Introducción a las FPGA

Introducción a la Microfabricación y las FPGA

Instituto Balseiro

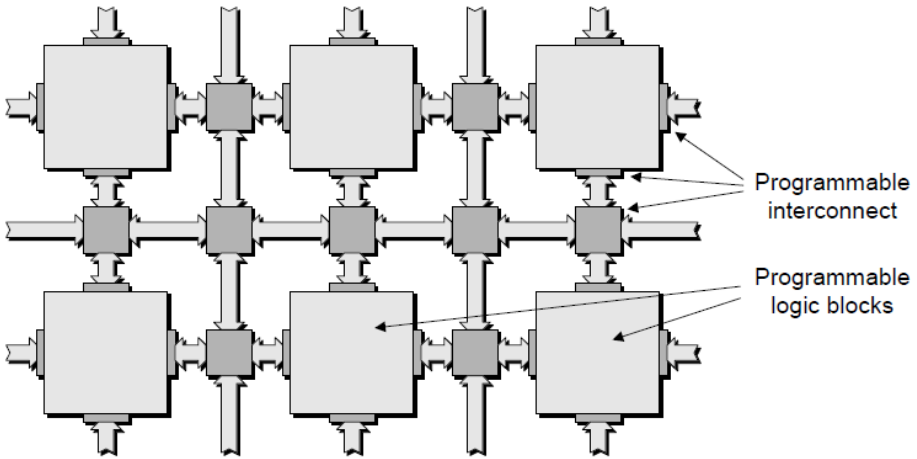
28 de Julio 2014

Menú del Día

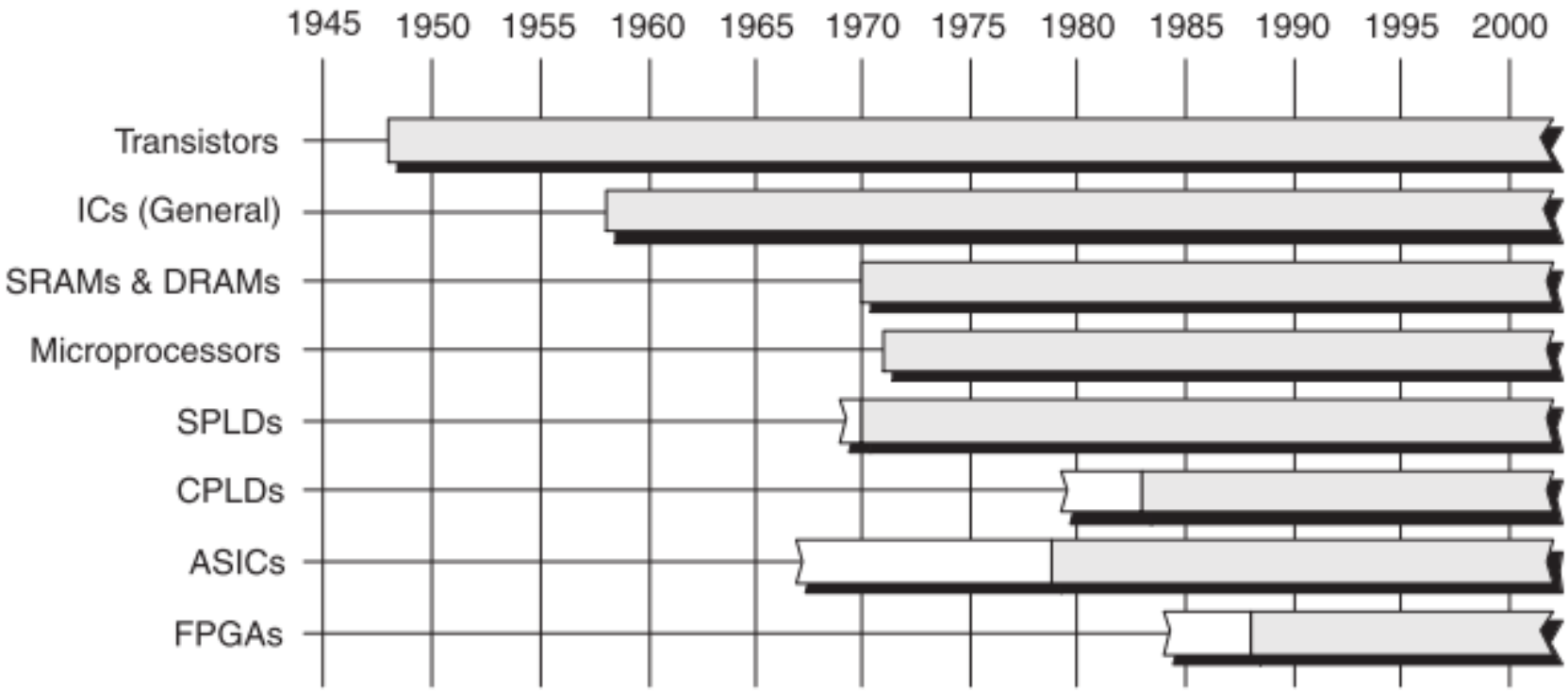
- Qué es una FPGA. Para qué se usan. Arquitecturas.
- Flujo de diseño. Introducción a VHDL.
- El "Hola Mundo!´´ del VHDL.
- ISE/ISim, y la primera vez que prendemos un led.

¿Qué es una FPGA?

- Circuitos Integrados que contienen bloques configurables de lógica junto con conexiones configurables entre esos bloques.
- Los FPGA se programan "in the field", o sea, no los programa el fabricante, sino que lo puede programar el desarrollador "en el campo".



¿De dónde vienen?

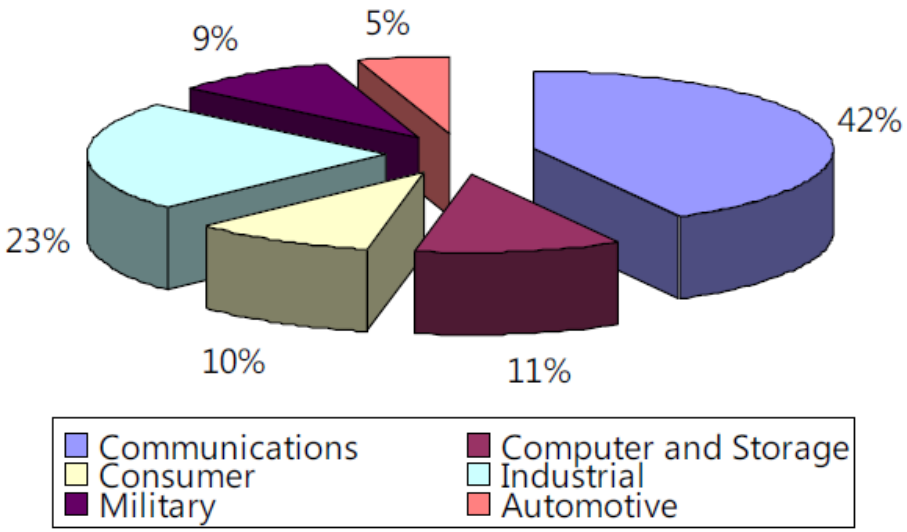


¿Porqué son de interés?

- Para SE que requieren procesamiento de datos masivo que no puede resolverse con procesadores. ASICs o FPGAs.
- Hoy en día las FPGAs tienen millones de gates y se pueden utilizar para implementar funciones extremadamente complejas que antes sólo se podían lograr con ASICs.
  - Costo de diseño menor.
  - Mas facil de cambiar.
  - Menor time-to-market.
- Permiten pequeñas empresas que testeen sus ideas/diseños con inversion inicial mucho menor. Verificación.

