Introducción a Verilog y al entorno de xilinx ISE

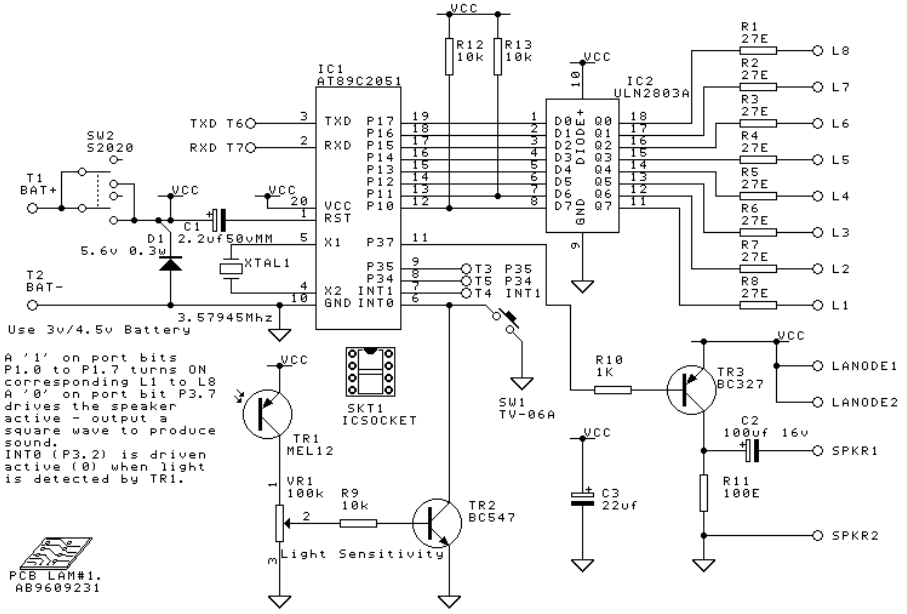
Diseño de Sistemas con FPGA

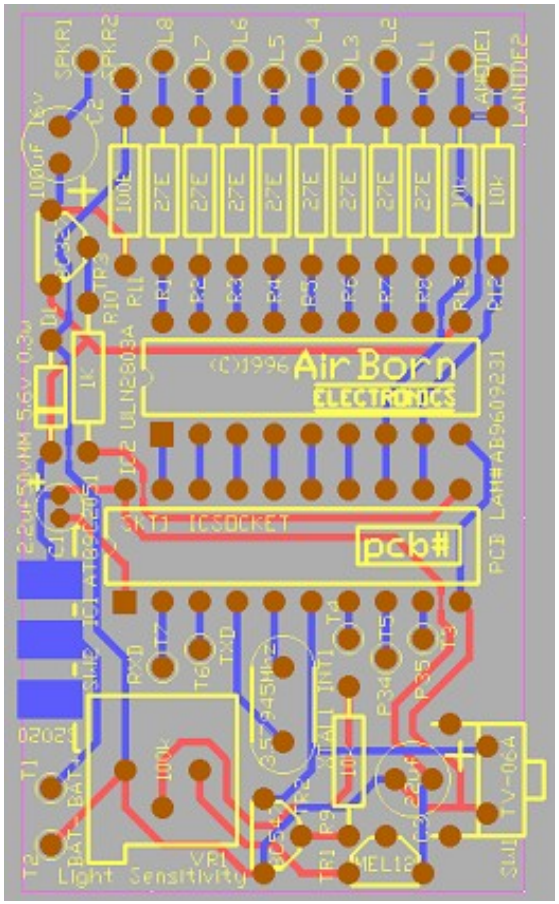
Patricia Borensztejn.

Vistas de un Sistema

- ¿Como describimos un sistema que puede tener millones de transistores? Por medio de vistas (view) y abstracciones
- Vistas: son maneras o puntos de vista diferentes de describir un sistema complejo
  - Estructural
    - Es un diagrama de bloques describiendo los componentes y sus interconexiones. Es el diagrama del sistema o “netlist”
  - De comportamiento
    - Describe la funcionalidad, trata al sistema como una caja negra e ignora su representación interna. Pone el foco en la relación entre las entradas y las salidas
  - Físico
    - Agrega mas información a la descripción estructural: por ejemplo el tamaño de los componentes, su ubicación…

Vistas Estructural y Física





Niveles de Abstracción

- Independientemente de la manera que “vemos” el sistema, éste puede ser descrito con diversos (cuatro) niveles de abstracción (o detalle):
  - Transistor
  - Gate
  - RTL (register transfer)
  - Procesador (memoria,
- Estos niveles nos permiten ir encarando el diseño de un sistema complejo, de forma abstracta, hasta llegar a los niveles mas bajos del diseño
- Lo que caracteriza a cada nivel de abstracción es:
  - Bloques básicos
  - Representación de las señales
  - Representación del tiempo
  - Representacion del comportamiento
  - Representación física

Características de cada nivel 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

Verilog

- Verilog es un HDL, desarrollado por Phil Moorby en 1985 mientras trabajaba en Automated Integrated Design Systems, más tarde renombrada Gateway Design Automation. El objetivo de Verilog era ser un lenguaje de modelado de hardware. Gateway Design Automation fue comprada por Cadence Design Systems en 1990. Cadence ahora tiene todos los derechos sobre los simuladores lógicos de Verilog y Verilog-XL hechos por Gateway.
- Con el incremento en el éxito de VHDL, Cadence decidió hacer el lenguaje abierto y disponible para estandarización. Cadence transfirió Verilog al dominio público a través de Open Verilog International, actualmente conocida como Accellera. Verilog fue después enviado a la IEEE que lo convirtió en el estándar IEEE 1364-1995, habitualmente referido como Verilog 95.
- Extensiones a Verilog 95 fueron enviadas a la IEEE para cubrir las deficiencias que los usuarios habían encontrado en el estándar original de Verilog. Estas extensiones se volvieron el estándar IEEE 1364-2001 conocido como Verilog 2001.
- El advenimiento de los lenguajes de verificación de alto nivel como OpenVera y el lenguaje E de Verisity, impulsaron el desarrollo de Superlog, por Co-Design Automation Inc. Co-Design fue más tarde comprada por Synopsis. Las bases de Superlog y Vera han sido donadas a Accellera. Todo ello ha sido transformado y actualizado en forma de SystemVerilog, que probablemente se convierta en el próximo estándar de la IEEE.
- Las últimas versiones del lenguaje incluyen soporte para modelado analógico y de señal mixta. Todos estos están descritos en Verilog-AMS

