Introducción a la programación en Verilog

Desarrollo de aplicaciones con Dummy System FPGA para Red Pitaya

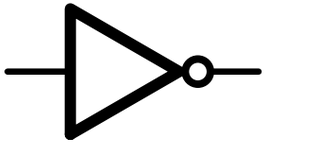
Resumen

Diseño de electrónica programable FPGA
Síntesis e implementación
Tecnología FPGA

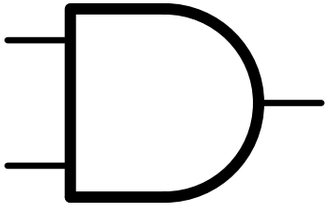
Programación de aplicación para Red Pitaya
Estructura de las aplicaciones
Dummy System

Introducción a programación en Verilog
Lenguaje básico
Ejemplos y demostraciones

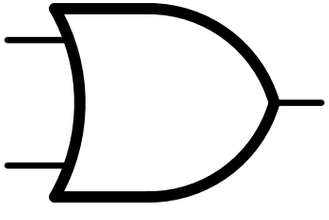
Diseño de circuitos en FPGA



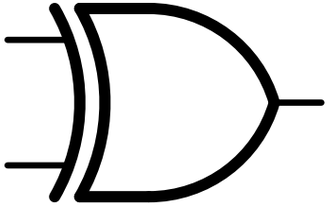
NOT



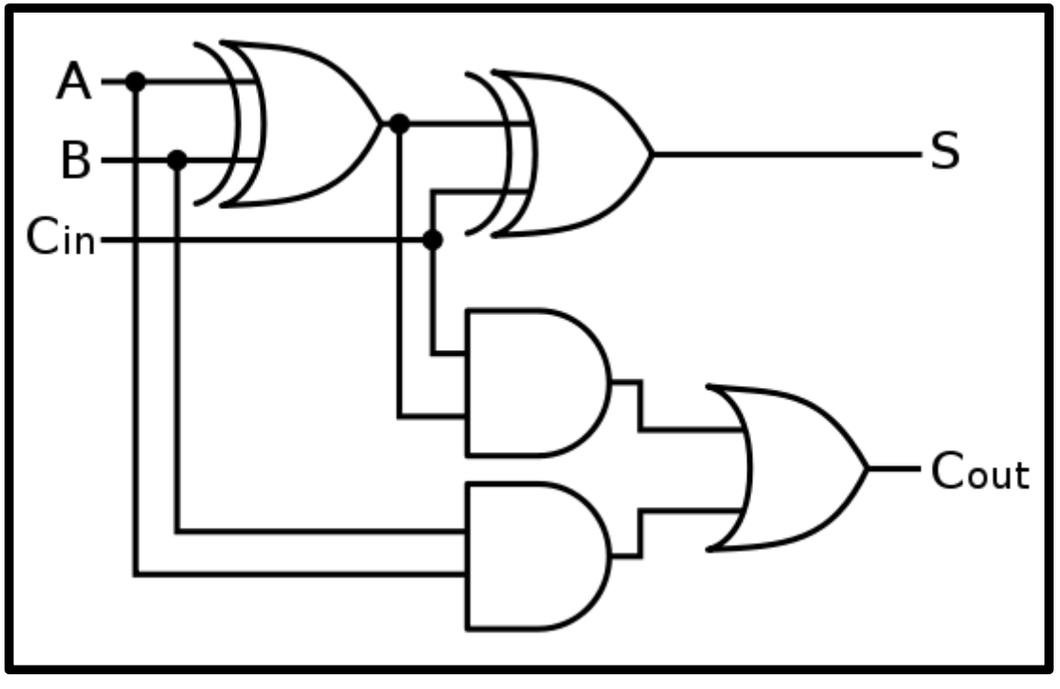
AND



OR



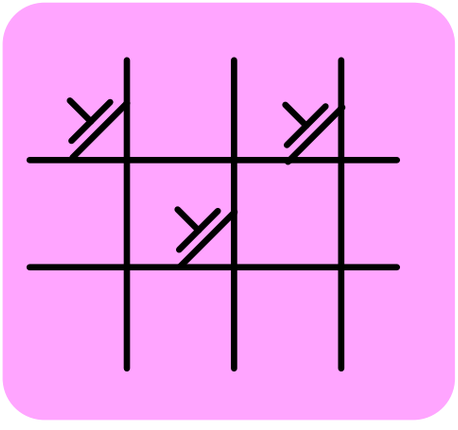
XOR

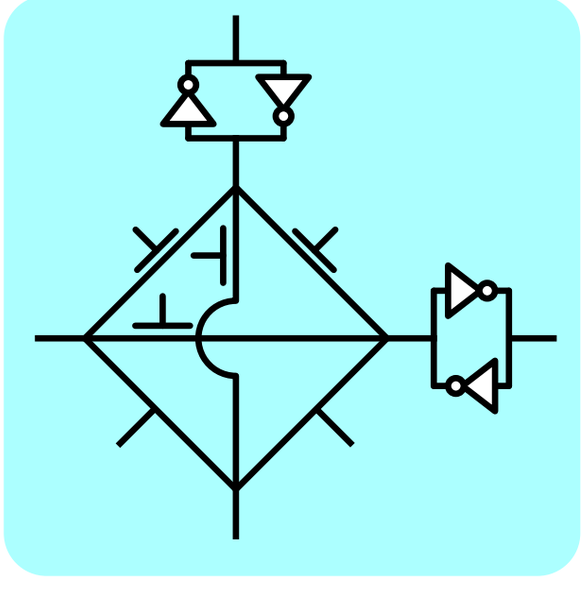


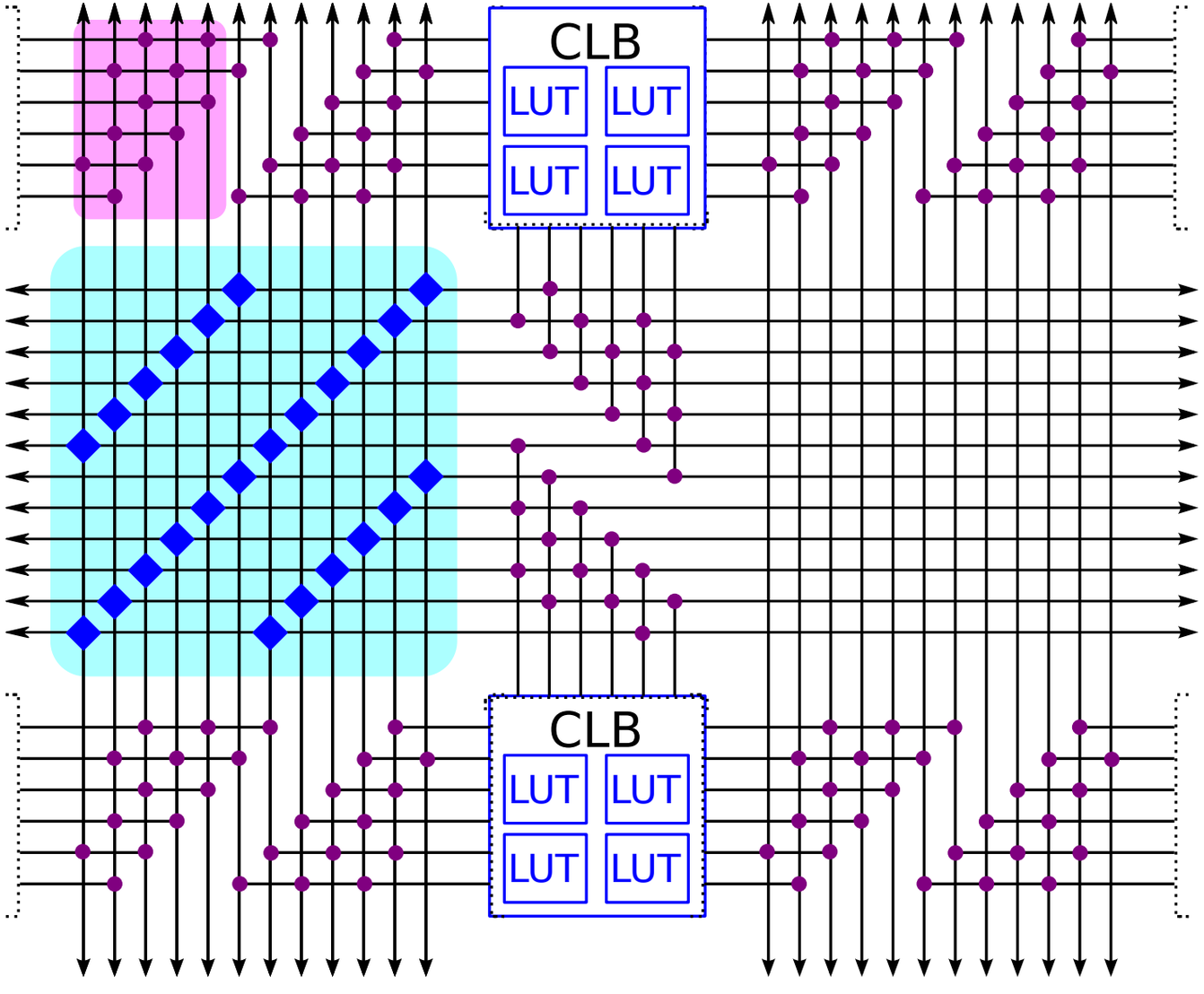
Circuito sumador

Diseño de circuito de lógica digital

Diseño de circuitos en FPGA

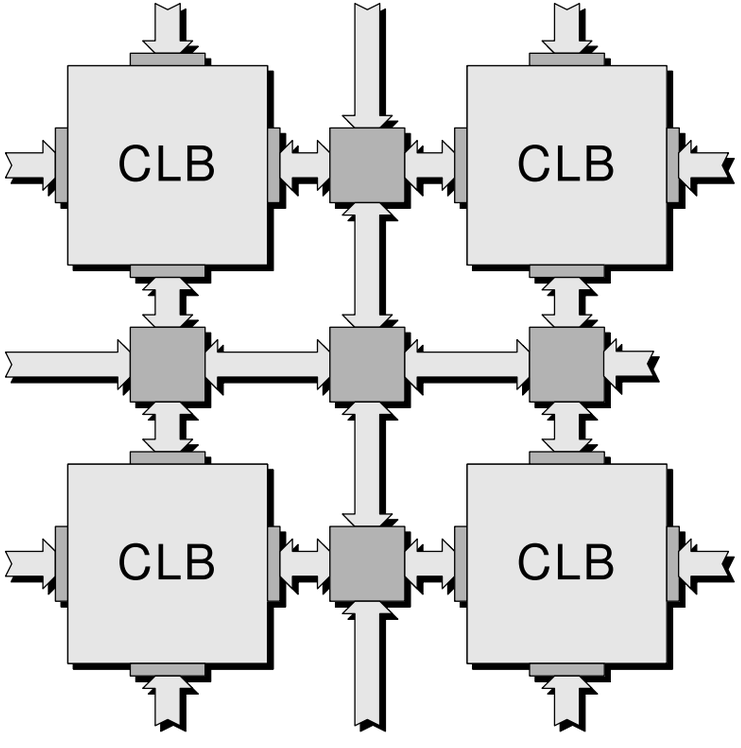




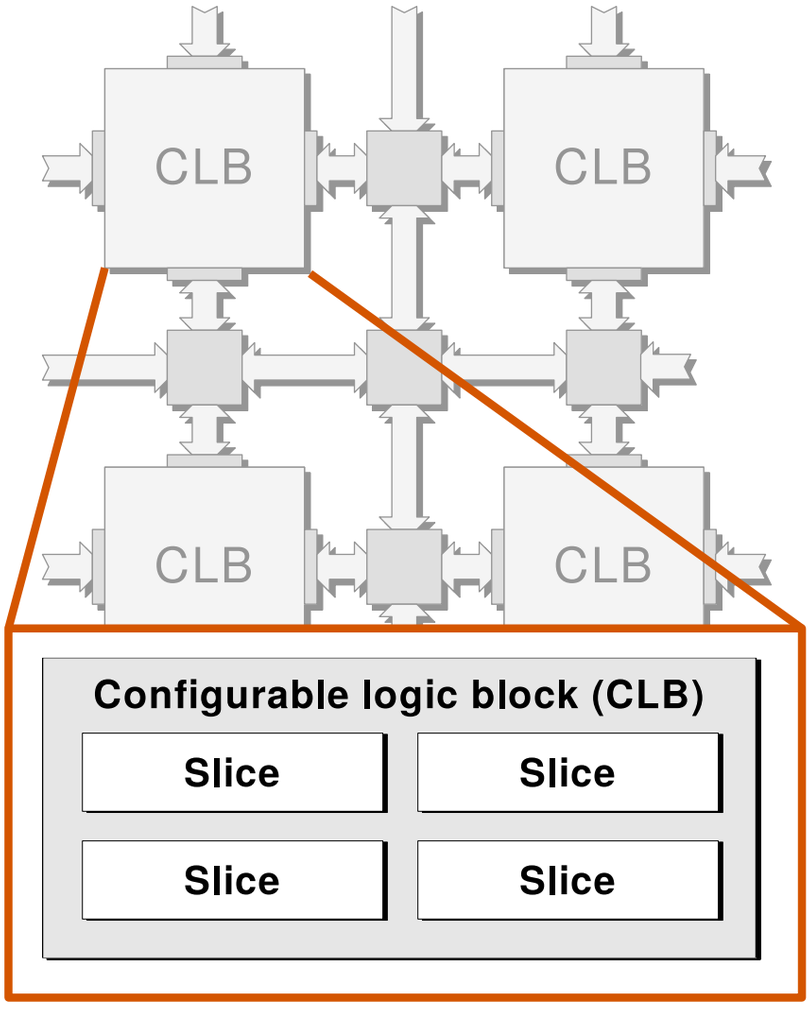


