FPGAs and Verilog Lab

Implement a chronograph

Objective

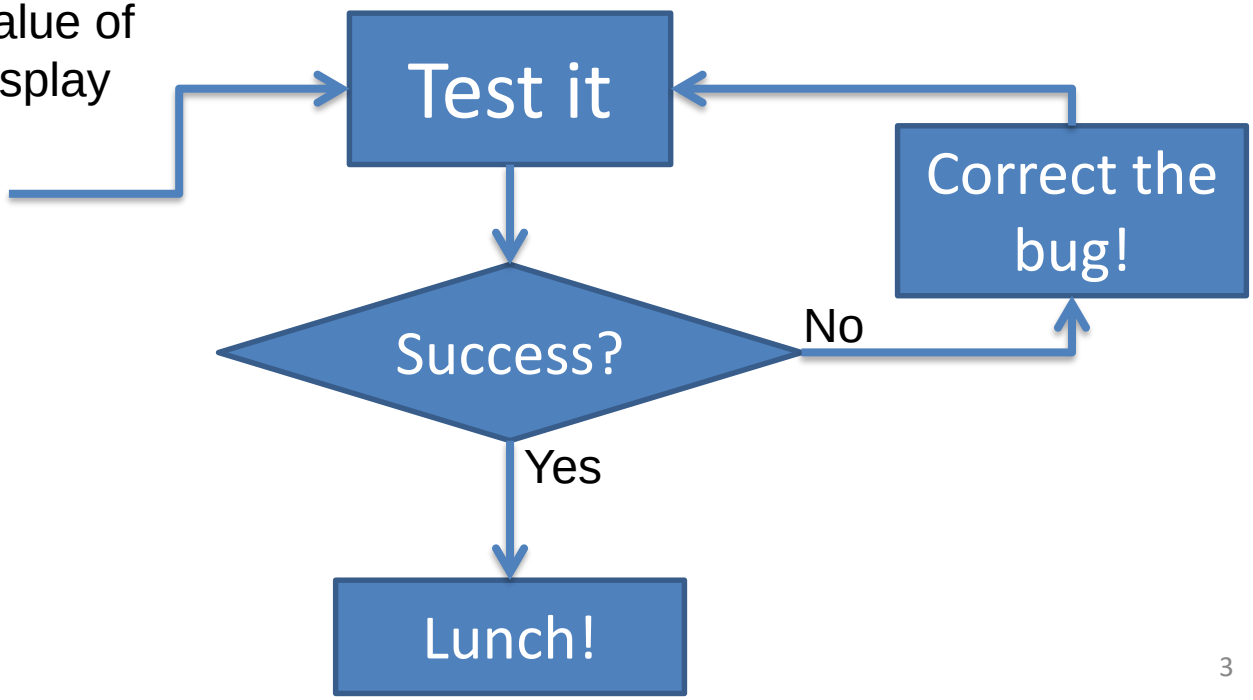
Implement in a FPGA development board a chronograph

Count seconds from 0 to 99 when a switch is up...

First Step

Open the Quartus Software;
Open the DE2_top project
compile;
program the board
-- The DE2_top gives you the definition of all pins and signals in DE2

Do Something different...
e.g. Change the value of the 7 Segments display
compile
program

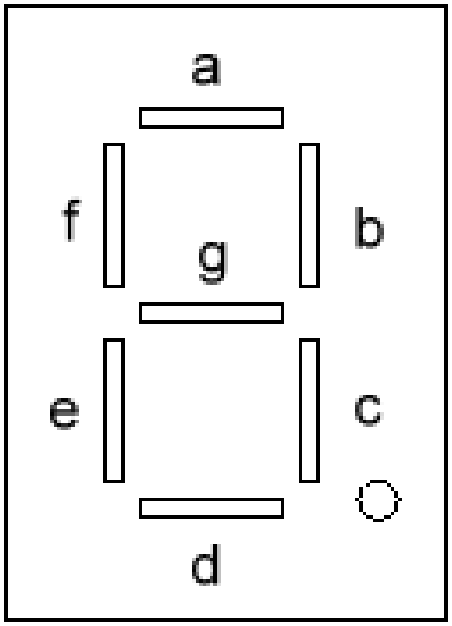


Decode the binary and put it in 7 seg display

Do a module that decodes binary to 7 segment display

Use the Switches as inputs and output to Hex6 and 7

SW[3:0] → Hex6
SW[7:4] → Hex7




```
case (bin)
//disp= "g f e d c b a“
     0: display=~7'b0111111;
     1: display=~7'b0000110;
     2: display=~7'b1011011;
     3: display=~7'b1001111;
     4: display=~7'b1100110;
     5: display=~7'b1101101;
     6: display=~7'b1111100;
     7: display=~7'b0000111;
     8: display=~7'b1111111;
     9: display=~7'b1100111;
     default:display=~7'b0000000;
endcase
```

Flash a LED with T=1s

Produce a one second clock and change a led each second

Identify the 50 MHz clock.

Divide the clock using a 32 bit counter:
Create a 32 bit counter and show in the LEDs

Now, everytime the counter value reaches 1s/(clock period)=1s/20ns
activate a flag and reset counter

Now you have a pulse each second...
Use it to turn on/off a LED

Count 1 second pulses

Make two counters: seconds and tens of seconds

Use the 1s pulse for the seconds.
Each 10 seconds give a pulse to the tens of seconds and reset the seconds counter

Control the counter

Use a switch to control when it counts

Use a Key for reset:

```
wire reset;
assign reset=Key[0];
always @ (posedge ... or negedge reset)
if (!reset)
        count <=0;
else .....
```