¿Para qué se usan?

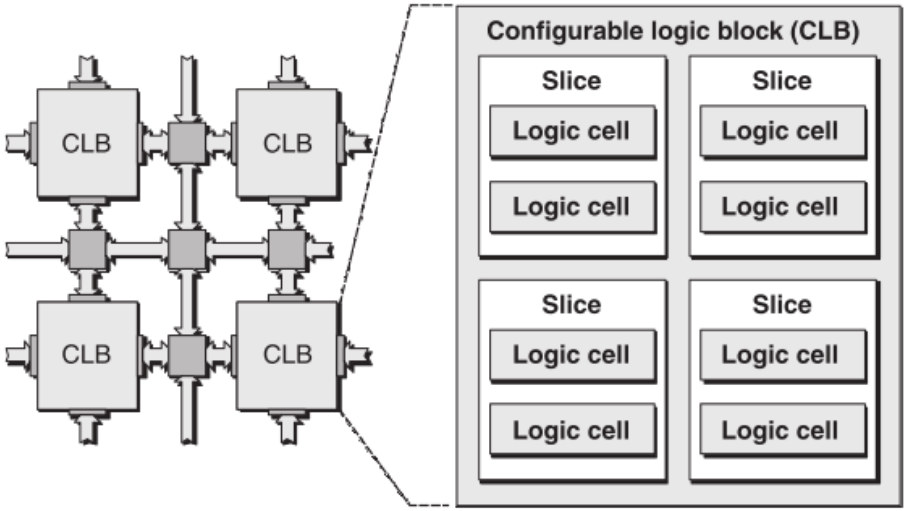
- 1980 - Glue logic.
- 1990 - Prototipado, verificación.
- 2000 - Cada vez mas complejas y con menor costo $\rightarrow$ empiezan a meterse en productos finales.
  - ASIC
  - DSP
  - Embedded microcontrollers.
  - Capa física de comunicación (routers).
  - Computacion Reconfigurable.
- Industria de 4 billones de dólares: Hoy, el 35 % de los ingenieros de SE usan FPGAs en sus diseños (2012 Embedded Market Survey)

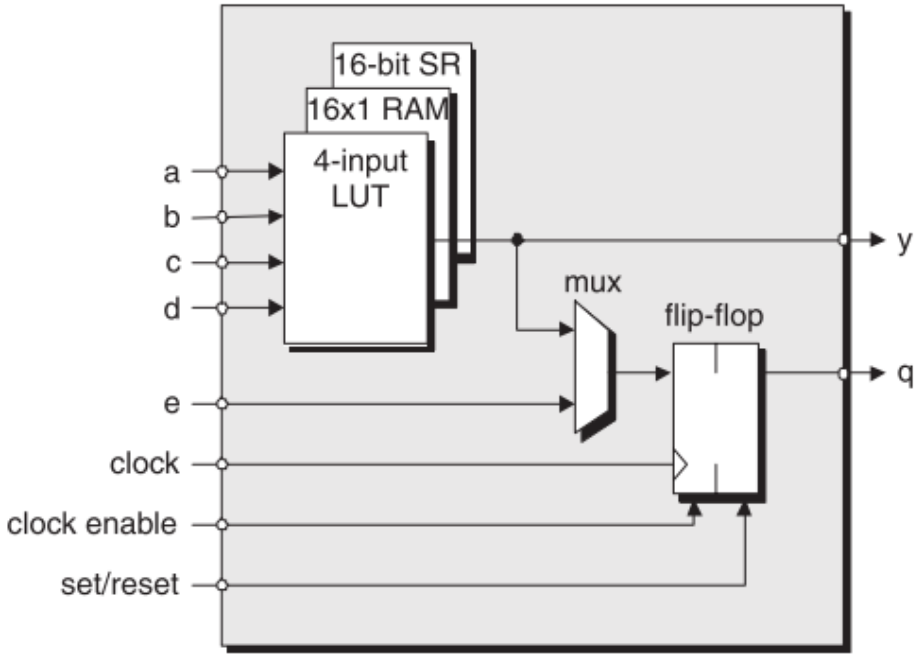


Arquitectura

- Bloques Lógicos
- Matriz de ruteo & Señales globales
- Bloques I/O
- Recursos de Reloj
- Memoria embebida
- Bloques multiplicadores, sumadores, DSP
- Caractaerísticas Avanzadas: procesadores hard embebidos, etc.