Diseño de circuito de lógica digital

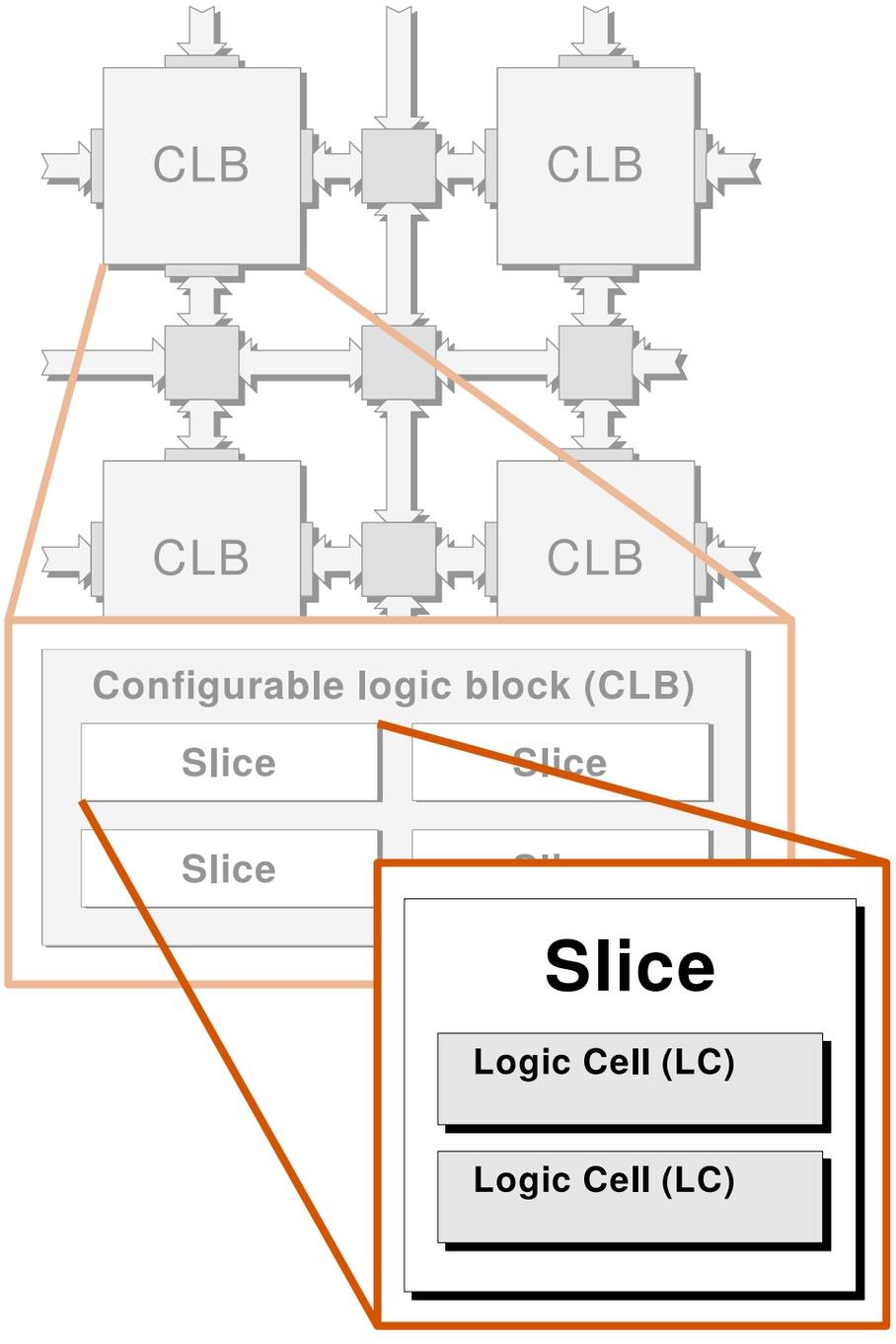
FPGA



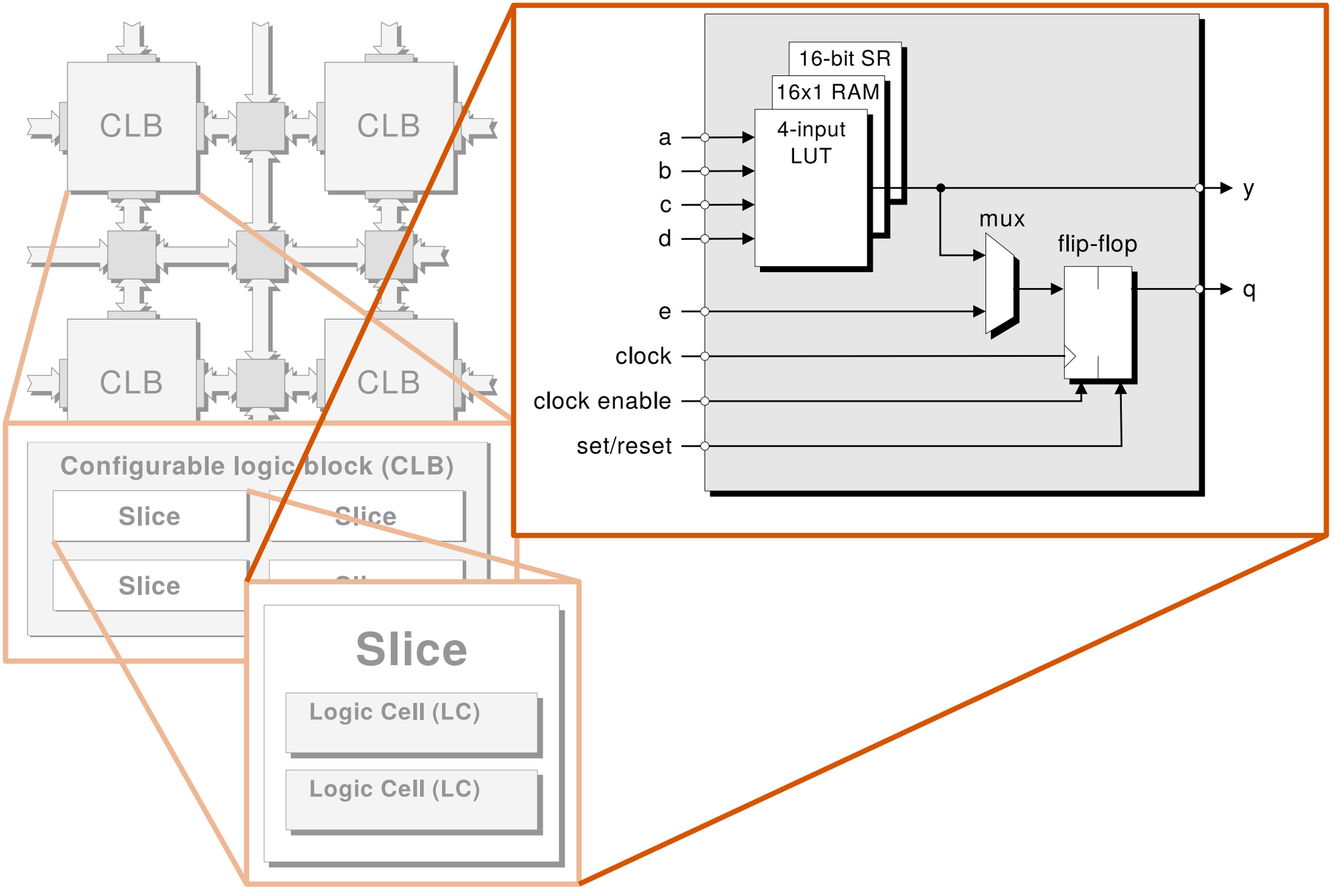
FPGA


CLB
CLB
CLB
CLB
Configurable logic block (CLB)
Slice
Slice
Slice
Slice

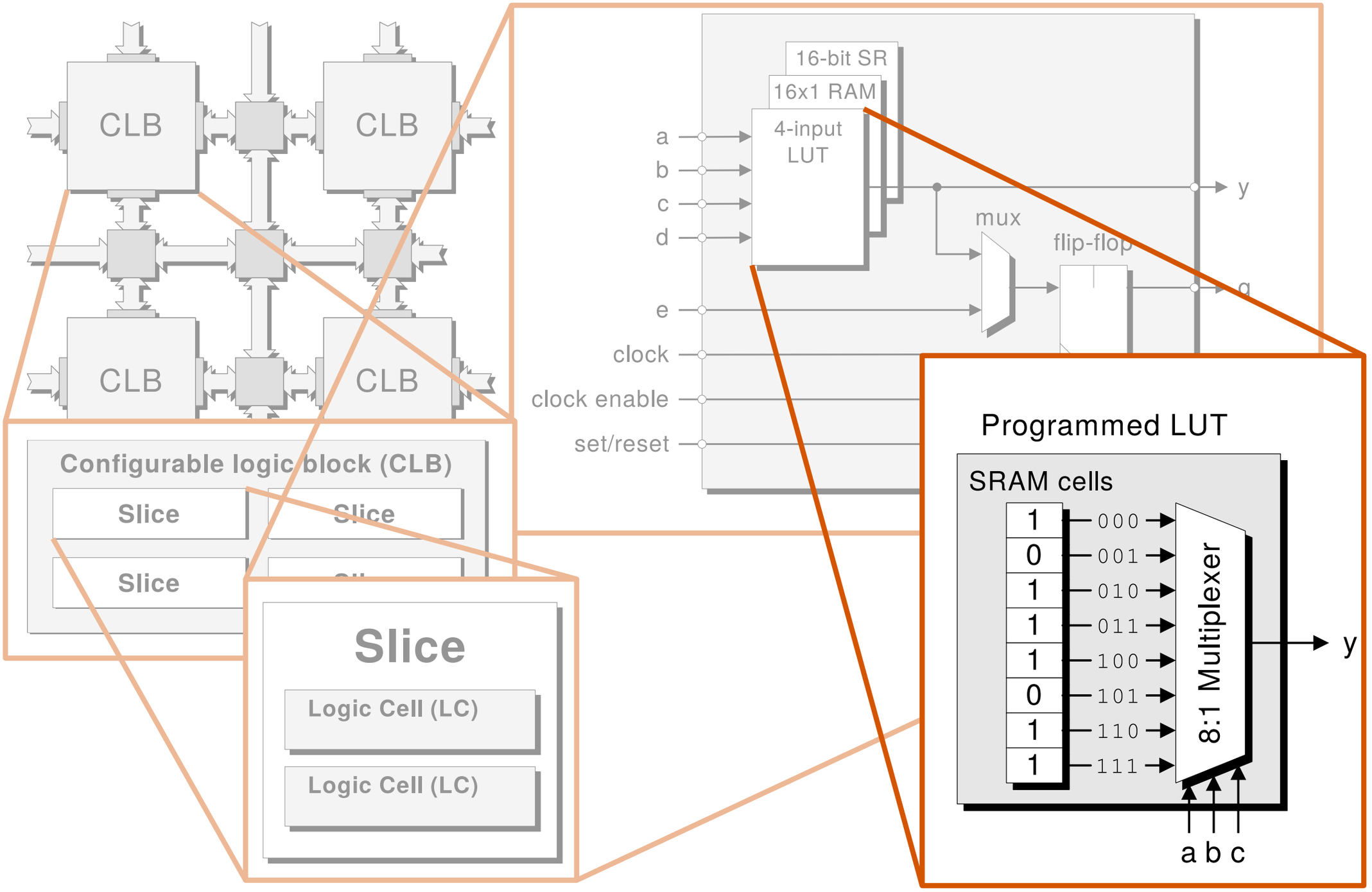
FPGA


CLB
CLB
CLB
CLB
Configurable logic block (CLB)
Slice
Slice
Slice
Slice
Logic Cell (LC)
Logic Cell (LC)

FPGA


CLB
CLB
CLB
CLB
Configurable logic block (CLB)
Slice
Slice
Slice
Slice
Logic Cell (LC)
Logic Cell (LC)
16-bit SR
16x1 RAM
4-input
LUT
a
b
c
d
e
clock
clock enable
set/reset
mux
flip-flop
y
q

FPGA

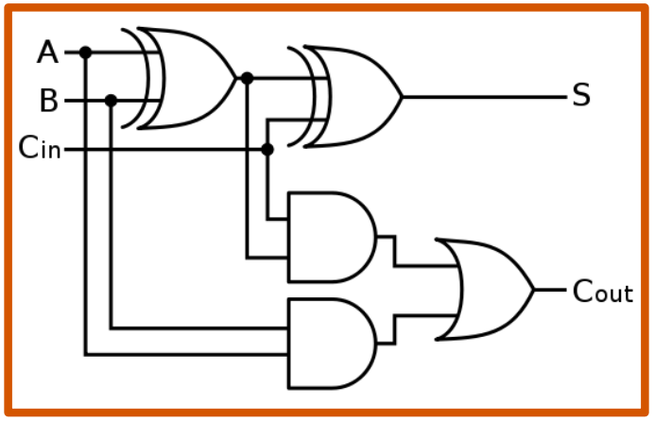

CLB
CLB
CLB
CLB
Configurable logic block (CLB)
Slice
Slice
Slice
Slice
Logic Cell (LC)
Logic Cell (LC)
16-bit SR
16x1 RAM
4-input LUT
a
b
c
d
e
clock
clock enable
set/reset
y
mux
flip-flop
q
Programmed LUT
SRAM cells
1
0
1
1
1
0
1
1
000
001
010
011
100
101
110
111
8:1 Multiplexer
y
a b c

