TÉCNICO LISBOA

Projecto e Controlo em Lógica Digital

www.lip.pt/~pedjor/PCLD2014

- Introduction
  - FPGAs
  - Laboratory
  - Verilog
  - combinatorial logic
  - Sequential Logic
- State Machines
- Advanced verilog (video, etc.)

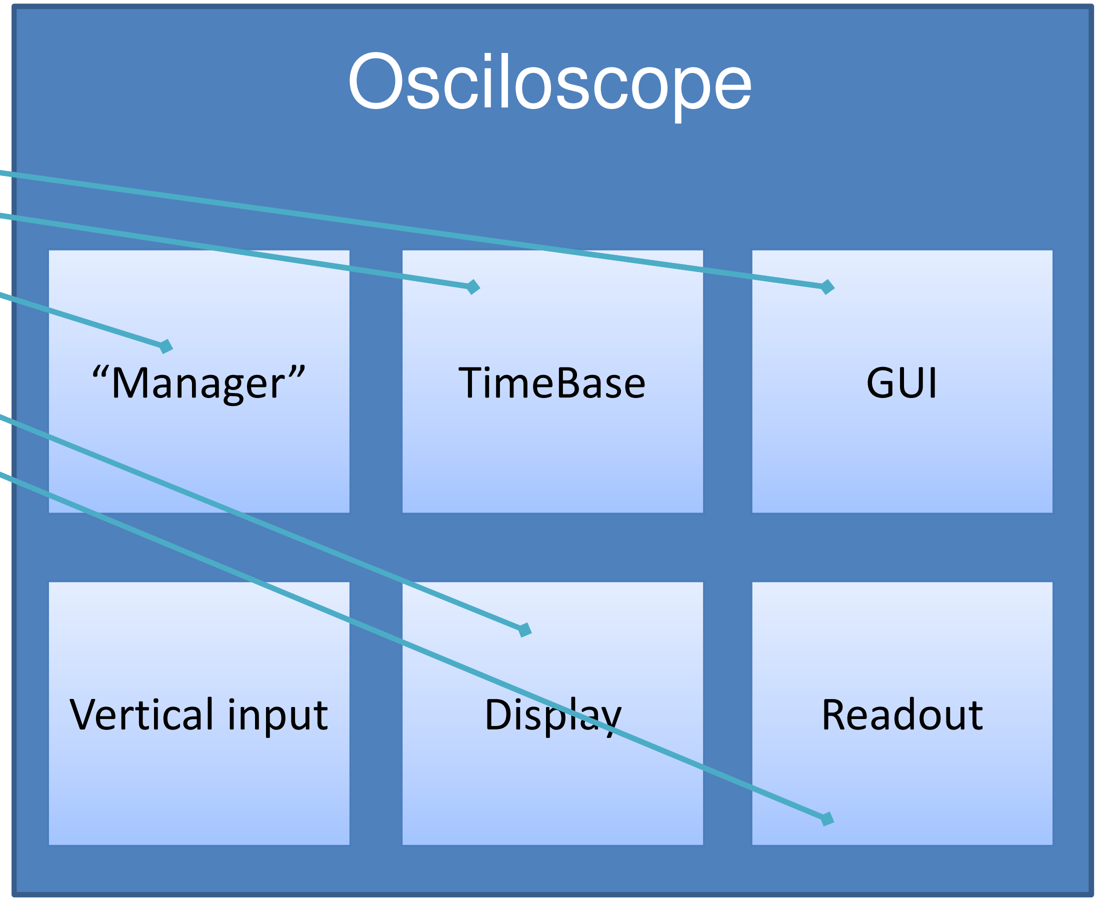
Refs:
Cyclone II device Handbook, Altera corp.
Quartus II Handbook , Altera corp.
DE2 documentation
Verilog HDL, S. Palnitkar, Prentice Hall

Programming Digital Logic...

Inspired on course 6.111 from MIT * Some support material taken from 6.111 (thank you)

Estrutura

Assignments:
Ex1: Ola Mundo
Ex2: StopWatch
Ex3: State Machine
Ex4: Test Image
Ex5: Microprocessor
W: Oscilocsope
Pr: To be negociated



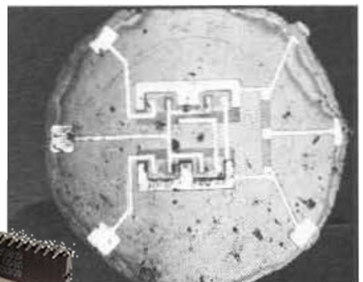
Evaluation

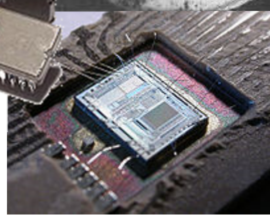
Exercises evaulated in trinary (0,1,2); worth 25%
Work: Evaluation 0-20; worth 25%
Project: Evaluation 0-20; worth 50%

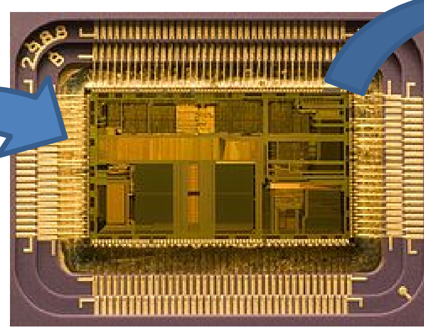
Nota= (Ex1+Ex2+Ex3+Ex4+Ex5)/5*10 *0.25+W * 0.25 + Pr * 0.5