Verilog

- *Verilog* es un *HDL* , hay otros*… como VHDL (Very High Speed Integrated Circuit) , Abel, SystemC…*
- El lenguaje surge originalmente para SIMULAR circuitos, pero luego se comenzó a utilizar también para SINTETIZAR circuitos.
- De hecho, aplicado a las herramientas de desarrollo de FPGA, usamos Verilog para las dos cosas:
  - Síntesis: archivo .v conteniendo el circuito
  - Simulación: archivo .v que simula el comportamiento del circuito anterior.
- Del conjunto del lenguaje HDL, solo un subconjunto muy pequeño “sintetiza”.
- Advertencia: la sintaxis de Verilog es muy parecida a la de C, pero no así su semántica.

Verilog

- Sólo permite la descripción de los sistemas usando vistas de comportamiento o estructurales. No físicas.
- Soporta como niveles de abstracción el nivel de puerta (GATE) y RTL.

Síntesis

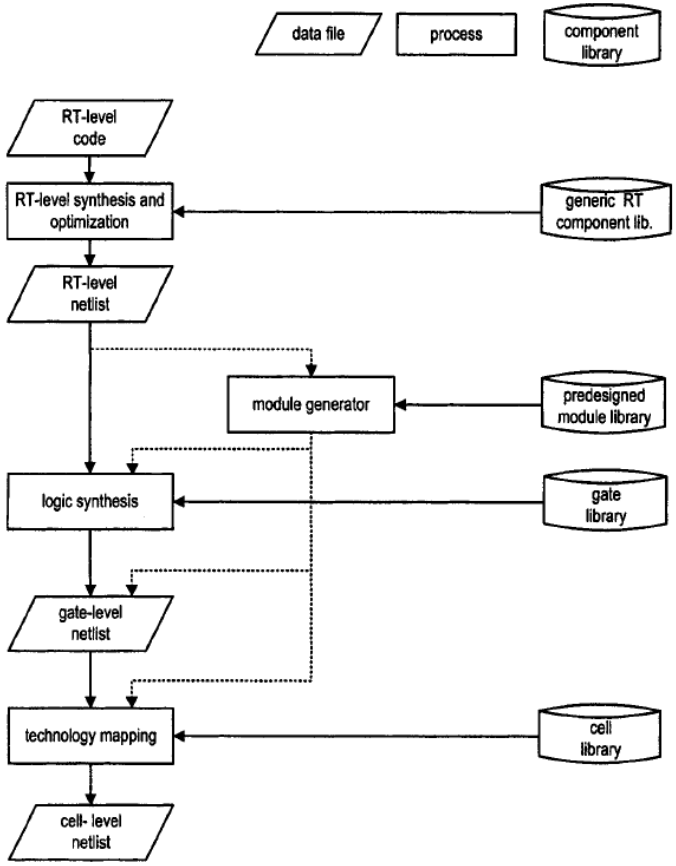


Figure 6.6 Synthesis flow.

- De HDL a una descripción física para la tecnología en uso.
- Netlist: descripción de la conectividad de los componentes. Los componentes pueden ser:
  - Comparadores, sumadores, bloques .. Si estamos en RT
  - Puertas si estamos en Gate Level
  - Celdas básicas de la FPGA

Verilog para simulación

```
module hello_world ;

initial
begin
        $display ("Hello World");

        #10 $stop;
end

endmodule // End of Module hello_world
```

Verilog contiene un conjunto de funciones de sistema predefinidas que se invocan mediante *$nombre_de_ función*

#10 : Sentencia secuencial que significa esperar 10 unidades de tiempo

Comparador

```
module eq1
    // I/O ports
    (
     input wire i0, i1,
     output wire eq
    );

    // signal declaration
    wire p0, p1;

    // body
    // sum of two product terms
    assign eq = p0 | p1;
    // product terms
    assign p0 = ~i0 & ~i1;
    assign p1 = i0 & i1;

endmodule
```

i1	i0	eq
0	0	1
0	1	0
3	0	0
1	1	1

0 1 Z : sintetizables

X : simulación

3 sentencias concurrentes. “continuous statement”

Una descripción estructural a nivel de puertas

Tipos de Datos en Verilog

- Dos tipos de datos:
  - Net: representan las conexiones físicas entre componentes hardware
    - wire:
    - Y otros tipos que no usaremos (wand, supply0, …)
  - Variable: representan almacenamiento abstracto en los módulos “de comportamiento”
    - reg
    - integer
    - Real, time, realtime: solamente en simulación

Como describir un circuito en Verilog

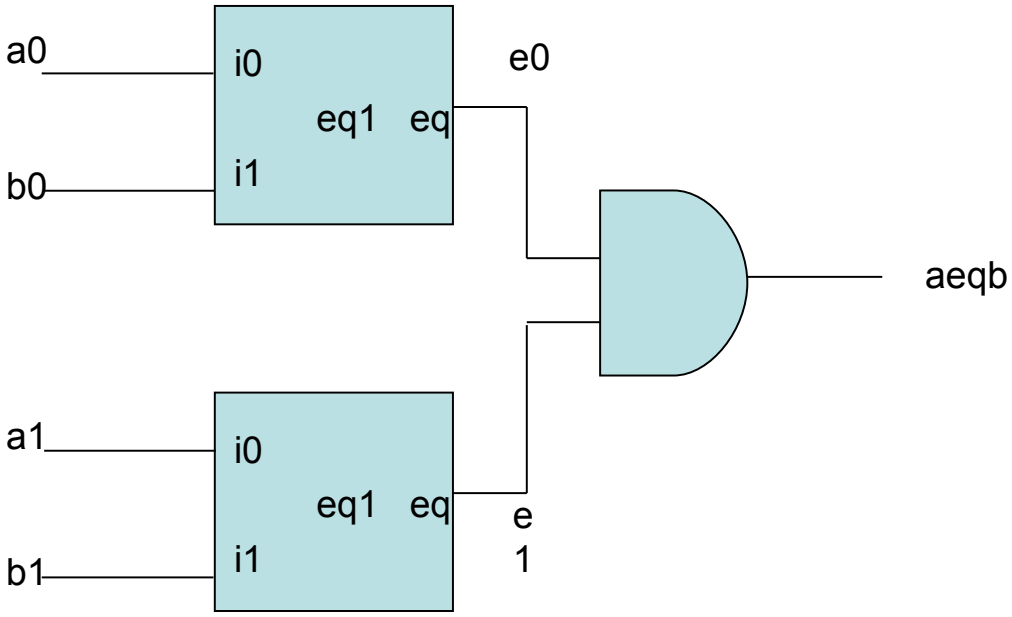
- Los programas o módulos escritos en HDL deben verse como una organización de hardware (colección de circuitos) y no como un “algoritmo secuencial”
- Tres maneras de describir circuitos:
  1. Por medio de “asignaciones continuas”
  2. Instanciación de módulos (vista estructural)
  - Construcción “always block” (vista de comportamiento) (*procesos* en VHDL)