Síntesis e implementación

Diseño RTL

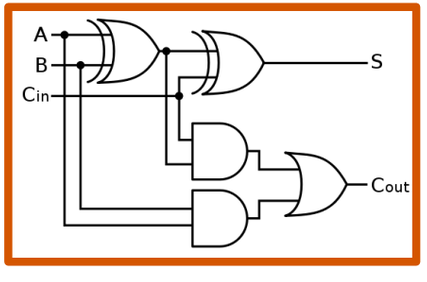
Código a lógica combinacional

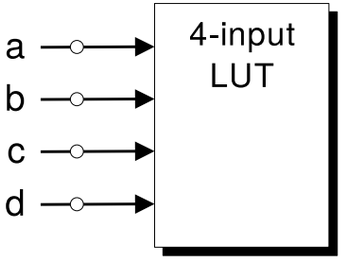
```
module  mux_using_assign(
din_0       , // Mux first input
din_1       , // Mux Second input
sel         , // Select input
mux_out       // Mux output
);
//--------Input Ports-----------
input  din_0, din_1, sel ;
//--------Output Ports----------
output mux_out;
//--------Internal Variables-----
wire   mux_out;
//---------Code Start-----------
assign mux_out = (sel) ? din_1 : din_0;

endmodule //End Of Module mux
```



Síntesis

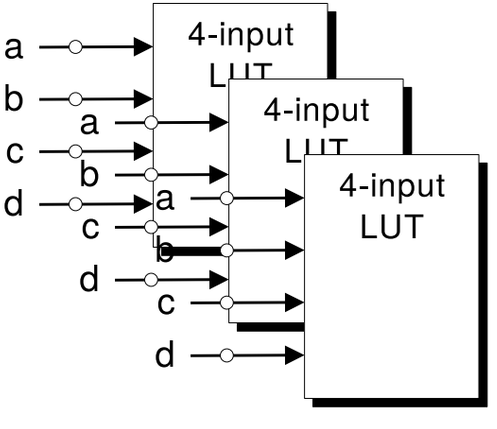
Lógica combinacional a combinación de LUTS

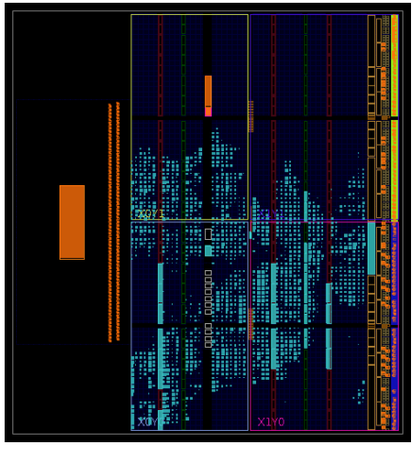




Implementación

Cableado según el hardware disponible





Síntesis e implementación

Diseño RTL
Código a lógica combinacional

Para saber qué entendió el intérprete de verilog.
Chequeo de sintaxis.