Have fun with it

****

****

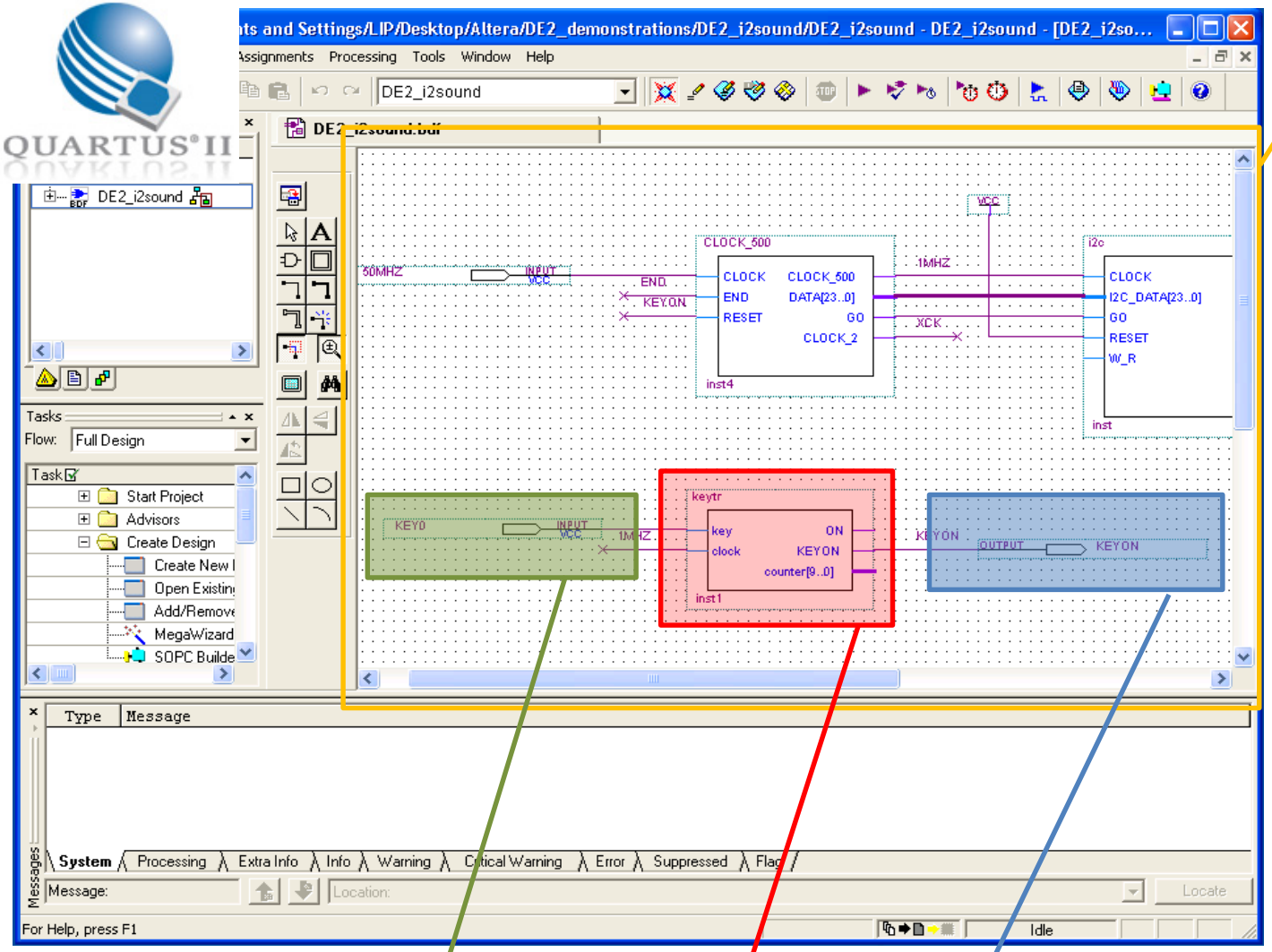
****

Nibble to 7 segment

```
case (nibble)
      0: segment <= ~8'b11111100;  // "a b c d e f g h"
      1: segment <= ~8'b01100000;
      2: segment <= ~8'b11011010;  //   --a--
      3: segment <= ~8'b11110010;  //  |     |
      4: segment <= ~8'b01100110;  //  f     b
      5: segment <= ~8'b10110110;  //  |     |
      6: segment <= ~8'b10111110;  //   --g--
      7: segment <= ~8'b11100000;  //  |     |
      8: segment <= ~8'b11111110;  //  e     c
      9: segment <= ~8'b11100110;  //  |     |
     10: segment <= ~8'b11101110;  //   --d--  (h)
     11: segment <= ~8'b00111110;
     12: segment <= ~8'b10011100;
     13: segment <= ~8'b01111010;
     14: segment <= ~8'b10011110;
     15: segment <= ~8'b10001110;
     default: segment <= 8'bx;
   endcase
```