Lógica Digital

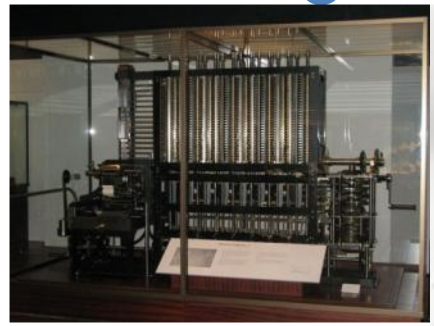
Programação

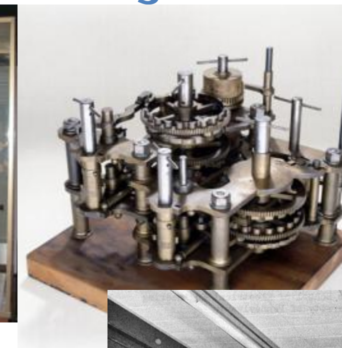


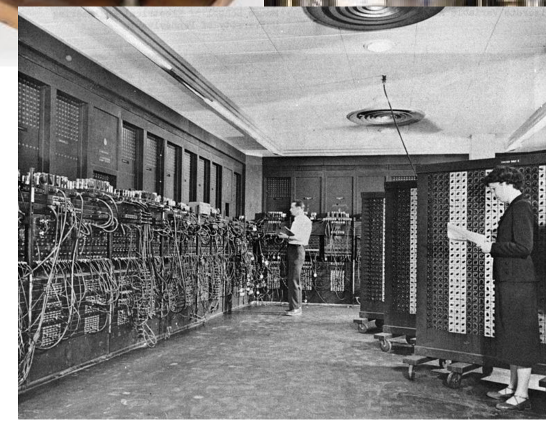




Babbage difference engine







Lógica Digital de Programação

Everything works either by magic or by gnomes...

Programming microprocessors
(assembly, c++, etc.)

Give the "gnome" a list (consecutive of operations to perform

1 Machine executes several consecutive tasks

Programming Digital Logic
(FPGAs+HDL)

Give the gnome a list of objects/machines to build

Several machines execute several tasks in parallel

In high level languages there is some confusion

2+2?
Result x 2?

A=2+2
Cout << A
A=A*2
Cout << A

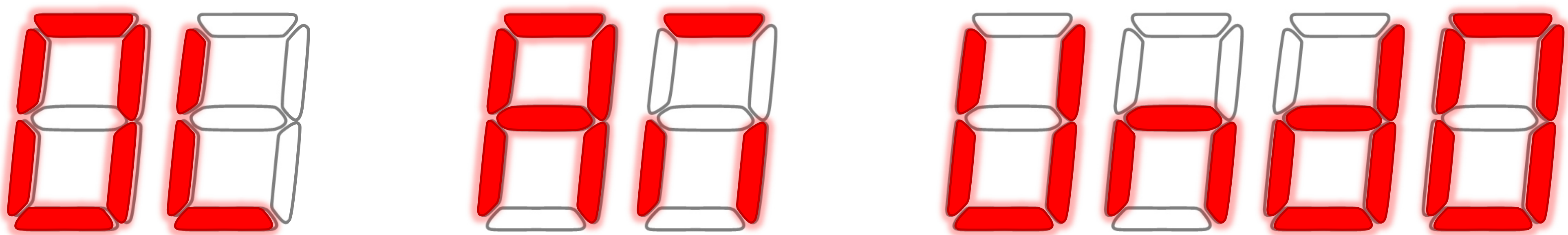
A=4
4!
A=8
8!

time

A=2+2
Output A
A=A*2
Output A

Adder: A=4
"at the same time"
Multiplier: A=A*2
Out ??? (glitches?)

An FPGA working...

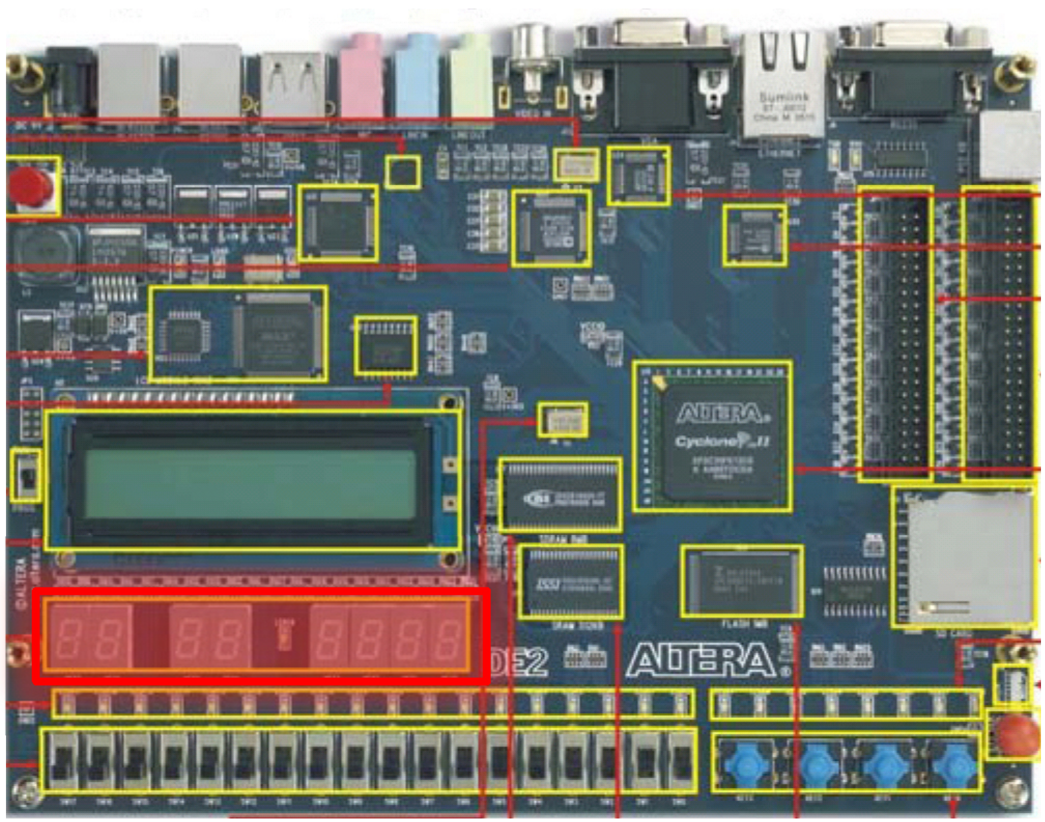


First exercise. . .