Comparador de 2 bits en Verilog (1)“asignación continua”

```
module eq2_sop
  (
  input wire[1:0] a, b,
  output wire aeqb
  );

  // internal signal declaration
  wire p0, p1, p2, p3;

  // sum of product terms
  assign aeqb = p0 | p1 | p2 | p3;
  // product terms
  assign p0 = (~a[1] & ~b[1]) & (~a[0] & ~b[0]);
  assign p1 = (~a[1] & ~b[1]) & (a[0] & b[0]);
  assign p2 = (a[1] & b[1]) & (~a[0] & ~b[0]);
  assign p3 = (a[1] & b[1]) & (a[0] & b[0]);

endmodule
```

b1	b0	a1	a0	eq
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

Comparador de 2 bits
(2) Descripción estructural.



Comparador de 2 bits en Verilog (2) Instanciación

```
module eq2
    (
     input  wire[1:0] a, b,
     output wire aeqb
    );

    // internal signal declaration
    wire e0, e1;

    // body
    // instantiate two 1-bit comparators
    eq1 eq_bit0_unit (.i0(a[0]), .i1(b[0]), .eq(e0));
    eq1 eq_bit1_unit (.eq(e1), .i0(a[1]), .i1(b[1]));

    // a and b are equal if individual bits are equal
    assign aeqb = e0 & e1;

endmodule
```

Comparador de 2 bits en Verilog. (3) Inferencia del sintetizador

Descripción funcional (de comportamiento) utilizando la construcción always block.

```
module eq3(
    input wire [1:0] a,b,
    output reg aeqb
    );

always @ *
            if (a==b)
                        aeqb = 1'b1;
            else
                        aeqb = 1'b0;
endmodule
```

```
Synthesizing Unit <eq3>.
    Related source file is "eq3.v".
    Found 2-bit comparator equal for signal <aeqb$cmp_eq0000> created at line 28.
    Summary:
            inferred   1 Comparator(s).
Unit <eq3> synthesized.
```

TestBench

- Para poder simular el funcionamiento del código, y antes de sintetizarlo a nivel físico, crearemos un programa, también en Verilog llamado testbench.
- Un bloque generará los estímulos o patterns de entrada para la UUT (Unit Under Test) que es nuestro código.
- Otro bloque examinará las respuestas de nuestra UUT.

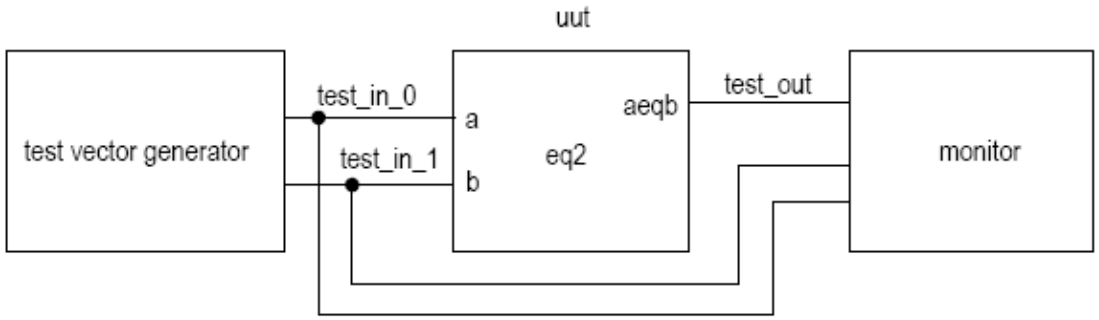


Figure 1.3 Testbench for a 2-bit comparator.

Testbench

```
// The 'timescale directive specifies that
// the simulation time unit is 1 ns  and
// the simulation timestep is 10 ps
`timescale 1 ns/10 ps

module eq2_testbench;
   // signal declaration
   reg  [1:0] test_in0, test_in1;
   wire  test_out;

   // instantiate the circuit under test
   eq2 uut
      (.a(test_in0), .b(test_in1), .aeqb(test_out));

   //  test vector generator
   initial
   begin
      // test vector 1
      test_in0 = 2'b00;
      test_in1 = 2'b00;
      # 200;
      // test vector 2
      test_in0 = 2'b01;
      test_in1 = 2'b00;
      # 200;
      // test vector 3
      test_in0 = 2'b01;
      test_in1 = 2'b11;
      # 200;
      // test vector 4
      test_in0 = 2'b10;
      test_in1 = 2'b10;
      # 200;
      // test vector 5
      test_in0 = 2'b10;
      test_in1 = 2'b00;
      # 200;
      // test vector 6
      test_in0 = 2'b11;
      test_in1 = 2'b11;
      # 200;
      // test vector 7
      test_in0 = 2'b11;
      test_in1 = 2'b01;
      # 200;
      // stop simulation
      $stop;
   end

endmodule
```

Testbenchs

- Algunas construcciones que usaremos en simulación:
  - Initial Block:
    - Es una construcción especial que solo se ejecuta una vez, cuando arranca la simulación. Está compuesto por sentencias que se ejecutan secuencialmente.
  - Asignación procedural
    - Variable = expresión