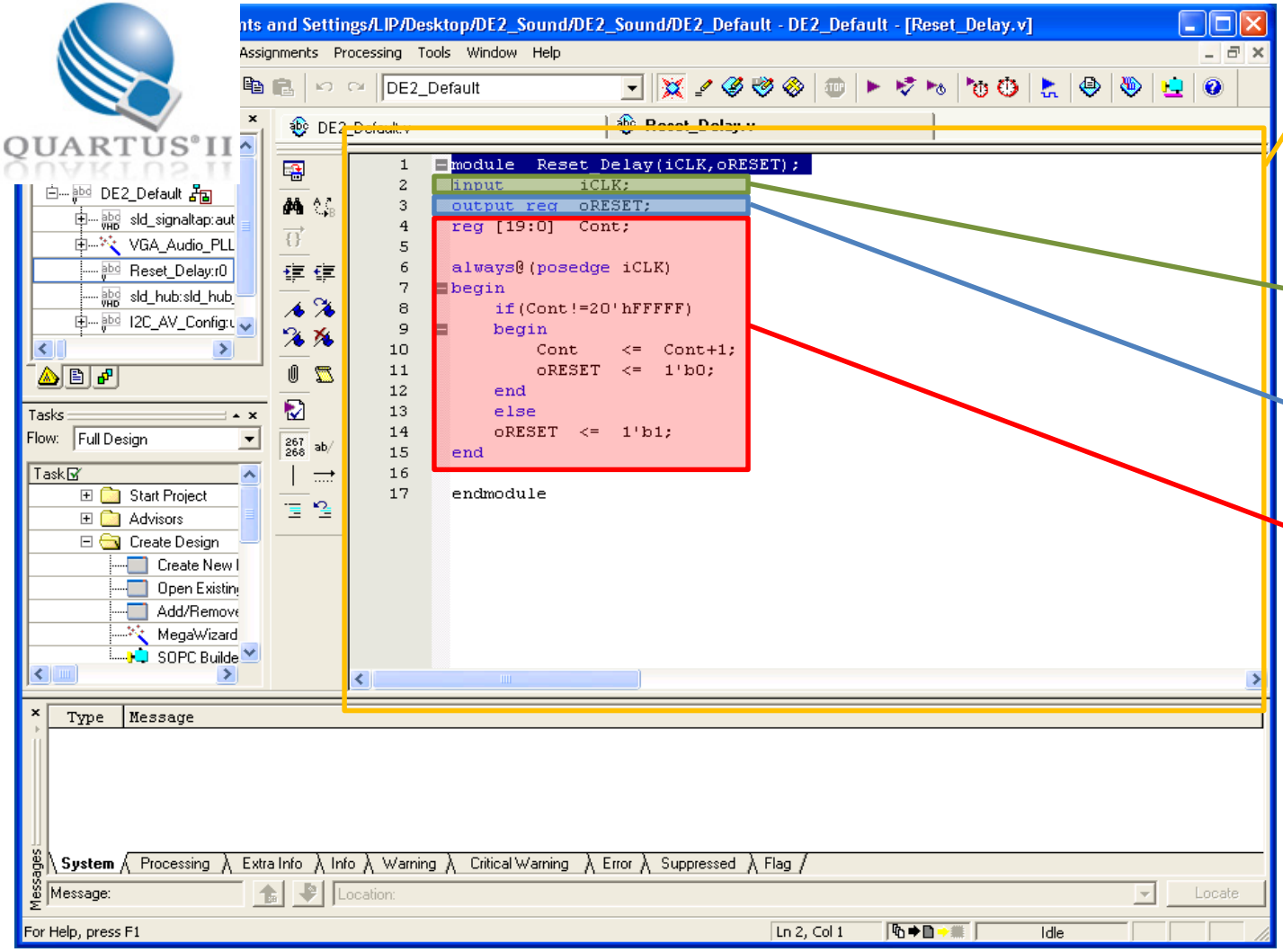
Software - Schematic

Schematic programming



Input **Logic** **Output**

Software - verilog



Verilog Programming

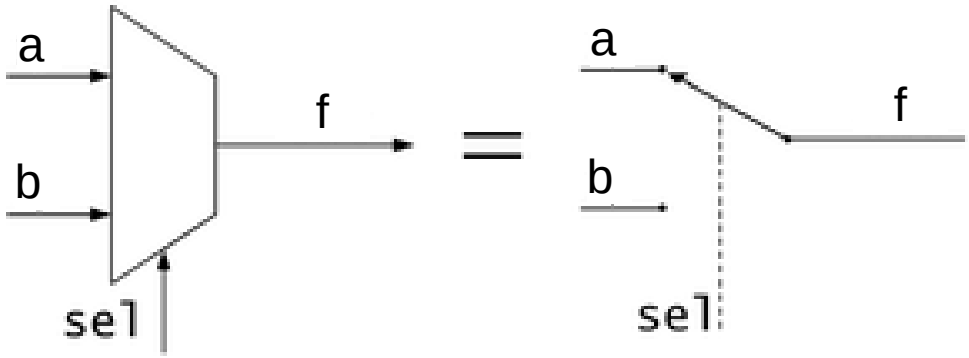
Input

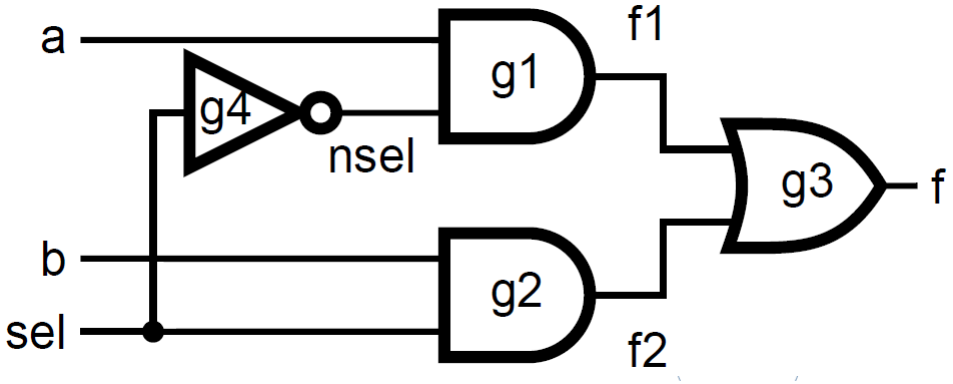
Output

Logic

Implementing a Multiplexer

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

and g1(f1, a, nsel),
    g2(f2, b, sel);
or  g3(f, f1, f2);
not g4(nsel, sel);


endmodule
```






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

reg f;

always @(a or b or sel)
  if (sel) f = a;
  else f = b;

endmodule
```

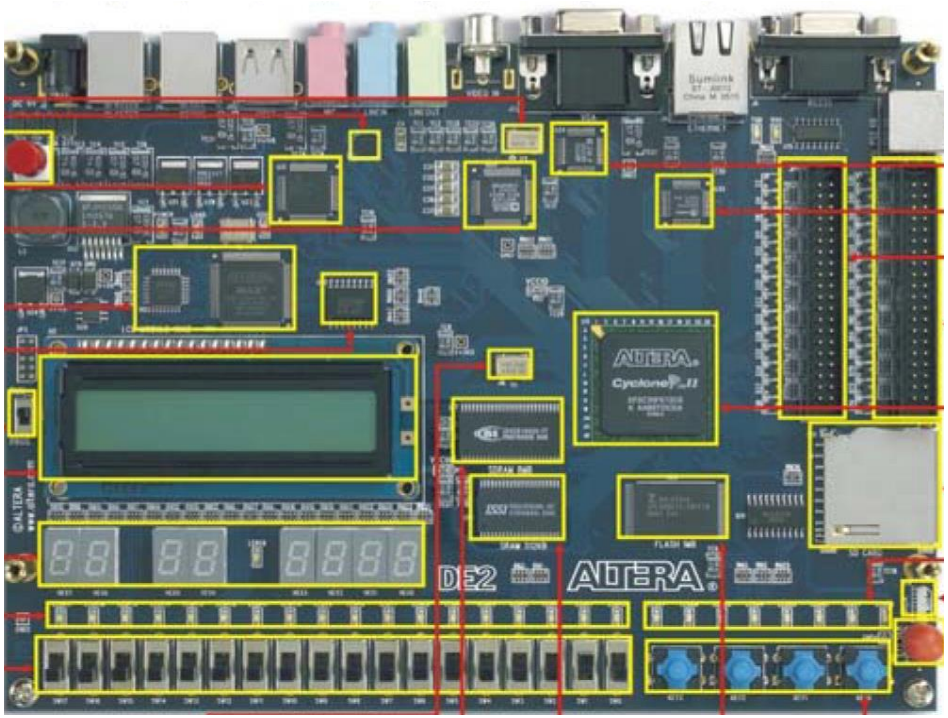
```
module mux(f, a, b, sel);
output f;
input a, b, sel;

assign f = sel ? a : b;

endmodule
```

