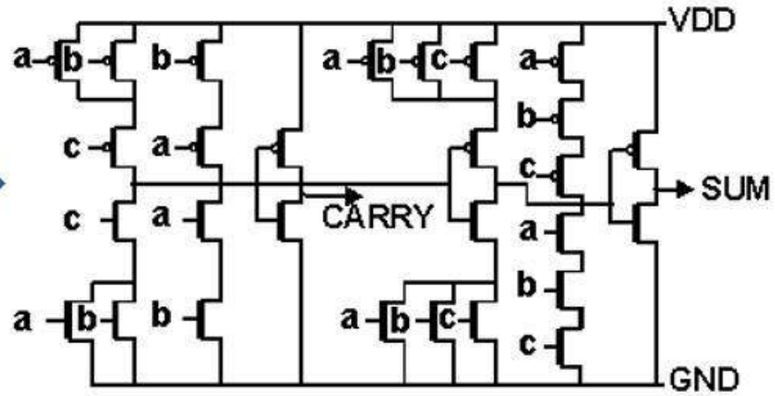
# 1. SYNTHESIS

verilog

register-transfer level (RTL)

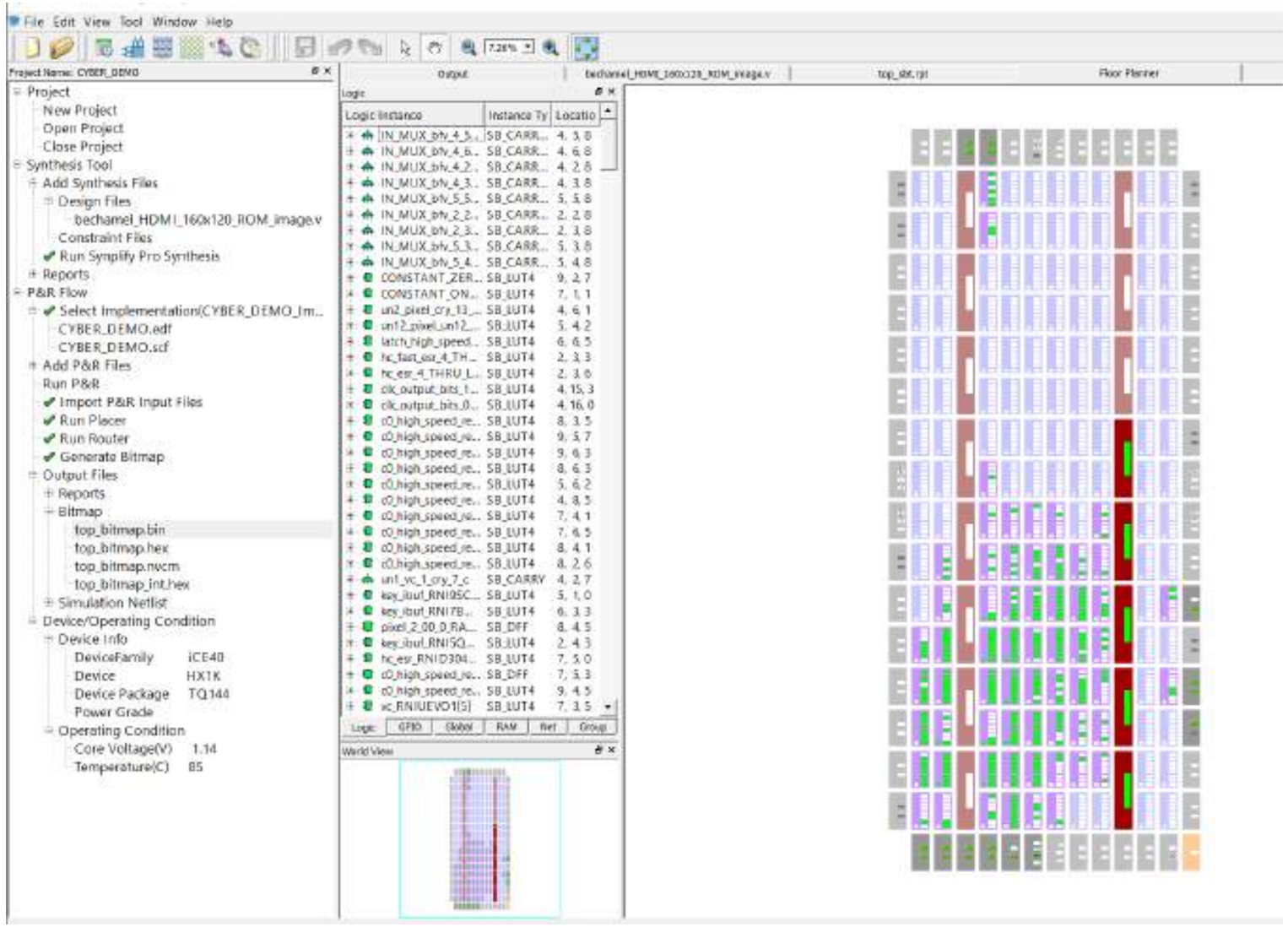
transistors



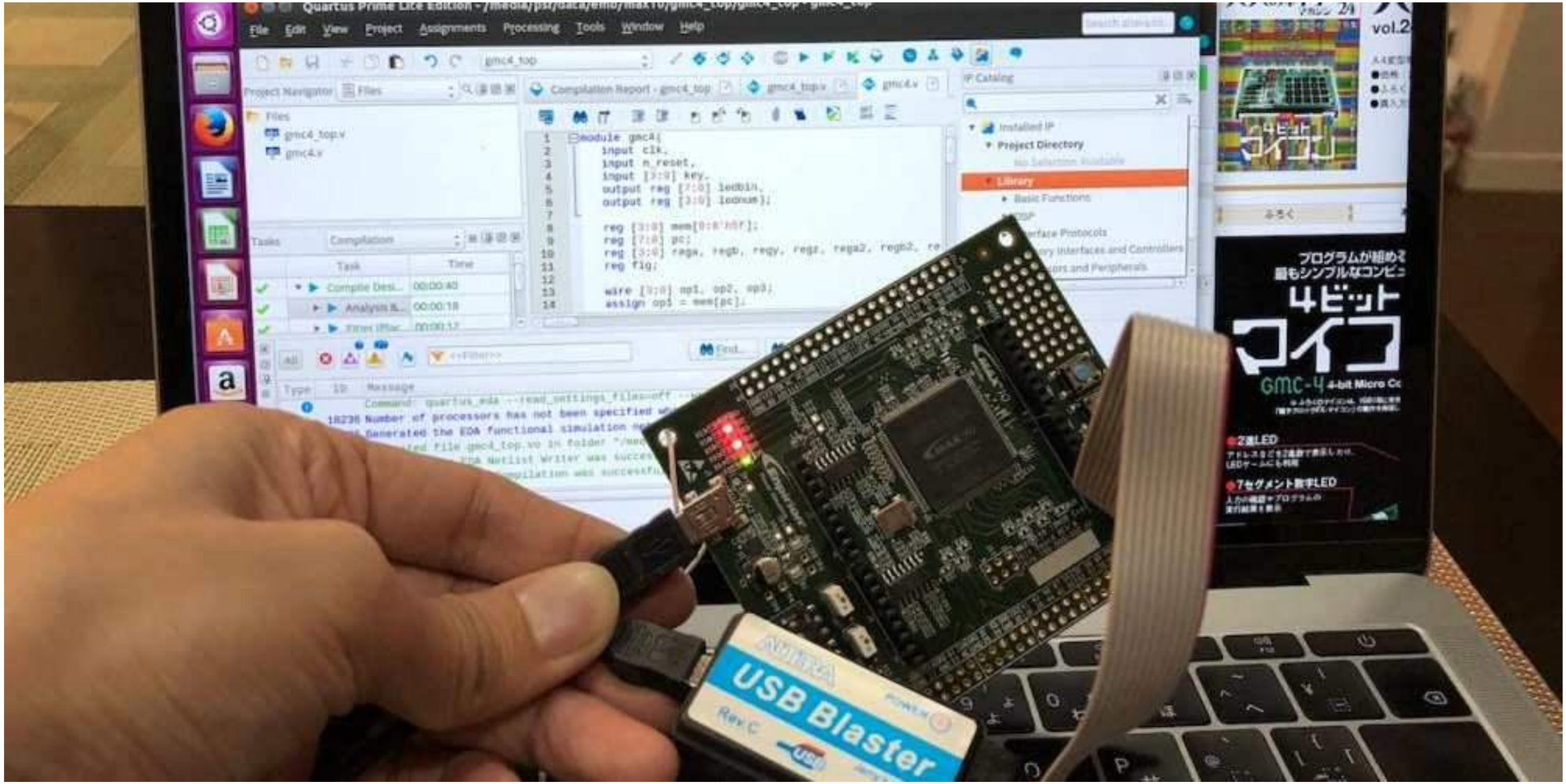

- $Sum = A \oplus B \oplus Cin$
- $Cout = AB + ACin + BCin$

# 2. PLACE AND ROUTE



# 3. SIMULATE W TESTBENCH

# 4. FPGA CONFIGURING



# FPGA Dev Board Manufacturers

1.


AMD Xilinx

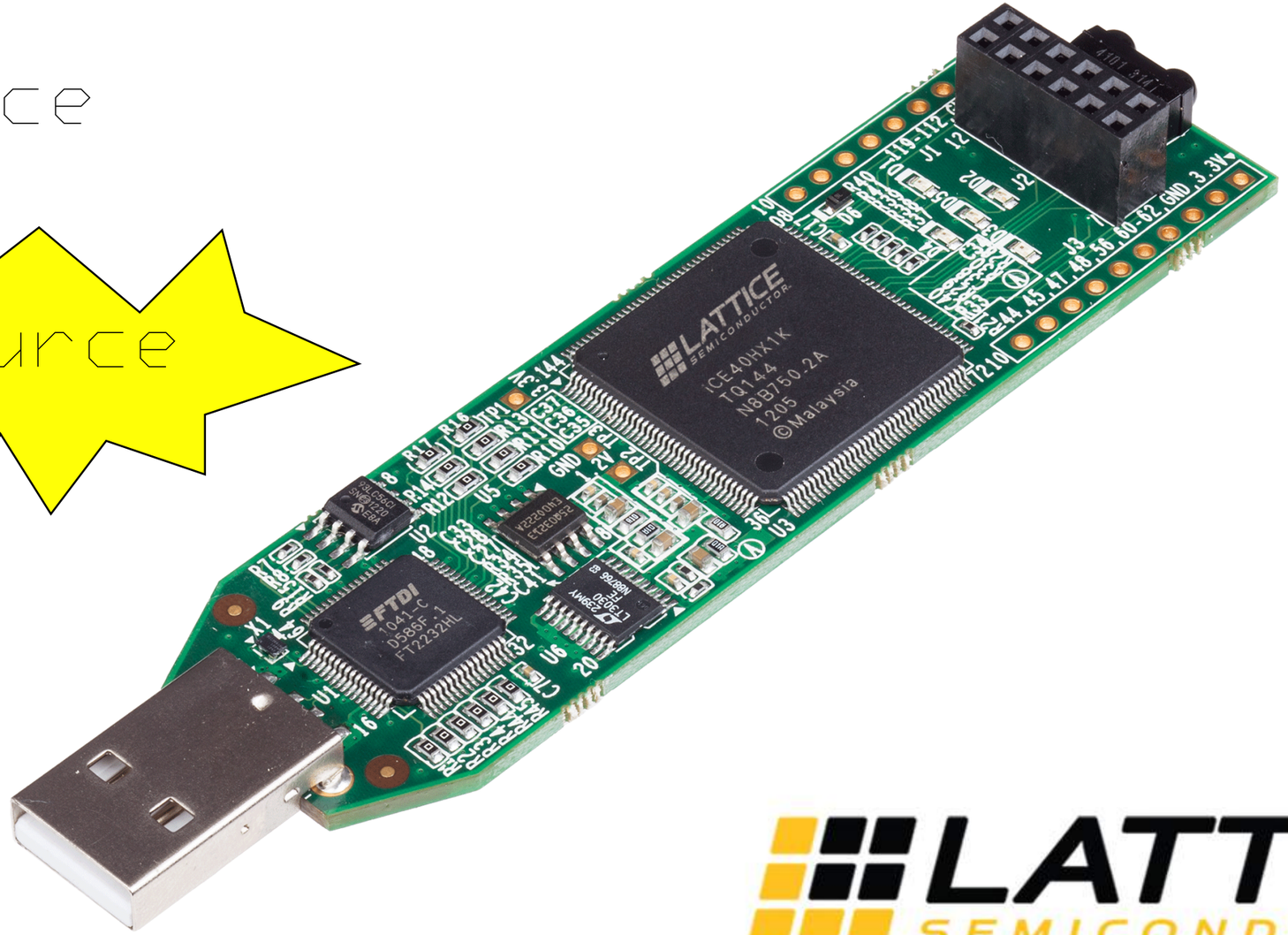
2.


Intel/Altera

# FPGA Dev Board Manufacturers

## 3. Lattice

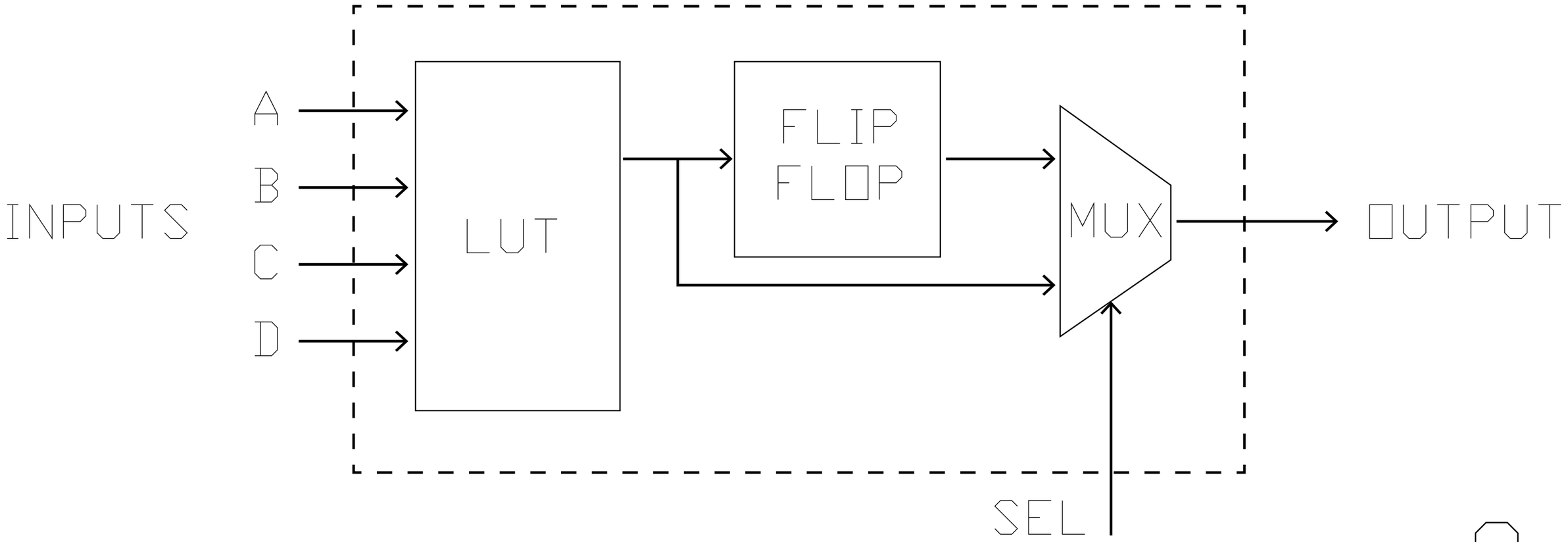
open-source




LATTICE SEMICONDUCTOR

# CONFIGURABLE LOGIC BLOCKS

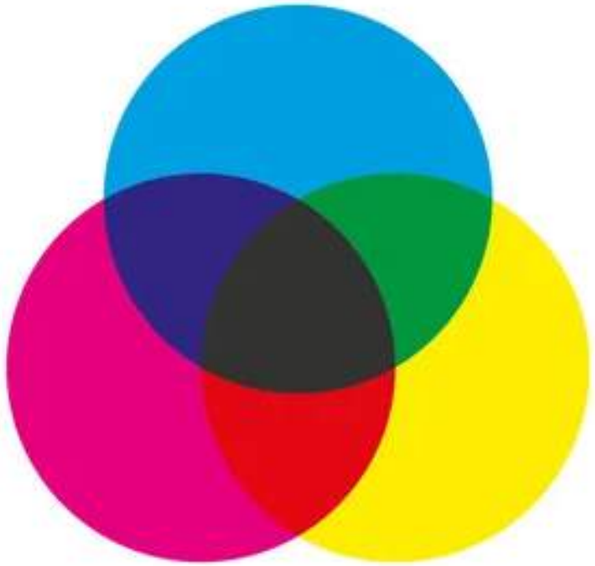
## SINGLE LOGIC CELL




×8

There are **two types** of digital logic we can implement in the FGPA...

## COMBINATIONAL



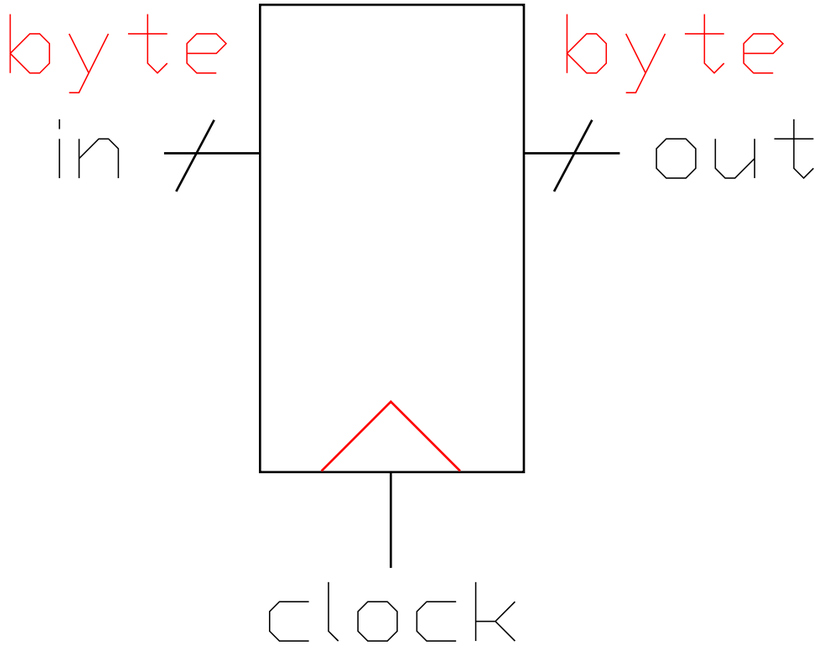
*Light travels a foot in one billionth of a second. Electric charge is about 90% as fast.