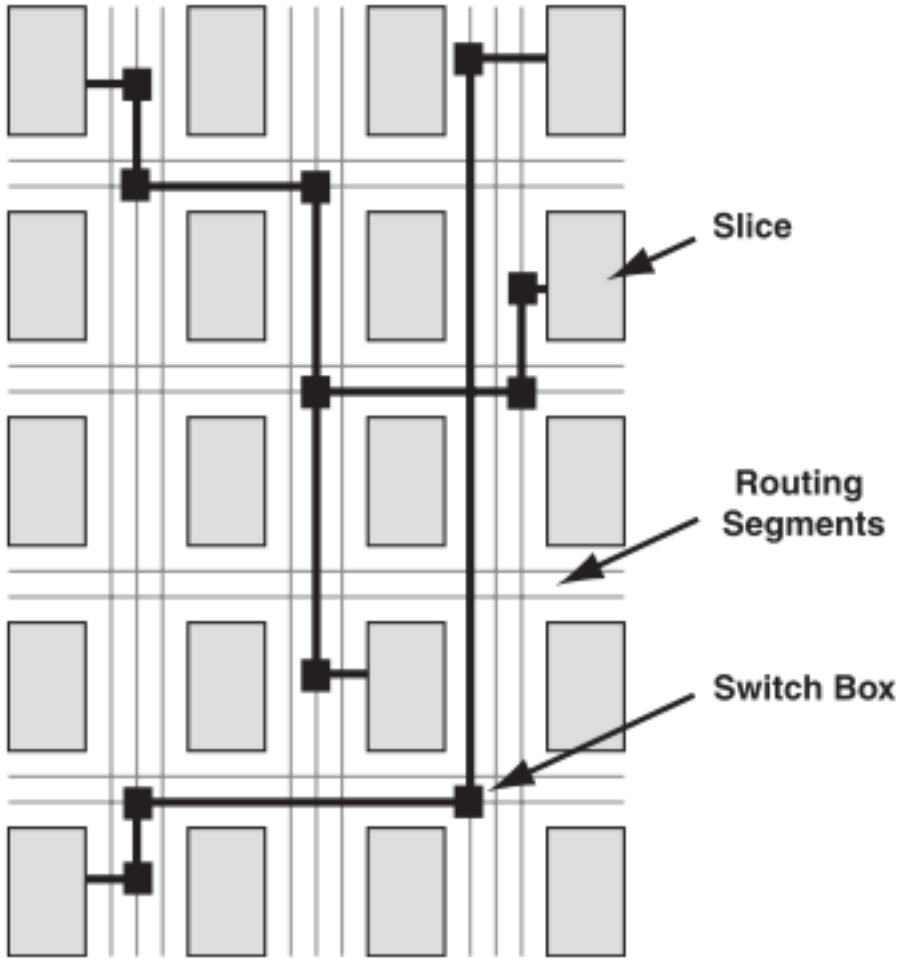
Bloques Lógicos





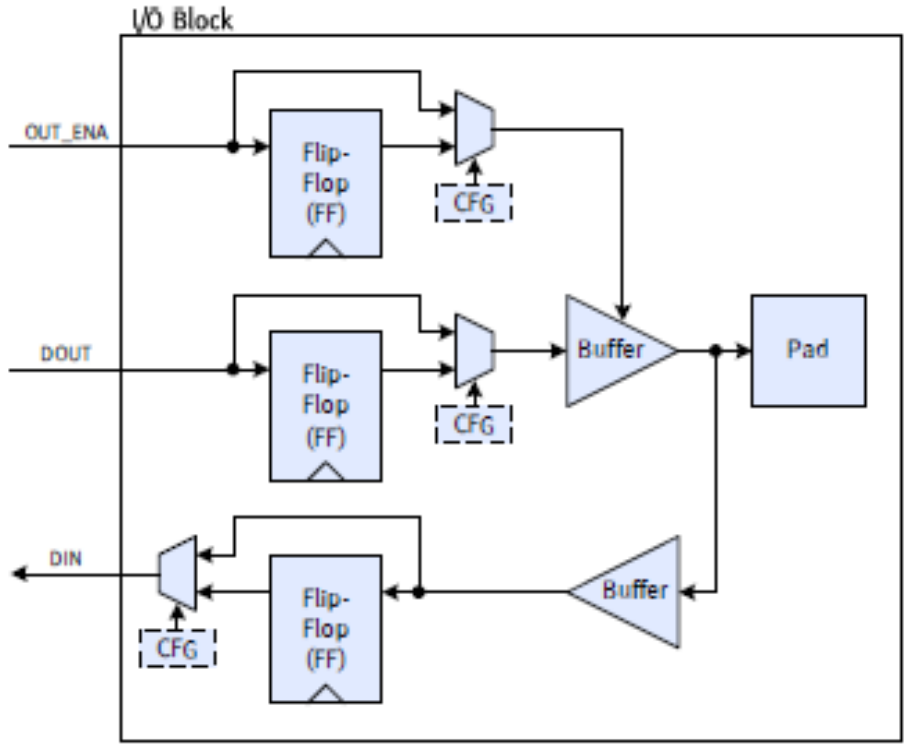
Matriz de ruteo & Señales globales

- Canales de ruteo Horizontales/Verticales y switches programables.
- Carry-chain.
- Global low-skew routing.



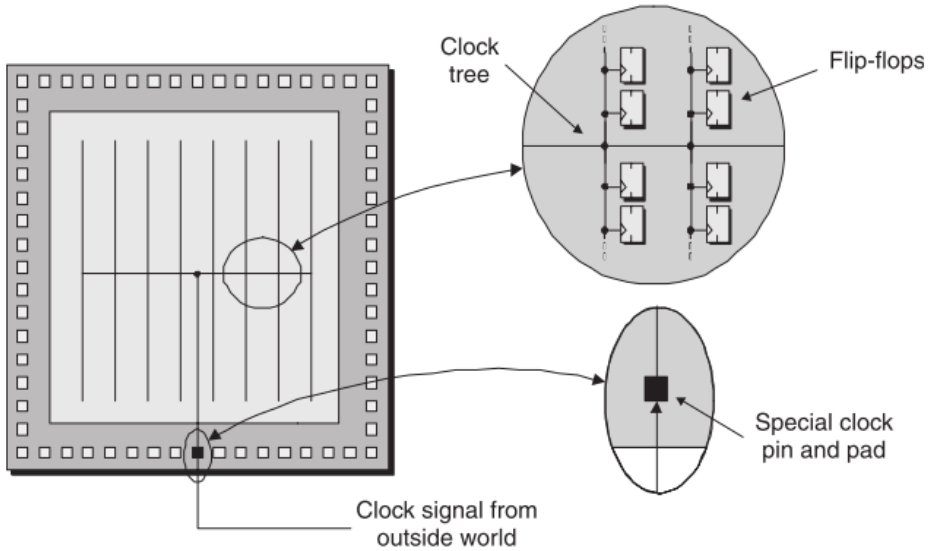
Bloques de I/O

- Dirección (I,O,bi)
- Data Rate (SDR, DDR, SERDES)
- Standard (single-ended, differential, referenced,etc).
- Voltaje (1.2 V to 3.3 V for single-ended standards)