Hardware: buy and hack it

Hardware

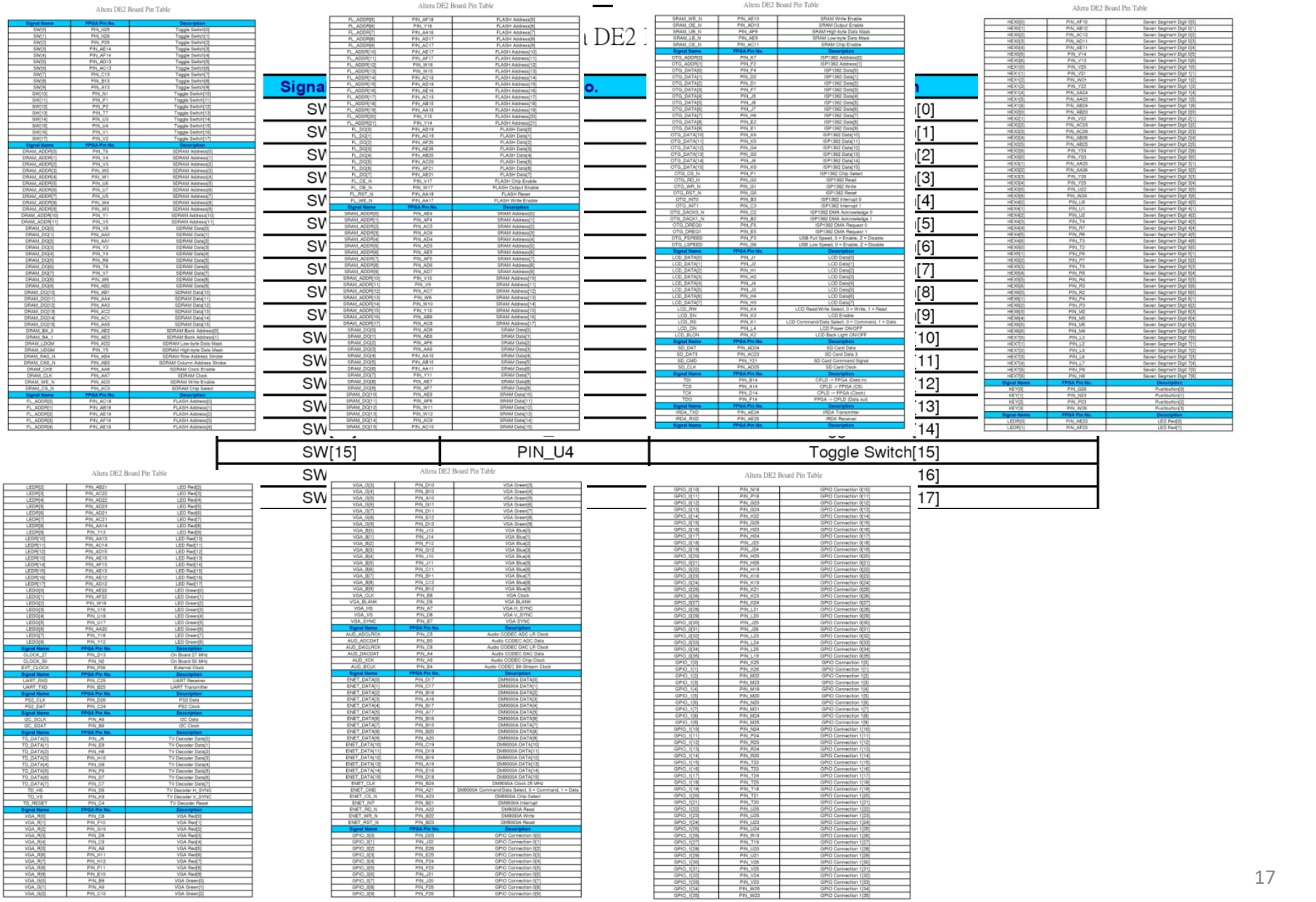
~200€ for academic

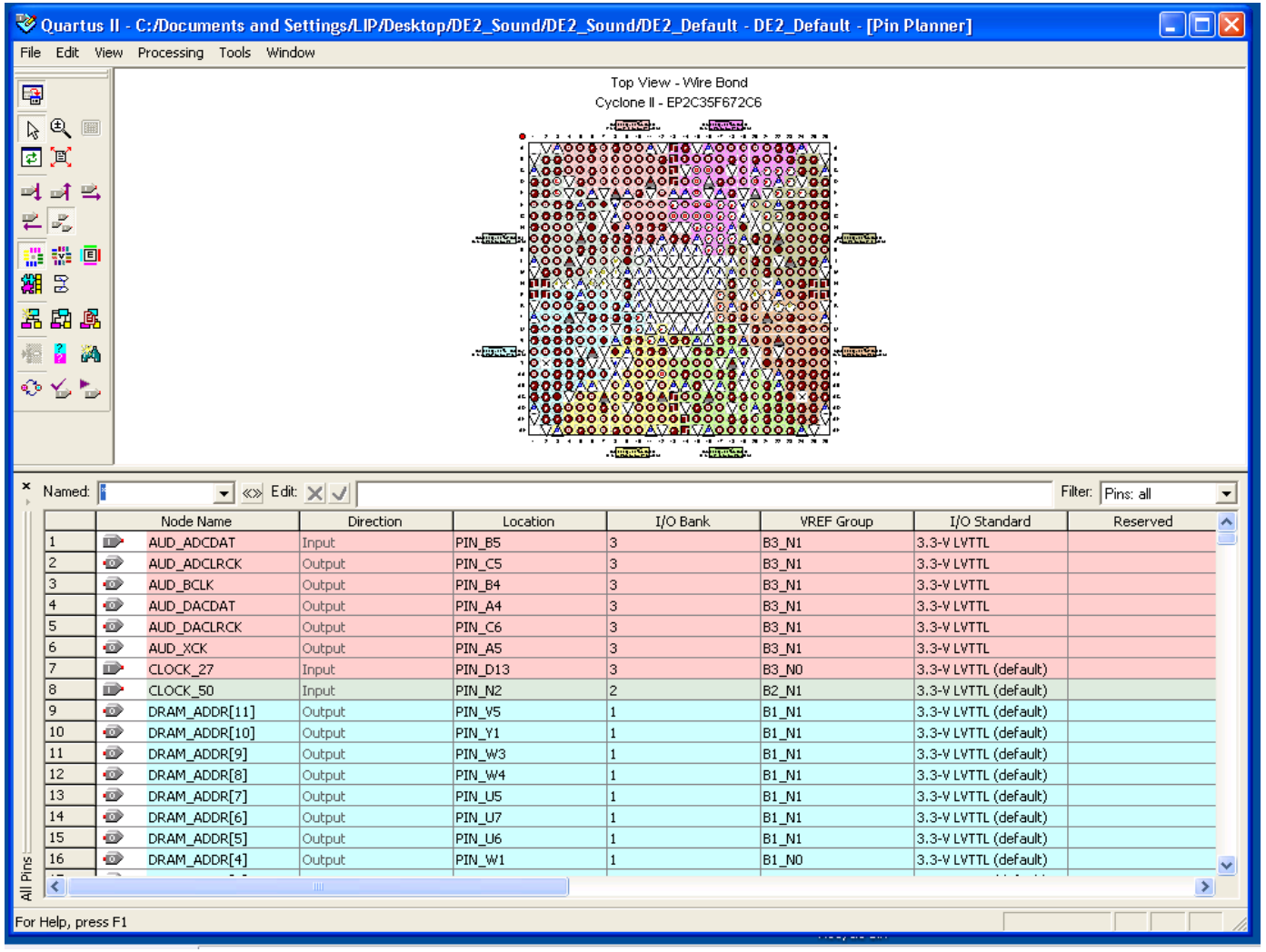
Hardware



Hardware

DE2_Pin_Table.pdf

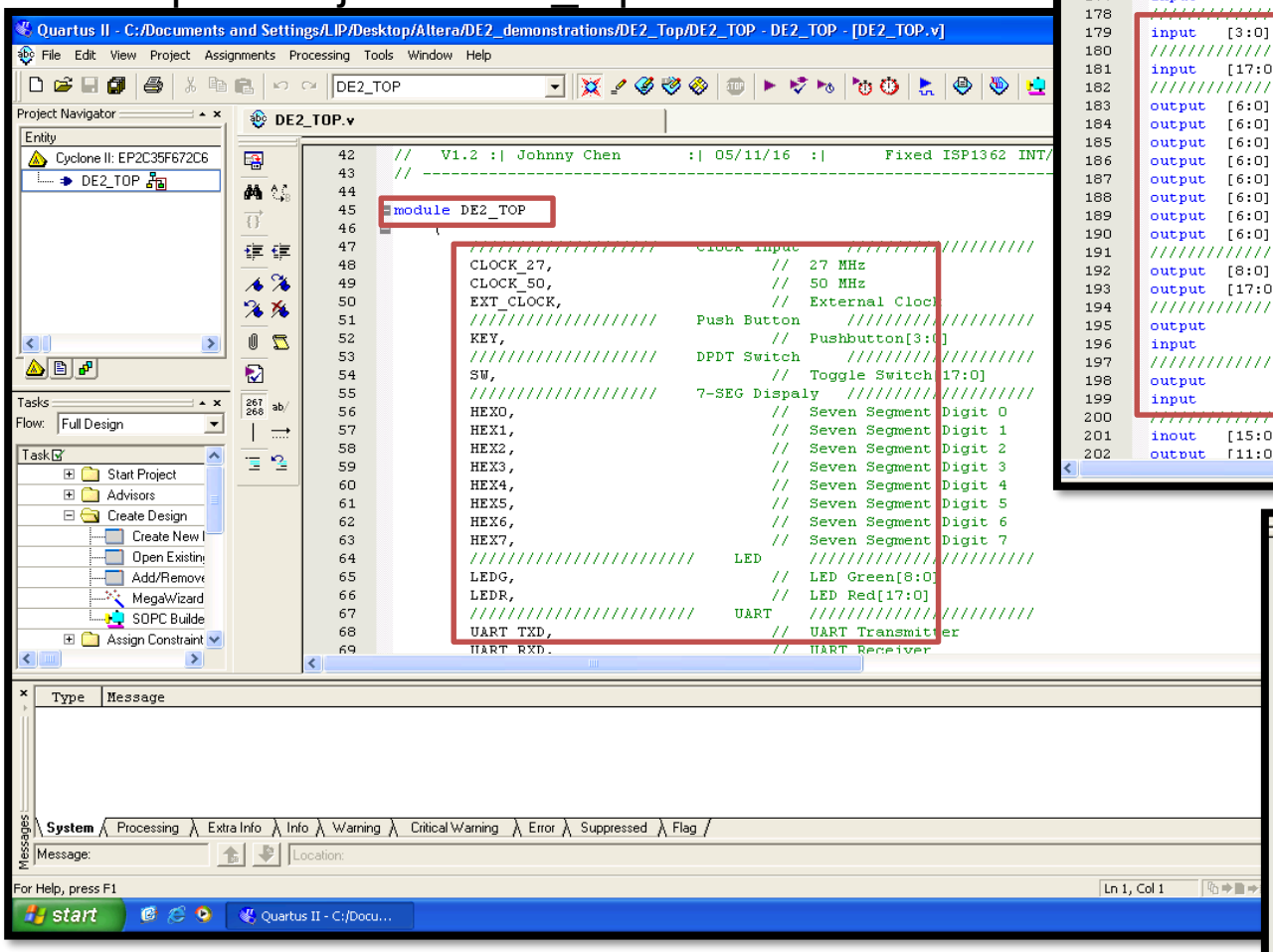




	Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
1	AUD_ADCDAT	Input	PIN_B5	3	B3_N1	3.3-V LVTTL	
2	AUD_ADCLRCK	Output	PIN_C5	3	B3_N1	3.3-V LVTTL	
3	AUD_BCLK	Output	PIN_B4	3	B3_N1	3.3-V LVTTL	
4	AUD_DACDAT	Output	PIN_A4	3	B3_N1	3.3-V LVTTL	
5	AUD_DACLRCK	Output	PIN_C6	3	B3_N1	3.3-V LVTTL	
6	AUD_XCK	Output	PIN_A5	3	B3_N1	3.3-V LVTTL	
7	CLOCK_27	Input	PIN_D13	3	B3_N0	3.3-V LVTTL (default)	
8	CLOCK_50	Input	PIN_N2	2	B2_N1	3.3-V LVTTL (default)	
9	DRAM_ADDR[11]	Output	PIN_V5	1	B1_N1	3.3-V LVTTL (default)	
10	DRAM_ADDR[10]	Output	PIN_Y1	1	B1_N1	3.3-V LVTTL (default)	
11	DRAM_ADDR[9]	Output	PIN_W3	1	B1_N1	3.3-V LVTTL (default)	
12	DRAM_ADDR[8]	Output	PIN_W4	1	B1_N1	3.3-V LVTTL (default)	
13	DRAM_ADDR[7]	Output	PIN_U5	1	B1_N1	3.3-V LVTTL (default)	
14	DRAM_ADDR[6]	Output	PIN_U7	1	B1_N1	3.3-V LVTTL (default)	
15	DRAM_ADDR[5]	Output	PIN_U6	1	B1_N1	3.3-V LVTTL (default)	
16	DRAM_ADDR[4]	Output	PIN_W1	1	B1_N0	3.3-V LVTTL (default)	

30 seconds to light a LED

File→Open Project→DE2_top




```
42  //   V1.2 :| Johnny Chen       :| 05/11/16  :|      Fixed ISP1362 INT/
43  // ---------------------------------------------------------------------
44
45  module DE2_TOP
46  (
47      ////////////////////    Clock Input     ////////////////////////
48      CLOCK_27,                       //  27 MHz
49      CLOCK_50,                       //  50 MHz
50      EXT_CLOCK,                      //  External Clock
51      ////////////////////    Push Button     ////////////////////////
52      KEY,                            //  Pushbutton[3:0]
53      ////////////////////    DPDT Switch     ////////////////////////
54      SW,                             //  Toggle Switch[17:0]
55      ////////////////////    7-SEG Dispaly   ////////////////////////
56      HEX0,                           //  Seven Segment Digit 0
57      HEX1,                           //  Seven Segment Digit 1
58      HEX2,                           //  Seven Segment Digit 2
59      HEX3,                           //  Seven Segment Digit 3
60      HEX4,                           //  Seven Segment Digit 4
61      HEX5,                           //  Seven Segment Digit 5
62      HEX6,                           //  Seven Segment Digit 6