Hardware

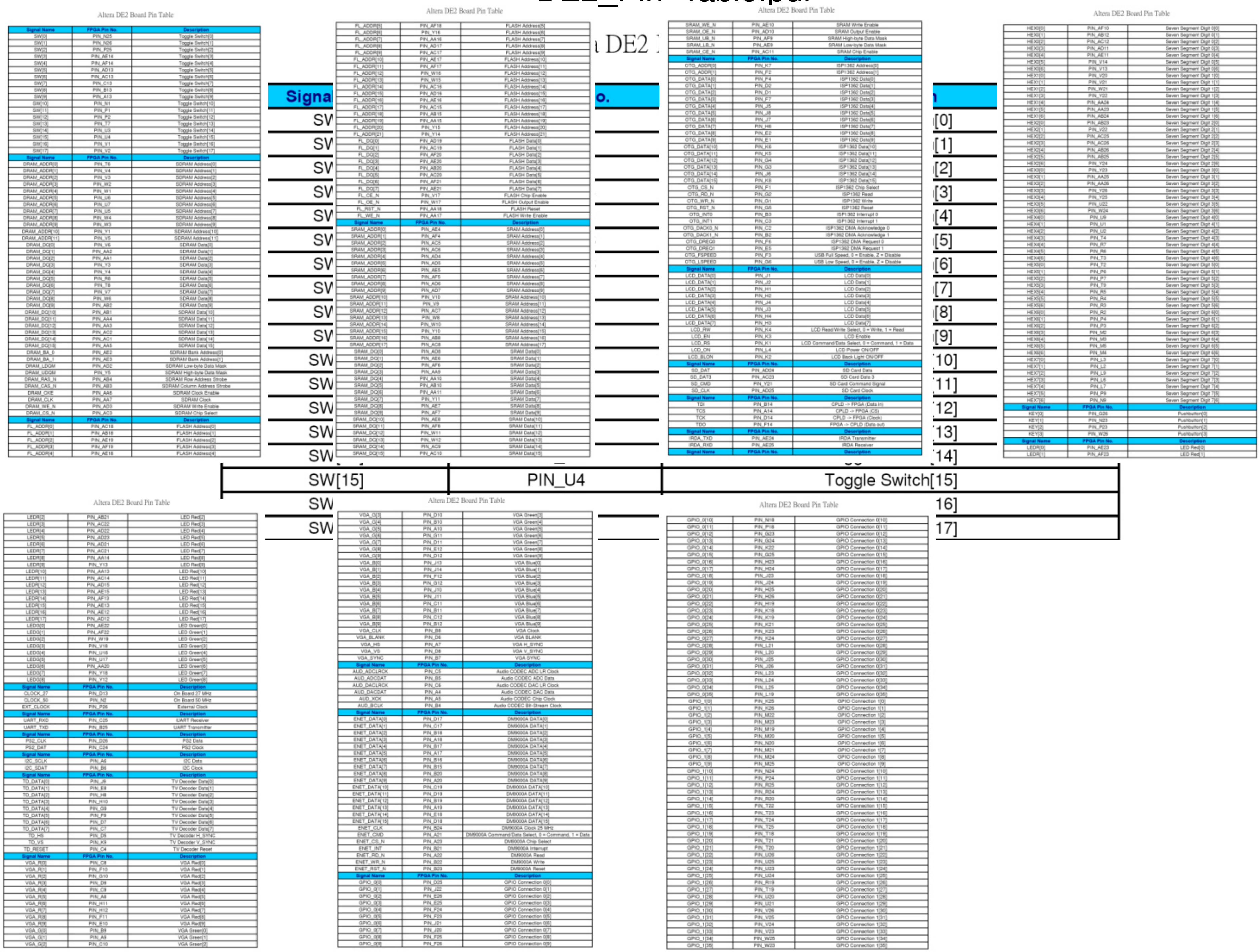
OLAMUndO

Hardware



Hardware

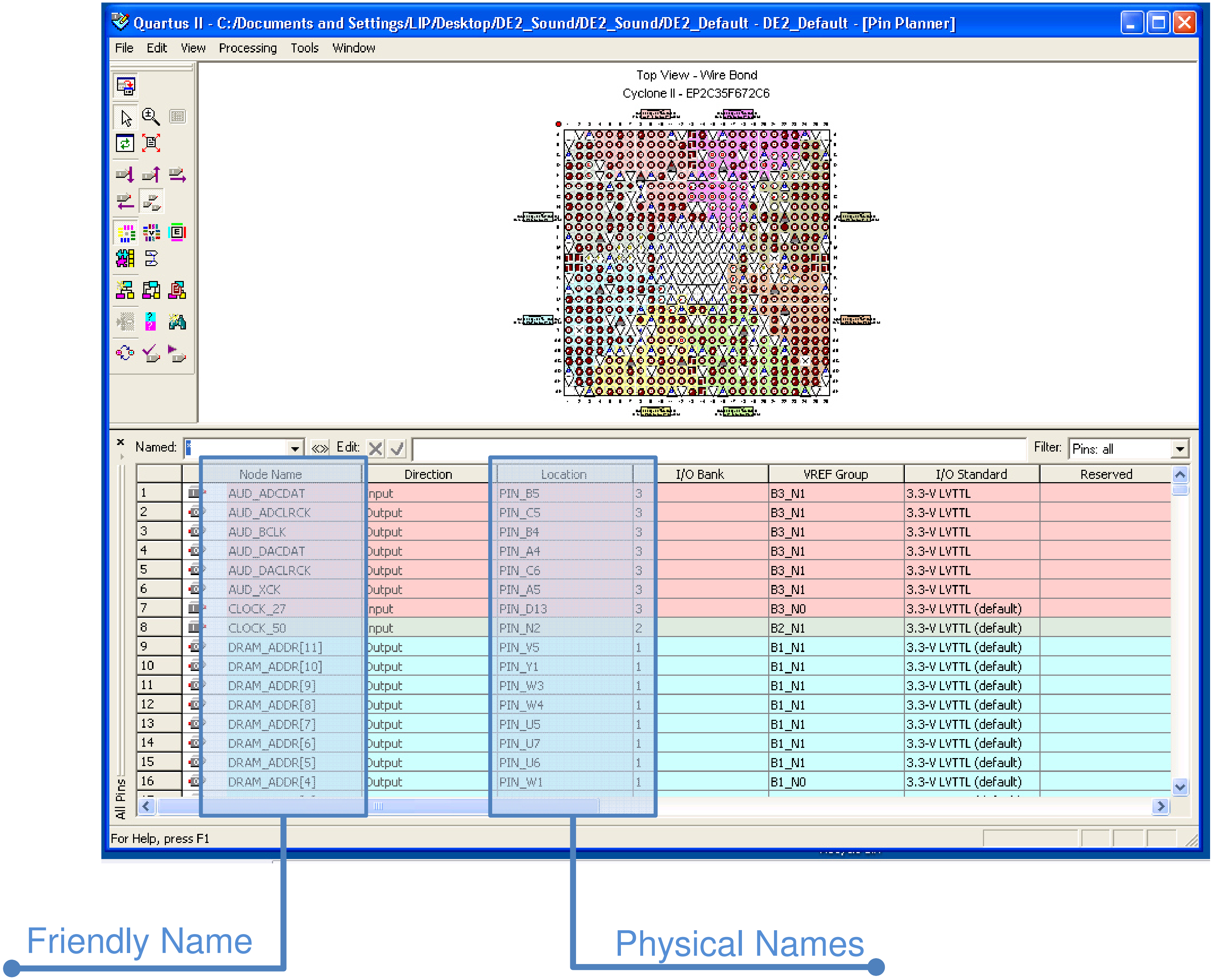
DE2_Pin Table.pdf



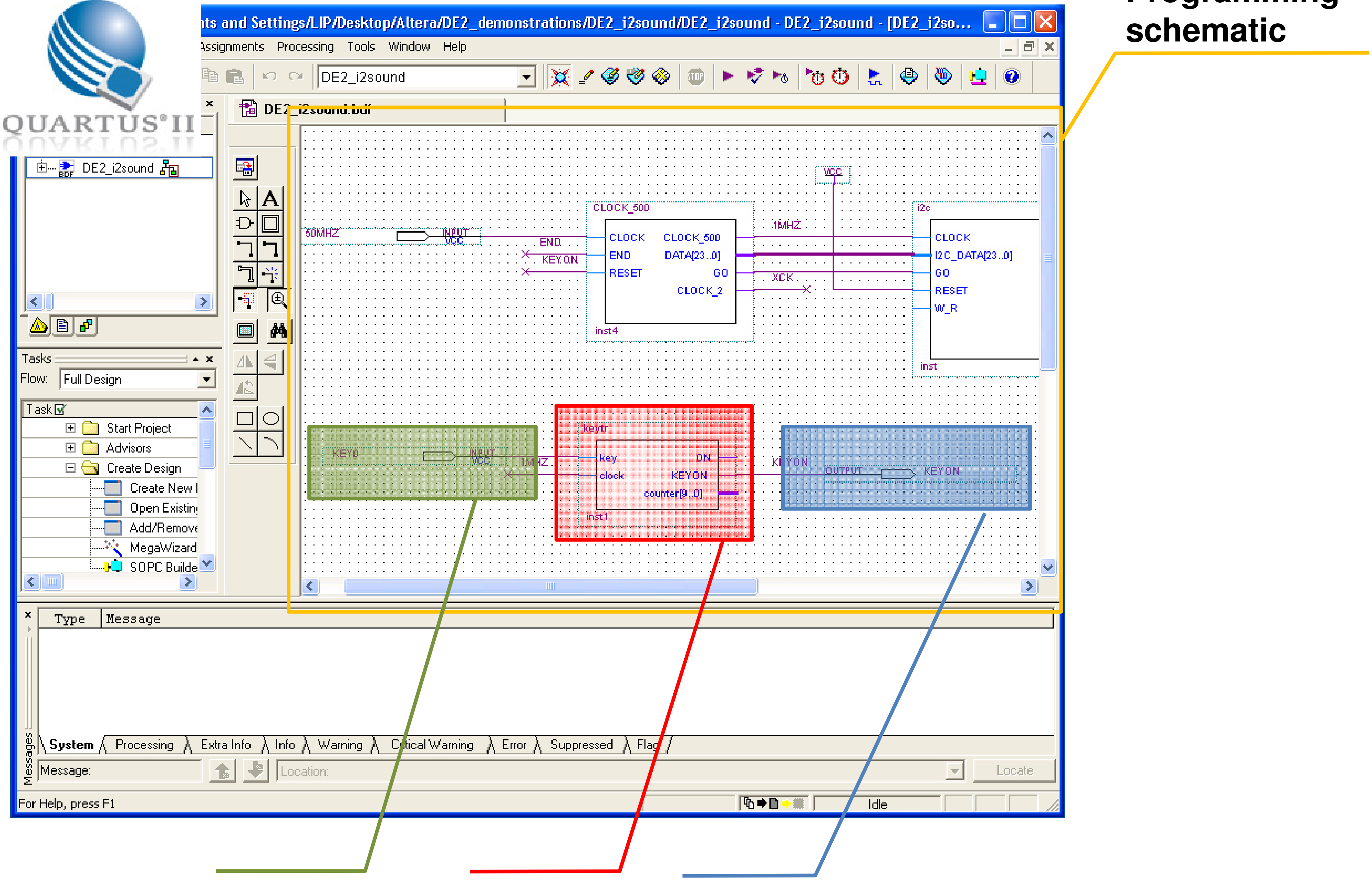
HEX3[5]	PIN_U22	Seven Segment Digit 3[5]
HEX3[6]	PIN_W24	Seven Segment Digit 3[6]
HEX4[0]	PIN_U9	Seven Segment Digit 4[0]
HEX4[1]	PIN_U1	Seven Segment Digit 4[1]
HEX4[2]	PIN_U2	Seven Segment Digit 4[2]
HEX4[3]	PIN_T4	Seven Segment Digit 4[3]
HEX4[4]	PIN_R7	Seven Segment Digit 4[4]
HEX4[5]	PIN_R6	Seven Segment Digit 4[5]
HEX4[6]	PIN_T3	Seven Segment Digit 4[6]
HEX5[0]	PIN_T2	Seven Segment Digit 5[0]
HEX5[1]	PIN_P6	Seven Segment Digit 5[1]
HEX5[2]	PIN_P7	Seven Segment Digit 5[2]
HEX5[3]	PIN_T9	Seven Segment Digit 5[3]