## SEQUENTIAL

# What do these mean ?

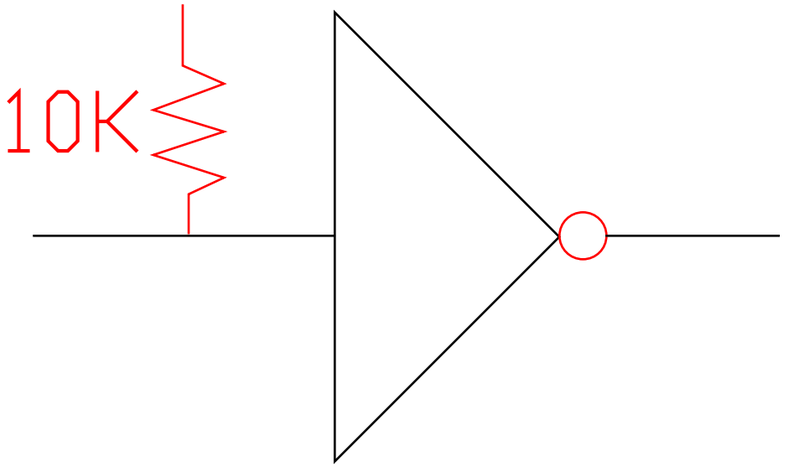
synchronous vs
asynchronous ?

«HIGH Z»

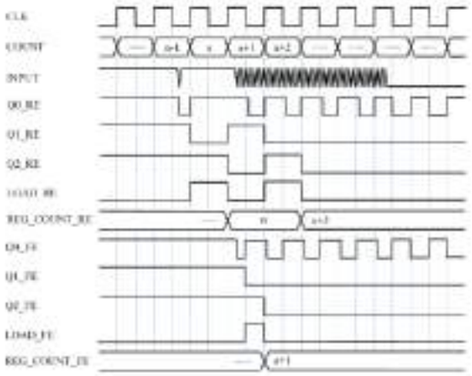


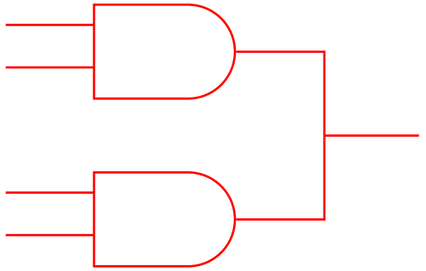

$\overline{\text{RESET}}$

!RESET



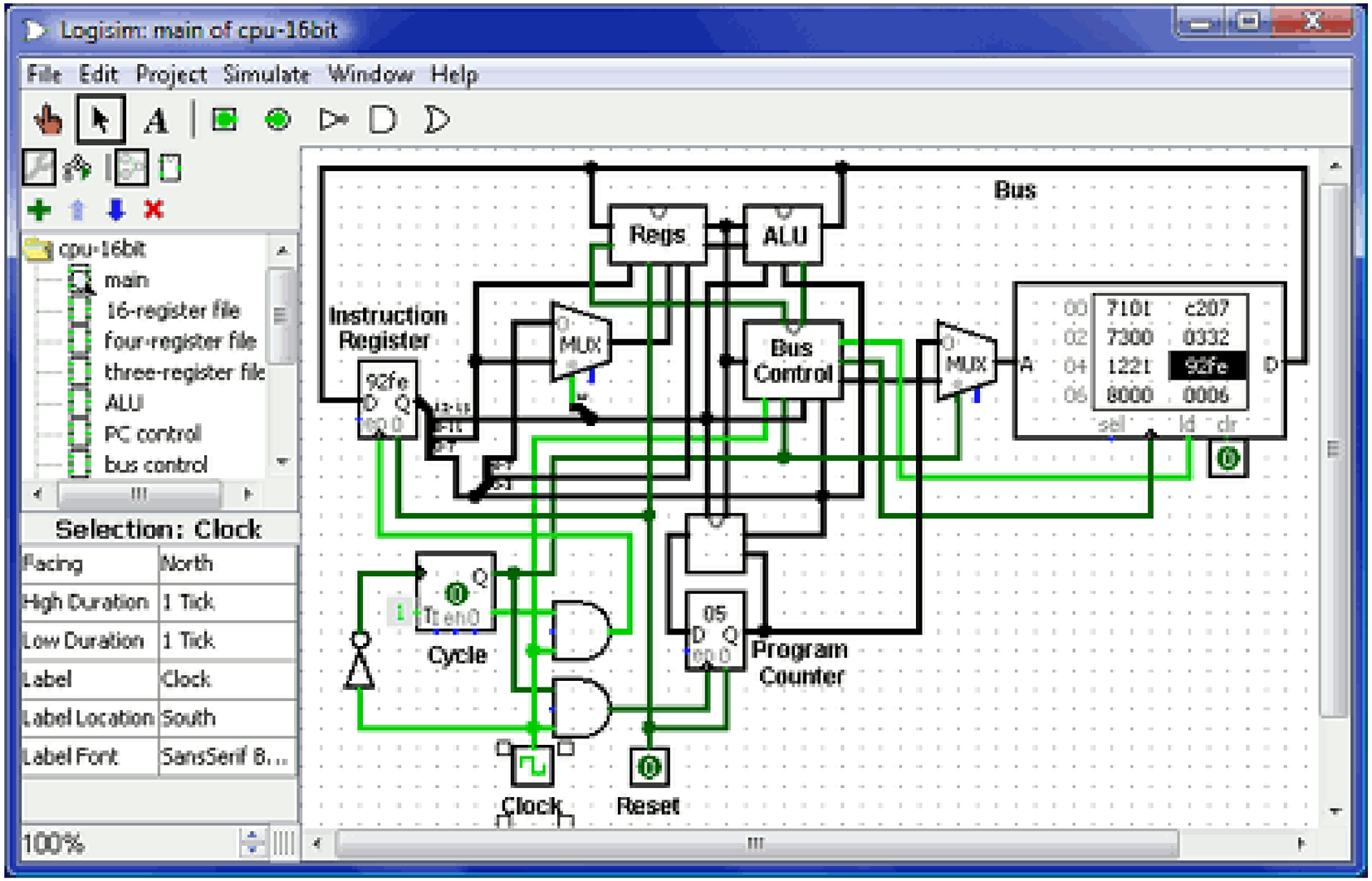

???





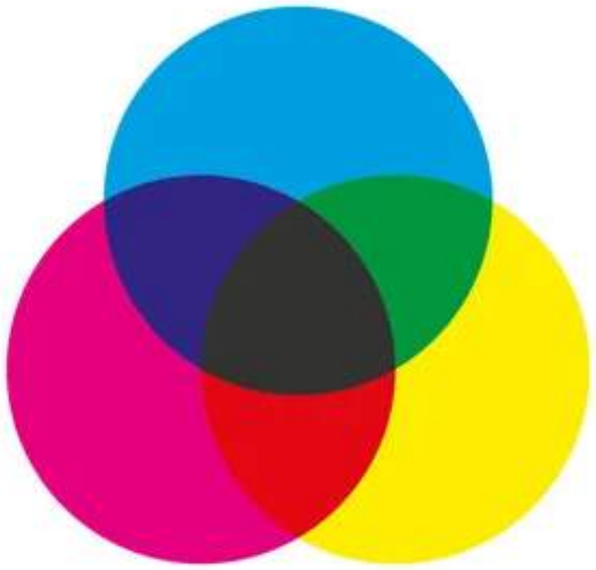
«FLOATING INPUT»

# DIGITAL ELECTRONICS REFRESHER




http://www.cburch.com/logisim/

# Combinatorial circuits



**No memory required:** These circuits don't "remember" past inputs; they work in real-time based only on the present input.

**Fast and straightforward:** The output is calculated instantly as soon as the input changes.

**Made up of logic gates:** Combinational circuits are primarily built using logic gates like AND, OR, and NOT.

ex : Multiplexer

https://www.jaroeducation.com/blog/difference-between-combinational-and-sequential-circuits/

# Sequential circuits

ex : Counter

**Memory-based:** Sequential circuits can store past inputs and use this information along with current inputs to determine the output.

**Time-dependent:** They often work in sync with a clock signal, which controls when the circuit updates its state.

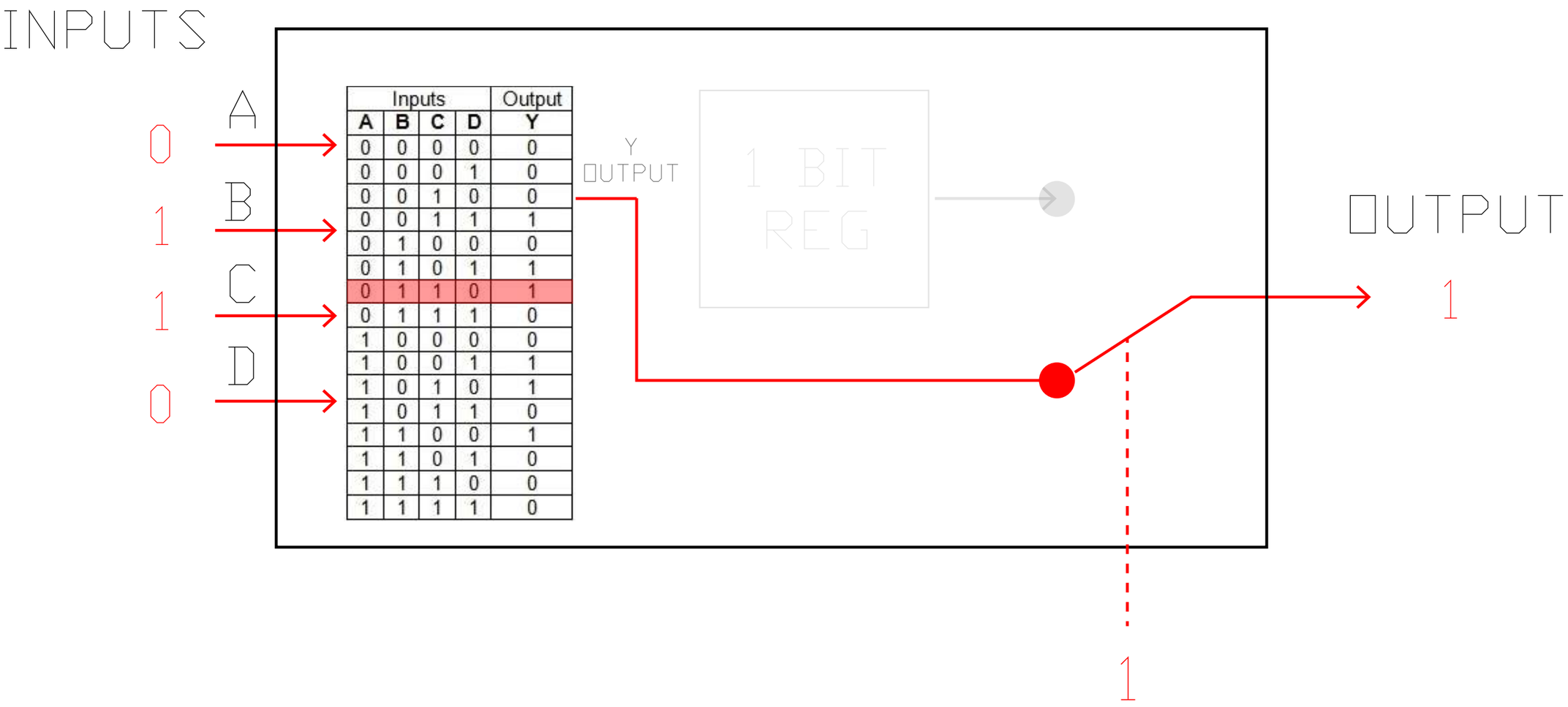
**Built with flip-flops:** The basic building blocks of digital electronics sequential circuits are flip-flops, which are used to store data.

ex :
Shift register
RAM

https://www.jaroeducation.com/blog/difference-between-combinational-and-sequential-circuits/

# CONFIGURABLE LOGIC BLOCK

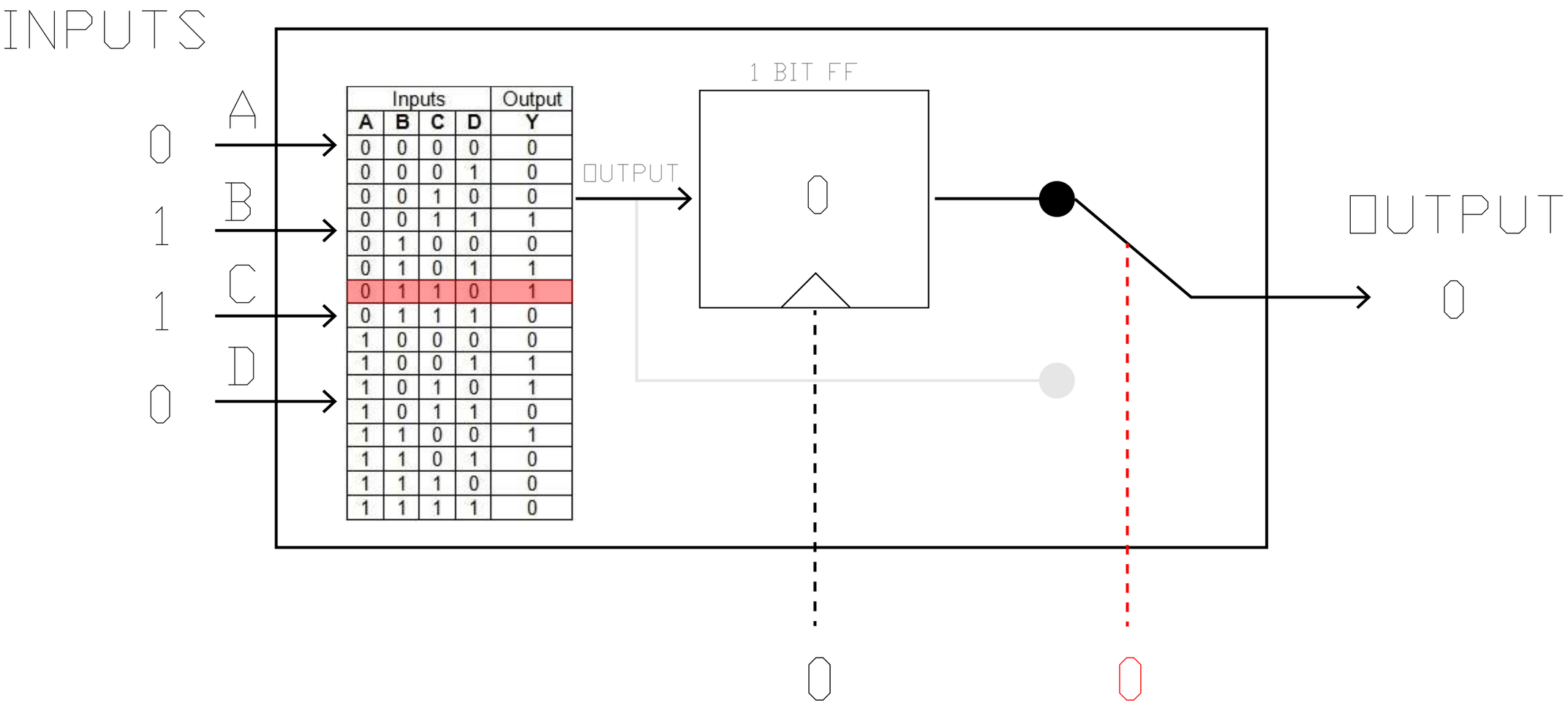
## COMBINATIONAL MODE




| Inputs | | | | Output |
|---|---|---|---|---|
| A | B | C | D | Y |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