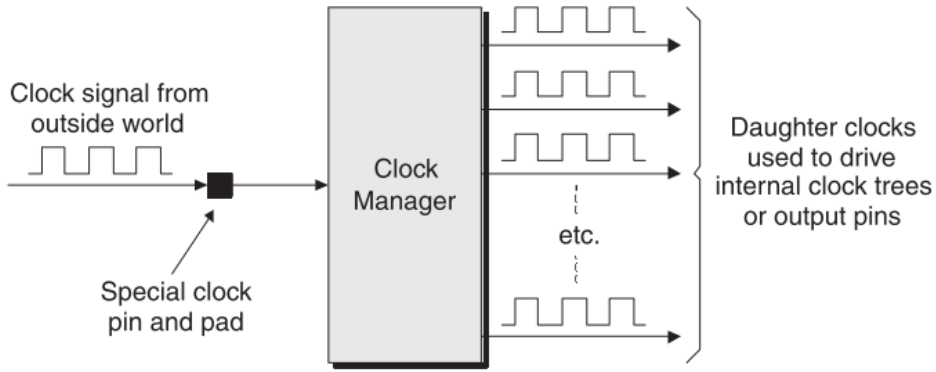


Clock Resources

Árbol de Clock

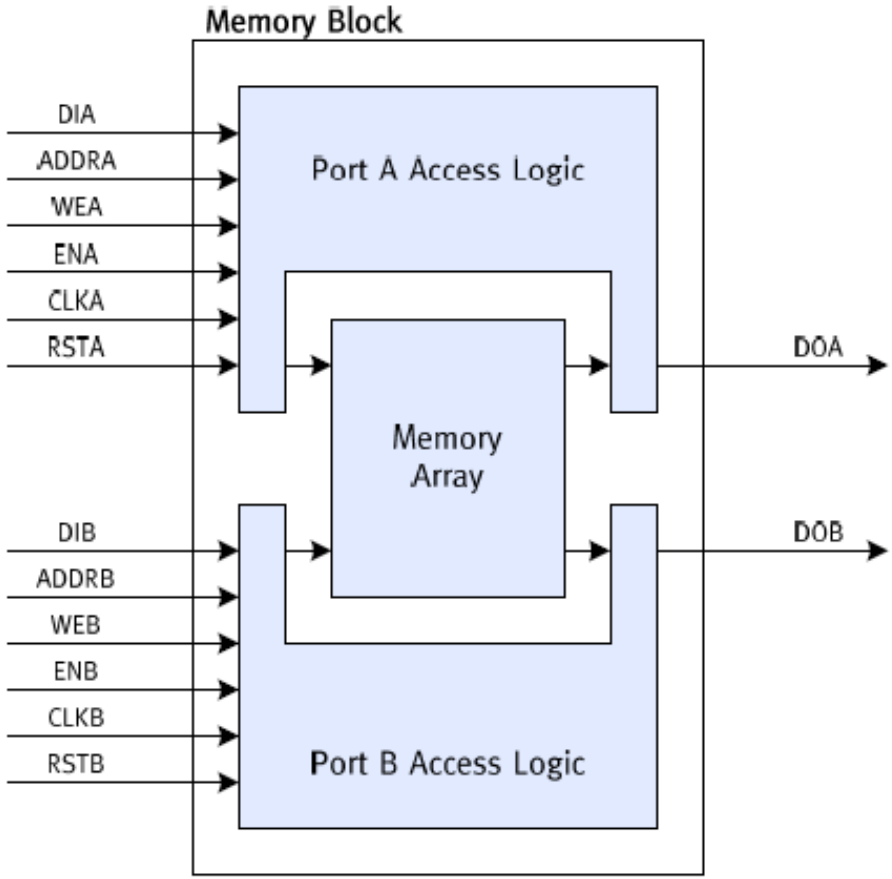


Digital Clock Manager



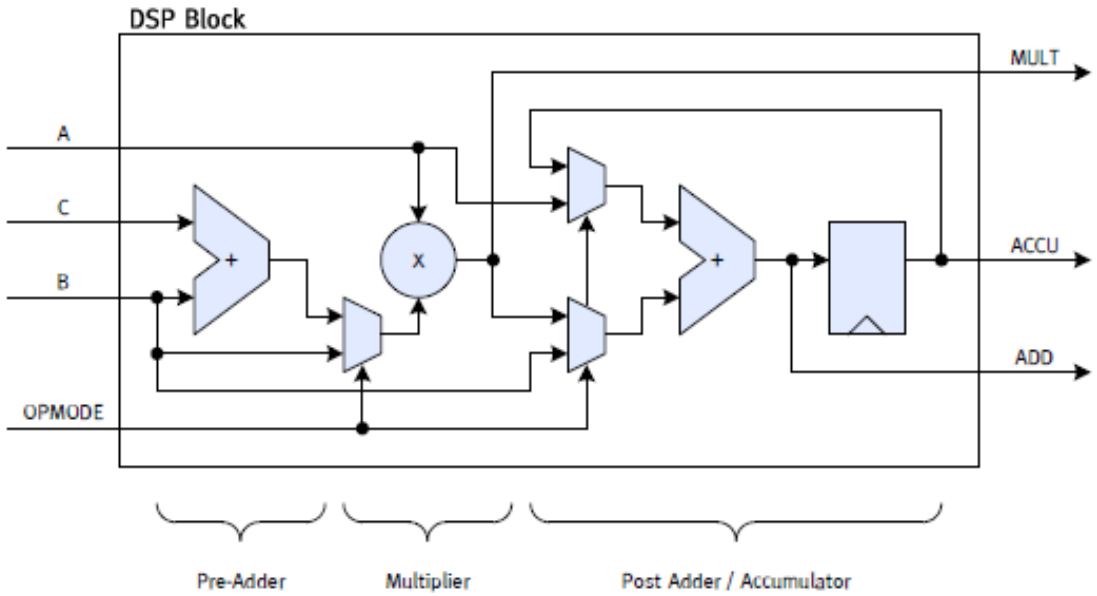
Memoria Embebida (Block RAM)

- Dentro y/o alrededor de la lógica.
- Cada BRAM se puede usar de manera independiente o combinada para implementar bloques mas grandes.
- Se pueden usar para implementar single y dual-port BRAMs, FIFOs, FSMs, etc.



Bloques DSP

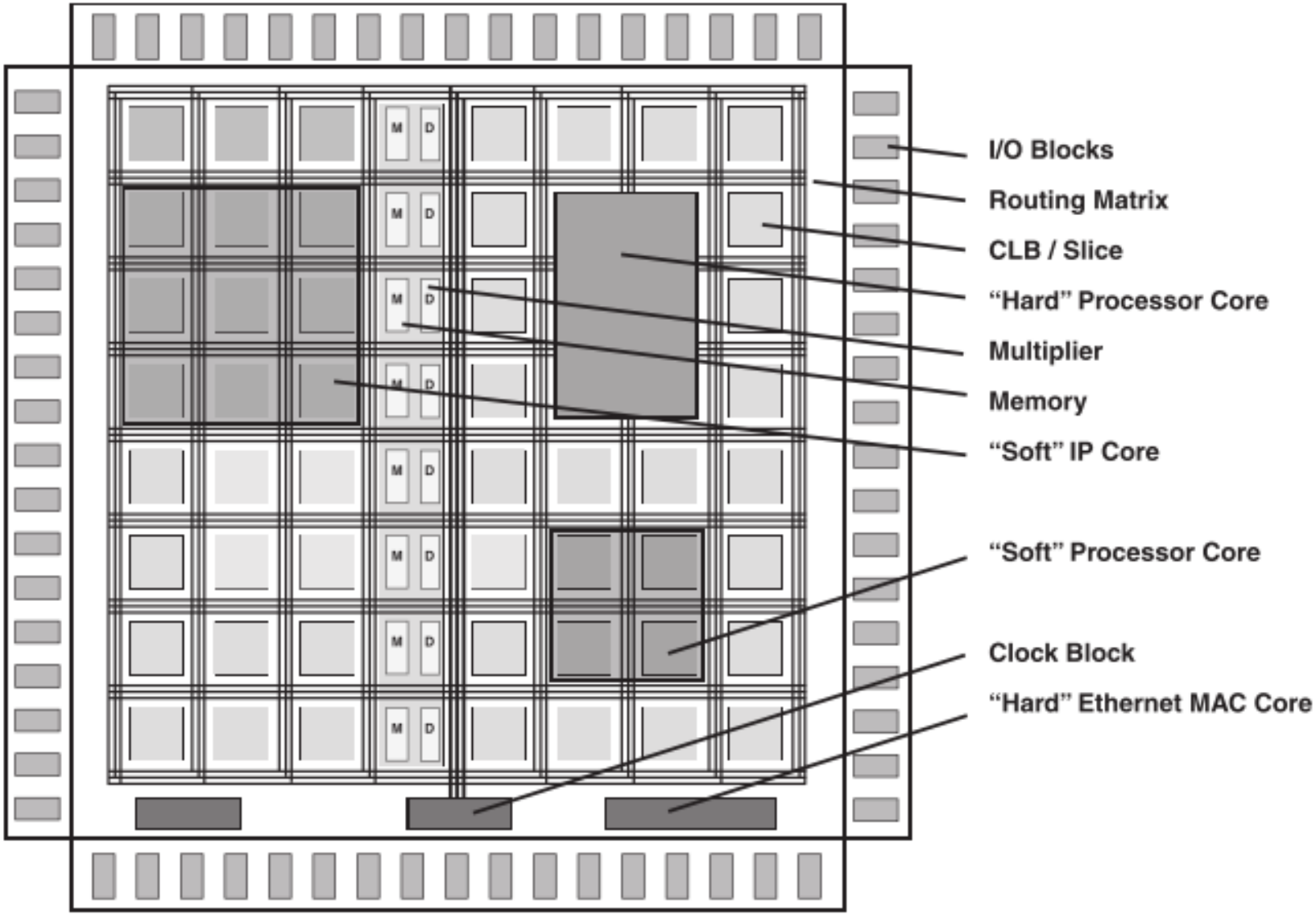
- Multiplicación, Sumador, MACC.
- En general cerca de las BRAMs.



Características Avanzadas

- Características particulares: controladores de memoria externa.
- I/O de alta velocidad con controladores para protocolos avanzados.
- Procesadores hard embebidos.