HEX5[4]	PIN_R5	Seven Segment Digit 5[4]
HEX5[5]	PIN_R4	Seven Segment Digit 5[5]
HEX5[6]	PIN_R3	Seven Segment Digit 5[6]
HEX6[0]	PIN_R2	Seven Segment Digit 6[0]
HEX6[1]	PIN_P4	Seven Segment Digit 6[1]
HEX6[2]	PIN_P3	Seven Segment Digit 6[2]
HEX6[3]	PIN_M2	Seven Segment Digit 6[3]
HEX6[4]	PIN_M3	Seven Segment Digit 6[4]
HEX6[5]	PIN_M5	Seven Segment Digit 6[5]
HEX6[6]	PIN_M4	Seven Segment Digit 6[6]
HEX7[0]	PIN_L3	Seven Segment Digit 7[0]
HEX7[1]	PIN_L2	Seven Segment Digit 7[1]
HEX7[2]	PIN_L9	Seven Segment Digit 7[2]
HEX7[3]	PIN_L6	Seven Segment Digit 7[3]
HEX7[4]	PIN_L7	Seven Segment Digit 7[4]
HEX7[5]	PIN_P9	Seven Segment Digit 7[5]
HEX7[6]	PIN_N9	Seven Segment Digit 7[6]
Signal Name	**FPGA Pin No.**	**Description**