Síntesis
Lógica combinacional a combinación de LUTS

Muchas formas de impementar lo mismo (equivalenvias Morgan)

Implementación
Cableado según el hadware disponible

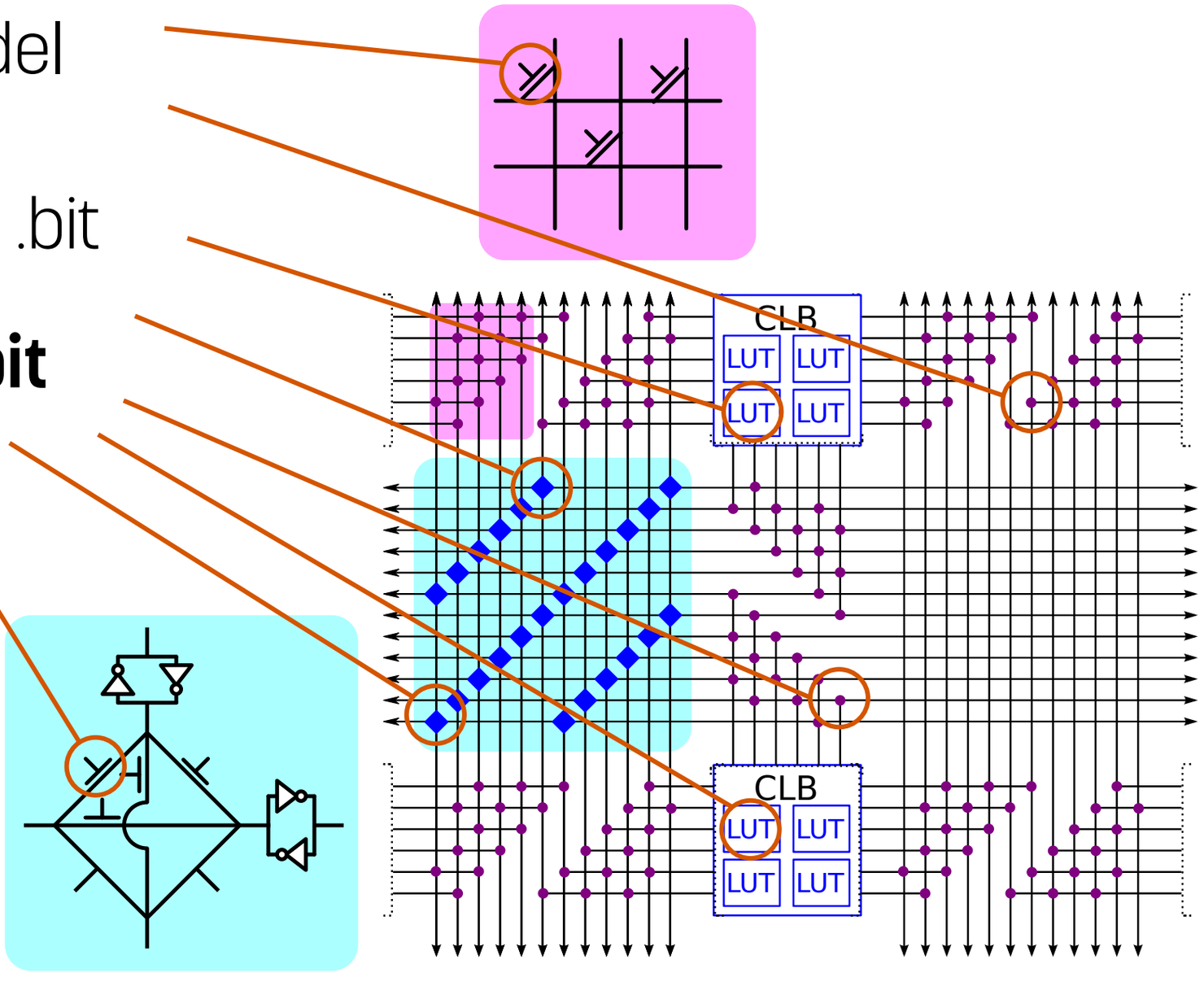
Tiempos de estabilizacion
Uso de superficie (recursos)
Consumo de energía
Líneas de relojes

Síntesis e implementación

Producción del archivo de programación .bit

red_pitaya.bit

```
1101100010010
1111100011010
0100111100110
0001000110001
0100110100010
0110100111010
1010001010100
0010000000001
0101000111001
```



Síntesis e implementación

Producción del
a
prog

red

1101100
1111100
0100111
0001000
0100110
0110100
1010001
0010000
0101000111001

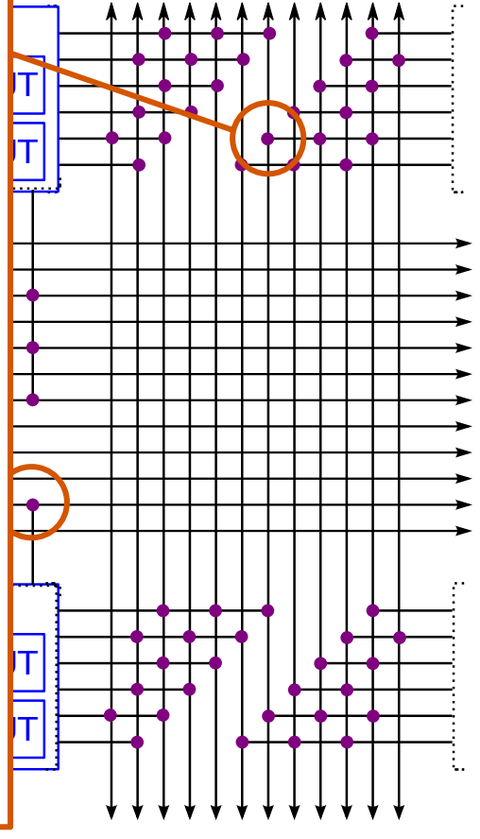
To appear: Proc. 1st Int. Conf. on Evolvable Systems (ICES96). Springer LNCS.

An evolved circuit, intrinsic in silicon, entwined with physics.

Adrian Thompson*

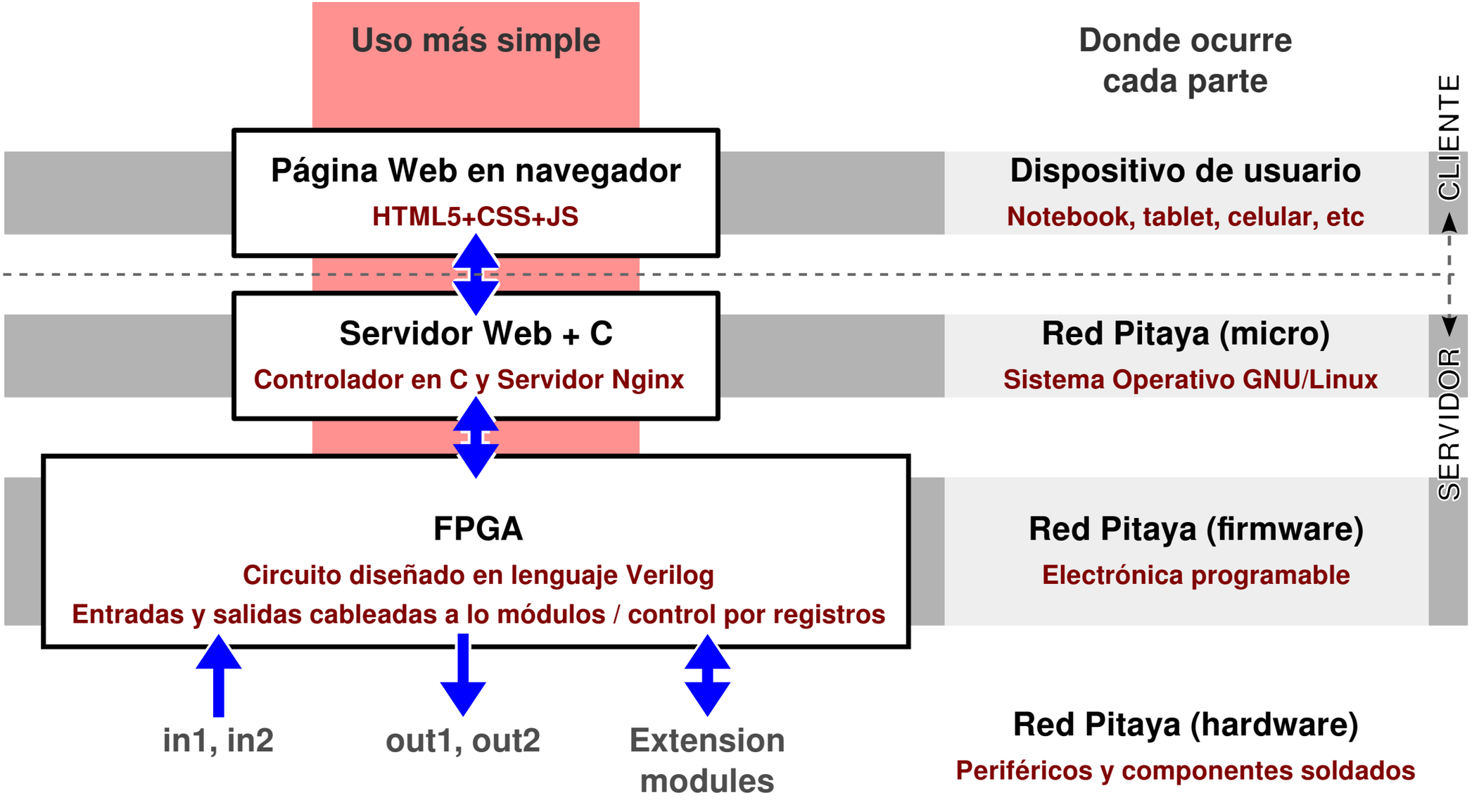
COGS, University of Sussex, Brighton, BN1 9QH, UK