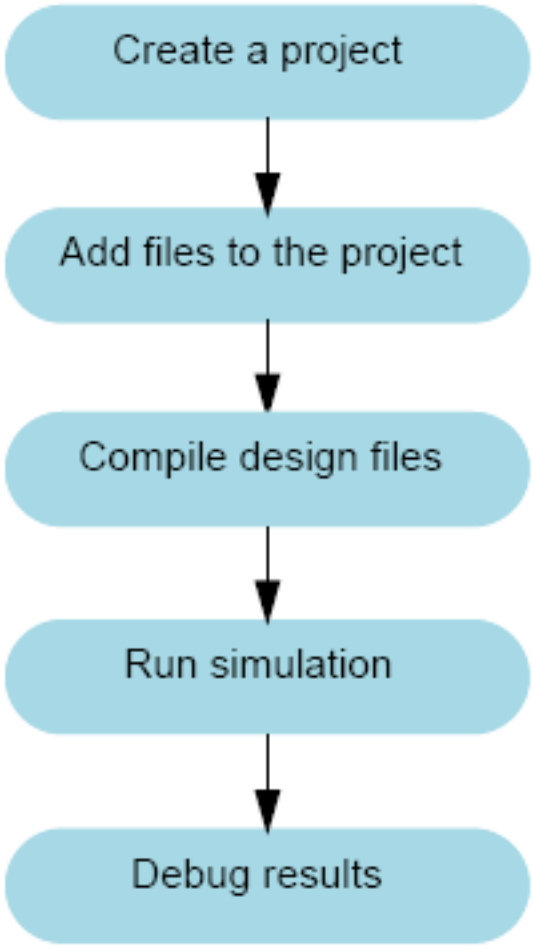
    En este caso, el tipo de la variable de la izquierda debe ser de tipo reg.
  - Directivas:
    - Timescale
  - Llamadas al sistema
    - $stop

Testbench

- Los testbench no se sintetizan. Usualmente son código secuencial.
- El testbench es una entidad sin ports
- En síntesis, el tiempo no tiene sentido
- En los testbenchs, el tiempo es la principal magnitud.
- Toda la potencia de Verilog puede utilizarse en simulación.

ModelSim Simulator

Figure 2-2. Project Flow

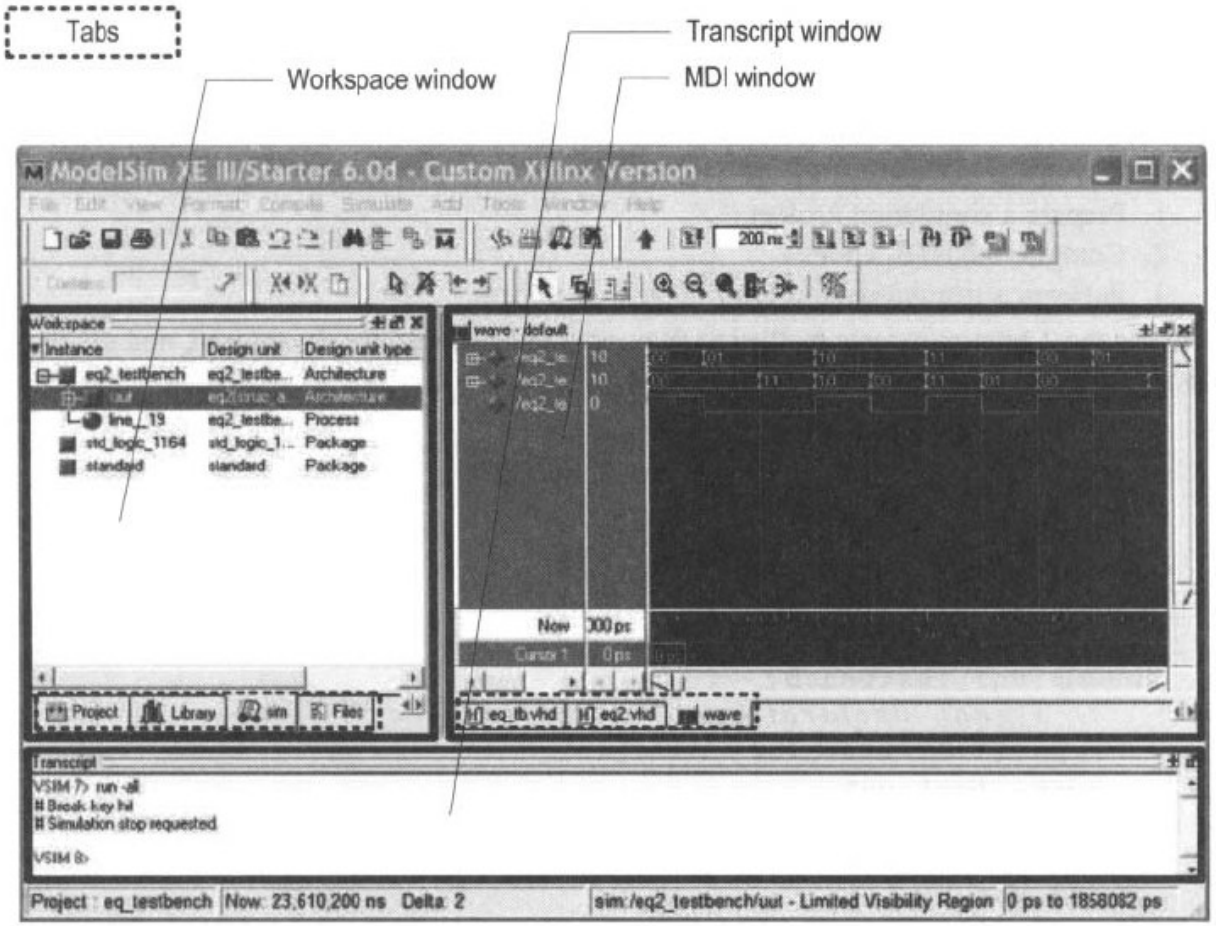


- Puede correr independientemente del ISE. Es de Mentor Graphics Corporation, empresa especializada en testing.
- Es una herramienta para verificación y simulación de sistemas escritos en Verilog, VHDL y sistemas mixtos.