Physical Pins of FPGA

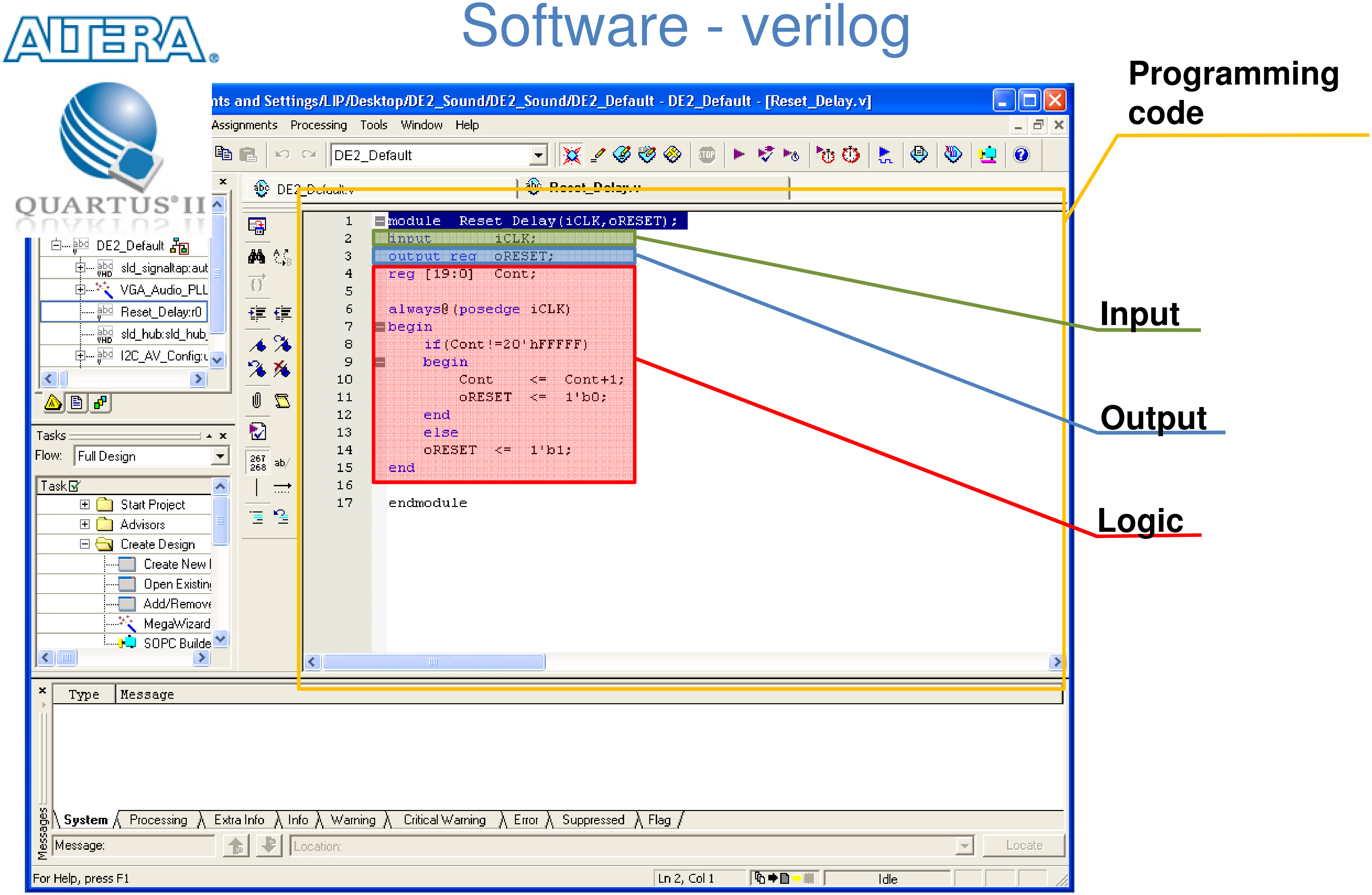
On the “compiler” / “programmer”