La figura completa



La Spartan-6 de la placa Nexsys3

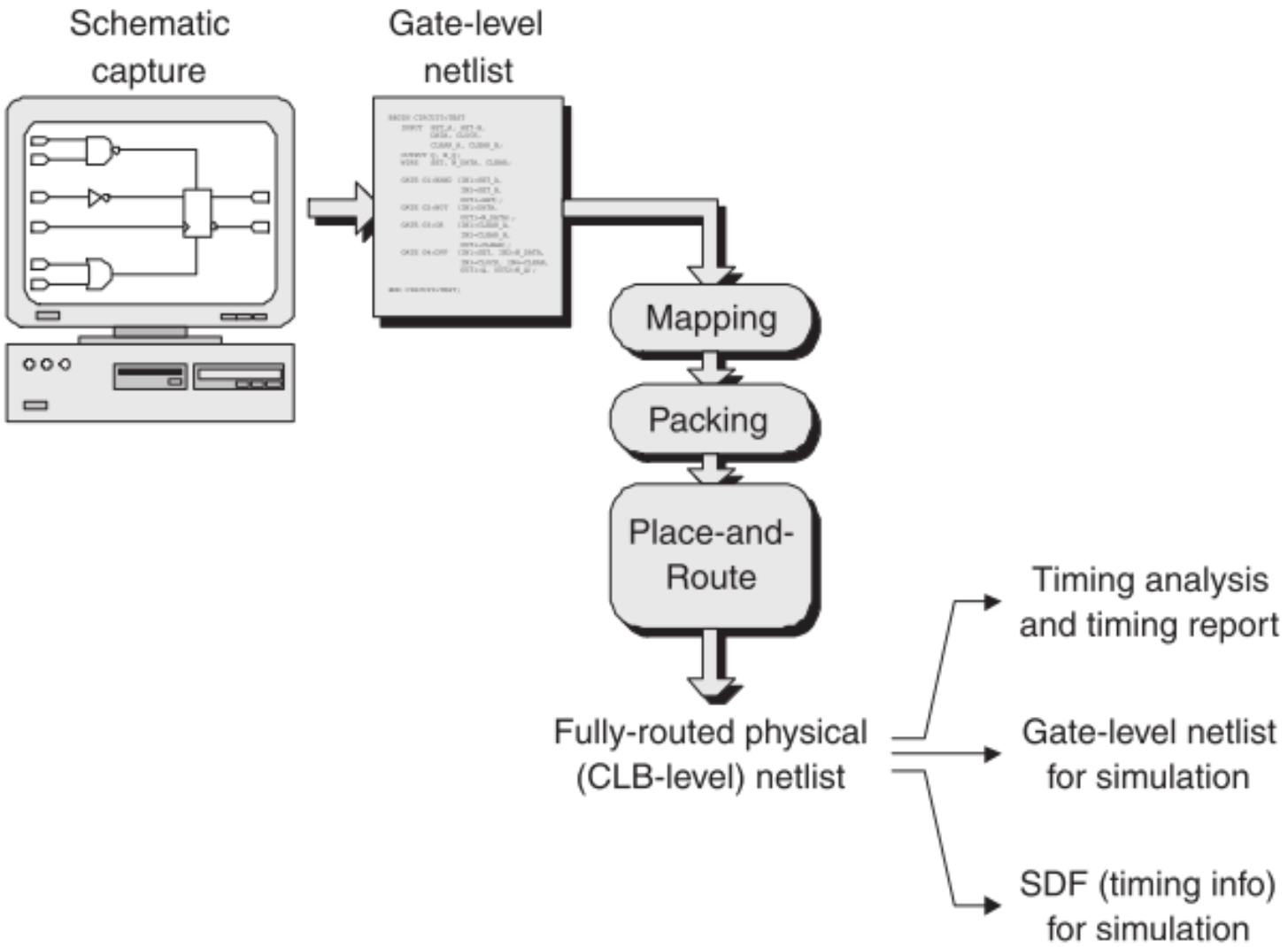
Device	Logic Cells[1]	Configurable Logic Blocks (CLBs)			DSP48A1 Slices[3]	Block RAM Blocks		CMTs[5]	Memory Controller Blocks (Max)[6]	Endpoint Blocks for PCI Express	Maximum GTP Transceivers	Total I/O Banks	Max User I/O
		Slices[2]	Flip-Flops	Max Distributed RAM (Kb)		18 Kb[4]	Max (Kb)						
XC6SLX4	3,840	600	4,800	75	8	12	216	2	0	0	0	4	132
XC6SLX9	9,152	1,430	11,440	90	16	32	576	2	2	0	0	4	200
XC6SLX16	14,579	2,278	18,224	136	32	32	576	2	2	0	0	4	232
XC6SLX25	24,051	3,758	30,064	229	38	52	936	2	2	0	0	4	266
XC6SLX45	43,661	6,822	54,576	401	58	116	2,088	4	2	0	0	4	358
XC6SLX75	74,637	11,662	93,296	692	132	172	3,096	6	4	0	0	6	408
XC6SLX100	101,261	15,822	126,576	976	180	268	4,824	6	4	0	0	6	480
XC6SLX150	147,443	23,038	184,304	1,355	180	268	4,824	6	4	0	0	6	576
XC6SLX25T	24,051	3,758	30,064	229	38	52	936	2	2	1	2	4	250
XC6SLX45T	43,661	6,822	54,576	401	58	116	2,088	4	2	1	4	4	296
XC6SLX75T	74,637	11,662	93,296	692	132	172	3,096	6	4	1	8	6	348
XC6SLX100T	101,261	15,822	126,576	976	180	268	4,824	6	4	1	8	6	498
XC6SLX150T	147,443	23,038	184,304	1,355	180	268	4,824	6	4	1	8	6	540

Notes:

1. Spartan-6 FPGA logic cell ratings reflect the increased logic cell capability offered by the new 6-input LUT architecture.
2. Each Spartan-6 FPGA slice contains four LUTs and eight flip-flops.
3. Each DSP48A1 slice contains an 18 x 18 multiplier, an adder, and an accumulator.
4. Block RAMs are fundamentally 18 Kb in size. Each block can also be used as two independent 9 Kb blocks.
5. Each CMT contains two DCMs and one PLL.
6. Memory Controller Blocks are not supported in the -3N speed grade.

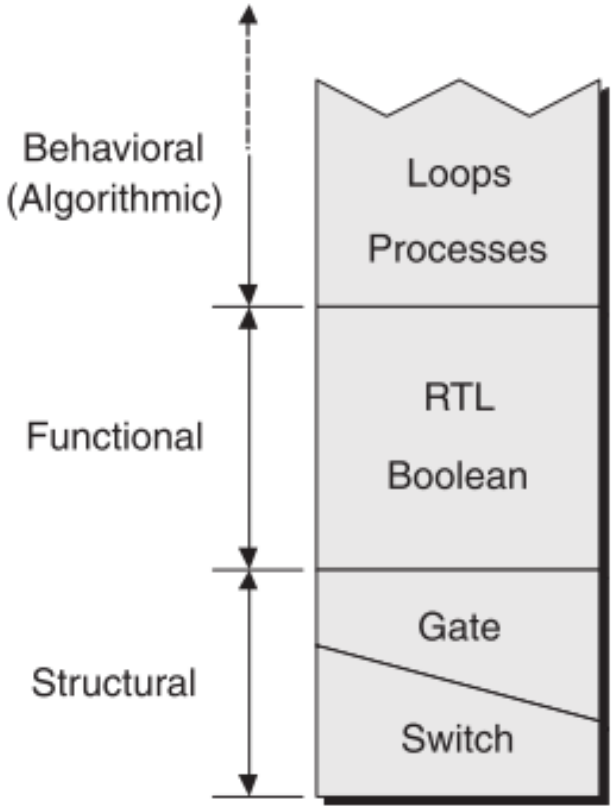
Basado en esquemático

Hace tiempo...