# CONFIGURABLE LOGIC BLOCK

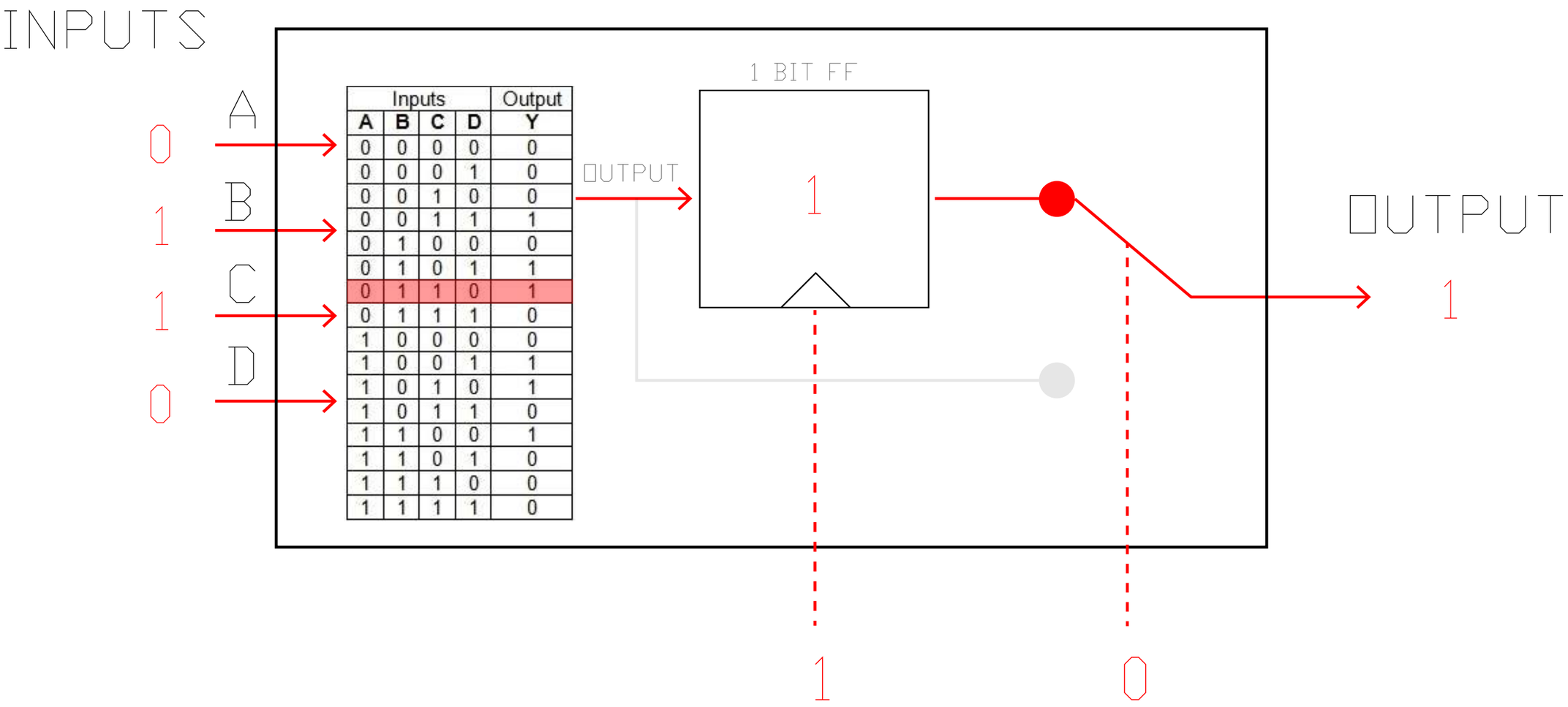
## CLOCKED MODE




| Inputs | | | | Output |
|---|---|---|---|---|
| A | B | C | D | Y |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

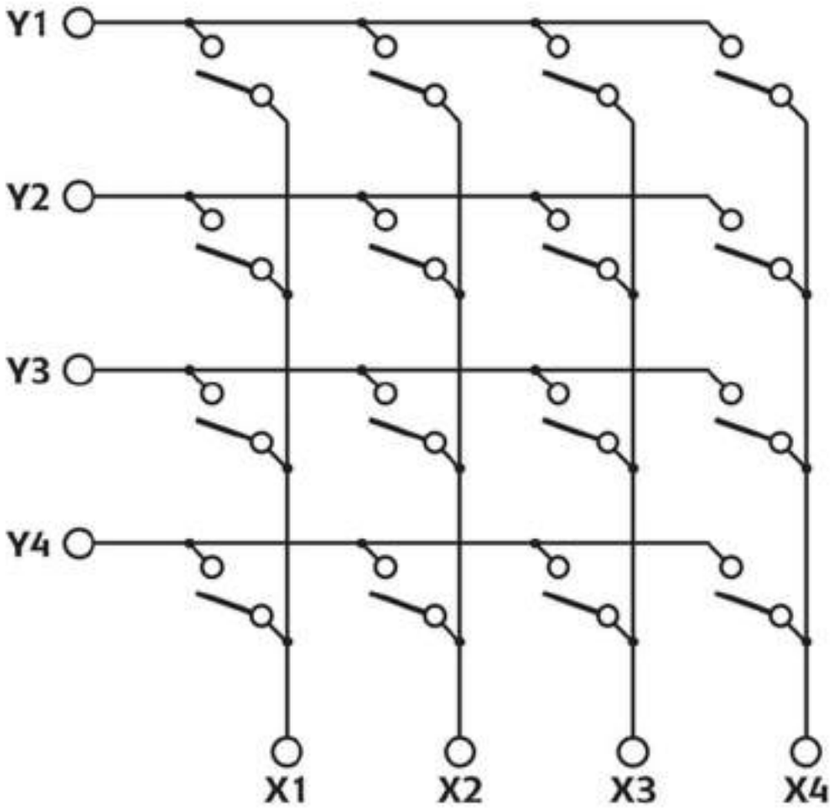
# CONFIGURABLE LOGIC BLOCK

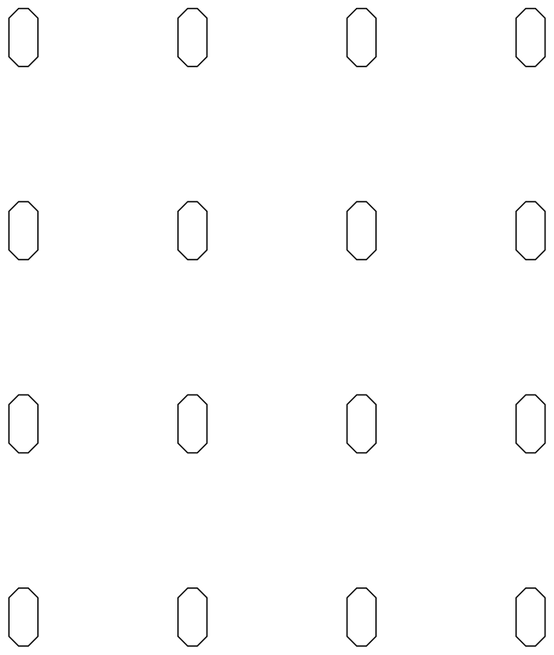
## CLOCKED MODE




| Inputs | | | | Output |
|---|---|---|---|---|
| A | B | C | D | Y |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

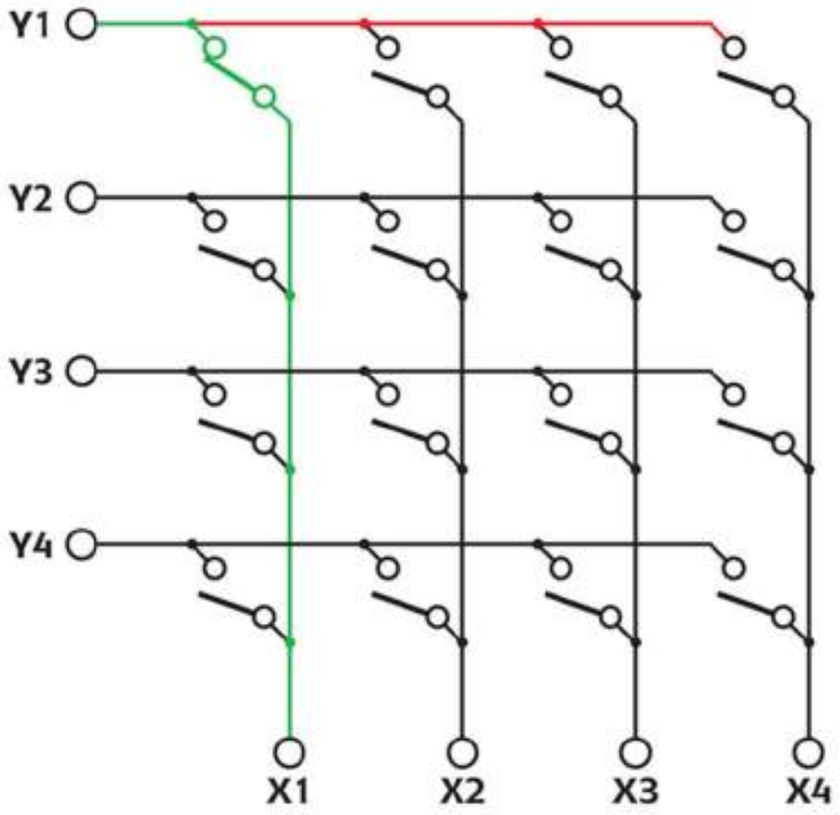
# CONFIGURABLE INTERCONNECT



BINARY REP

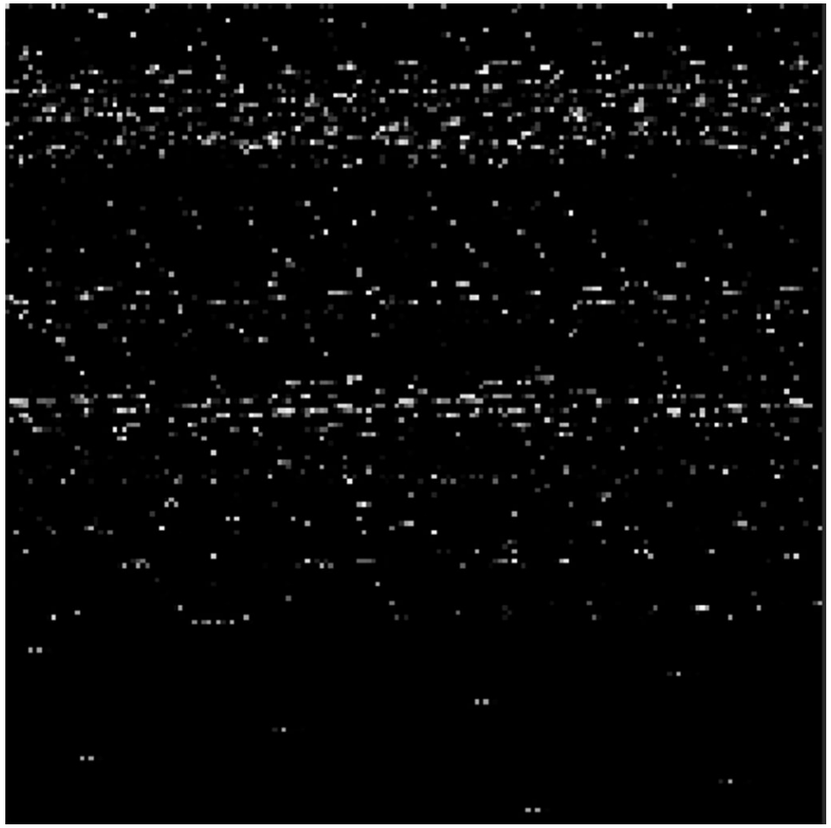
# CONFIGURABLE INTERCONNECT




| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

BINARY REP

# ACTUAL BINARY FILE

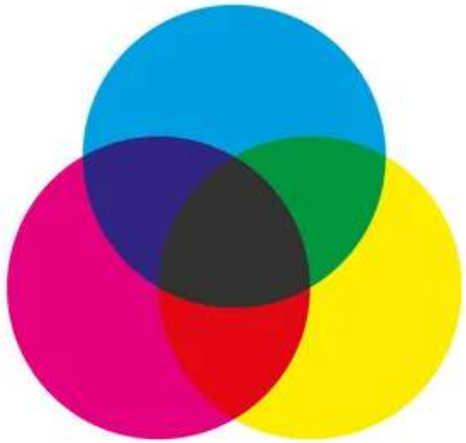


Configures interconnects
and logic blocks !

always @(*) or assign

=

COMBINATIONAL



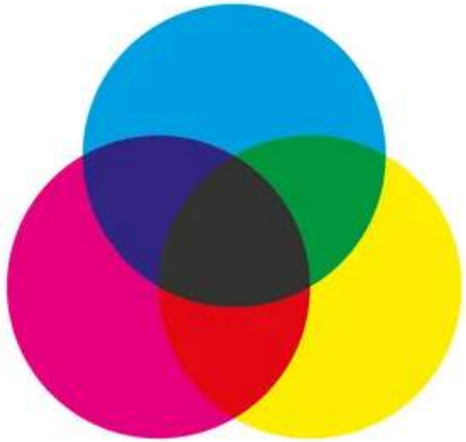
always @(posedge clk)

<=

CLOCKED

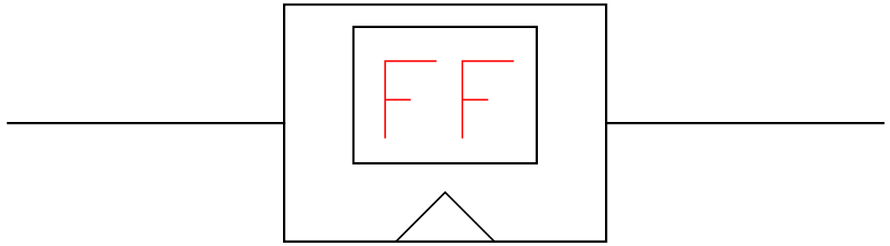
wire

VALUE «DRIVEN»



reg

has a value retained in memory until changed by a subsequent assignment*




*but doesn't necessarily use flip flops

# Problem sets

## Getting Started

- Getting Started
- Output Zero

## Verilog Language

### Basics

- Simple wire
- Four wires
- Inverter
- AND gate
- NOR gate
- XNOR gate
- Declaring wires
- 7458 chip

### Vectors

- Vectors
- Vectors in more detail
- Vector part select
- Bitwise operators
- Four-input gates
- Vector concatenation operator
- Vector reversal 1
- Replication operator
- More replication

### Modules: Hierarchy

- Modules
- Connecting ports by position
- Connecting ports by name
- Three modules
- Modules and vectors
- Adder 1
- Adder 2
- Carry-select adder
- Adder-subtractor

### Procedures

Procedures include **always**, initial, task, and function blocks. Procedures allow sequential statements (which cannot be used outside of a procedure) to be used to describe the behaviour of a circuit.

- Always blocks (combinational)
- Always blocks (clocked)
- If statement
- If statement latches
- Case statement

Contents



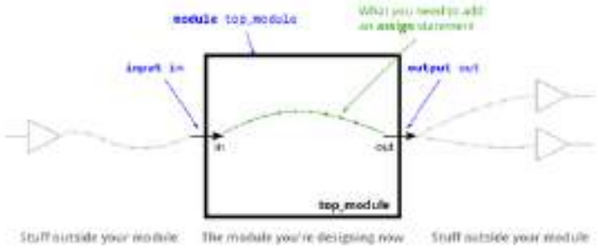
# Wire

Create a module with one input and one output that behaves like a wire.

Unlike physical wires, in Verilog (and other signals) in Verilog are directional. This means information flows in only one direction, from (usually one) source to one sink. (This source is also often called a driver that drives a value onto a wire). In a Verilog "continuous assignment" (`assign left_side = right_side;`), the value of the signal on the right side is driven onto the wire on the left side. The assignment is "continuous" because the assignment continues all the time even if the right sides value changes. A continuous assignment is not a one-time event.

The ports on a module also have a direction (usually input or output). An input port is driven by something from outside the module, while an output port drives something outside. When viewed from inside the module, an input port is a driver or source, while an output port is a sink.

The diagram below illustrates how each part of the circuit corresponds to each bit of Verilog code. The module and port declarations create the black portions of the circuit. Your task is to create a wire (in green) by adding an `assign` statement to connect `in` to `out`. The parts outside the box are not your concern, but you should know that your circuit is tested by connecting signals from our test harness to the ports on your `top_module`.




In addition to continuous assignments, Verilog has three other assignment types that are used in procedural blocks, two of which are synthesizable. We won't be using them until we start using procedural blocks.

**Expected solution length:** Around 1 line

## Module Declaration

```
module top_module( input in, output out );
```

Hint...

## Write your solution here

```
module top_module( input in, output out );

endmodule
```

Submit | Submit (new window)

## Solution

https://hdlbits.01xz.net/wiki/Main_Page

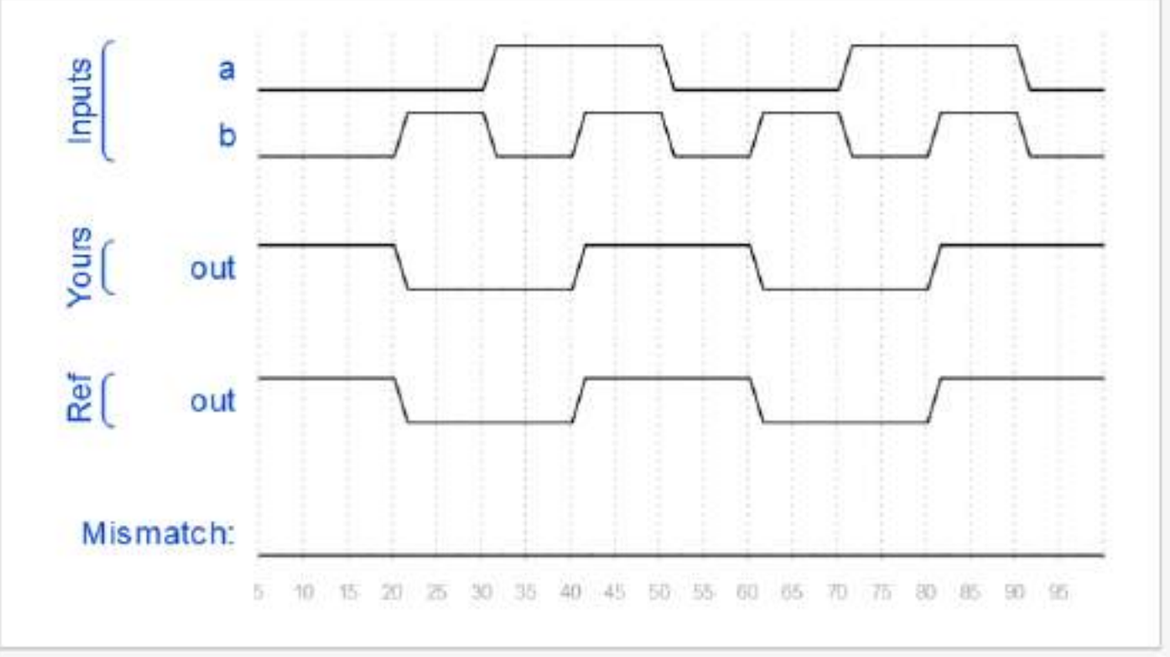
# Status: Success!

You have solved 47 problems. See my progress...

## Timing diagrams for selected test cases

These are timing diagrams from some of the test cases we used. They may help you debug your circuit. The diagrams show inputs to the circuit, outputs from your circuit, and the expected reference outputs. The "Mismatch" trace shows which cycles your outputs don't match the reference outputs (0 = correct, 1 = incorrect).