Software - schematic

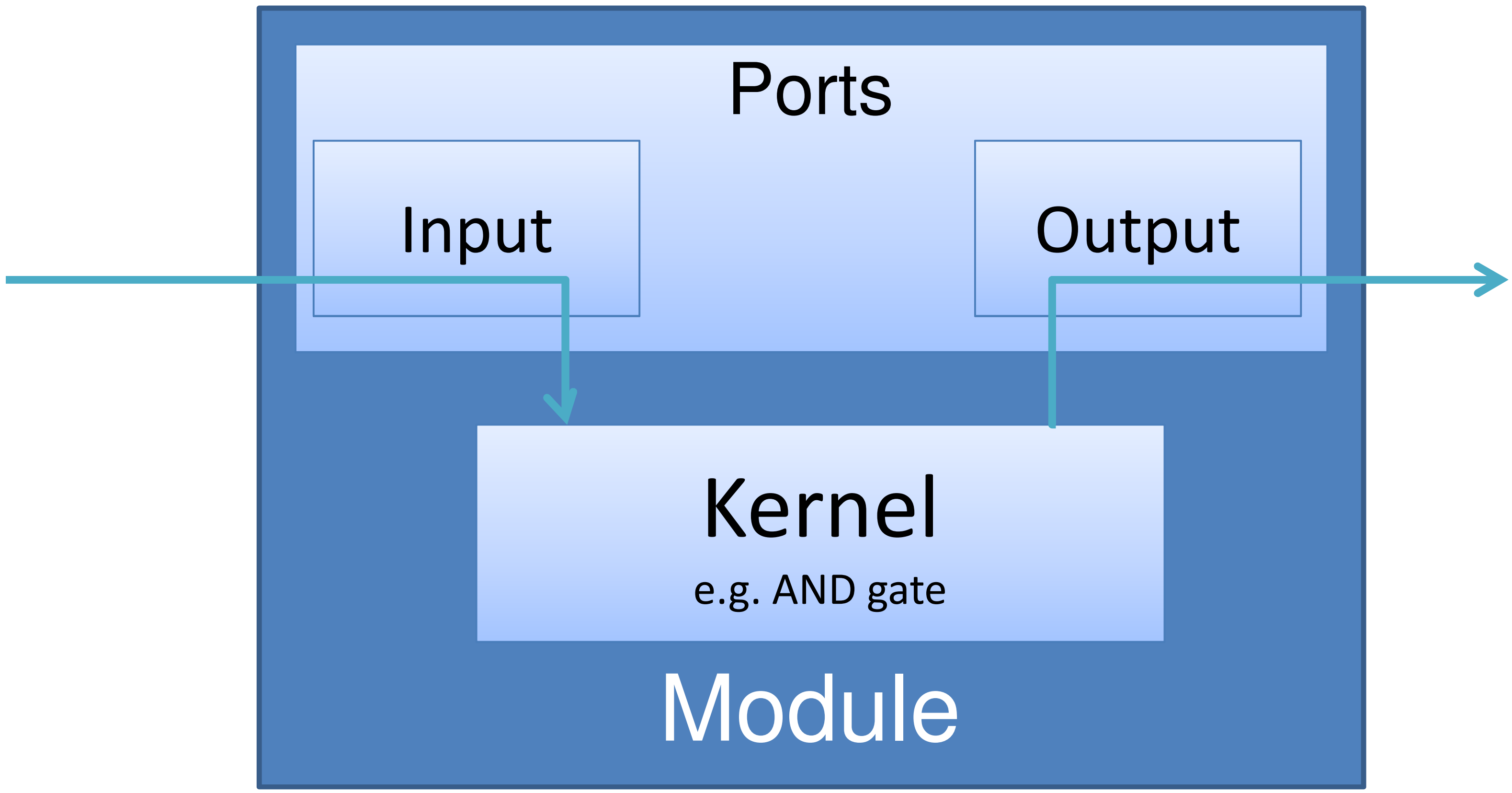


Software - verilog



Structuring... Modules

Modules are the building blocks in Verilog!
There is at least one module: The top one that has the name of the project!



Ports in the top level module link directly to the pins

Now, the text you need to write that the compiler will interpret with the configuration for the FPGA to do what you want.
a.k.a. code

```
module mux(input a,b,sel, output q, qbar);
   wire selbar, q1, q2;
   not n1(selbar, sel);
   and a1(q1, a, selbar);
   and a2(z2, b, sel);
   or o1(q, q1, q2);
   not n2(qbar, q)
endmodule
```

Selbar, q1, q2 are names given to wires

not, and, or are modules!
n1, a1, a2, o1, n2 are names of instances

Another way...

```
module mux(input a,b,sel, output q, qbar);

   assign q = sel? a : b;
   assign qbar = ~ q;

endmodule
```

Same as
If (sel) then a else b

Assign means that
Left Hand Side (LHS)
of ‘=‘ takes ‘immediately’ the value that results from
Right Hand Side (RHS)

Yet another way...