63      HEX7,                           //  Seven Segment Digit 7
64      ////////////////////////    LED     ////////////////////////////
65      LEDG,                           //  LED Green[8:0]
66      LEDR,                           //  LED Red[17:0]
67      ////////////////////////    UART    ////////////////////////////
68      UART_TXD,                       //  UART Transmitter
69      UART_RXD,                       //  UART Receiver
```

```
174 //////////////////////    Clock Input     //////////////////////////
175 input           CLOCK_27;               //  27 MHz
176 input           CLOCK_50;               //  50 MHz
177 input           EXT_CLOCK;              //  External Clock
178 //////////////////////    Push Button     //////////////////////////
179 input   [3:0]   KEY;                    //  Pushbutton[3:0]
180 //////////////////////    DPDT Switch     //////////////////////////
181 input   [17:0]  SW;                     //  Toggle Switch[17:0]
182 //////////////////////    7-SEG Dispaly   //////////////////////////
183 output  [6:0]   HEX0;                   //  Seven Segment Digit 0
184 output  [6:0]   HEX1;                   //  Seven Segment Digit 1
185 output  [6:0]   HEX2;                   //  Seven Segment Digit 2
186 output  [6:0]   HEX3;                   //  Seven Segment Digit 3
187 output  [6:0]   HEX4;                   //  Seven Segment Digit 4
188 output  [6:0]   HEX5;                   //  Seven Segment Digit 5
189 output  [6:0]   HEX6;                   //  Seven Segment Digit 6
190 output  [6:0]   HEX7;                   //  Seven Segment Digit 7
191 ////////////////////////////    LED     ////////////////////////////
192 output  [8:0]   LEDG;                   //  LED Green[8:0]
193 output  [17:0]  LEDR;                   //  LED Red[17:0]
194 ////////////////////////////    UART    ////////////////////////////
195 output          UART_TXD;               //  UART Transmitter
196 input           UART_RXD;               //  UART Receiver
197 ////////////////////////////    IRDA    ////////////////////////////
198 output          IRDA_TXD;               //  IRDA Transmitter
199 input           IRDA_RXD;               //  IRDA Receiver
200 //////////////////////    SDRAM Interface //////////////////////////
201 inout   [15:0]  DRAM_DQ;                //  SDRAM Data bus 16 Bits
202 output  [11:0]  DRAM_ADDR;              //  SDRAM Address bus 12 Bits
```