**XNOR gate**



filename

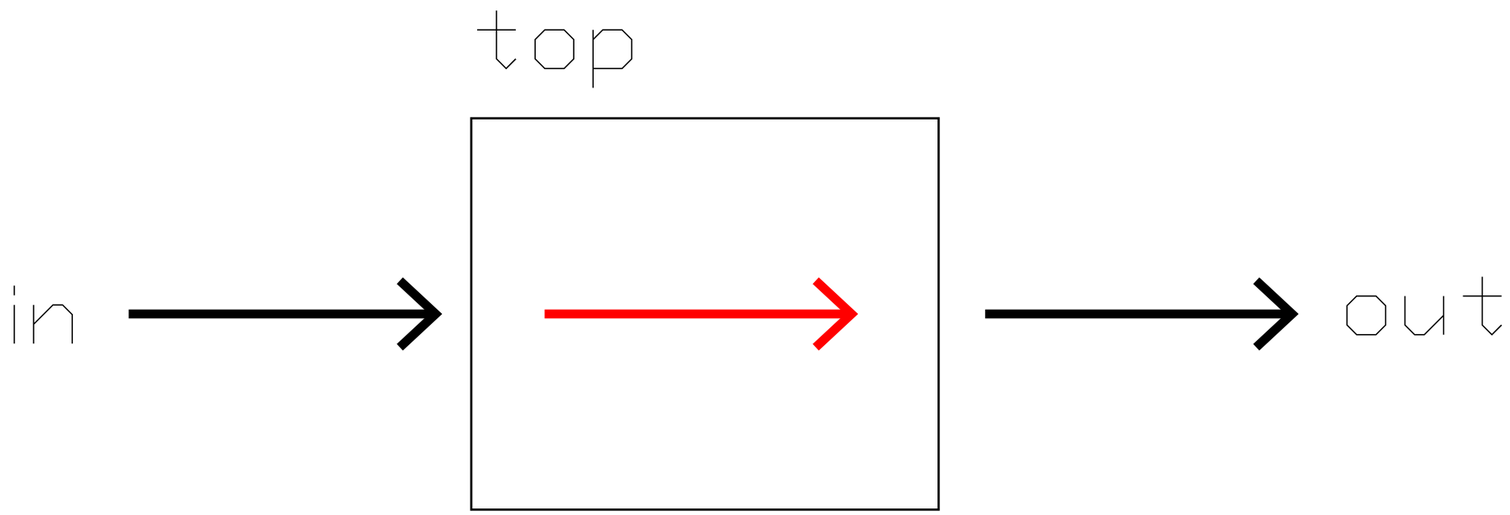
list inputs and outputs

(default to wire type)

```
module top ( input in, output out );

    assign out = in;

endmodule
```

the section that does something

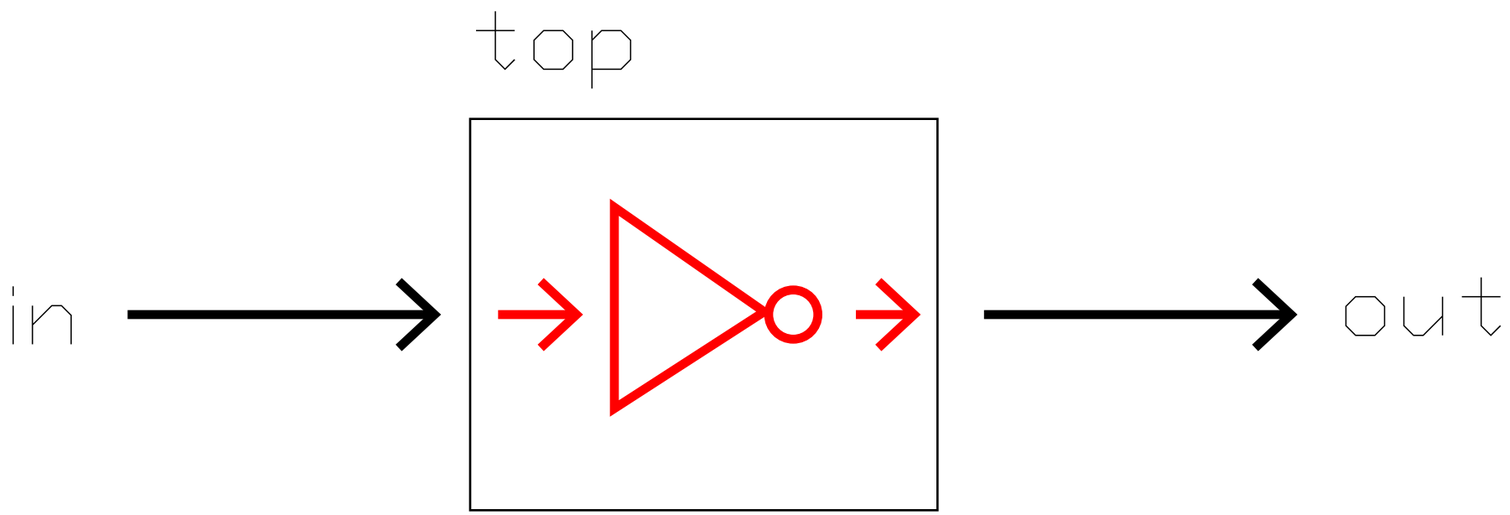
assign defines logic linking inputs and outputs



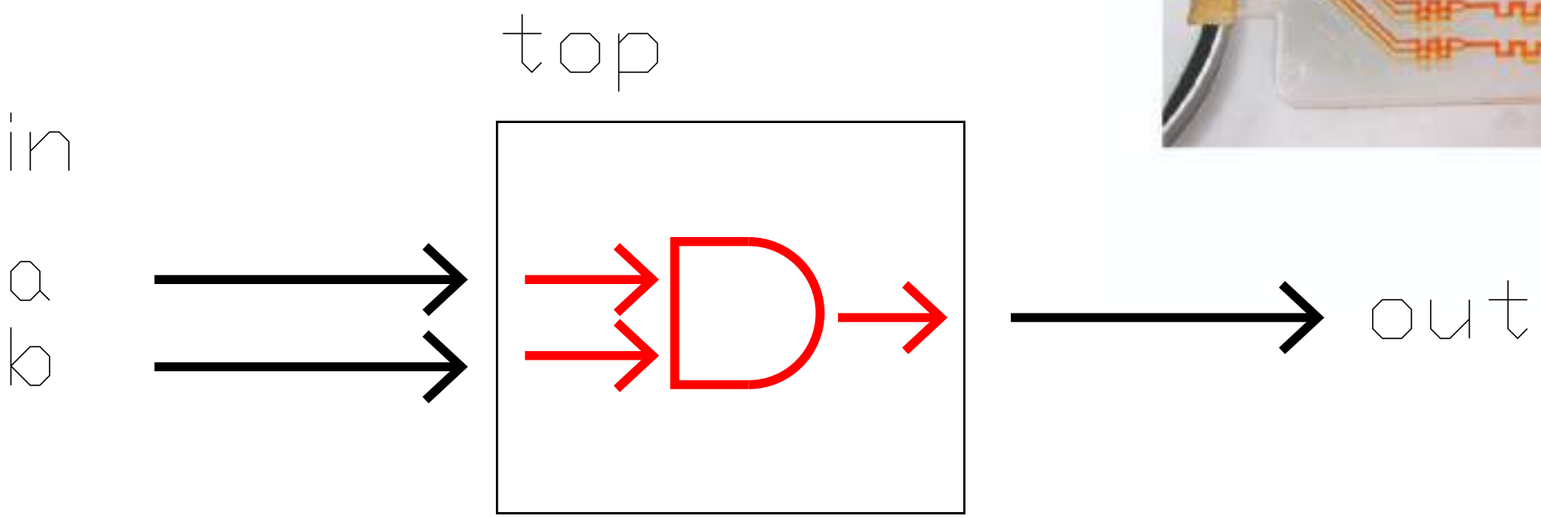

**assign **continuously drives** in into out**

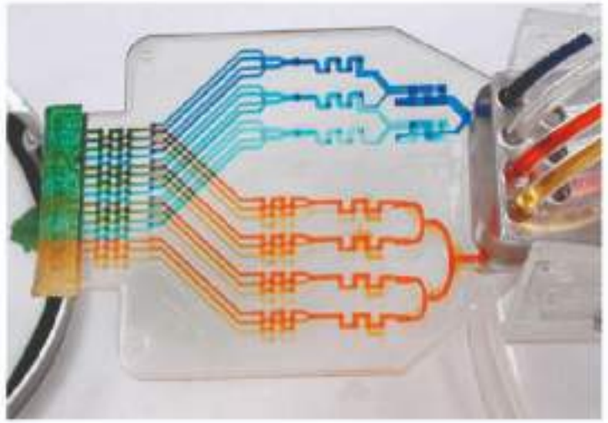
**there is no memory here**

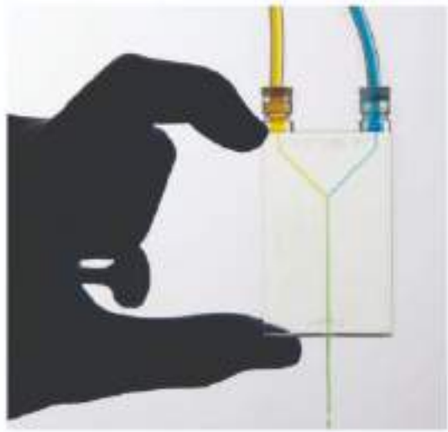
```
module top ( input in, output out );

    assign out = ! in;

endmodule
```



```
module top ( input a, input b, output out );

  assign out = a&b;

endmodule
```





microfluidics

# VECTORS

wire/
reg

↓

```
type [upper:lower] vector_name;
```

e.g.

```
reg [7:0] a;

assign a [7:0] = 8'b00001111; // name then index

assign a [2] = 1'b0; // only sets position 2 bit !
```

*Verilog is case sensitive

# VECTORS v. SCALARS

```
reg r_A = 1'b1;
reg r_B = 1'b0;

reg [3:0] r_X = 4'b0101;
reg [3:0] r_Y = 4'b1100;
wire [3:0] w_AND_VECTOR, w_OR_VECTOR,  w_XOR_VECTOR, w_NOT_VECTOR;

assign w_AND_SCALAR = r_A & r_B;
assign w_OR_SCALAR  = r_A | r_B;
assign w_XOR_SCALAR = r_A ^ r_B;
assign w_NOT_SCALAR = ~r_A;

assign w_AND_VECTOR = r_X & r_Y;
assign w_OR_VECTOR  = r_X | r_Y;
assign w_XOR_VECTOR = r_X ^ r_Y;
assign w_NOT_VECTOR = ~r_X;
```

```
# AND of 1 and 0 is 0
# OR  of 1 and 0 is 1
# XOR of 1 and 0 is 1
# NOT of 1 is 0
# AND of 0101 and 1100 is 0100
# OR  of 0101 and 1100 is 1101
# XOR of 0101 and 1100 is 1001
# NOT of 0101 is 1010
```

# ~verilog gotchas~


use counters to increment but **not** in the form of a for loop and also, **++ doesn't exist** in verilog !

**use if/else** or a tertiary op (it's like an AND gate)

ex. paint_r = (square) ? 4'hF : 4'h1;

**use case** (it's like a MUX) if you have several paths

beware the **begin** and **end** tags, format with indentation !

# DAY 2

# HDL pitfalls

## 1. Latches

Latch problems in Verilog coding typically arise when a signal is unintentionally inferred as a latch due to **incomplete assignments** in always blocks. To avoid these issues, here are several strategies:

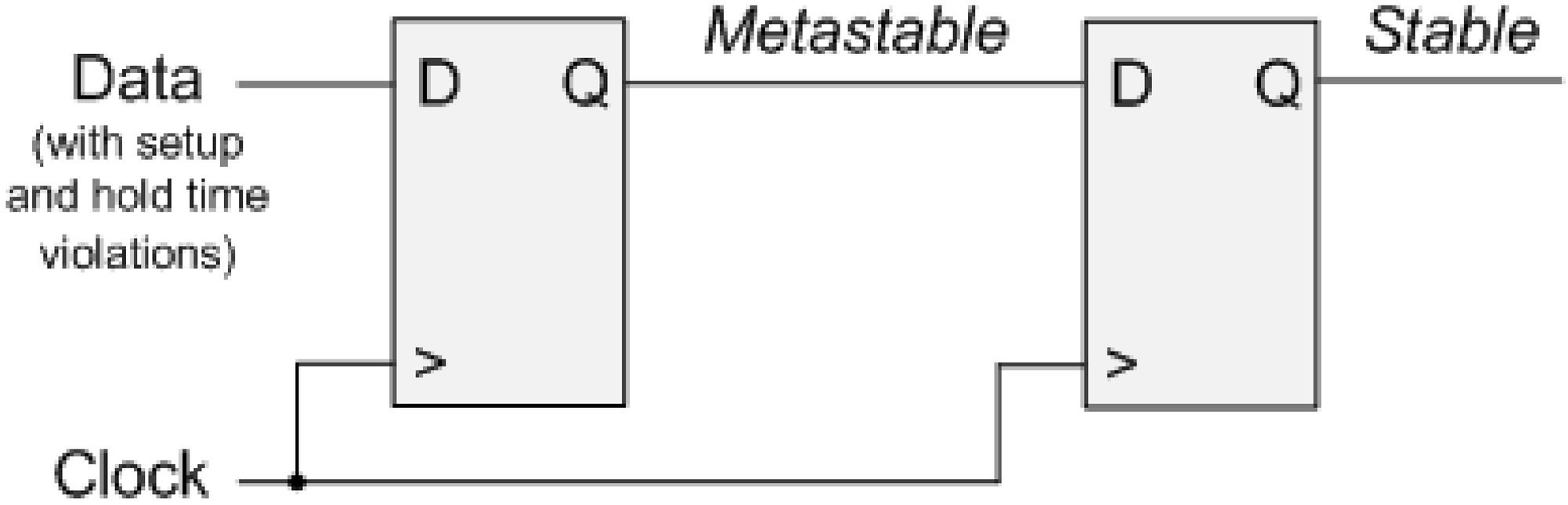
1. Complete Assignments in Always Blocks

Ensure that **all possible conditions** for a signal are covered in your always block. If a signal is not assigned in all paths, a latch may be inferred. For example:

```
   always @(posedge clk or posedge reset)
begin
      if (reset) begin
          signal <= 0; // Reset condition
      end else begin
          signal <= input_signal; // Always
assign a value
      end
   end
```

# 3. Metastability

A metastable condition occurs when setup or hold times are violated.

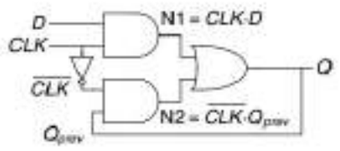
# 2. Race conditions

A race condition is any case where the outcome depends on unknown / uncontrolled event ordering.

Asynchronous circuits are infamous for having race conditions where the behavior of the circuit depends on **which of two paths through logic gates is fastest.**

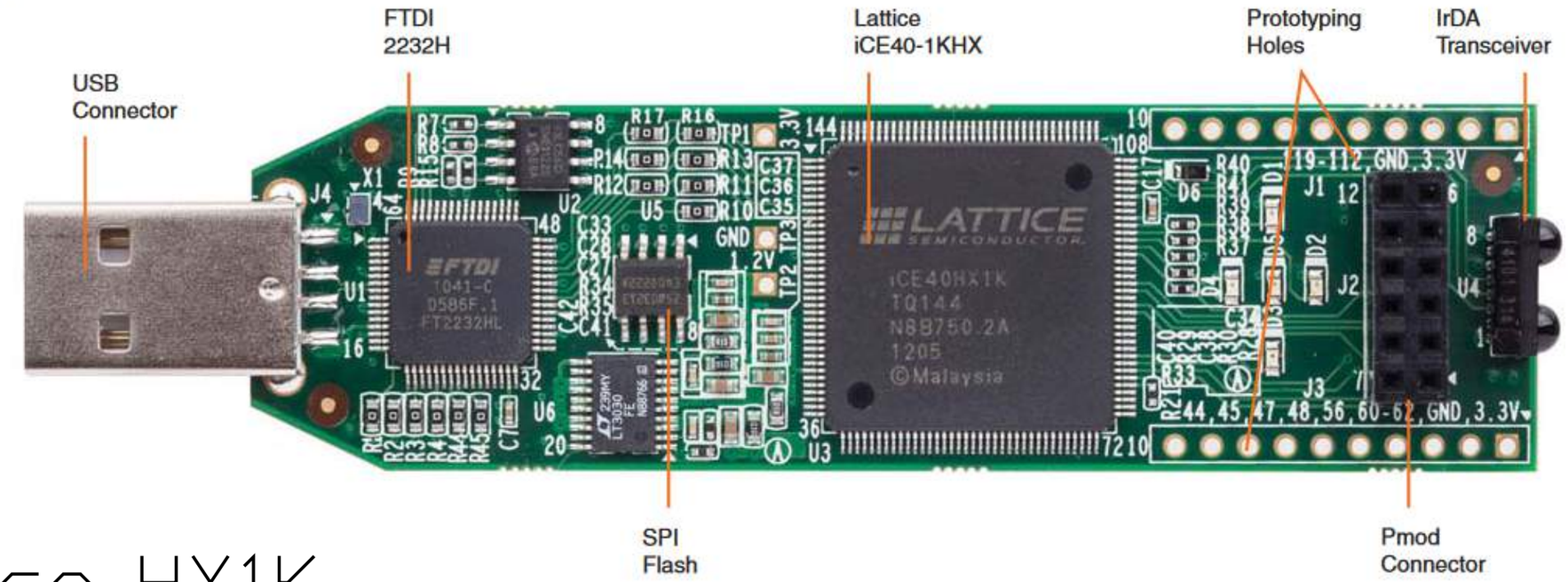
| CLK | D | $Q_{prev}$ | Q |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

$$Q = CLK \cdot D + \overline{CLK} \cdot Q_{prev}$$




Suppose CLK = D = 1. The latch is transparent and passes D through to make Q = 1. Now, CLK falls. The latch should remember its old value, keeping Q = 1. However, suppose the delay through the inverter from CLK to is rather long compared to the delays of the AND and OR gates. Then nodes N1 and Q may both fall before rises. In such a case, N2 will never rise, and Q becomes stuck at 0.