HDL

- Visualizar, capturar, debuggear, entender y mantener un diseño a nivel de compuertas lógicas se torna imposible pasado cierta complejidad – Fines de 1980.
- Lenguajes de Descripción de Hardware: primeros orientados a gate level, luego simulación.
- Clave: **herramientas** cada vez mas poderosas de síntesis lógica, mapeo, etc.

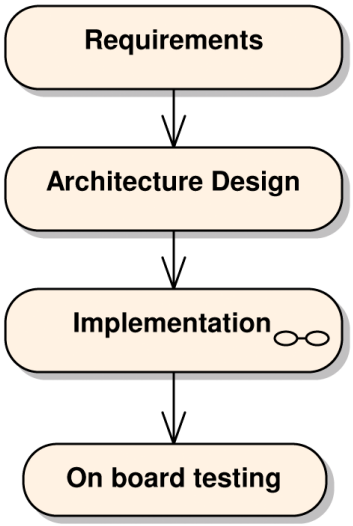


Niveles de abstracción

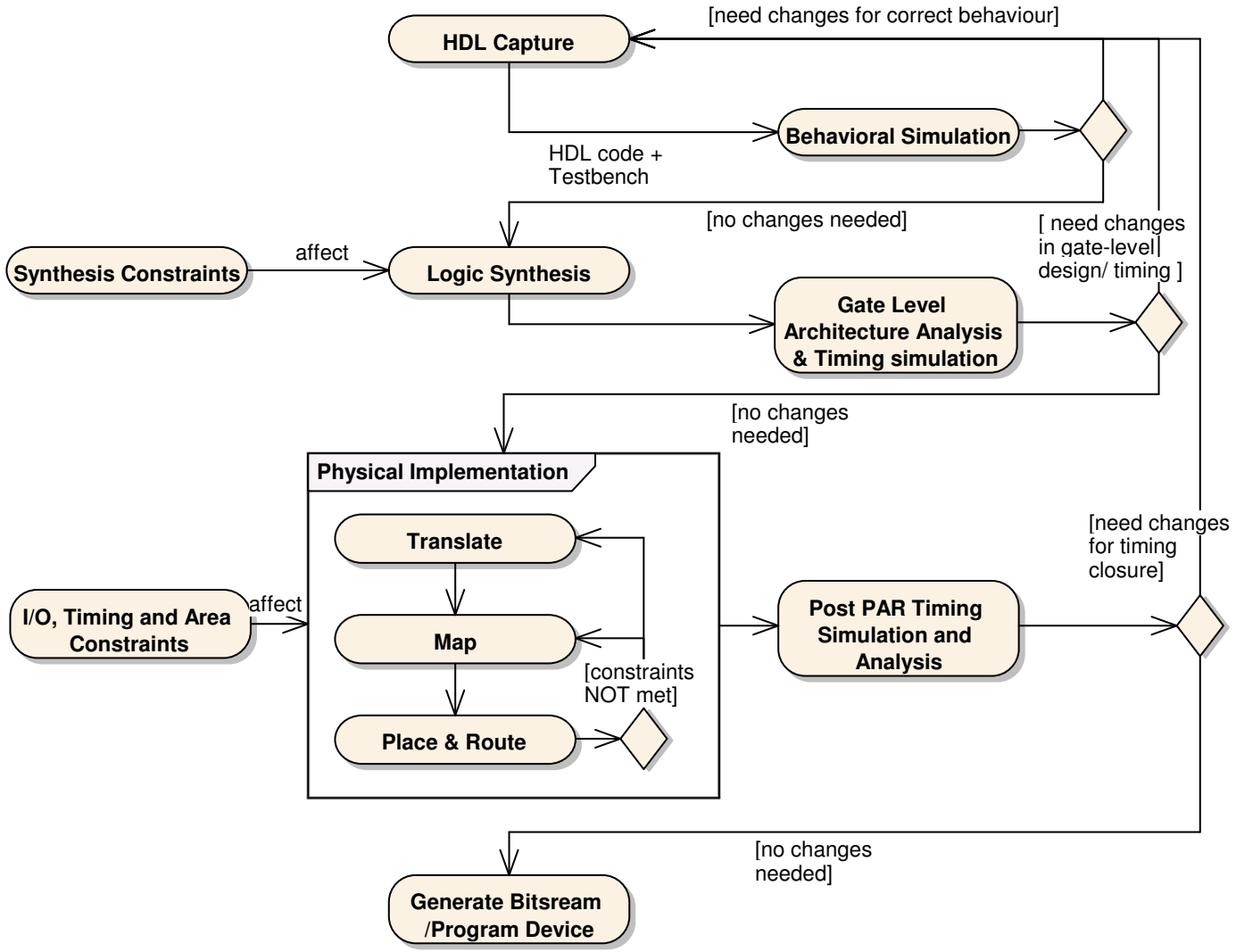
	typical blocks	signal representation	time representation	behavioral description	physical description
transistor	transistor, resistor	voltage	continuous function	differential equation	transistor layout
gate	and, or, xor, flip-flop	logic 0 or 1	propagation delay	Boolean equation	cell layout
RT	adder, mux, register	integer, system state	clock tick	extended FSM	RT level floor plan
processor	processor, memory	abstract data type	event sequence	algorithm in C	IP level floor plan

Basado en HDL

General:



Implementación:



Hola Mundo

Hoy vamos a ver:

- Nuestros primeros diseños. Usando VHDL en nivel de gates (compuertas lógicas básicas).
- Nuestros primeros testbenches y simulaciones en nivel de comportamiento (no timing accurate).
- Todo el flujo desde el diseño, codificación, simulación, síntesis y configuración de la FPGA.

Igualdad de 1 bit

input		**output**
i0	*i1*	*eq*
0	0	1
0	1	0
1	0	0
1	1	1

- A nivel de gates, o sea, expresado utilizando sólo compuertas lógicas básicas (*and*, *not*, *or*, *xor*).
- Usamos suma de productos: *eq* = *i0.i1* + *i0'.i1'*

Primer código

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity eq1bit is
5
6     port (
7        i0,i1: in std_logic;
8        eq1: out std_logic
9     );
10 end eq1bit;
11
12 architecture Behavioral of eq1bit is
13    signal p0,p1:std_logic;
14
15 begin
16    -- suma de dos productos
17    eq1 <= p0 or p1;
18    -- terminos producto
19    p0 <= (not i0) and (not i1);
20    p1 <= i0 and i1;
21 end Behavioral;
```

- VHDL es case insensitive.
- – indica comentarios.

Library y package

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity eq1bit is
5
6     port (
7        i0,i1: in std_logic;
8        eq1: out std_logic
9     );
10 end eq1bit;
11
12 architecture Behavioral of eq1bit is
13    signal p0,p1:std_logic;
14
15 begin
16    -- suma de dos productos
17    eq1 <= p0 or p1;
18    -- terminos producto
19    p0 <= (not i0) and (not i1);
20    p1 <= i0 and i1;
21 end Behavioral;
```

- Nos permiten agregar tipos, operadores, funciones, etc.
- VHDL es *fuertemente tipado*.
- Tiene muchos tipos, pero los *sintetizables* no son tantos!