Notice the **reg** keyword...

```
module mux(input a,b,sel, output reg q, qbar);

   always @ (a,b,sel) begin
     if (sel) q=a;
     else q=b;
   end
   assign qbar = ~ q;

endmodule
```

"*Whenever a,b or sel changes*"

Can be replaced by
always @ (*)
"*Whenever anything changes*"

If and cases can be used

*Beware of c-like syntax... You are **NOT** programming a microprocessor*

Verilog data values

Value	Meaning
0	Logic zero, “Low”
1	Logic one, “High”
Z or ?	High Impedance (tri-state)
X	Unknow (simulation)

Numeric constants

Full format: <Width>’<Radix>value
Width: number
Radix: d=decimal, h=hex, o=ocatl, b=binary

Value	Meaning
123	Default: decimal radix
‘d123	‘d=decimal radix
‘h7B	‘h=hexadecimal radix
‘o173	‘o=ocatl radix
‘b111_101	‘b=binary radix
16’b11111	A binary with 16 bits
16’d5	A 16 bit decimal = ‘b0000_0000_0000_0101

How many bits?

Wire ab;	A 1 bit wire called ab
Wire ab,cd;	Two 1 bit wires called ab and cd
wire [31:0] ef;	A 32 bits wire bus called ef;
{ab,cd}	Concatenation of ab and cd;
ef[15]	Bit #15 (the sixteenth) of ef
ef[7:0]	First 8 bits of ef (the ones to the right

What does this means?

```
wire [31:0] kk;
Wire [7:0] a,b,c,d;
Assign kk={d,c,b,a}
```

Boolean operators

- Bitwise operators perform bit-oriented operations on vectors
  - ~(4'b0101) = {~0,~1,~0,~1} = 4'b1010
  - 4'b0101 & 4'b0011 = {0&0, 1&0, 0&1, 1&1} = 4'b0001
- Reduction operators act on each bit of a single input vector
  - &(4'b0101) = 0 & 1 & 0 & 1 = 1'b0
- Logical operators return one-bit (true/false) results
  - !(4'b0101) = 1'b0

Bitwise

~a	NOT
a & b	AND
a | b	OR
a ^ b	XOR
a ~^ b a ^~ b	XNOR

Reduction

&a	AND
~&a	NAND
|a	OR
~|a	NOR
^a	XOR
~^a ^~a	XNOR

Logical

!a	NOT
a && b	AND
a || b	OR
a == b a != b	[in]equality returns x when x or z in bits. Else returns 0 or 1
a === b a !== b	case [in]equality returns 0 or 1 based on bit by bit comparison

Note distinction between ~a and !a when operating on multi-bit values

Other operators

Conditional

a ? b : c	If a then b else c

Relational

a > b	greater than
a >= b	greater than or equal
a < b	Less than
a <= b	Less than or equal

Arithmetic

-a	negate
a + b	add
a - b	subtract
a * b	multiply
a / b	divide
a % b	modulus
a ** b	exponentiate
a << b	logical left shift
a >> b	logical right shift
a <<< b	arithmetic left shift
a >>> b	arithmetic right shift

And now our lab session...

Two simple exercises:

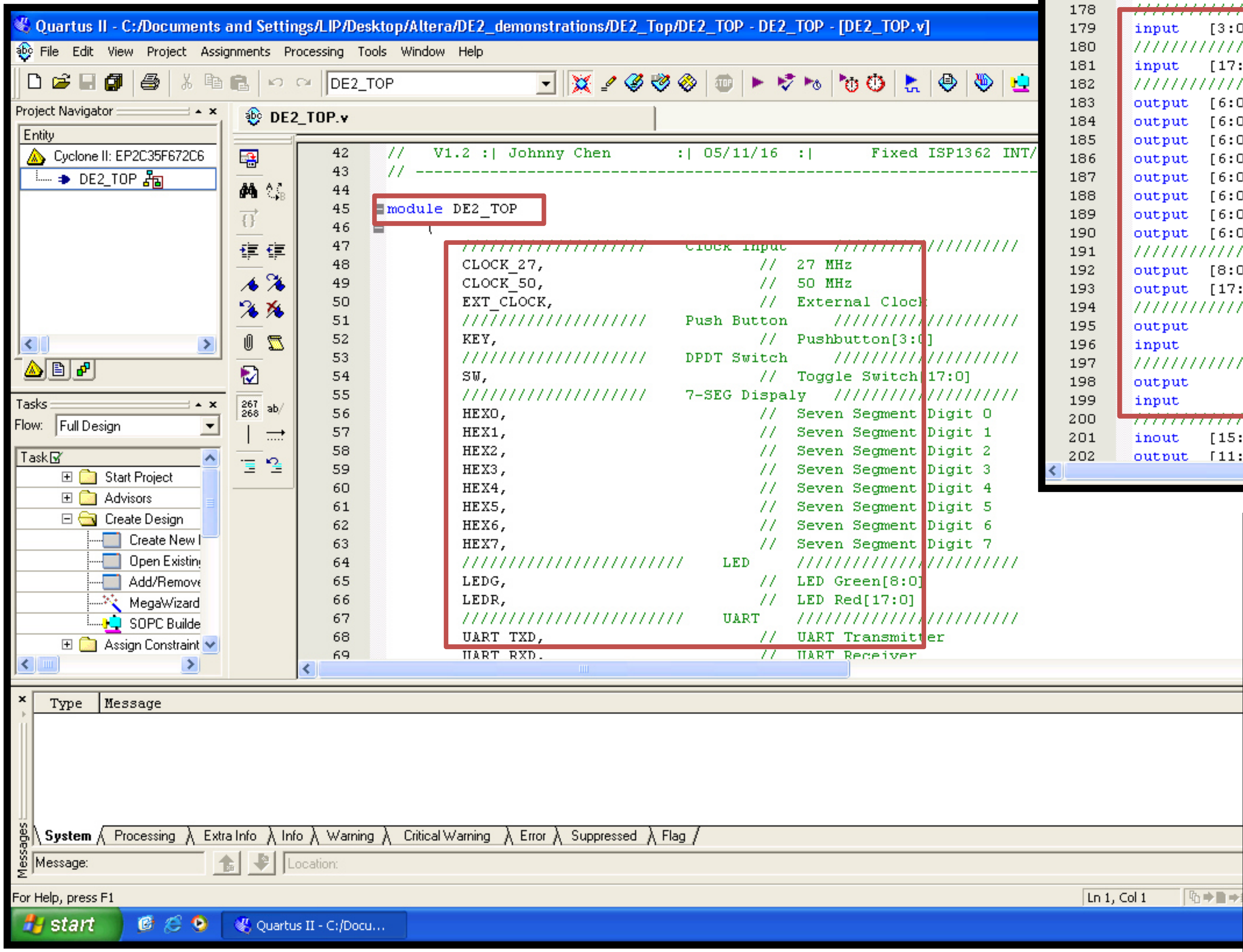
1) "OLA MUNDO" just play directly with bits