# iCE40




Lattice HX1K

144 I/O PINS

1280 LOGIC BLOCKS

12 MHz

≈6 euros

## iCE40HX1K-VQ100

Les images sont fournies à titre indicatif
Voir les caractéristiques du produit

Partager

| | |
|---|---|
| **N° Mouser :** | 842-ICE40HX1K-VQ100 |
| **N° de fab. :** | iCE40HX1K-VQ100 |
| **Fab. :** | Lattice |
| **N° client:** | N° client |
| **Description :** | FPGA - Réseau prédiffusé programmable par l'utilisateur iCE40HX 1280 LUTs 1.2V Ultra Low-Power |
| **Fiche technique:** | iCE40HX1K-VQ100 Fiche technique |
| **Modèle de ECAO:** | Symbole PCB, empreinte et modèle 3D |

Téléchargez gratuitement le chargeur de bibliothèque pour convertir ce fichier pour votre outil ECAD. En savoir plus sur le modèle ECAD.

| | |
|---|---|
| **Plus d'informations** | En savoir plus à propos de Lattice iCE40HX1K-VQ100 |

Comparer un produit

Ajouter au projet | Ajouter des notes

## En stock: 721

| | |
|---|---|
| **Stock:** | 721 Expédition possible immédiatement |
| **Délai usine :** | 16 Semaines |
| **Entrez la quantité:** | Minimum : 1 Multiples : 1 — Acheter |

### Prix (EUR)

| Qté. | Prix unitaire | Ext. Prix |
|---|---|---|
| 1 | 6,98 € | 6,98 € |
| 25 | 6,13 € | 153,25 € |
| 90 | 5,87 € | 528,30 € |

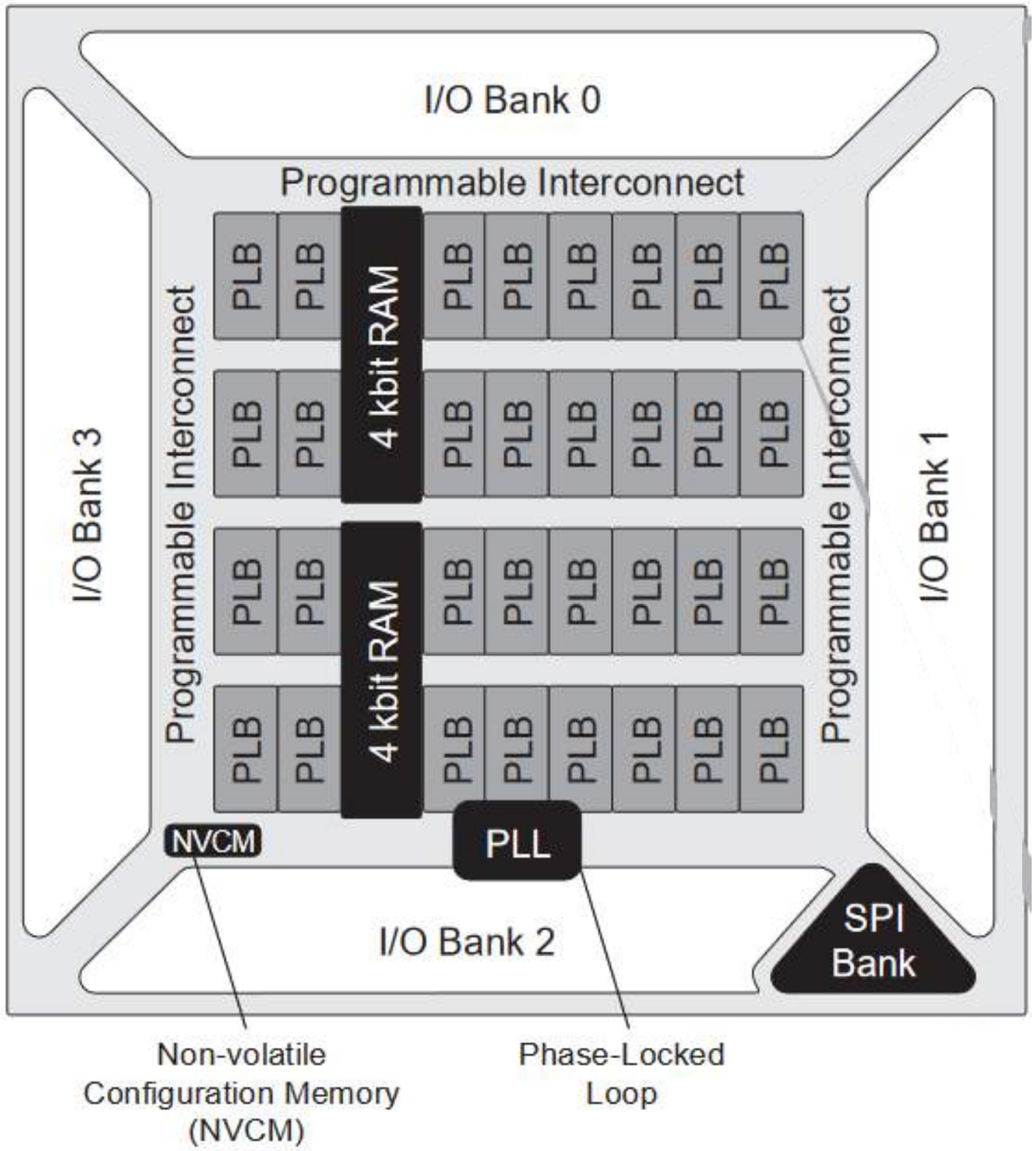
## PRODUITS PRÉSENTÉS LATTICE

# iCE40

not just a
sea of logic...

specialized
modules :

BRAM
PLL
SPI
I/O (w LVDS)
NVCM

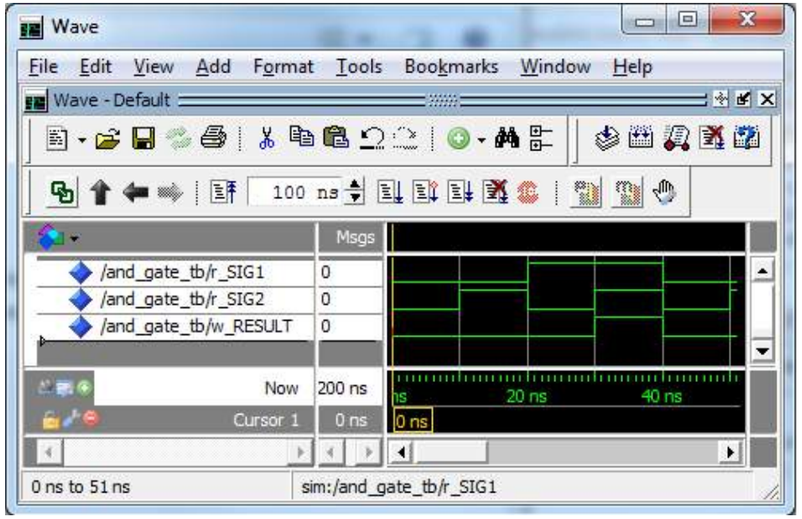


# Lattice Toolchain

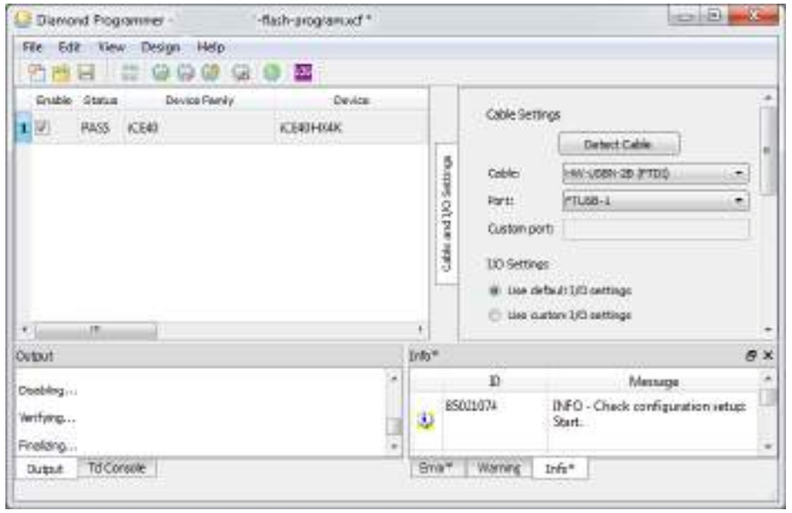
SYNTHESIS → TESTBENCH → CONFIGURATION

iCEcube 2


ModelSim

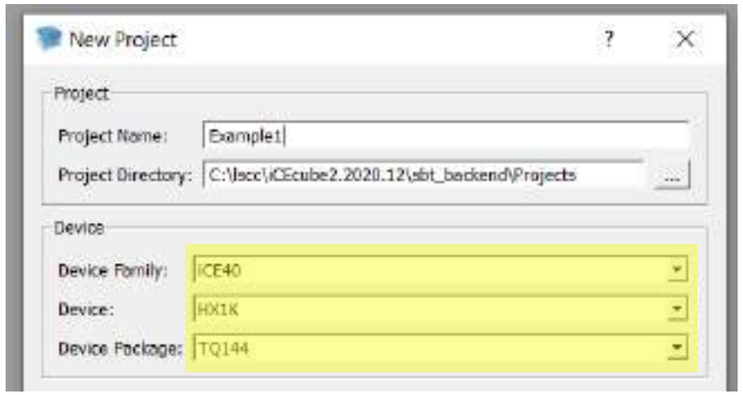



Diamond Programmer

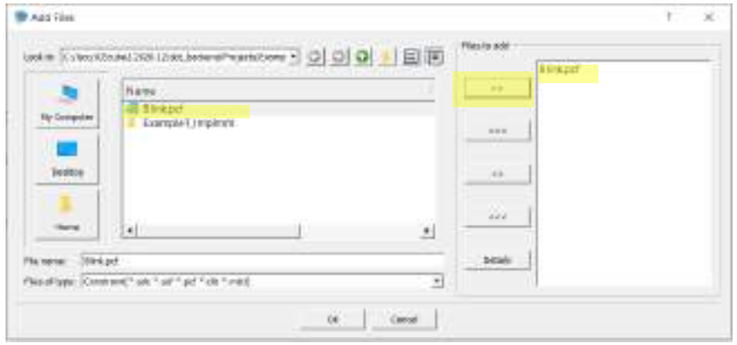



free licence for enthusiasts

# IceCube 2




① Make new project

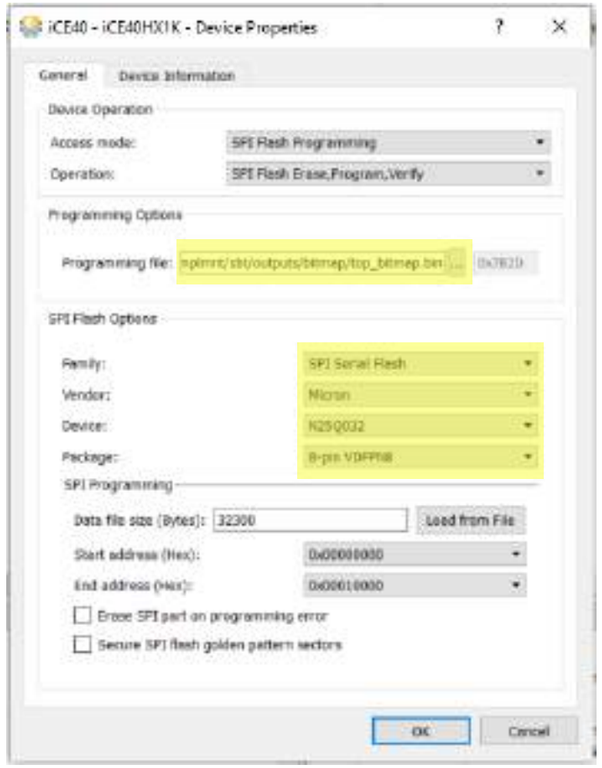


② ADD .V and .PCF

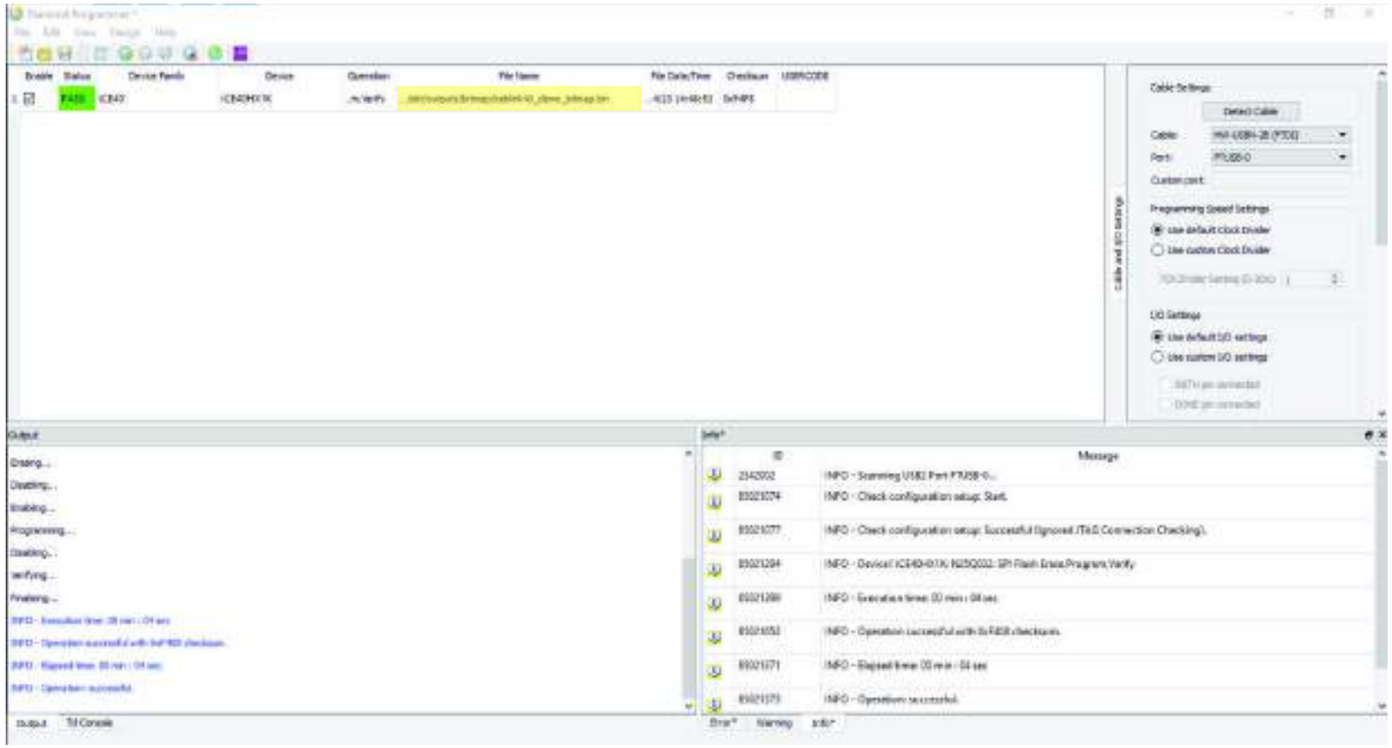

③ Tool > Run All

Output Files >
Bitmap >
Open File Location

# Diamond Programmer




④ Operation

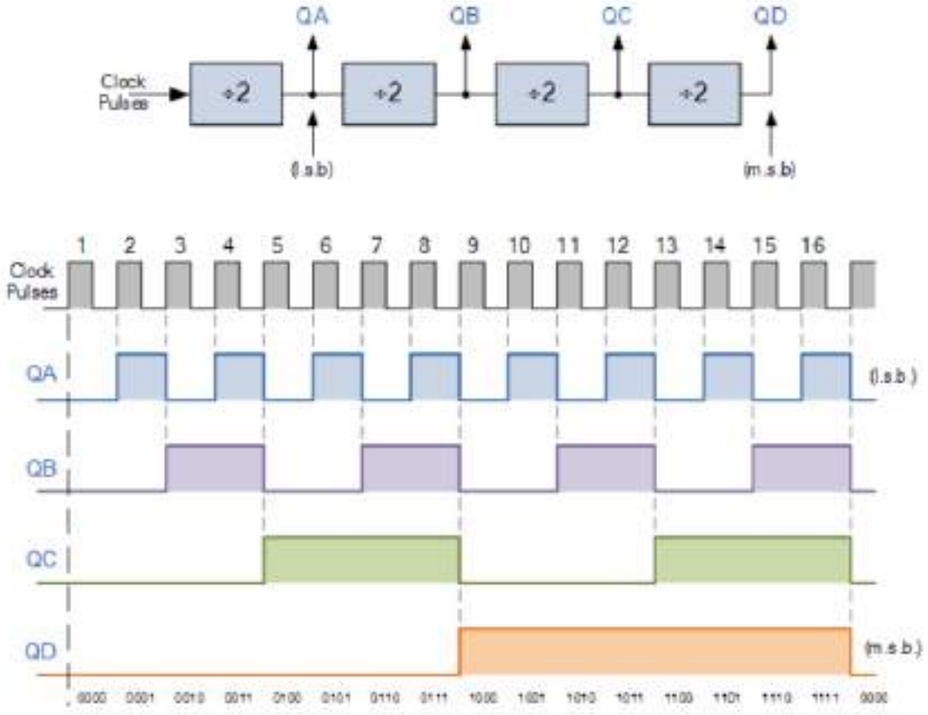


⑤ Upload

Challenge

# BLINK w FPGA!

HINT :




Use a register to divide the clock frequency to a 1Hz (green LED) blink.