Entity

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity eq1bit is
5
6     port (
7        i0,i1: in std_logic;
8        eq1: out std_logic
9     );
10 end eq1bit;
11
12 architecture Behavioral of eq1bit is
13    signal p0,p1:std_logic;
14
15 begin
16    -- suma de dos productos
17    eq1 <= p0 or p1;
18    -- terminos producto
19    p0 <= (not i0) and (not i1);
20    p1 <= i0 and i1;
21 end Behavioral;
```

- Declaración de las señales I/O del circuito: in, out, inout.
- `std_logic`. Sintetizables '0', '1', 'Z', Simulación 'U', 'X'.
- Operaciones lógicas.

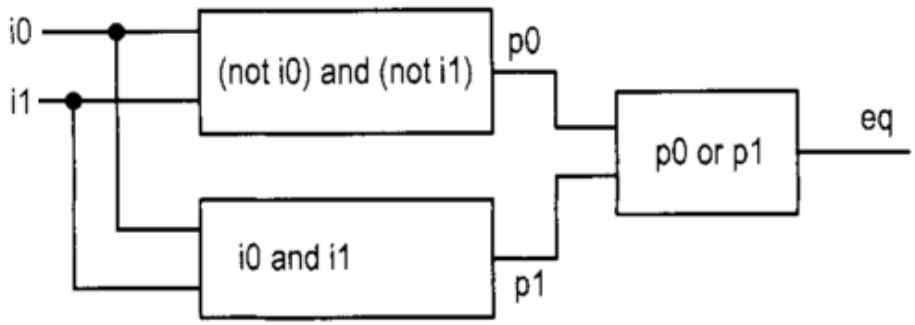
Arquitectura

```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3
 4  entity eq1bit is
 5
 6     port (
 7        i0,i1: in std_logic;
 8        eq1: out std_logic
 9     );
10  end eq1bit;
11
12  architecture Behavioral of eq1bit is
13     signal p0,p1:std_logic;
14
15  begin
16     -- suma de dos productos
17     eq1 <= p0 or p1;
18     -- terminos producto
19     p0 <= (not i0) and (not i1);
20     p1 <= i0 and i1;
21  end Behavioral;
```

- Define la operación del circuito. Puedo tener más de una arquitectura por entidad.
- Sección de declaración (signal, constant, etc)
- Descripción principal: tres **sentencias concurrentes**

Representación gráfica

```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3
 4  entity eq1bit is
 5
 6     port (
 7        i0,i1: in std_logic;
 8        eq1: out std_logic
 9     );
10  end eq1bit;
11
12  architecture Behavioral of eq1bit is
13     signal p0,p1:std_logic;
14
15  begin
16     -- suma de dos productos
17     eq1 <= p0 or p1;
18     -- terminos producto
19     p0 <= (not i0) and (not i1);
20     p1 <= i0 and i1;
21  end Behavioral;
```



Igualdad de 2 bits

input				**output**
a0	*a1*	*b0*	*b1*	*eqab*
0	**0**	**0**	**0**	**1**
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	**1**	**0**	**1**	**1**
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	**0**	**1**	**0**	**1**
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	**1**	**1**	**1**	**1**

Igual que antes, usamos suma de productos:
$eqab = a0'.a1'.b0'.b1' + a0.a1'.b0.b1' + a0'.a1.b0'.b1 + a0.a1.b0.b1'$

Codigo VHDL - suma de productos

```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3
 4  entity eq2bit is
 5     port(
 6        a,b: in std_logic_vector(1 downto 0);
 7        aeqb: out std_logic
 8     );
 9  end eq2bit;
10
11  architecture Behavioral of eq2bit is
12     signal p0,p1,p2,p3 : std_logic;
13  begin
14     -- suma de productos
15     aeqb <= p0 or p1 or p2 or p3;
16     -- productos
17     p0 <= (not a(0)) and (not a(1)) and
18           (not b(0)) and (not b(1));
19     p1 <= (not a(0)) and a(1) and
20           (not b(0)) and b(1);
21     p2 <=  a(0) and (not a(1)) and
22           b(0) and (not b(1));
23     p3 <=  a(0) and a(1) and b(0) and b(1);
24  end Behavioral;
```

Lo único nuevo es el tipo de datos `std_logic_vector`. Como su nombre lo indica, un vector de `std_logic`.

Descripción estructural

- Una misma entidad puede tener muchas arquitecturas diferentes.
- Cada arquitectura pueden sintetizar a una implementación diferente, o puede ser que dos arquitecturas sean diferentes maneras de expresar la misma implementación.
- Arquitectura estructural:
  - Descripción basada en dividir la entidad en diferentes componentes.
  - *Instanciamos* módulos ya existentes, asignando sus puertos a señales del diseño.

Descripción estructural - VHDL