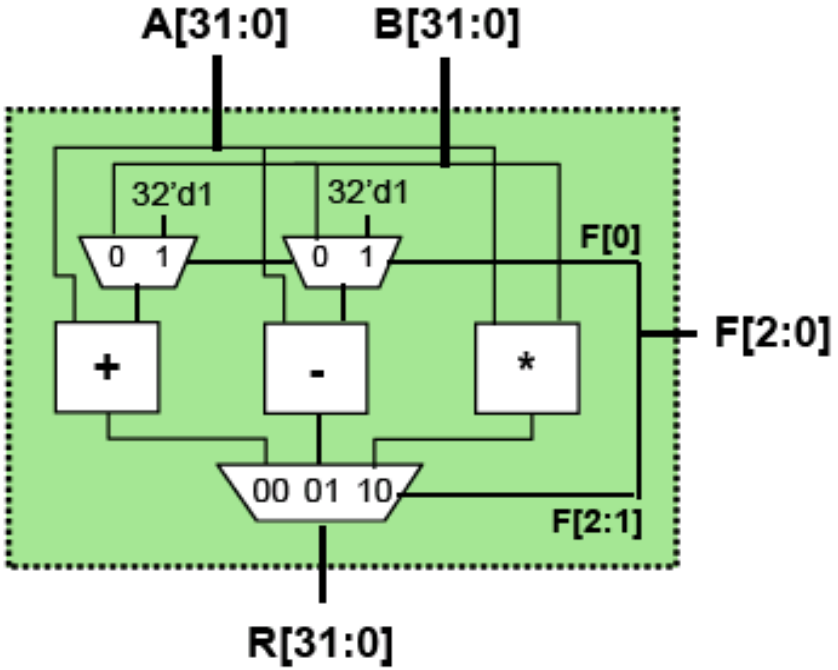
```
300 //  Turn on all display
301 assign  HEX0        =   7'h00;
302 assign  HEX1        =   7'h00;
303 assign  HEX2        =   7'h00;
304 assign  HEX3        =   7'h00;
305 assign  HEX4        =   7'h00;
306 assign  HEX5        =   7'h00;
307 assign  HEX6        =   7'h00;
308 assign  HEX7        =   7'h00;
309 assign  LEDG    =   9'h1FF;
310 assign  LEDR        =   18'h3FFFF;
311 assign  LCD_ON      =   1'b1;
312 assign  LCD_BLON    =   1'b1;
313
314 //  All inout port turn to tri-state
315 assign  DRAM_DQ     =   16'hzzzz;
316 assign  FL_DQ       =   8'hzz;
317 assign  SRAM_DQ     =   16'hzzzz;
318 assign  OTG_DATA    =   16'hzzzz;
319 assign  LCD_DATA    =   8'hzz;
320 assign  SD_DAT      =   1'bz;
321 assign  I2C_SDAT    =   1'bz;
322 assign  ENET_DATA   =   16'hzzzz;
323 assign  AUD_ADCLRCK =   1'bz;
324 assign  AUD_DACLRCK =   1'bz;
325 assign  AUD_BCLK    =   1'bz;
326 assign  GPIO_0      =   36'hzzzzzzzzz;
327 assign  GPIO_1      =   36'hzzzzzzzzz;
328
```

Build your own computer...

Connecting modules to build complex machines

- Modularity is essential to the success of large designs
- A Verilog `module` may contain submodules that are "wired together"
- High-level primitives enable direct synthesis of behavioral descriptions (functions such as additions, subtractions, shifts (<< and >>), etc.

Example: A 32-bit ALU



Function Table

F2	F1	F0	Function
0	0	0	A + B
0	0	1	A + 1
0	1	0	A - B
0	1	1	A - 1
1	0	X	A * B

Modules

2-to-1 MUX

```
module mux32two(i0,i1,sel,out);
input [31:0] i0,i1;
input sel;
output [31:0] out;

assign out = sel ? i1 : i0;

endmodule
```

3-to-1 MUX

```
module mux32three(i0,i1,i2,sel,out);
input [31:0] i0,i1,i2;
input [1:0] sel;
output [31:0] out;
reg [31:0] out;

always @ (i0 or i1 or i2 or sel)
begin
  case (sel)
    2'b00: out = i0;
    2'b01: out = i1;
    2'b10: out = i2;
    default: out = 32'bx;
  endcase
end
endmodule
```

32-bit Adder

```
module add32(i0,i1,sum);
input [31:0] i0,i1;
output [31:0] sum;

assign sum = i0 + i1;

endmodule
```

32-bit Subtracter

```
module sub32(i0,i1,diff);
input [31:0] i0,i1;
output [31:0] diff;

assign diff = i0 - i1;

endmodule
```