Answer

# BLINK w FPGA!

## (2^n-1) / clock speed = frequency

If we want to know what the final frequency of an LED blinking will be with a given counter and clock speed :

(2^24 -1) / 12MHz = 1.4Hz = blink frequency (if toggling every roll over of a 24 bit register with a 12MHz clock)

top.v

```
`default_nettype none

module top(
input wire clk,
output wire LED
);


reg[23:0] counter;

assign LED = counter[23];

always@(posedge clk)
begin

counter <= counter + 1;

end

endmodule
```

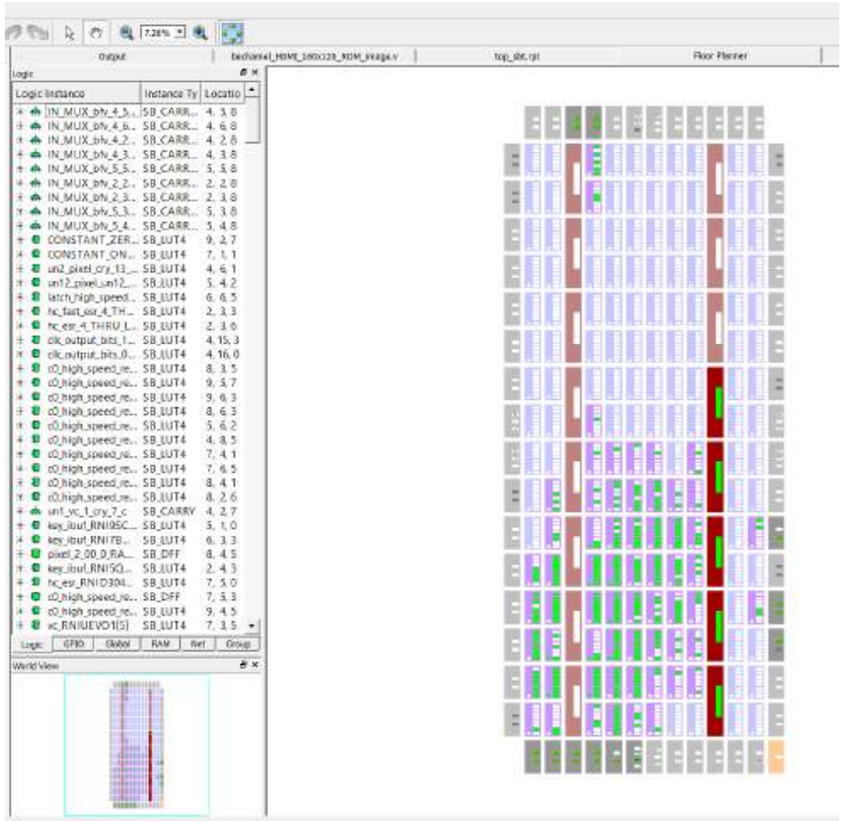
# BLINK w FPGA!

RUN SYNTHESIS

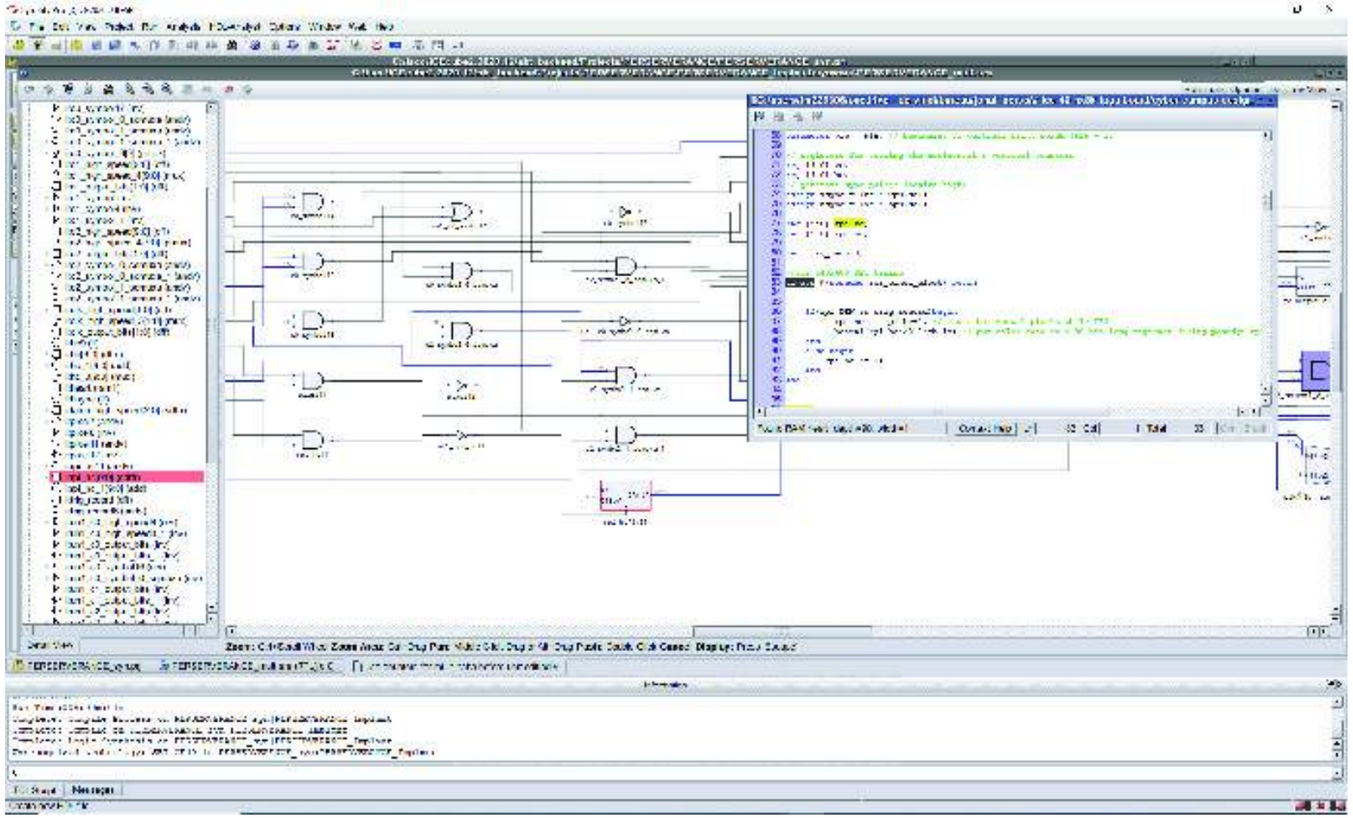
INSPECT REPORT

```
I2078: Design legalization is completed successfully
I2088: Phase 1, elapsed time : 0.0 (sec)
Phase 2
I2088: Phase 2, elapsed time : 0.0 (sec)
Phase 3
Info-1404: Inferred PLL generated clock at uut/PLLOUTCORE
Info-1404: Inferred PLL generated clock at uut/PLLOUTGLOBAL
Design Statistics after Packing
    Number of LUTs          :     280
    Number of DFFs          :     90
    Number of DFFs packed to IO   :     0
    Number of Carrys        :     43
Device Utilization Summary after Packing
    Sequential LogicCells
        LUT and DFF         :     73
        LUT, DFF and CARRY  :     17
    Combinational LogicCells
        Only LUT            :     168
        CARRY Only          :     4
        LUT with CARRY      :     22
    LogicCells                : 284/1280
    PLBs                      : 39/160
    BRAMs                     : 5/16
    IOs and GBIOs             : 17/96
    PLLs                      : 1/1
I2088: Phase 3, elapsed time : 0.4 (sec)
Phase 4
I2088: Phase 4, elapsed time : 0.1 (sec)
Phase 5
I2088: Phase 5, elapsed time : 0.4 (sec)
Phase 6
```

# BLINK w FPGA!

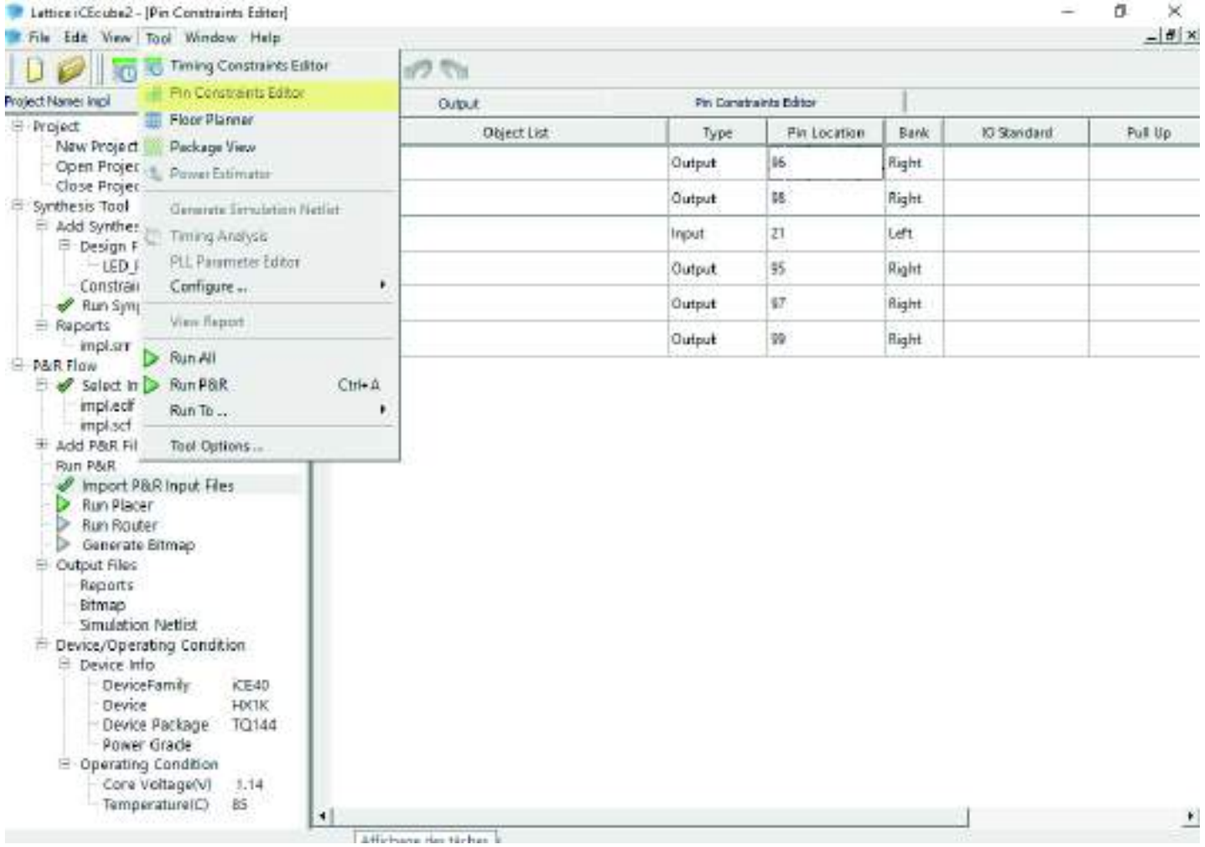


INSPECT P&R



INSPECT RTL

# BLINK w FPGA!




From : EB82-iCEstick_User_Manual-2.pdf

### Preprogrammed Design and Board LEDs

There are a total of 5 LEDs on the iCEstick board. All are controlled by I/Os of the iCE40HX-1k device. The default bitstream loads the iCE40HX-1k device and the green LED lights up signifying that the device has loaded correctly and power is good. The other four red LEDs arranged in a diamond pattern begins to flash in a clockwise direction. This is the intended function of the default bitstream.

***Table 1. User I/O and LEDs***

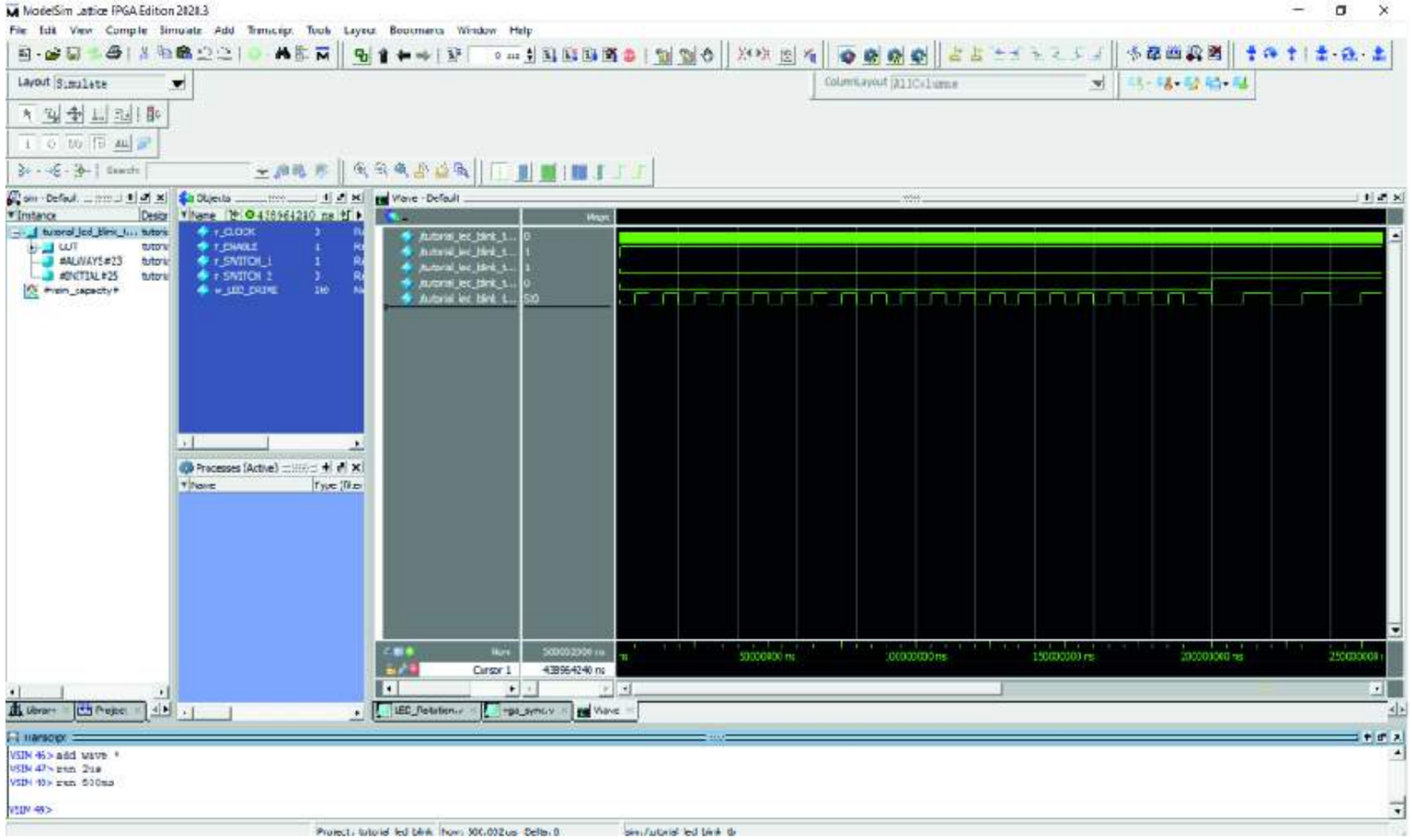
| LED location | CPLD pin (All in Bank 1) | CPLD I/O | LED color |
|---|---|---|---|
| D1 | 99 | PIO1_14 | Red |
| D2 | 98 | PIO1_13 | Red |
| D3 | 97 | PIO1_12 | Red |
| D4 | 96 | PIO1_11 | Red |
| D5 | 95 | PIO1_10 | Green |

top.pcf

```
set_io clk 21
set_io LED 95
```

PIN CONSTRAINTS (pcf)

# Testbench




*you can code things in testbenches that are not synthesizable

```
// TEST BENCHES

//for test benches
use # to make things
happen at certain times
:

x = 0;
#10 x = x + 1;
#20 x = x + 2;
#30 x = x + 3;
#20 x = x + 2;
#10 x = x + 1;
```

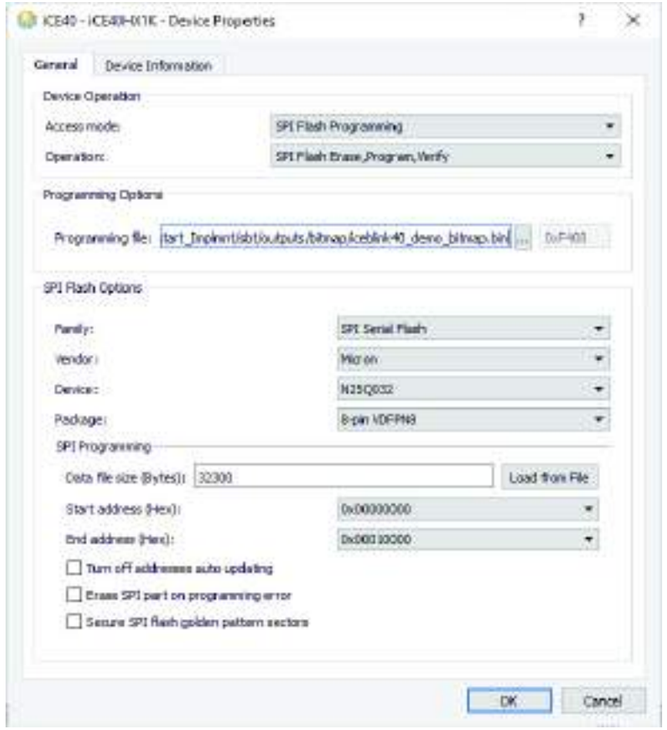
| time: | value of x : |
|---|---|
| 0 | 0 |
| 10 | 1 |
| 30 | 3 |
| 60 | 6 |
| 80 | 8 |
| 90 | 9 |