Abstract. 'Intrinsic' Hardware Evolution is the use of artificial evolution — such as a Genetic Algorithm — to design an electronic circuit automatically, where each fitness evaluation is the measurement of a circuit's performance when physically instantiated in a real reconfigurable VLSI chip. This paper makes a detailed case-study of the first such application of evolution directly to the configuration of a Field Programmable Gate Array (FPGA). Evolution is allowed to explore beyond the scope of conventional design methods, resulting in a highly efficient circuit with a richer structure and dynamics and a greater respect for the natural properties of the implementation medium than is usual. The application is a simple, but not toy, problem: a tone-discrimination task. Practical details are considered throughout.



Diseño de aplicaciones en Red Pitaya

Diseño de aplicaciones en Red Pitaya



Diseño de aplicaciones en Red Pitaya

Cliente: Navegador

Donde ocurre cada parte

Uso más simple

Página Web en navegador

HTML5+CSS+JS

Dispositivo de usuario

Notebook, tablet, celular, etc

CLIENTE

Servidor W

Controlador en C y S

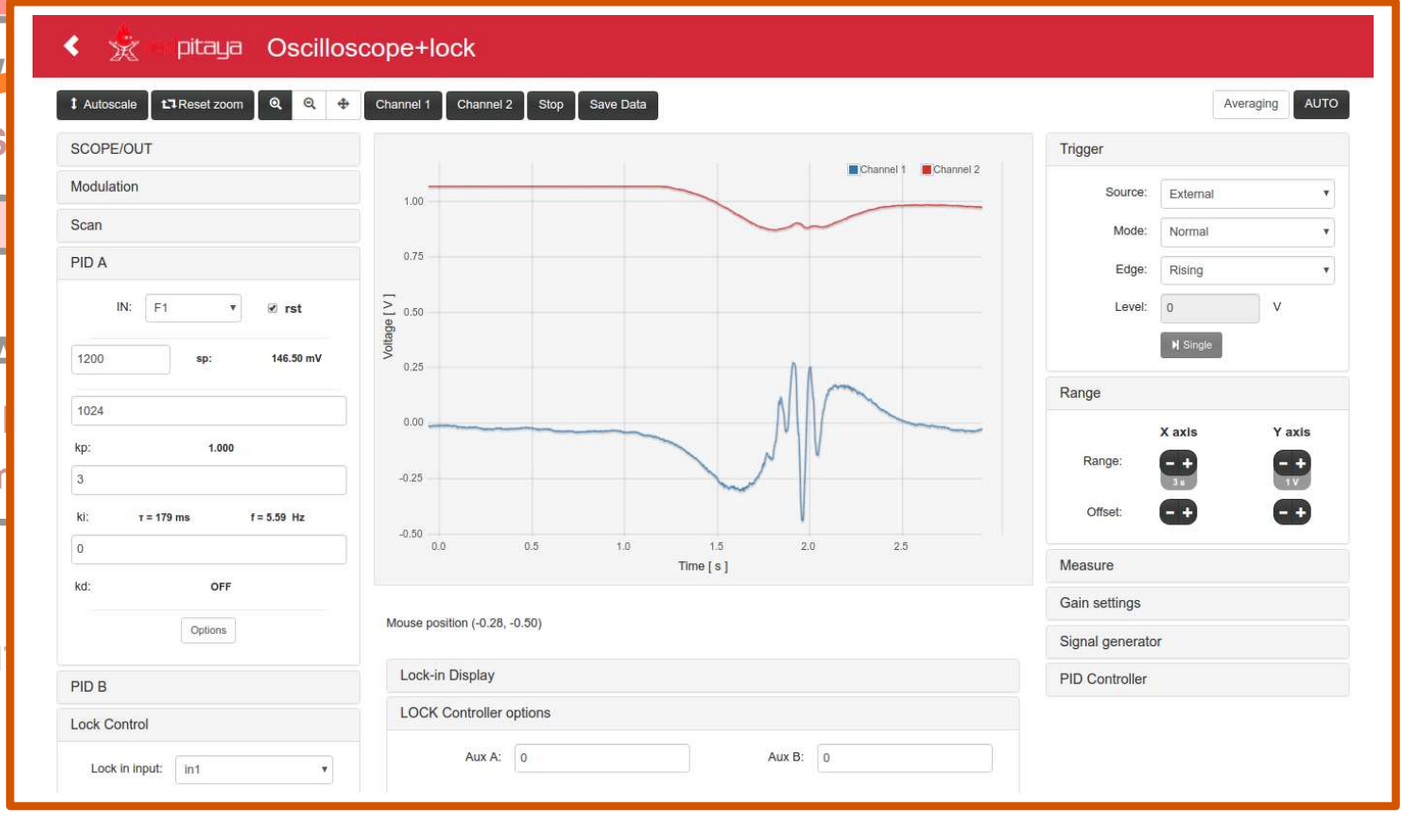
FPGA

Circuito diseñado en