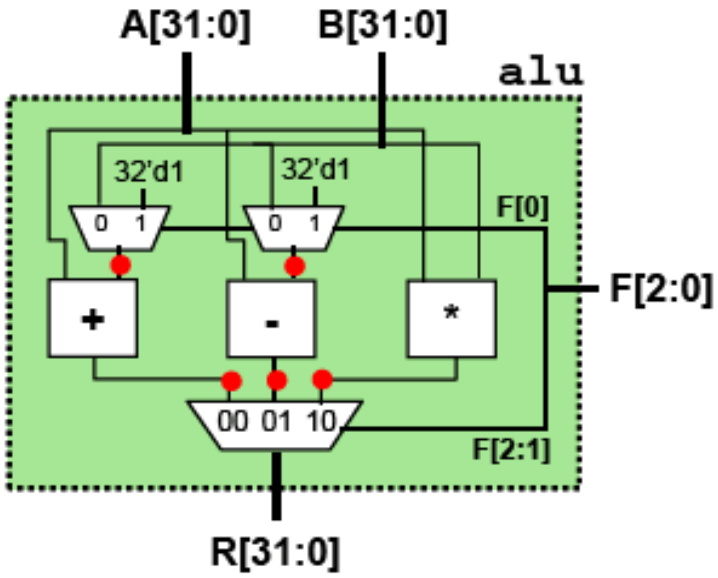
16-bit Multiplier

```
module mul16(i0,i1,prod);
input [15:0] i0,i1;
output [31:0] prod;

// this is a magnitude multiplier
// signed arithmetic later
assign prod = i0 * i1;

endmodule
```

Top-Level: connect the modules

- **Given submodules:**

```
module mux32two(i0,i1,sel,out);
module mux32three(i0,i1,i2,sel,out);
module add32(i0,i1,sum);
module sub32(i0,i1,diff);
module mul16(i0,i1,prod);
```



- **Declaration of the ALU Module:**

```
module alu(a, b, f, r);
  input [31:0] a, b;
  input [2:0] f;
  output [31:0] r;

  wire [31:0] addmux_out, submux_out;
  wire [31:0] add_out, sub_out, mul_out;

  mux32two   adder_mux(b, 32'd1, f[0], addmux_out);
  mux32two   sub_mux(b, 32'd1, f[0], submux_out);
  add32      our_adder(a, addmux_out, add_out);
  sub32      our_subtracter(a, submux_out, sub_out);
  mul16      our_multiplier(a[15:0], b[15:0], mul_out);
  mux32three output_mux(add_out, sub_out, mul_out, f[2:1], r);

endmodule
```

intermediate output nodes

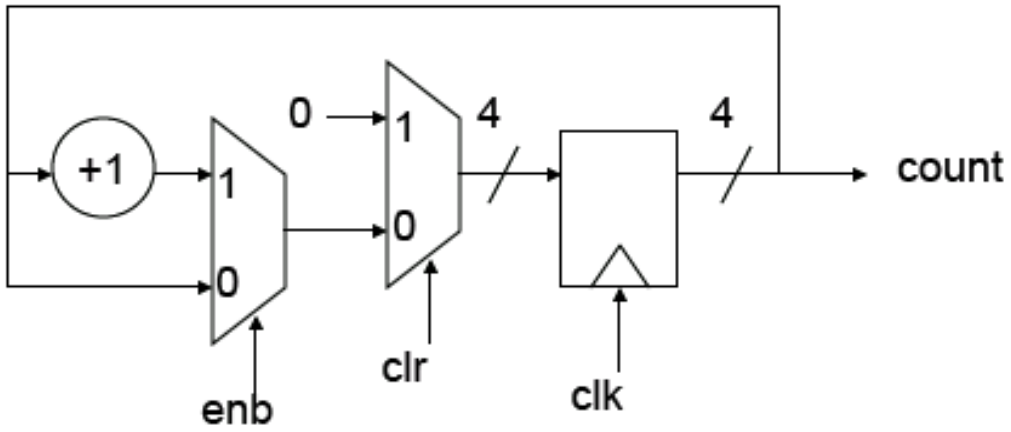
module names

(unique) instance names

corresponding wires/regs in module alu

Example: A simple counter

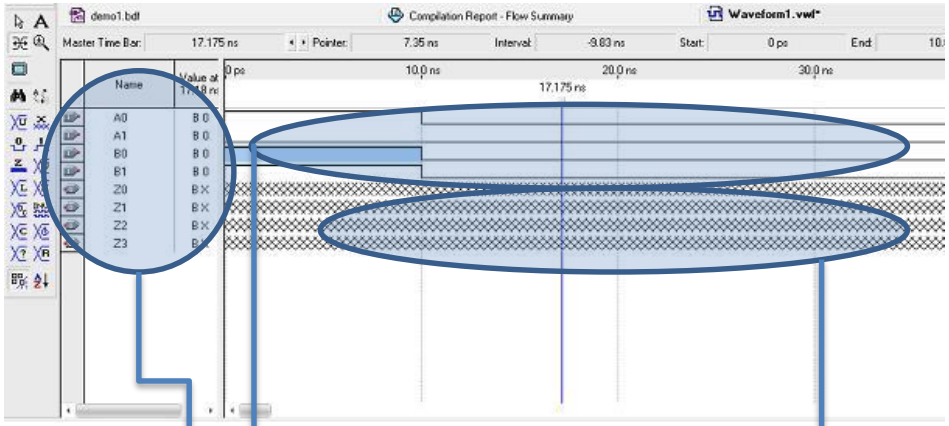
Do you know a simple way to count 10 ns pulses? Lots of them?




```
// 4-bit counter with enable and synchronous clear
module counter(input clk,enb,clr,
               output reg [3:0] count);
  always @(posedge clk) begin
    count <= clr ? 4'b0 : (enb ? count+1 : count);
  end
endmodule
```

Tools - Simulation

Quartus II



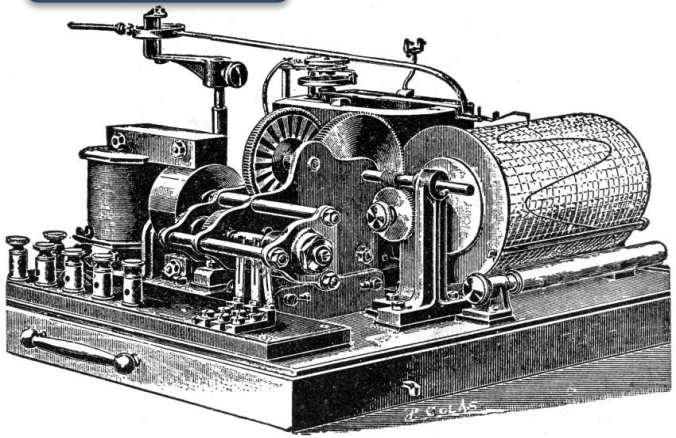
Define the signals :
Inputs – stimulus
Outputs - results

Define the stimulus to the system

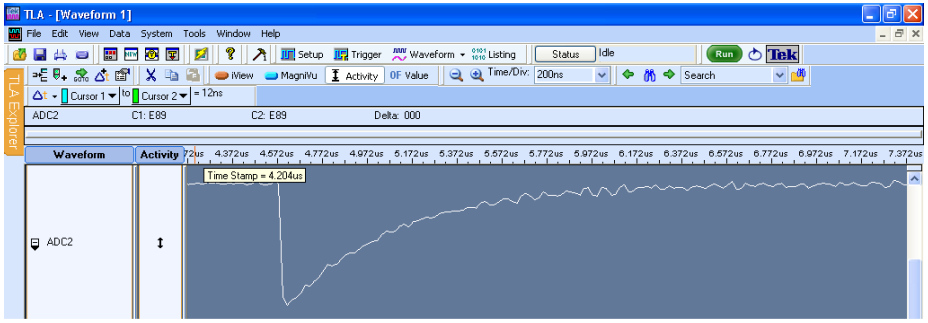
Run the simulation and you get the result