ModelSim

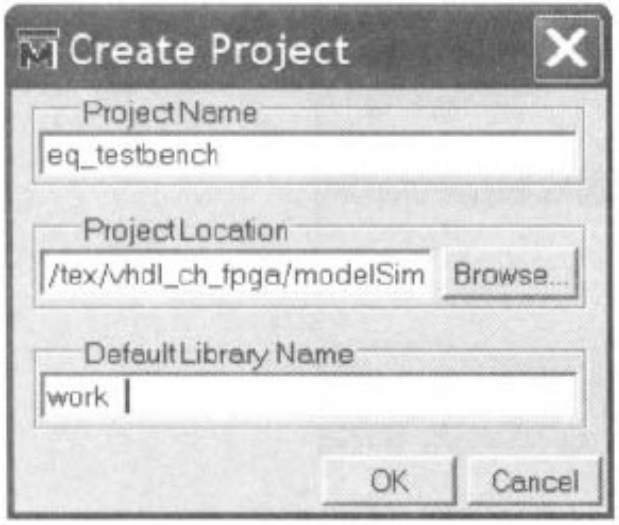
1. Abrir ModelSim
2. Preparar un proyecto
3. Compilar los códigos
4. Simular y Examinar los resultados

1. Abrimos ModelSim

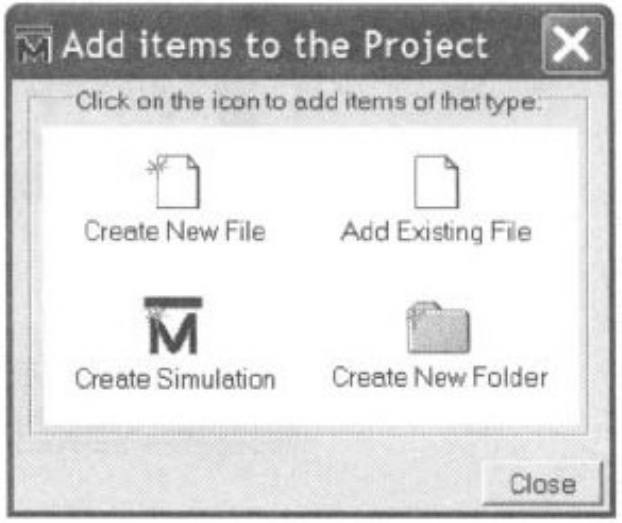


1. Preparar un proyecto

- Un proyecto está formado por :
  - Colección de archivos HDL: uno de ellos ha de ser el testbench, los otros formarán la UUT
  - Librería (“work”) donde se “compilará” nuestro diseño
- Los archivos HDL los tenemos en un directorio. (ModelSim también permite editarlos)
- Para crear el proyecto, en el menu File, seleccionamos New y luego Proyect.
- A continuación, mediante la ventana Add Items, seleccionamos los archivos HDL (eq1.v, eq2.v y test_eq.v)



(a) Create Project dialog



(b) Add items dialog

1. Compilar un proyecto

- “Compilar” significa convertir el código HDL al formato que entiende el simulador.
- Es necesario compilar cada uno de los módulos por separado, comenzando desde el nivel mas bajo de la jerarquía. (En nuestro caso, eq1.v)
- Para compilar, se selecciona el módulo y con el mouse derecho se selecciona Compile, Compile Selected.

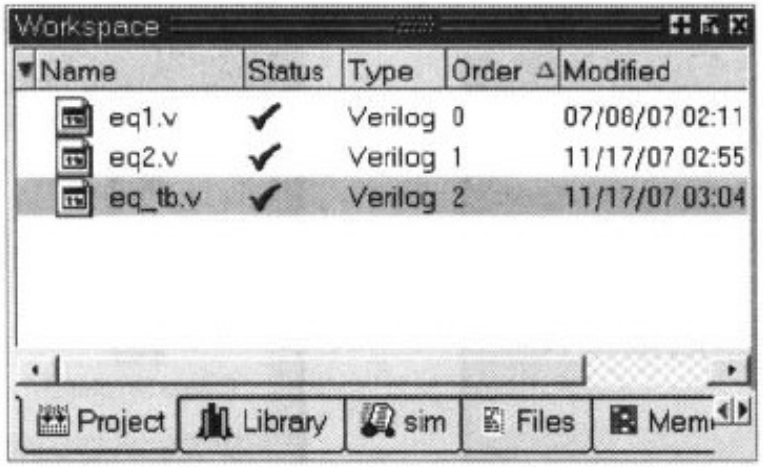
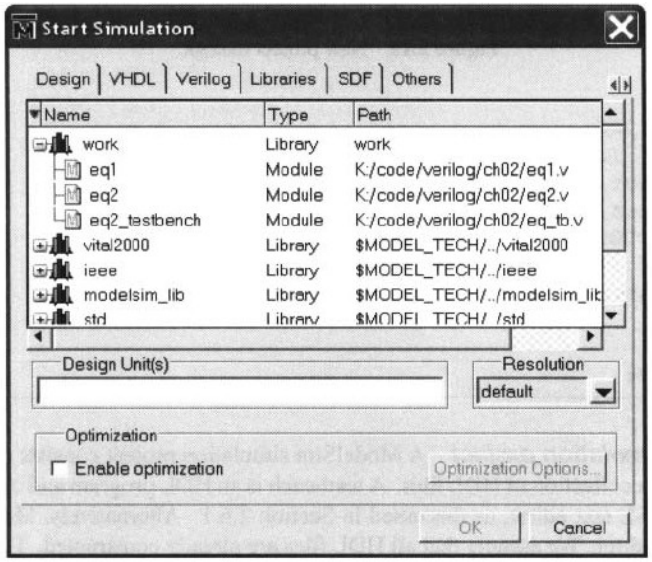
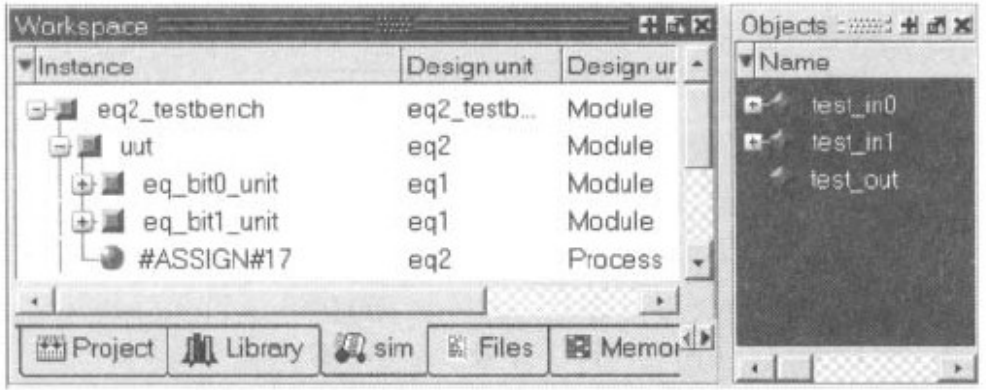


Figure 2.13 Project tab of the workplace panel.