Entradas y salidas cableadas a lo m

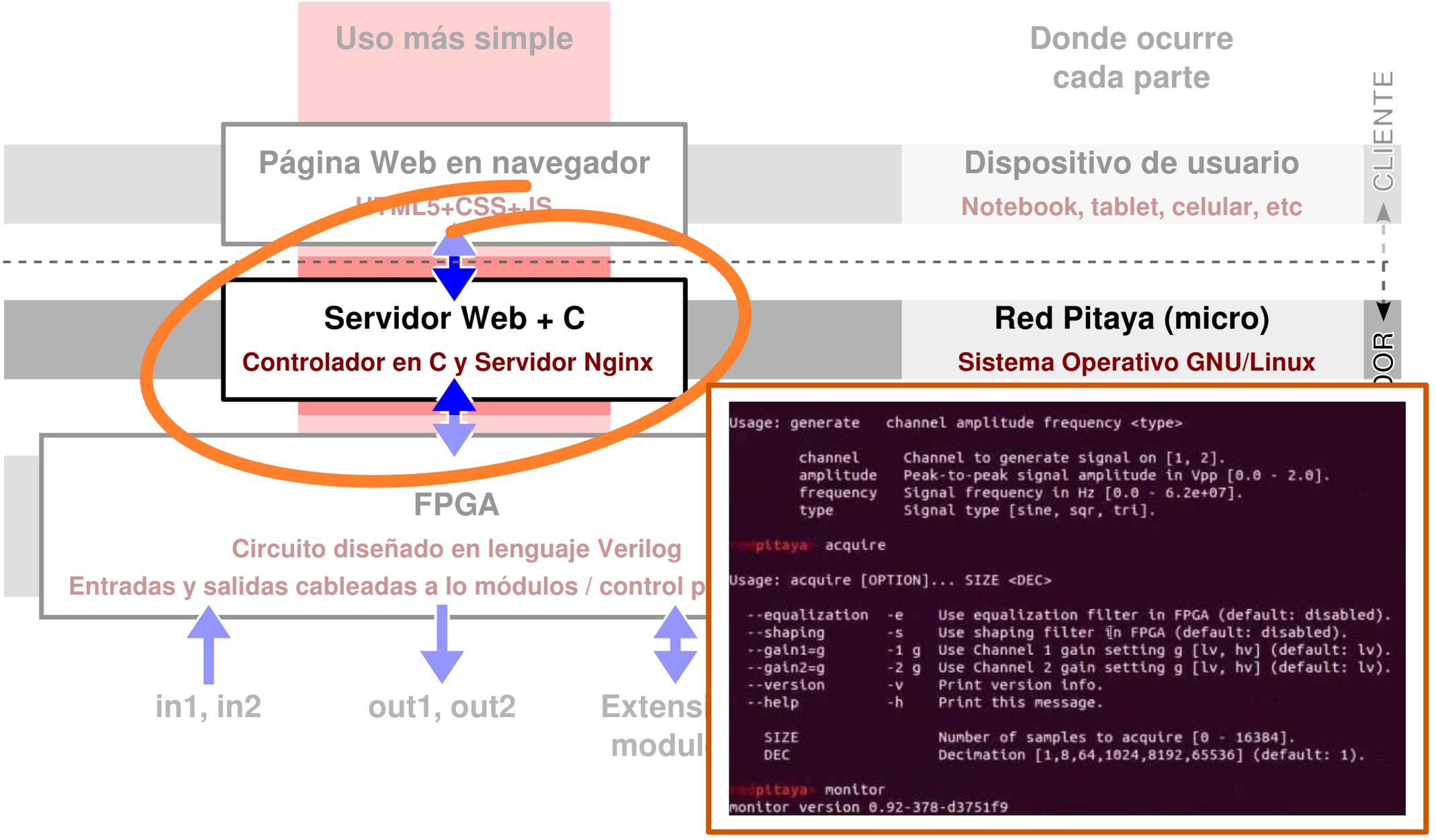
in1, in2

out1, ou

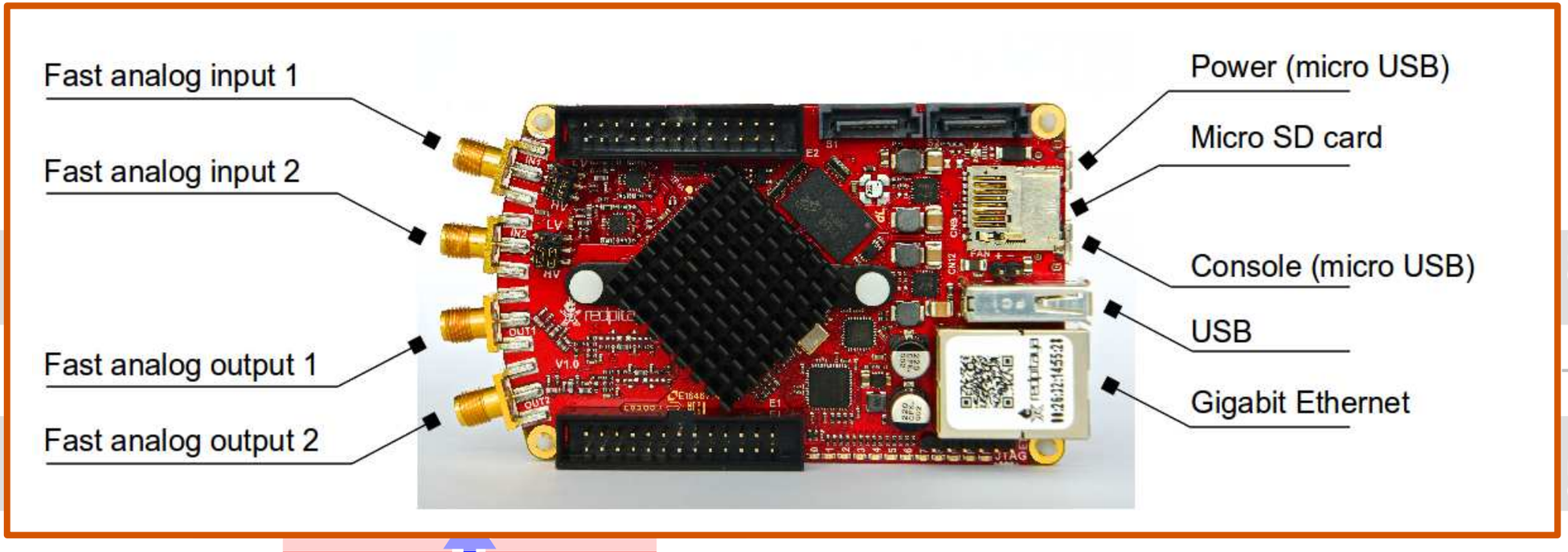


Diseño de aplicaciones en Red Pitaya

Servidor: GNU/Linux Red Pitaya



Diseño de aplicaciones en Red Pitaya



SERVIDOR ◄- - - - ► CLIENTE

FPGA

Circuito diseñado en lenguaje Verilog

Entradas y salidas cableadas a lo módulos / control por registros

Red Pitaya (firmware)

Electrónica programable

in1, in2

out1, out2

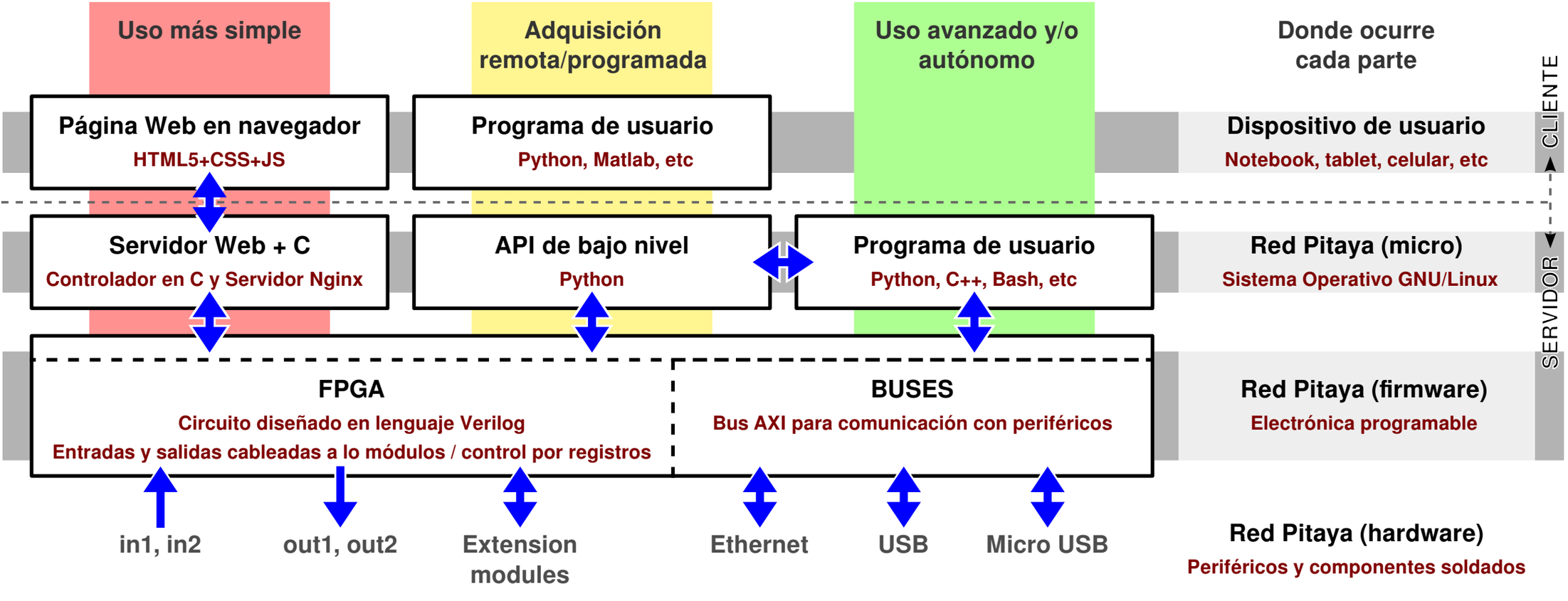
Extension modules

Red Pitaya (hardware)

Periféricos y componentes soldados

FPGA + Hardware

Diseño de aplicaciones en Red Pitaya



Dummy System

Sólo diseñamos FPGA
Automatización de creación y vinculación de controles web

Requisitos
Linaro + GCC + Make
Xilinx 2015.2

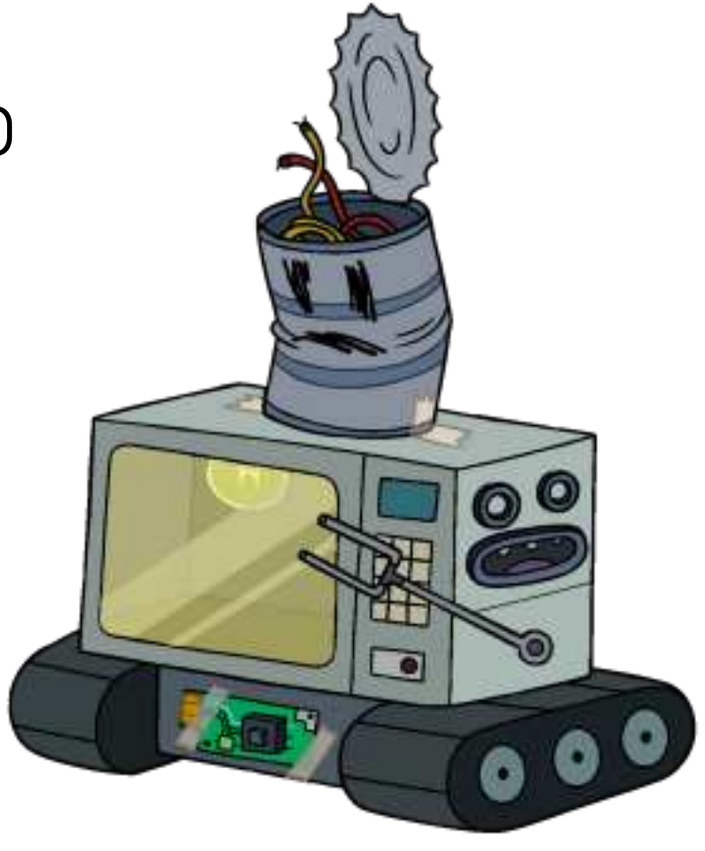
```
git clone https://github.com/marceluda/rp_dummy.git
```

marceluda.github.io/rp_dummy

Dummy System

Editamos archivo config.ini

Para configurar los controles web



Creamos proyecto nuevo

```
source settings.sh
./new_project.py NOMBRE
```

Editamos el archivo dummy.v

```
cd dummy_NOMBRE
geany fpga/rtl/dummy.v
```

Compilamos y subimos a la Red Pitaya

```
./config_tool.py -a
make
./upload_app.sh rp-XXXXXX.local
```