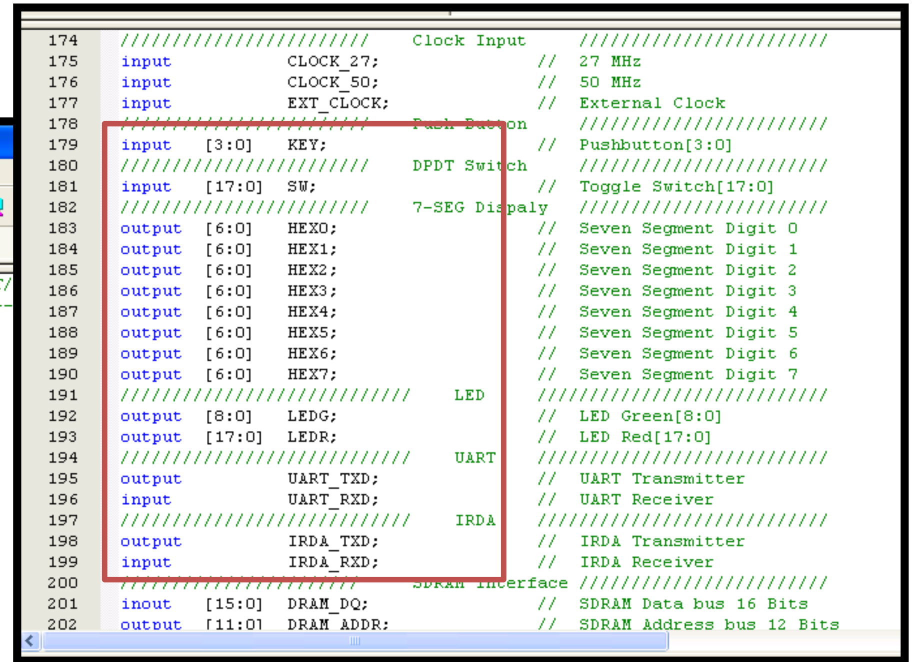
2) Make a module that has:
   - Inputs: 4 bits code binary number
   - Ouputs: the 7 lines of one 7-segment display
   - kernel: activate display segmentsto show decimal number

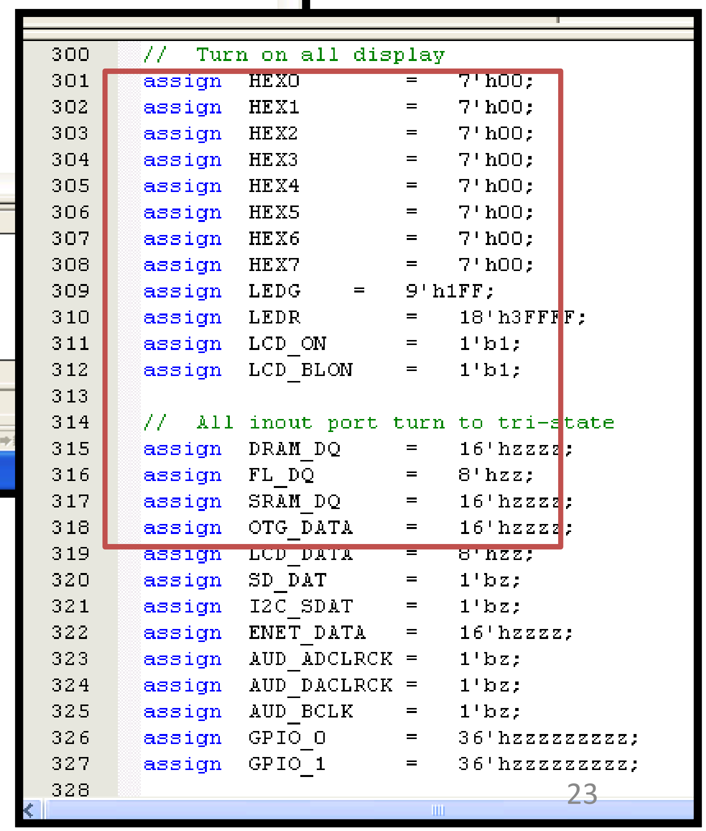
Helps…

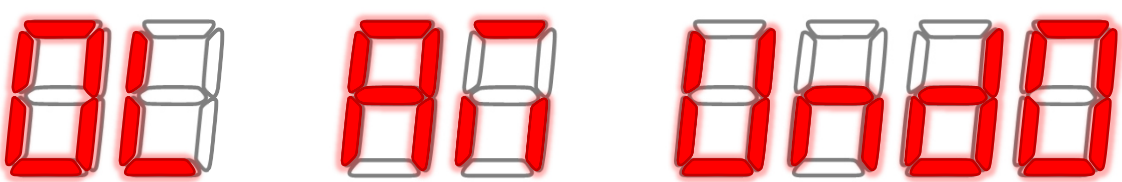
Usign DE2 you get all hardware issues "solved"
DE2_TOP project is a bare project: use it
DE2_TOP already has "friendly names"
DE2_TOP has some assigns... Play with them

File→Open Project→DE2_top










STOP