```
26  architecture struct_arch of eq2bit is
27     signal e0, e1 : std_logic;
28  begin
29     -- instanciamos dos comparadores de 1 bit
30     eq_bit0_unit: entity work.eq1bit(Behavioral)
31        port map(i0 => a(0), i1 => b(0), eq1 => e0);
32
33     eq_bit1_unit: entity work.eq1bit(Behavioral)
34        port map(i0 => a(1), i1 => b(1), eq1 => e1);
35
36     -- a y b son iguales si sus bits individuales
37     -- son iguales
38     aeqb <= e0 and e1;
39
40  end struct_arch;
```

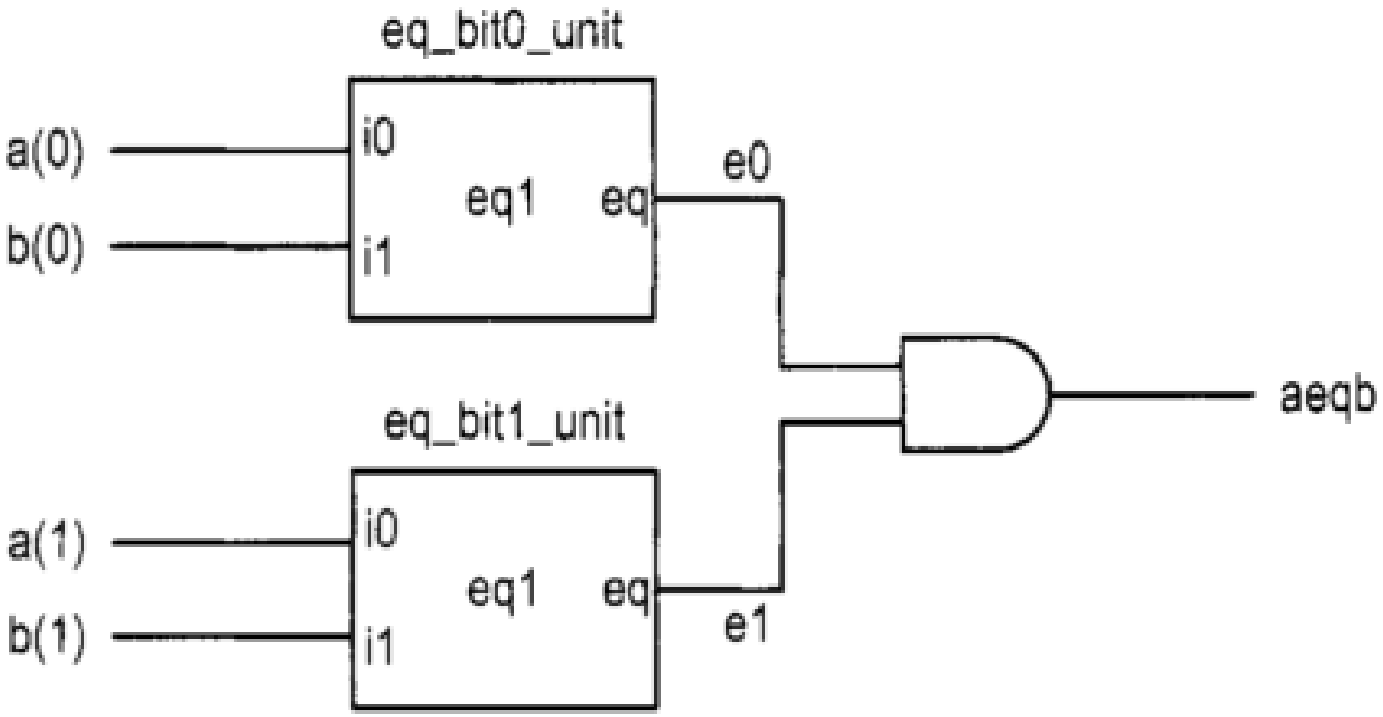
- El módulo `eq1bit` tiene que estar definido en el proyecto.
- `work`: librería donde están definidos todos los módulos del proyecto.
- Entre paréntesis va la arquitectura que queremos usar (opcional si el componente tiene una sóla arquitectura)

Descripción estructural VHDL 87

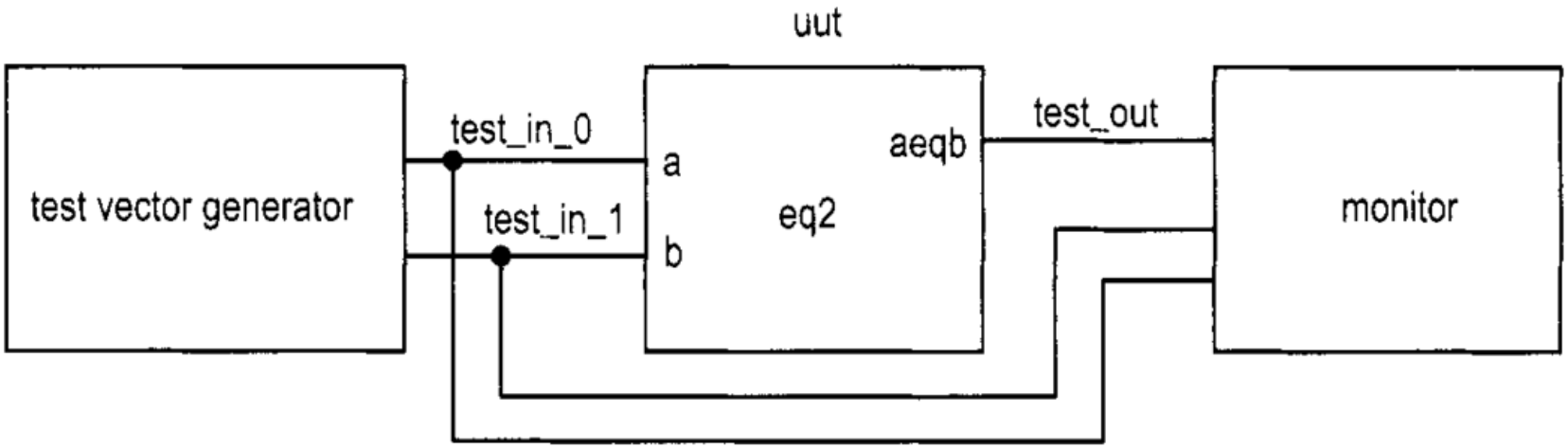
```
architecture struct_arch_vhdl87 of eq2bit is

   -- declaramos los componentes que vamos a
   -- usar
   component eq1bit is
      port(
         i0,i1: in std_logic;
         eq1:out std_logic
      );
   end component;

   signal e0, e1 : std_logic;
begin
   -- instanciamos dos comparadores de 1 bit
   eq_bit0_unit: eq1bit
      port map(i0 => a(0), i1 => b(0), eq1 => e0);

   eq_bit1_unit: eq1bit
      port map(i0 => a(1), i1 => b(1), eq1 => e1);

   -- a y b son iguales si sus bits individuales
   -- son iguales
   aeqb <= e0 and e1;

end struct_arch_vhdl87;
```

Declarar *explícitamente* los componentes a utilizar en la parte de declaraciones de la arquitectura.

Descripción gráfica



Testbench


uut
test vector generator
test_in_0
test_in_1
a
b
eq2
aeqb
test_out
monitor

Testbench en VHDL

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY test_eq1bit IS
5  END test_eq1bit;
6
7  ARCHITECTURE behavior OF test_eq1bit IS
8      -- Component Declaration for the
9      -- Unit Under Test (UUT)
10     COMPONENT eq1bit
11     PORT(
12          i0 : IN  std_logic;
13          i1 : IN  std_logic;
14          eq1 : OUT  std_logic
15         );
16     END COMPONENT;
17
18    --Inputs
19    signal i0 : std_logic := '0';
20    signal i1 : std_logic := '0';
21
22    --Outputs
23    signal eq1 : std_logic;
24
25 BEGIN
```

- Simulación de Comportamiento (NO es timing accurate).
- Inclusión de librerias y definición de entidad igual que antes.
- Arquitectura: instanciación del componente, y definición de señales para estimular el diseño y chequear los resultados.

Testbench en VHDL - cont

```
27    -- Instantiate the Unit Under Test (UUT)
28    uut: eq1bit PORT MAP (
29           i0 => i0,
30           i1 => i1,
31           eq1 => eq1
32         );
33
34    -- Stimulus process
35    stim_proc: process
36    begin
37
38      -- PRIMER CASO : vector de test (0,0)
39      -- 1) genero vector de test y estimulo el diseño
40      i0 <= '0';
41      i1 <= '0';
42      -- 2) espero el tiempo de propagación
43      wait for 1 ns;
44      -- 3) El monitor chequea si la salida es correcta
45      assert eq1 <= '1' report "Igualdad falló en caso i0 =0, i1=0"
46             severity failure;
                              ...
65      -- FINAL: si llegué hasta acá, está todo bien.
66      -- Hago assert para terminar simulacion
67      assert false report "Simulación terminó OK" severity failure;
68    end process;
69  END;
```

Manos a la obra

¡ESO!

Hoy vimos

- Introducción a FPGA: qué son, para qué se usan, arquitecturas, cómo se programan.
- Introducción a VHDL.
  - VHDL para síntesis: nuestro primer programa a nivel de Gates.
  - VHDL para simulación: nuestro primer testbench.
- Herramientas:
  - *ISE*: Crear proyecto, fuente, ucf (user constraint file) y todo el proceso de síntesis hasta generar el archivo de programa (bitstream).
  - *ISim*: Usando el ISE, crear un testbench y simularlo con el ISE Simulator (ISim).
  - *Digilent Adept*: Configurar la FPGA.

Bibliografía

- **Diseño: Pong. P. Chu "FPGA Prototyping by VHDL examples"** a.k.a "El libro del chino"
- Diseño: Wayne Wolf "FPGA-Based Design"
- Diseño: Cofer y Harding "Rapid System Prototyping with FPGAs"
- Desarrollo histórico, overview de lo que anda dando vueltas (¡muy bueno!): Clive Maxfield "The Design Warrior's Guide to FPGAs"
- **Manual de Referencia VHDL: Ashenden "The Designer's Guide to VHDL"**
- Survey: EETimes y Embedded.com: 2012 Embedded Market Survey
- **Manuales de Referencia de la placa de desarrollo (Digilent) y de la FPGA (Xilinx).**