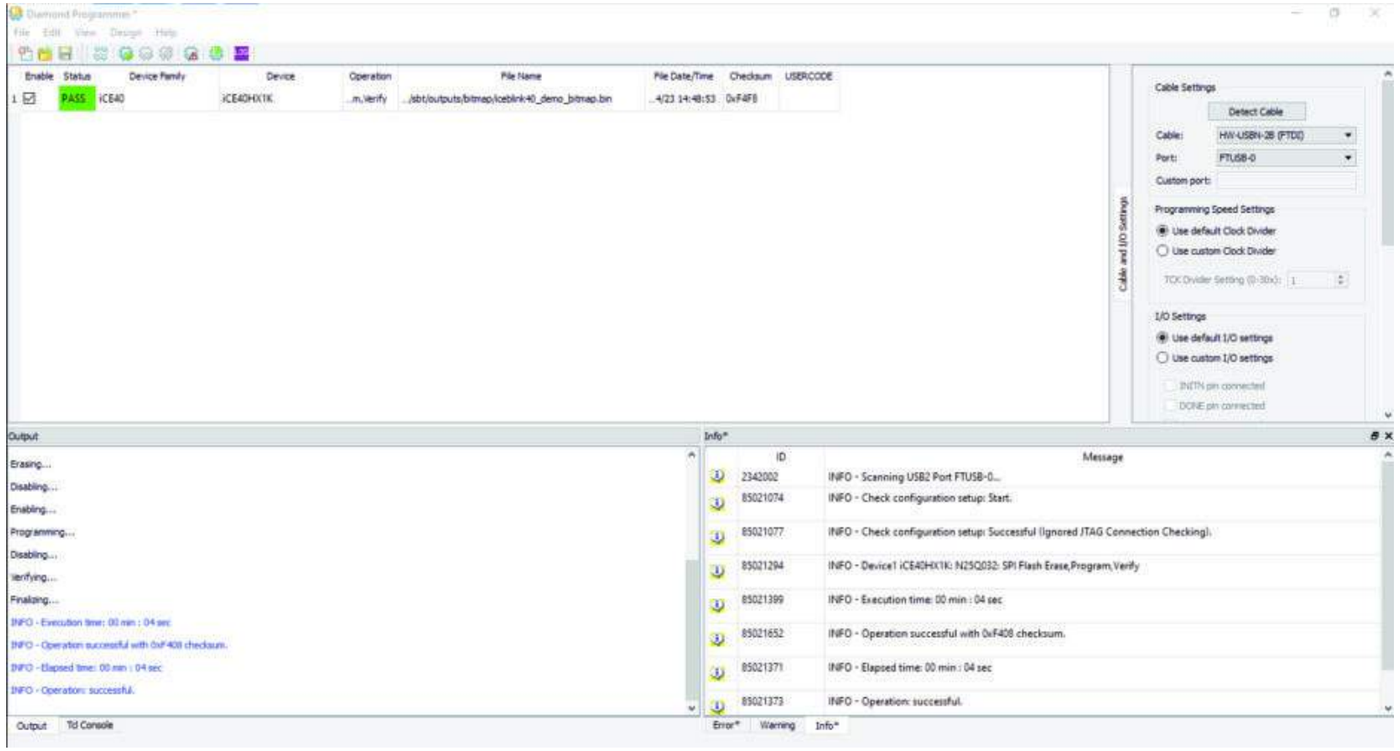
```
// for test bench
clocks :
always
begin
#period/2 clk = ~clk;
end
```

Open ModelSim, reselect the licence.dat file
Open a "new project"
Click Add Existing Files (pick only the test bench verilog file) and then Close
Now Compilation > Compile All
Now Simulation > Start Simulation
In the console type <<view wave>>...
...then <<add wave * >>... (note the space between wave and *)
right click on the signal and selected Clock and then set the speed.
...and a time period for example : <<run 10000ms>>.

https://nandland.com/tutorial-your-fpga-program-an-led-blinker-part-2/

# FLASHING BINARY



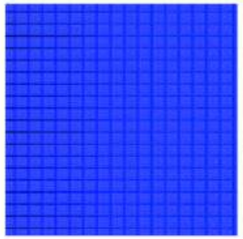

Launch Diamond Programmer and take the default options (or click scan and take those). Important to note that you must launch Lattice Diamond Programmer directly, not through Lattice Diamond, in order to find the iCE40HX1K chip. It will mistake the board for a JTAG interface and give the following errors : "Failed to scan board", "Scan Failed - Creating Blank Programmer Project."
Now change the Device Family to iCE40, the device to iCE40HX1K, and under Program enter the following information : Change Access mode to SPI Flash Programming and now some other options appear. Select > Vendor: Micron, Device: N25Q032, Package: 8-pin VDFPN8
Now load the bitmap (.bin) file and check that it has a non-zero datasize

# CHALLENGE :

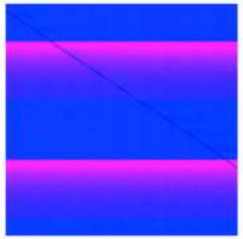
Make your own interactive
FPGA video synthesizer !!!

# DAY 3

# BECHAMEL Lo-Fi FPGA Video Art Board

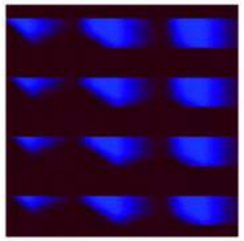


```
h_count%10 == 0 | v_count%13 == 0
```

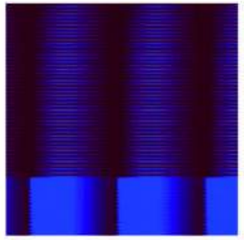


```
r0 <= v_count[6];
r1 <= v_count[6];
r2 <= v_count[6];
g0 <= 1'b1;
g1 <= 1'b1;
g2 <= 1'b1;
b0 <= 1'b1;
b1 <= 1'b1;
b2 <= 1'b1;
```

```
r0 <= v_count[6];
r1 <= v_count[1];
r2 <= v_count[4];
g0 <= h_count[3];
g1 <= h_count[5];
g2 <= h_count[1];
b0 <= v_count[4];
b1 <= h_count[1];
b2 <= v_count[1];
```



```
if (div_cntr2[12]^h_count[5]) begin
r0 <= 1'b1;
r1 <= 1'b1;
r2 <= 1'b1;
g0 <= 1'b0;
g1 <= 1'b0;
g2 <= 1'b0;
b0 <= v_count[5];
b1 <= div_cntr2[5];
b2 <= 1'b1;
end
```



```
if (div_cntr2[12]^h_count[5] ==
v_count%100>>h_count) begin
r0 <= 1'b1;
r1 <= 1'b1;
r2 <= 1'b1;
g0 <= 1'b0;
g1 <= 1'b1;
g2 <= 1'b0;
b0 <= v_count[1];
b1 <= div_cntr2[3];
b2 <= 1'b1;
end
```

```
wire[1:0] bit_or;
assign bit_or =
h_count[1:0]|v_count[1:0];

[...]

if (bit_or[5] == 1 | bit_or[3] == 0) begin
```

```
wire[1:0] bit_xor;
assign bit_xor =
h_count[1:0]^v_count[1:0];

[...]

if (bit_xor[2] == 1 | bit_xor[4] == 1)
begin
```

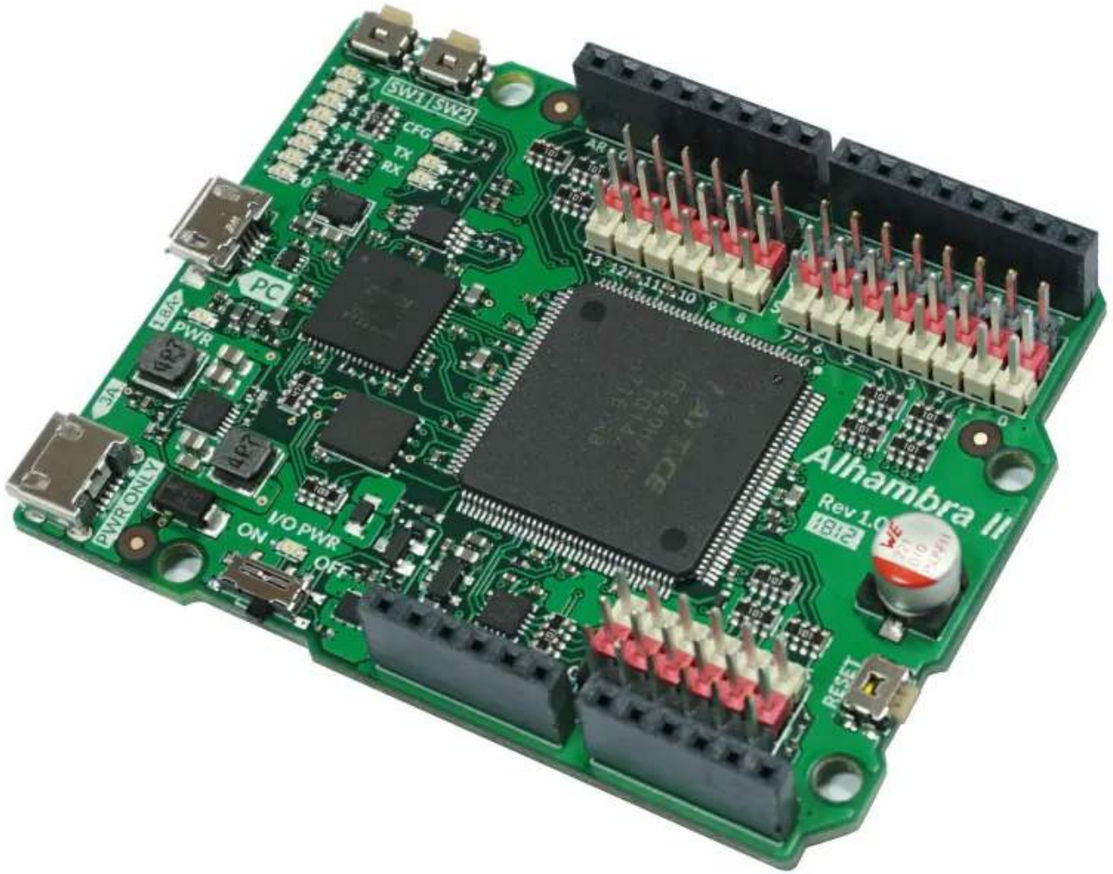
```
wire[1:0] bit_and;
assign bit_and =
h_count[1:0]&v_count[1:0];

...

if (bit_and[6] == 1 | bit_xor[7] == 1)
begin
```

marrs.io

https://github.com/preparedinstruments/bechamel

# Alhambra II

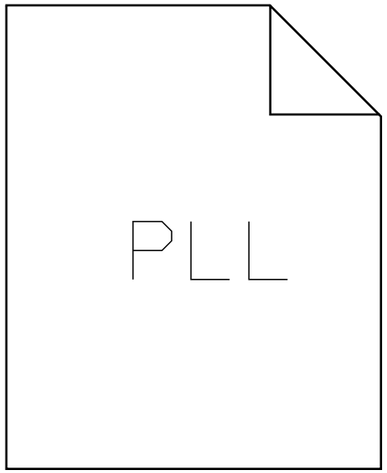
Iñigo Muguruza
imuguruza



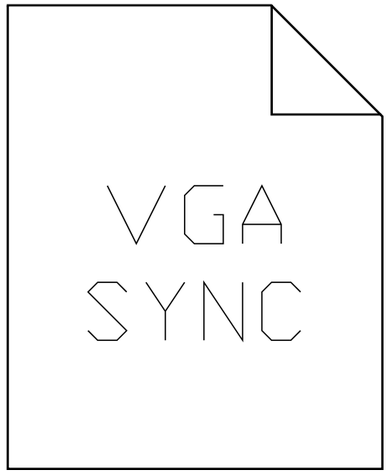

*your PCF will differ!