1. Simular

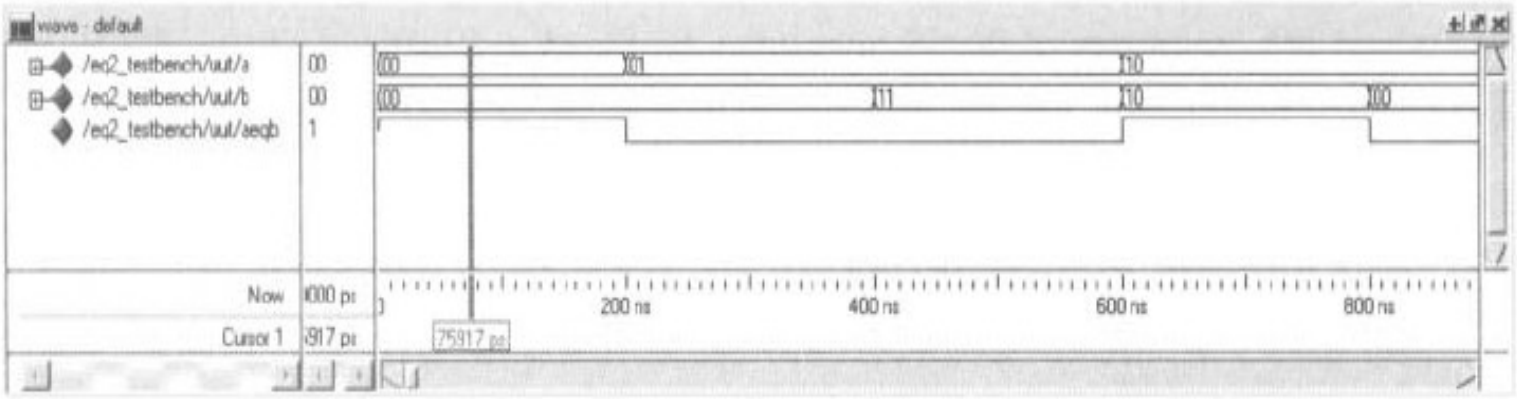
- Seleccionar Simulate, Simulate y aparece la ventana. En la solapa Design expandir la librería “work” y cargar el testbench.
- En la ventana workspace seleccionar la UUT y con el mouse derecho seleccionar ADD, ADD to Wave de esta forma todas las señales del UUT apareceran en la página del waveform





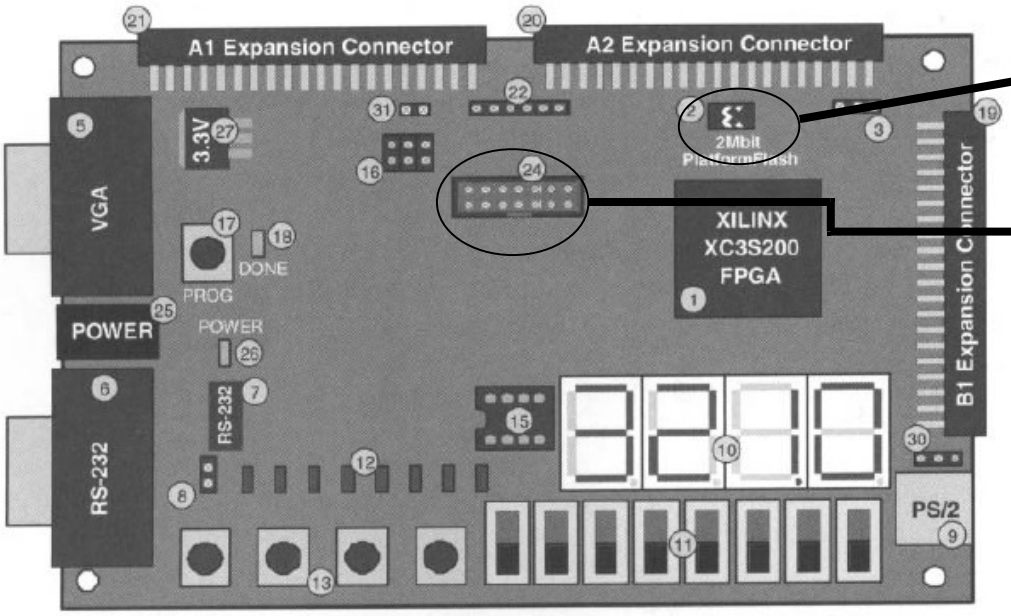
1. Simular

- Seleccionar Simulte, Run… o cualquiera de los otros comandos.
- Para volver a comenzar: restart



Desarrollo de un proyecto basado en FPGA

- Utilizamos una placa de prototipado que contiene un dispositivo FPGA.
- En nuestro caso, utilizamos una placa de la empresa Digilent que contiene una FPGA de Xilinx modelo Spartan 3.