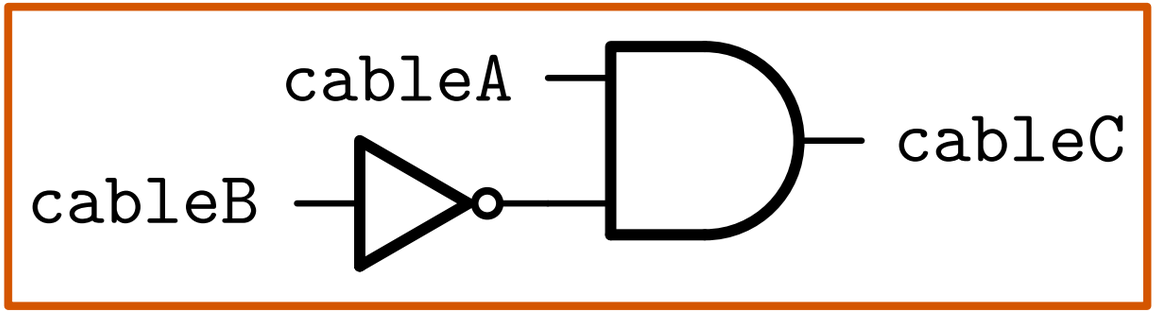
¿Cómo acceder a la Red Pitaya?

http://rp-XXXXXX.local

```
ssh -l root rp-XXXXXX.local
/opt/redpitaya/www/apps/
```

Introducción a Verilog

Diseño de cableado
No es procedimental



Sintaxis parecida a C

```
// Esto es un comenrario de una línea
/* Esto es un
   comentario de
   varias líneas  */
```

```
wire cableA ;
wire cableB, cableC;
assign  cableC = cableA && (~cableB) ;
```

Introducción a Verilog

Definiciones de cables

```
wire cable_simple;

wire [7:0] bus_de_8_bits_A; // sin signo
wire    [  7 : 0]  bus_de_8_bits_B  ; // lo mismo

wire signed [14-1:0] bus_14_bits_A ; // con signo
wire signed [  13:0] bus_14_bits_B ; // con signo
wire        [14-1:0] bus_14_bits_C ; // SIN signo
```

Asignaciones de valores

```
assign cable_simple = 1'b0 ;

assign bus_de_8_bits_A = 8'b01101011 ;
assign bus_de_8_bits_B = 8'd85 ;

assign bus_14_bits_A  = - 14'd24 ;
assign bus_14_bits_C  = $signed(8'b11010011010110);
assign bus_14_bits_D  = 14'd10 ;
```

Ejemplos

```
wire [4-1:0] A,B,C,D;
wire [8-1:0] E,F;

assign A = 4'b0100 ; // 4
assign B = 4'd7    ; // 0111
assign C = A && B  ; // 0100 and 0111 = 0100 = 4
assign D = A || B  ; // 0100  or 0111 = 0111 = 7

assign E = { A , B }; // 01000111 = 71
assign D = { E[6:3] , |A , &B , 2'b01 };
/*           1000   ,  1 ,  0 , 01  */
```

Introducción a Verilog

Operadores

```
// NOT
~A
// cocatenacion
{A, ..., B}
// aritméticos
A*B A+B A-B
// shift
A<<B A>>B
// comparacion
A>B A<B A>=B A<=B
A==B A!=B
// Bit-wise
A&B
A^B A~^B
A|B
// Logicos
A&&B
A||B
// Condicional
A ? B : C
```

Binario con signo

0	000
1	001
2	010
3	011
-1	111
-2	110
-3	101
-4	100

```
- A == (~ A) + 1
```

Módulos

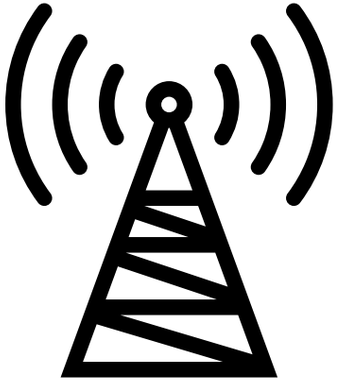
```
// Probamos un pasaaltos
filtro_pasaaltos #(.R(14)) NOMBRE (
     .clk(clk), .rst(rst),
     // inputs
     .tau( comboA      ),
     .dis( checkboxA   ),
     .in(  in1         ),
     // outputs
     .out(  salida_pasaaltos  )
);
```

Los módulos no son funciones

Se instancian: se replica el cableado

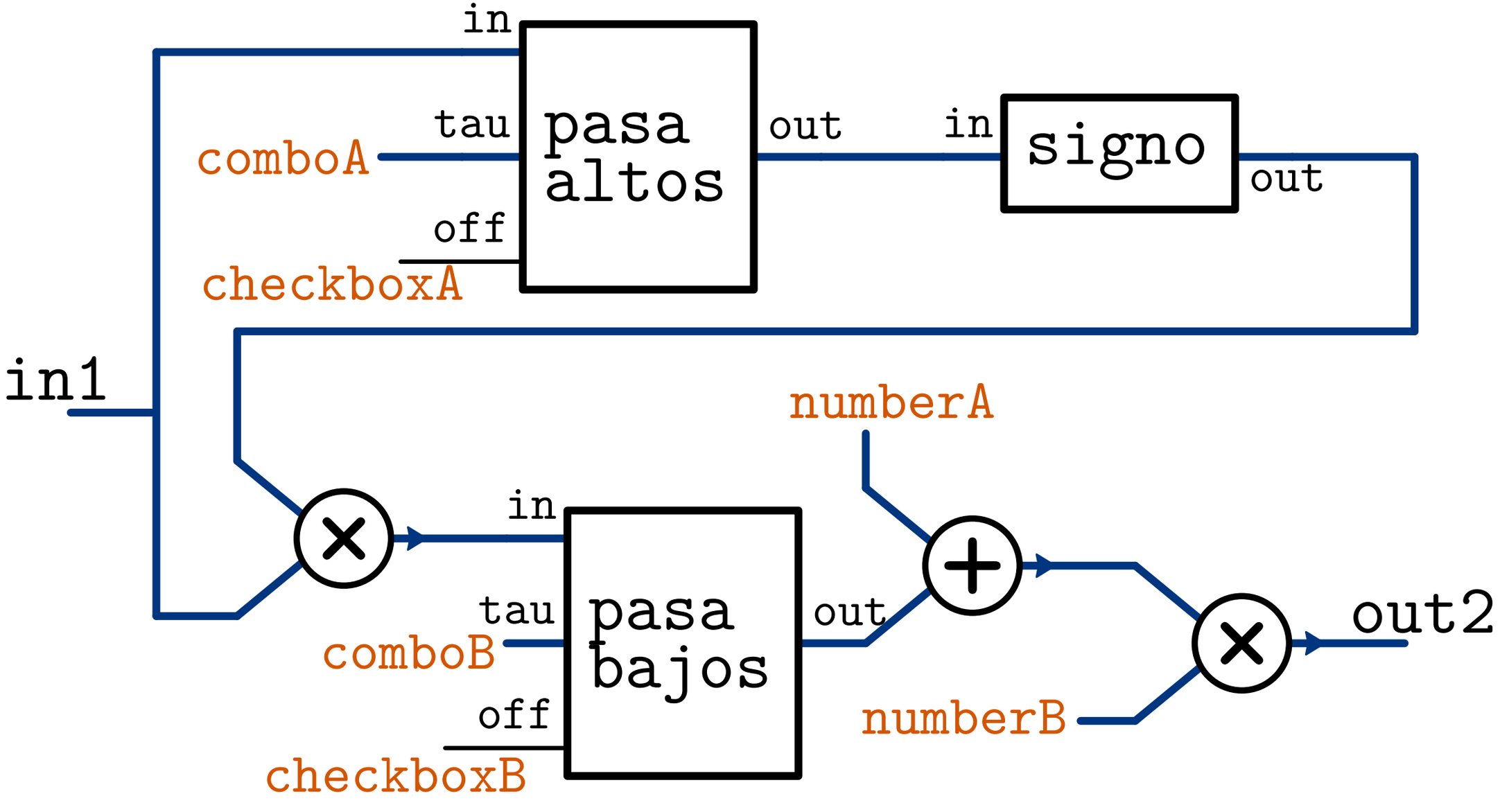
Parámetros en tiempo de diseño
E/S en tiempo de "ejecución"