https://github.com/imuguruza/alhambra_II_test/tree/master/vga

# Alhambra II



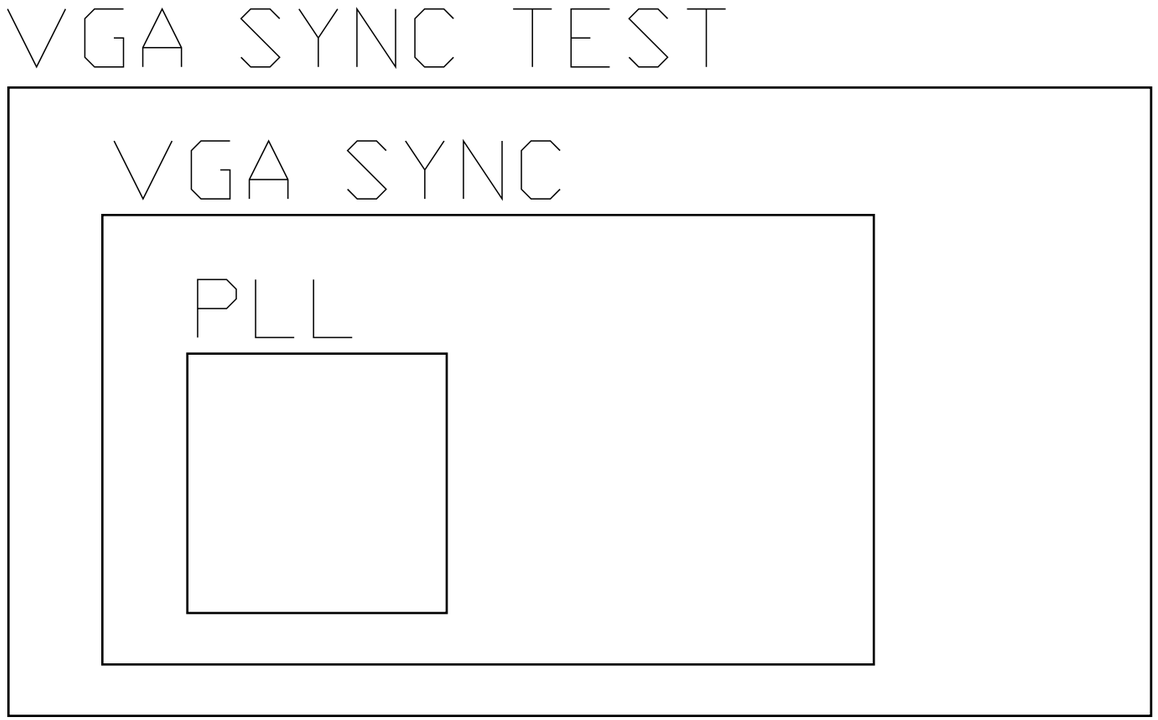

12MHz >
25MHz

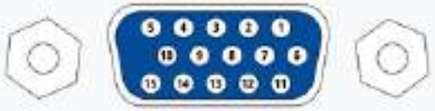



H SYNC
V SYNC


COLOR
PATTERNS

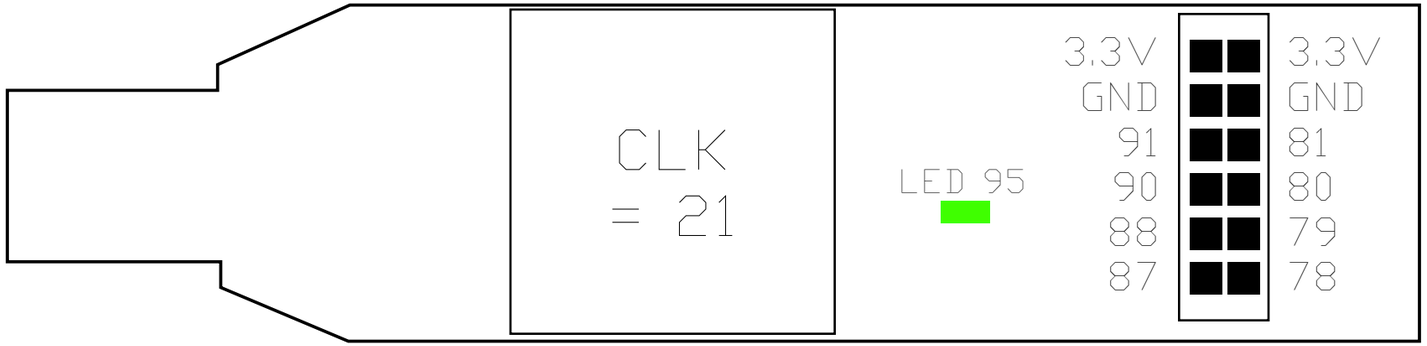
# Alhambra II

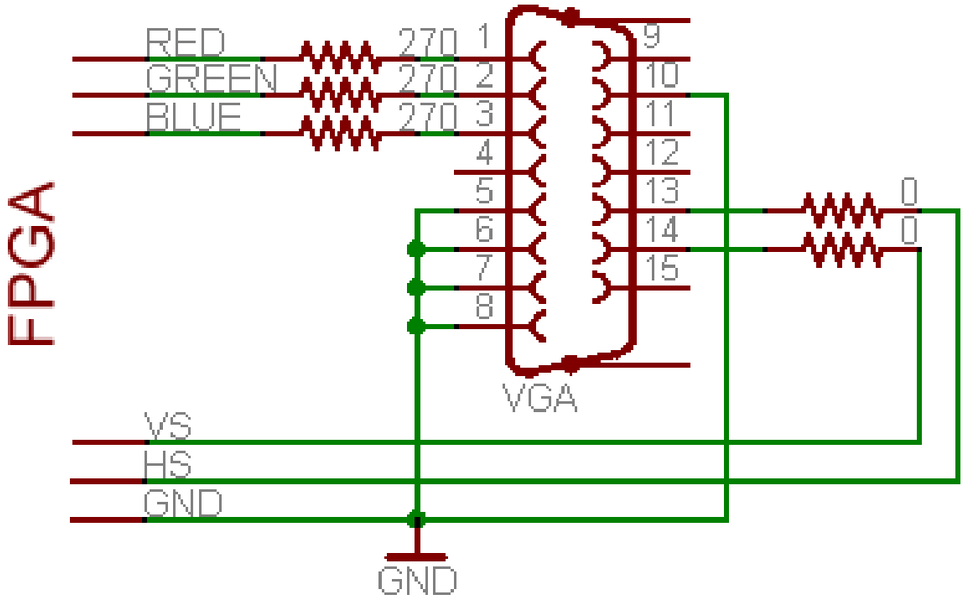


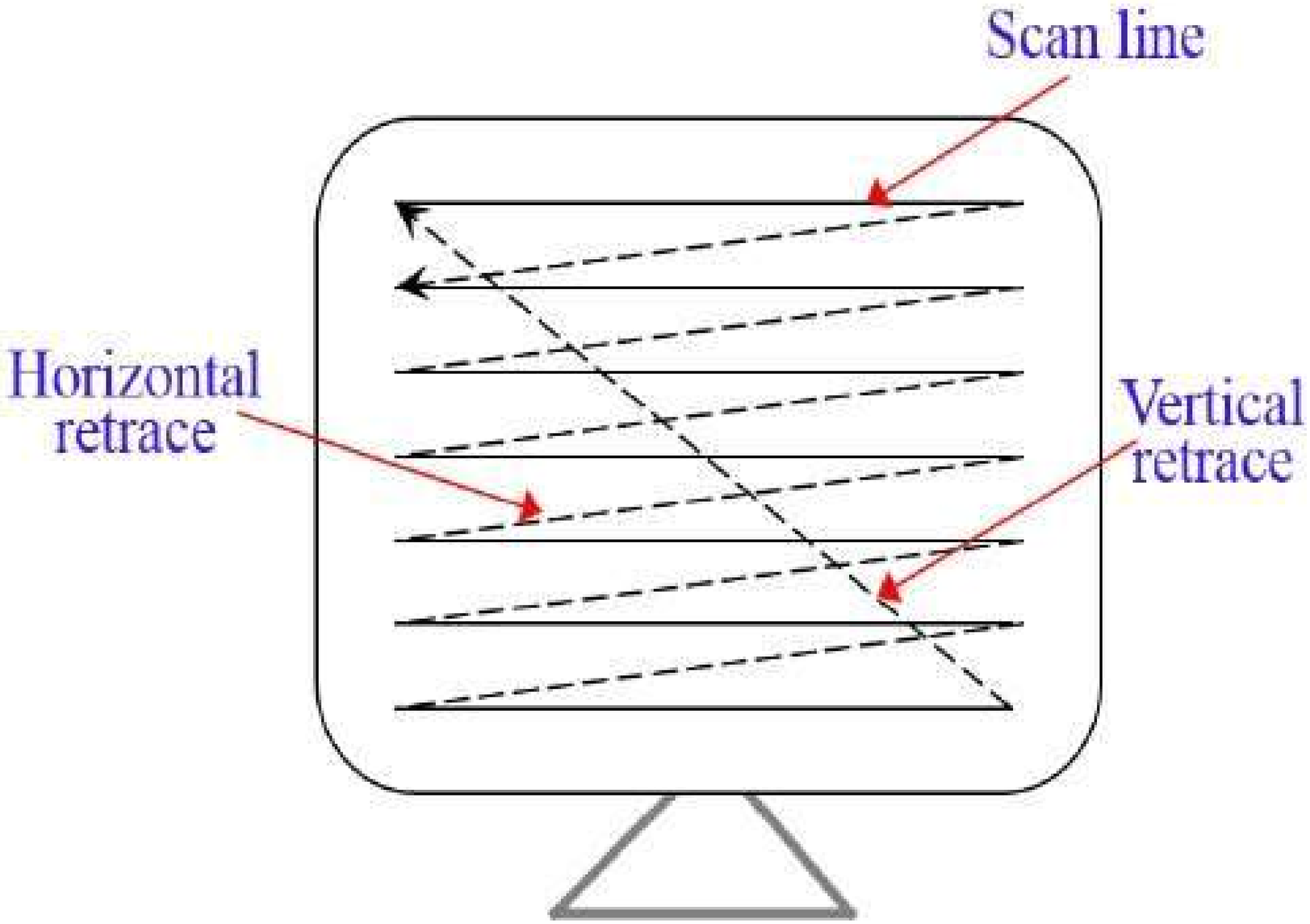


A female DE15 socket

| | | |
|---|---|---|
| **Pin 1** | RED | Red video |
| **Pin 2** | GREEN | Green video |
| **Pin 3** | BLUE | Blue video |
| **Pin 4** | ID2/RES | Reserved since E-DDC, formerly monitor id. bit 2 |
| **Pin 5** | GND | Ground (HSync) |
| **Pin 6** | RED_RTN | Red return |
| **Pin 7** | GREEN_RTN | Green return |
| **Pin 8** | BLUE_RTN | Blue return |
| **Pin 9** | KEY/PWR | +5 V DC (powers EDID EEPROM chip on some monitors), formerly key |
| **Pin 10** | GND | Ground (VSync, DDC) |
| **Pin 11** | ID0/RES | Reserved since E-DDC, formerly monitor id. bit 0 |
| **Pin 12** | ID1/SDA | I²C data since DDC2, formerly monitor id. bit 1 |
| **Pin 13** | HSync | Horizontal sync (or Composite sync) |
| **Pin 14** | VSync | Vertical sync |
| **Pin 15** | ID3/SCL | I²C clock since DDC2, formerly monitor id. bit 3 |



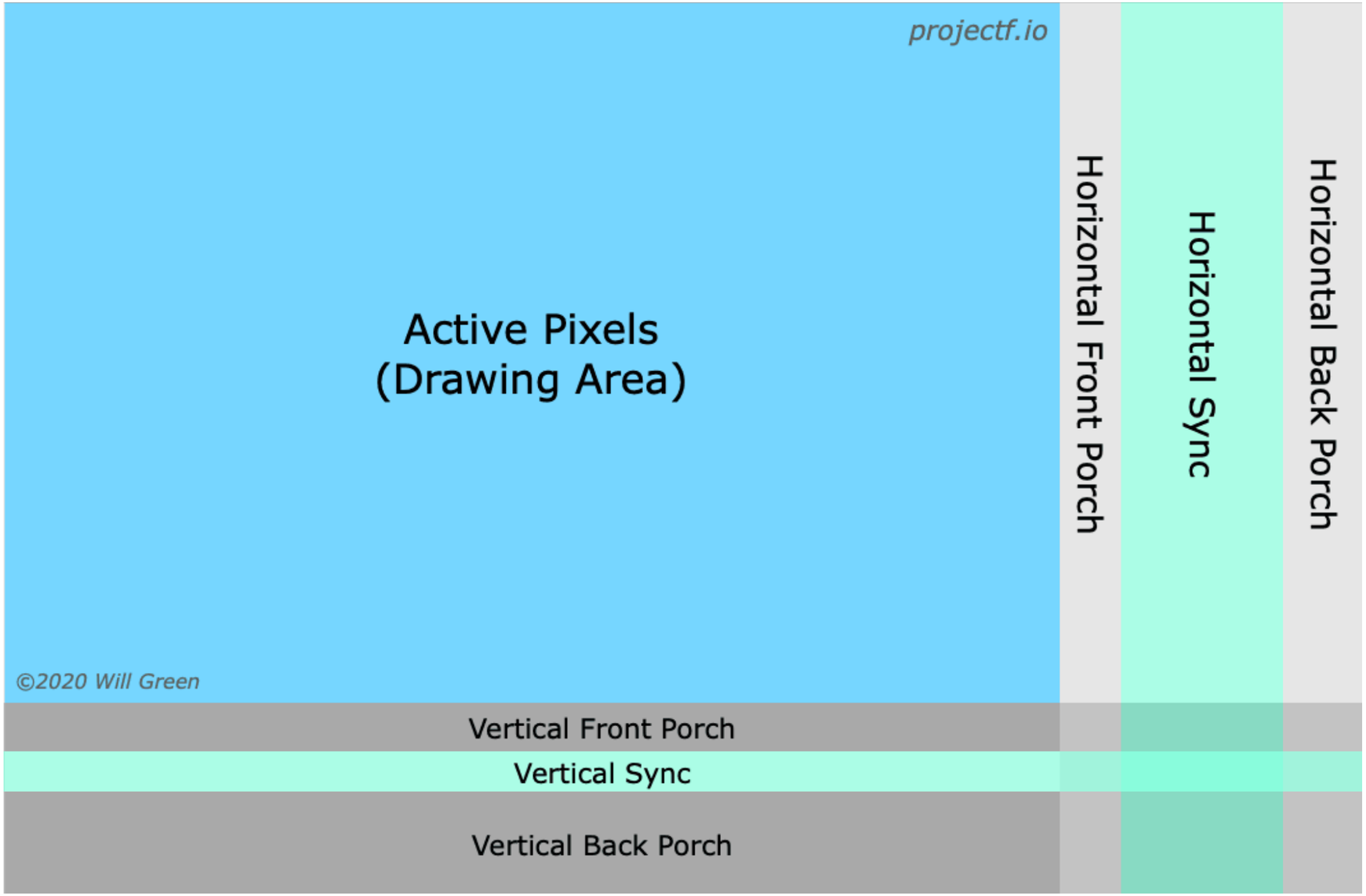




Scan line
Horizontal retrace
Vertical retrace

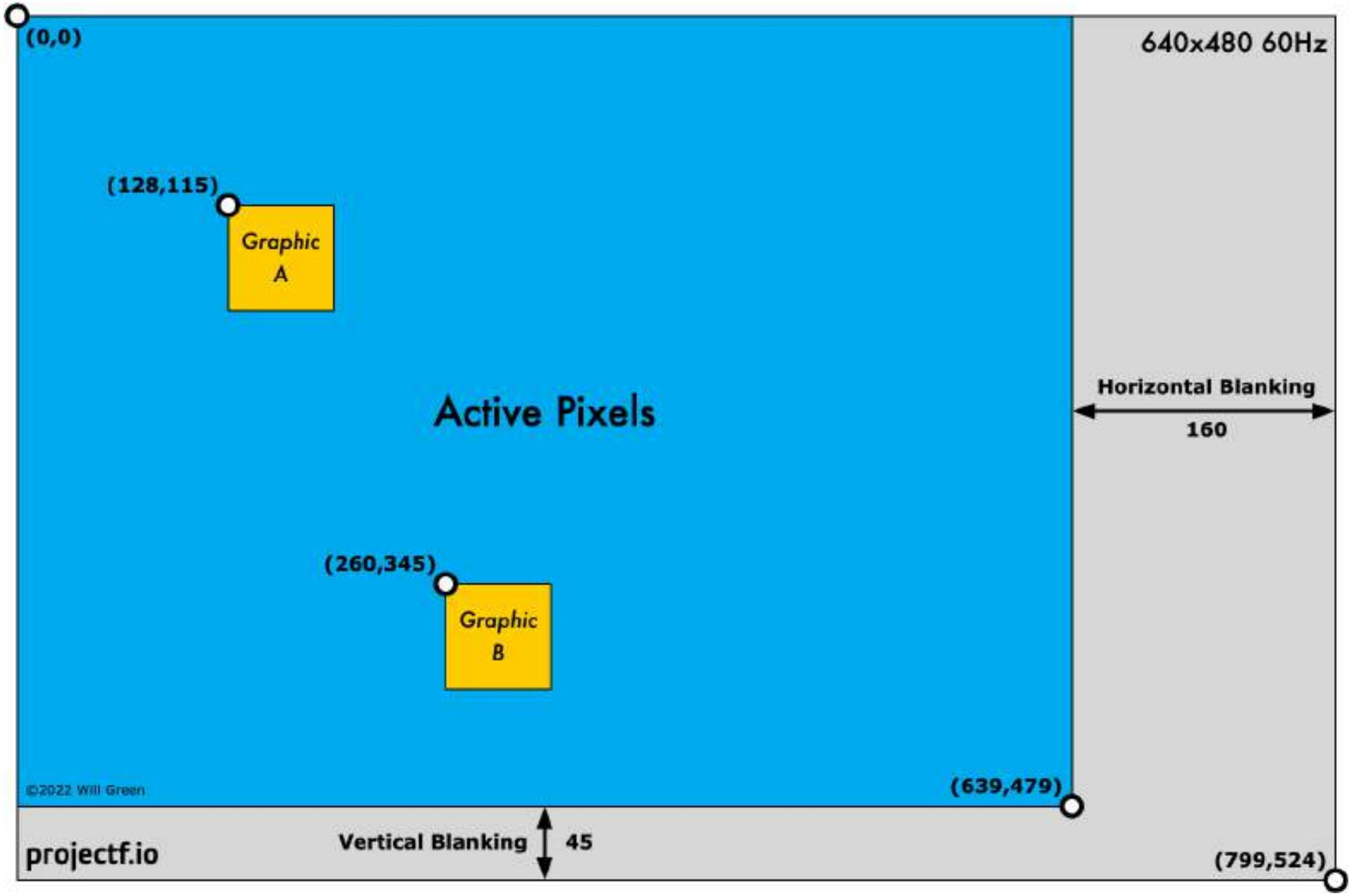
Display timings for 640x480 at 60Hz in units of pixels:

| Parameter | Horizontal | Vertical |
|---|---|---|
| Active Pixels | 640 | 480 |
| Front Porch | 16 | 10 |
| Sync Width | 96 | 2 |
| Back Porch | 48 | 33 |
| Total Blanking | 160 | 45 |
| Total Pixels | 800 | 525 |
| Sync Polarity | negative | negative |

https://projectf.io/posts/fpga-graphics/

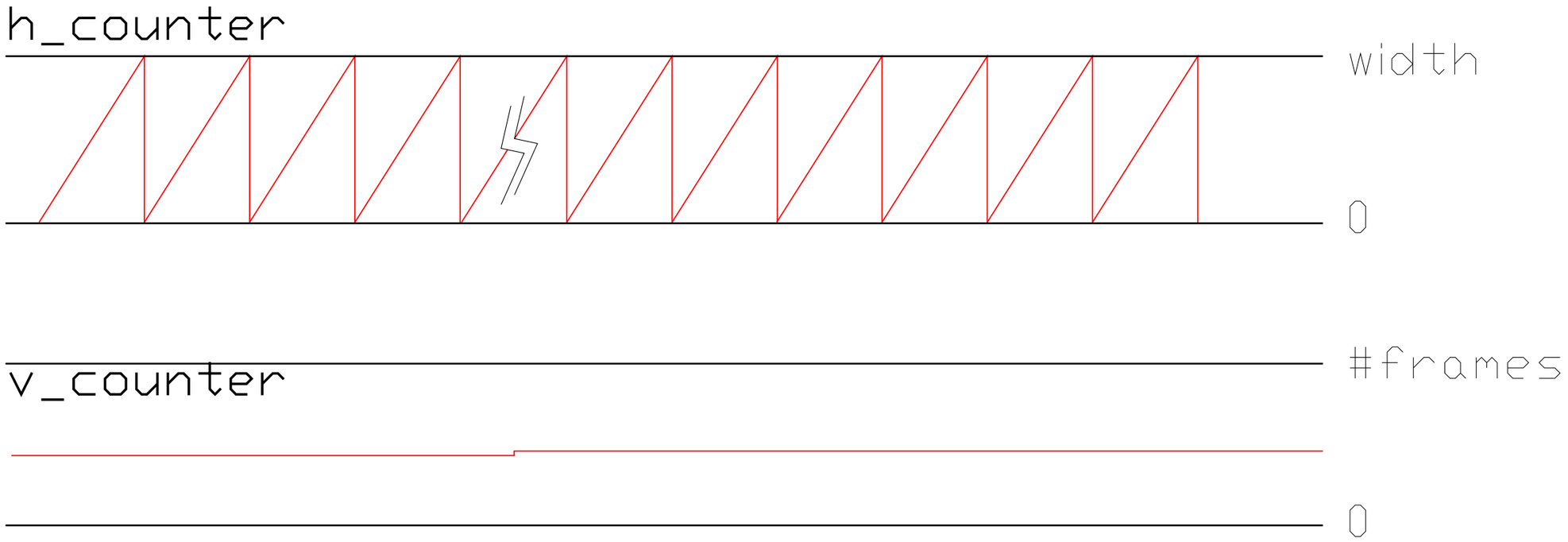



https://projectf.io/posts/fpga-graphics/




https://projectf.io/posts/fpga-graphics/

# BYTEBEATS




use h-count for x and y,
use v_counter for time animations

```
if(187==((h_count*5)+v_count) blue
else green
```

# GOING FURTHER

REACT TO BUTTONS
REACT TO MUSIC
GRADIENTS

DIFF RESOLUTIONS
SCREEN BUFFER
LINE BUFFER
TAKE VIDEO INPUT IN (from RPI)
SPRITES
FONTS

# USEFUL LINKS

https://hdlbits.01xz.net/wiki/Main_Page
http://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/
https://opencores.org/
https://www.fpga4fun.com/
https://nandland.com/
https://digitaljs.tilk.eu/

https://www.dcode.fr/binary-image

http://asic-world.com/verilog/index.html
https://projectf.io/tutorials/
http://fpgacpu.ca/fpga/index.html
https://zipcpu.com/tutorial/
http://www.doe.carleton.ca/~gallan/478/pdfs/PeterVrlR.pdf
https://fpgaer.tech/?p=191

https://projectf.io/posts/fpga-graphics/
https://projectf.io/posts/animated-shapes/

https://clifford.at/icestorm
https://m-labs.hk/gateware/migen/

analog input:
https://web.archive.org/web/20190615204217/https://hamsterworks.co.nz/mediawiki/index.php/Cheap_Analogue_Input

cold booting :
https://www.latticesemi.com/support/answerdatabase/3/3/4/3344

iCE Stick Manual :
https://www.mouser.fr/ProductDetail/Lattice/iCE40HX1K-TQ144?qs=F9A14TELRMtiYSwGl6R%2Fcw%3D%3D

Mentor ModelSim Simulator with Lattice iCEcube2:
https://www.latticesemi.com/-/media/LatticeSemi/Documents/ApplicationNotes/MP2/Modelsim_AN006_Dec2020.ashx?document_id=50795)

Diamond Programmer with the iCEstick :
https://www.youtube.com/watch?v=Df9k1T0bHmA&ab_channel=Dom

Bytebeats:
https://github.com/TuesdayNightMachines/Bytebeats

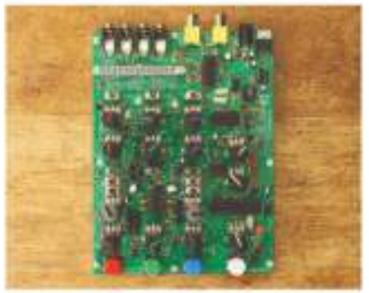
Critter & Guitari

LZX

KORG

Hypno

| | |
|---|---|
| 6.0 x 3.5 x 2.2' / 15.24 x 8.9 x 5.6 cm | $650 |
| RPI size | $640 |
| 213 x 58 x 10 mm | €430 |
| | £230 |
| | $479 |
| | $999 |



TACHYONS+