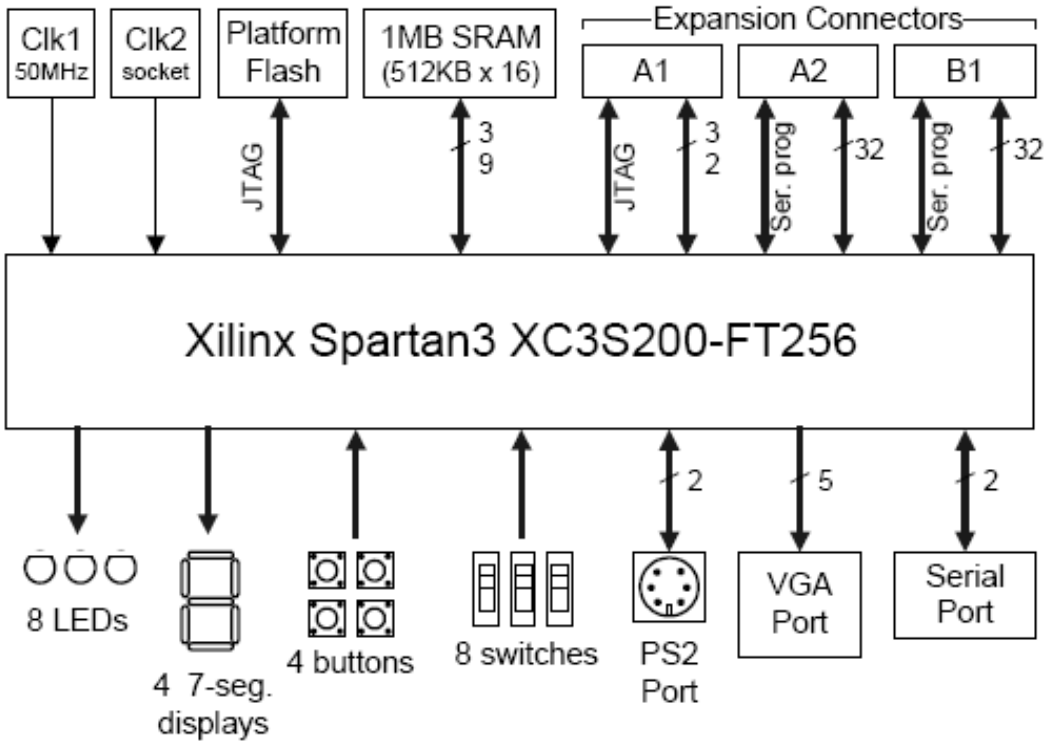


(a) Top view

PROM de configuración

JTAG port : se usa para bajar el programa a la FPGA.

Digilent S3: Principales Componentes



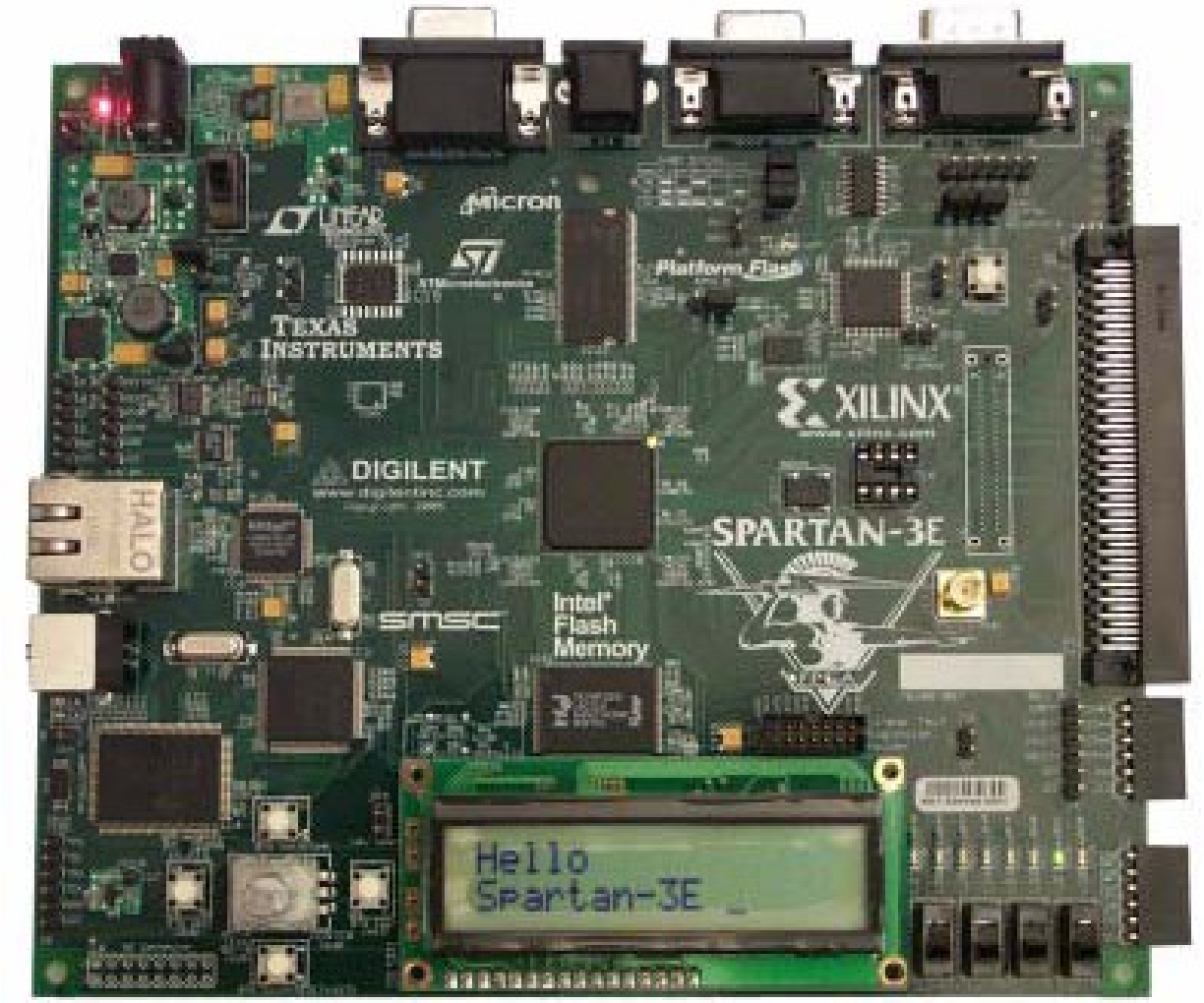
S3 Starter Board Block Diagram

FPGA Familia Spartan3

Table 2.1 Devices in the Spartan-3 family

Device	Number of LCs	Number of block RAMs	Block RAM bits	Number of multipliers	Number of DCMs
XC3S50	1,728	4	72K	4	2
XC3S200	4,320	12	216K	12	4
XC3S400	8,064	16	288K	16	4
XC3S1000	17,280	24	432K	24	4
XC3S1500	29,952	32	576K	32	4
XC3S2000	46,080	40	720K	40	4
XC3S4000	62,208	96	1,728K	96	4
XC3S5000	74,880	104	1,872K	104	4