Actividad propuesta

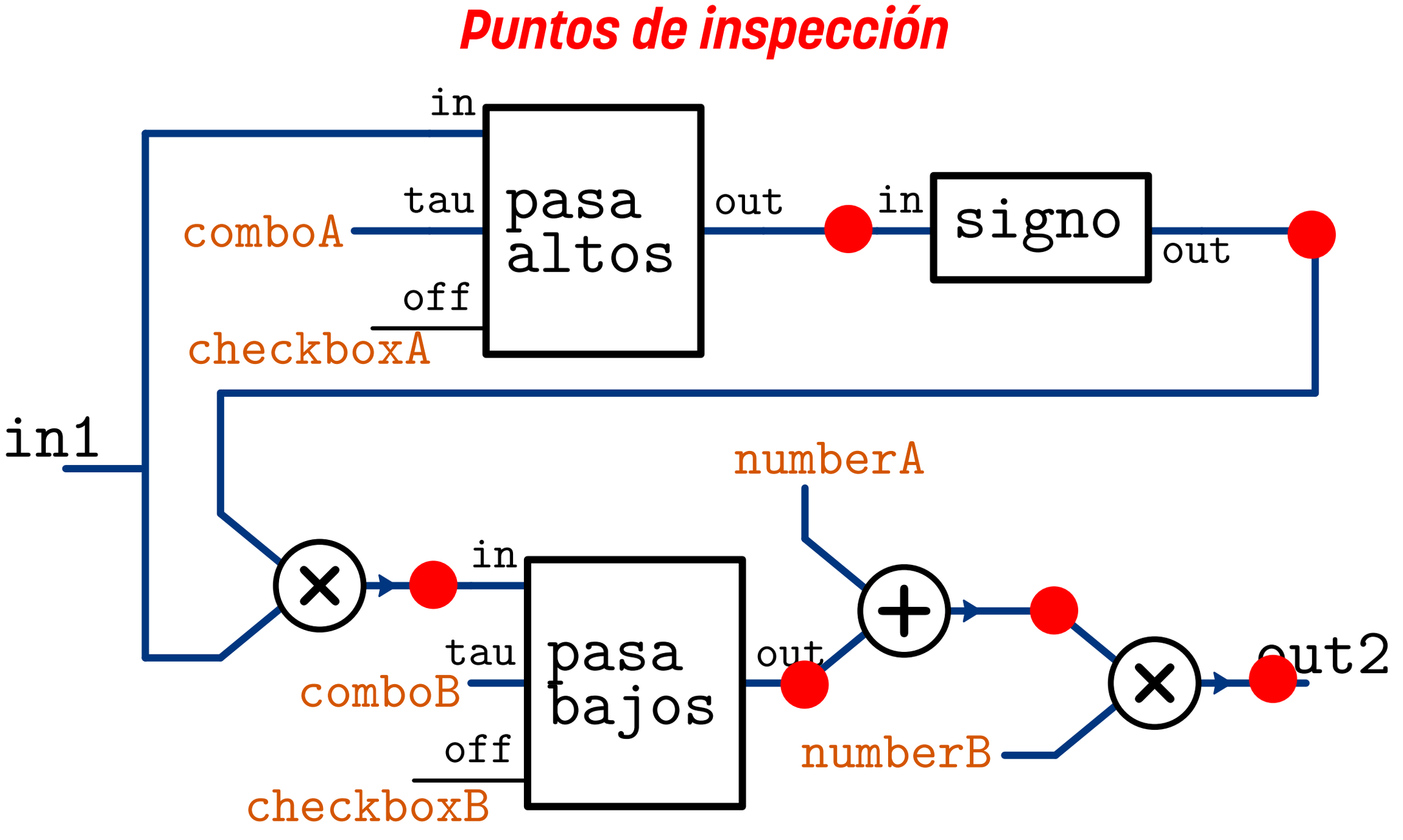


Señal de Amplitud Modulada (AM)
Portadora en 1 MHz
Interferencia en 100 Hz
Información: (?)

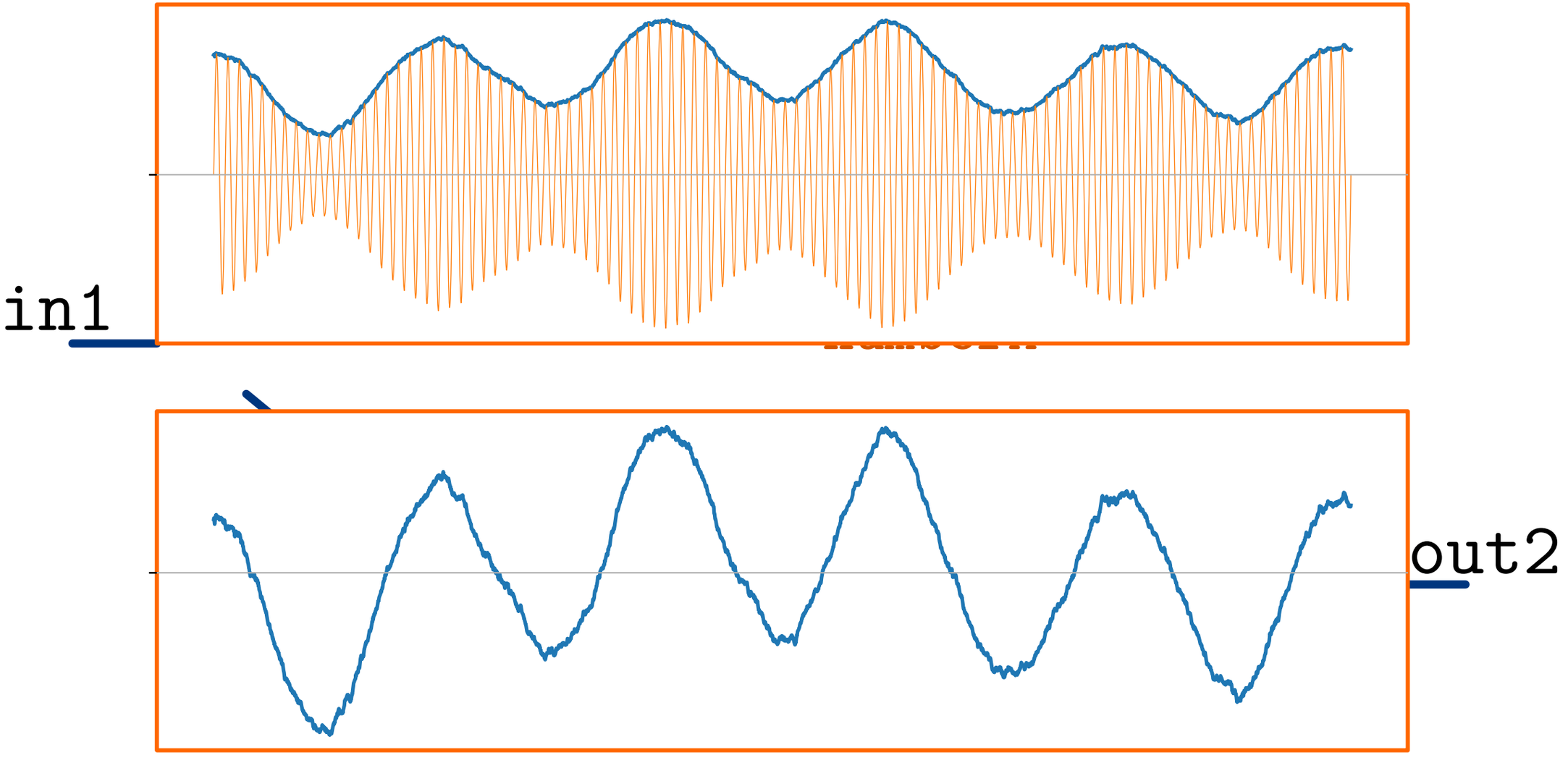
Actividad propuesta



Actividad propuesta

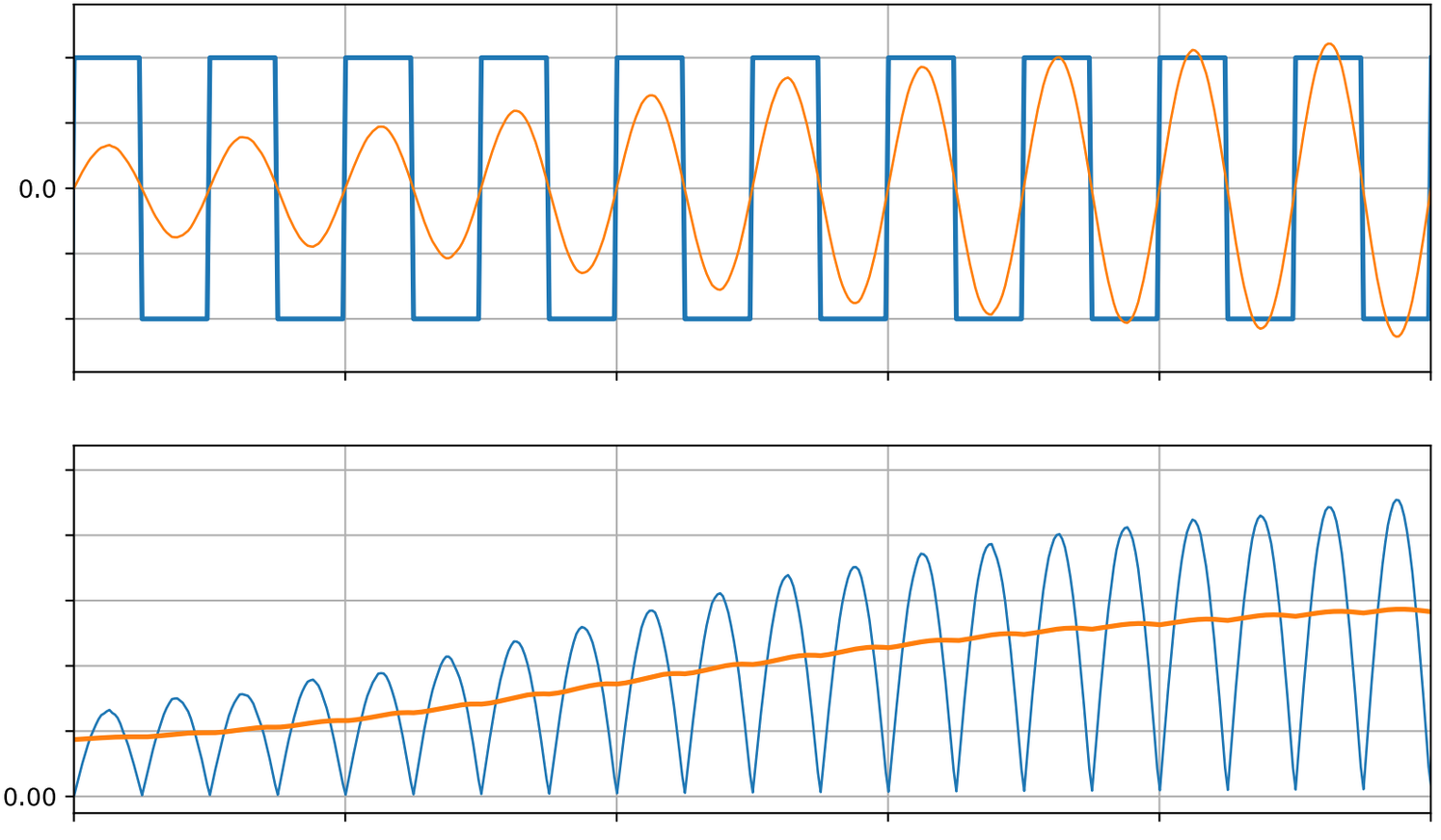


Actividad propuesta



Actividad propuesta

Demodulación



in1

out2

checkboxB

Actividad propuesta