Digilent Spartan3e Starter Kit



FPGA Familia Spartan3E

Table 1: **Summary of Spartan-3E FPGA Attributes**

Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB = Four Slices)				Distributed RAM bits(1)	Block RAM bits(1)	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs	Total Slices						
XC3S100E	100K	2,160	22	16	240	960	15K	72K	4	2	108	40
XC3S250E	250K	5,508	34	26	612	2,448	38K	216K	12	4	172	68
XC3S500E	500K	10,476	46	34	1,164	4,656	73K	360K	20	4	232	92
XC3S1200E	1200K	19,512	60	46	2,168	8,672	136K	504K	28	8	304	124
XC3S1600E	1600K	33,192	76	58	3,688	14,752	231K	648K	36	8	376	156

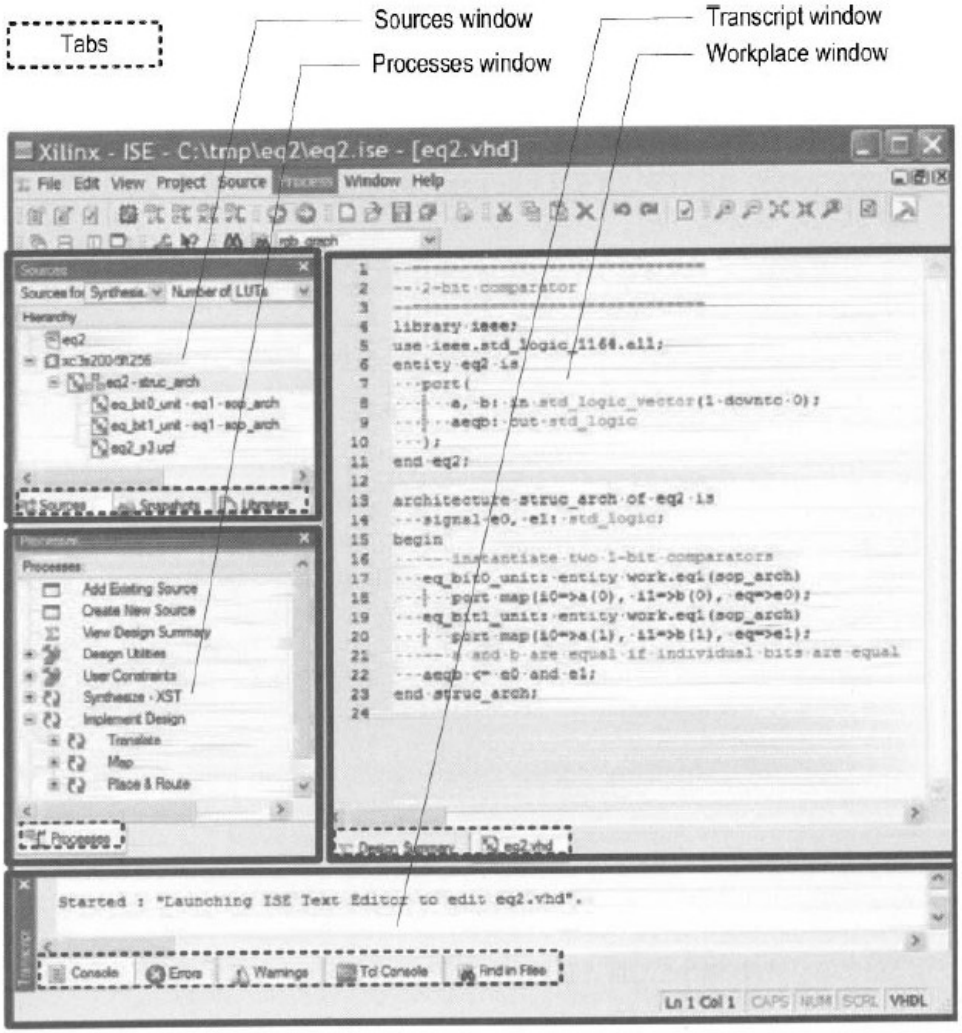
Notes:

1. By convention, one Kb is equivalent to 1,024 bits.

Desarrollo de un proyecto

- Utilizamos el entorno de desarrollo de Xilinx, Xilinx ISE que controla todos los aspectos del proceso de desarrollo.
  - Diseño y codificación en Verilog (o en otro HDL)
  - Síntesis => traducción de HDL a netlist compuesta por gate level components
  - Simulación funcional
  - Implementación
    - Translate: mezcla distintos componentes en un único netlist
    - Mapeo: Mapea los componentes lógicos a los componentes físicos (LC) de la FPGA. (Mapeo tecnológico)
    - Place and Route: deriva el layout físico
  - Generación del archivo de Programación
  - Bajada a la placa.

Xilinx ISE


Tabs
Sources window
Processes window
Transcript window
Workplace window
Xilinx - ISE - C:\tmp\eq2\eq2.ise - [eq2.vhd]
File Edit View Project Source Process Window Help
Sources
Hierarchy
eq2
xc3s200-5ft256
eq2 - struc_arch
eq_bit0_unit - eq1 - sop_arch
eq_bit1_unit - eq1 - sop_arch
eq2_s3.ucf
Processes
Add Existing Source
Create New Source
View Design Summary
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Translate
Map
Place & Route
Started : "Launching ISE Text Editor to edit eq2.vhd".
Console
Errors
Warnings
Tcl Console
Find in Files
Ln 1 Col 1
CAPS
NUM
SCRL
VHDL

Pequeño tutorial ISE

- Crear un proyecto y los códigos HDL
- Crear un testbench y simular el diseño
- Agregar un archivo de constraints y sintetizar e implementar el diseño
- Generar el archivo de configuración y bajarlo a la placa.
- Usaremos como ejemplo el comparador de dos bits.

Archivo de constraints

- Son condiciones para los procesos de síntesis e implementación.
- En particular, nosotros vamos a usar las restricciones que permiten asignar las salidas de nuestra entidad a los pines de la FPGA.
- Podemos usar 4 switches para conectar las entradas, y un Led para la salida.

```
# 4 slide switches
NET "a<0>"   LOC = "F12"   ;   # switch 0
NET "a<1>"   LOC = "G12"   ;   # switch 1
NET "b<0>"   LOC = "H14"   ;   # switch 2
NET "b<1>"   LOC = "H13"   ;   # switch 3
# led
NET "aeqb"   LOC = "K12"   ;   # led 0
```