Tools – Measurement instruments

Waveforms



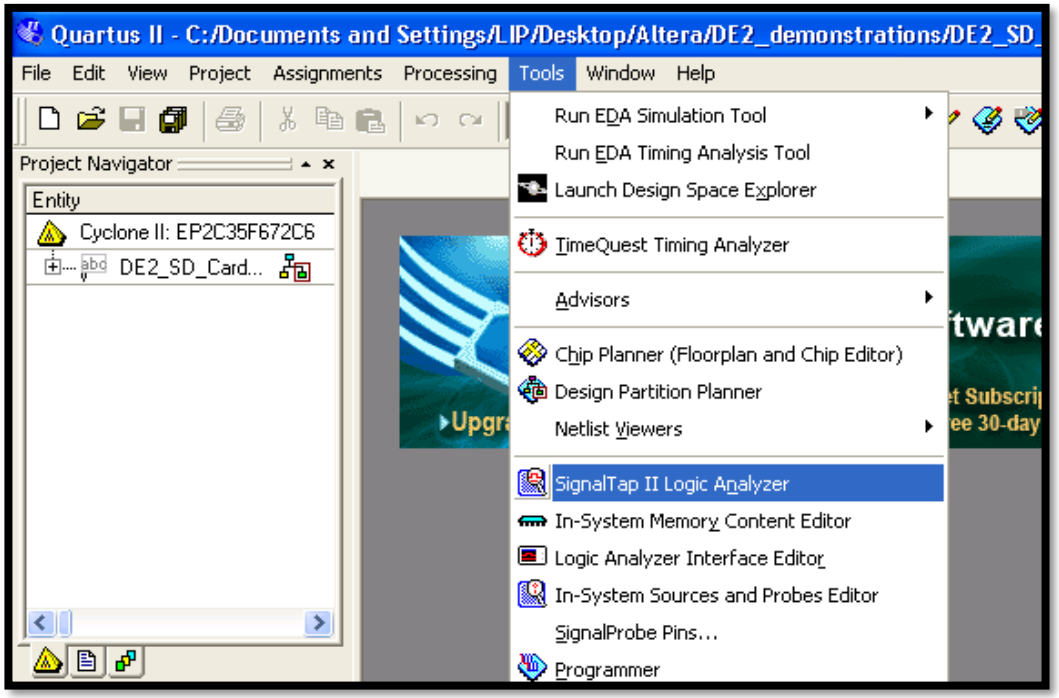
Logic Levels



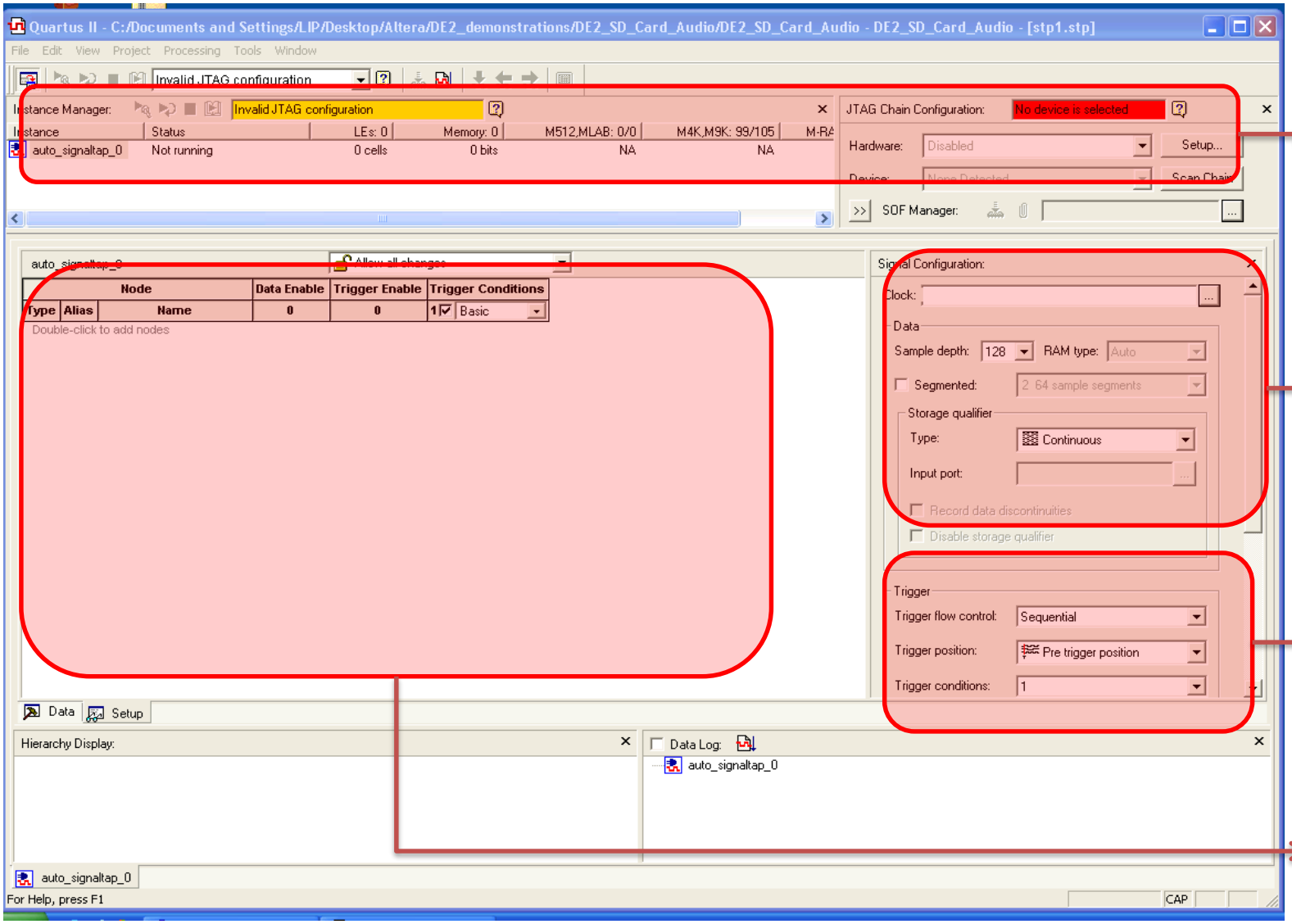
Tools – Internal Logic Analyser

Signal-TAP embedded Logic Analyser

Quartus II Handbook Version 9.0 Volume 3: Verification
14. Design Debugging Using the
SignalTap II Embedded Logic Analyzer



Tools – Internal Logic Analyser



CONTROL

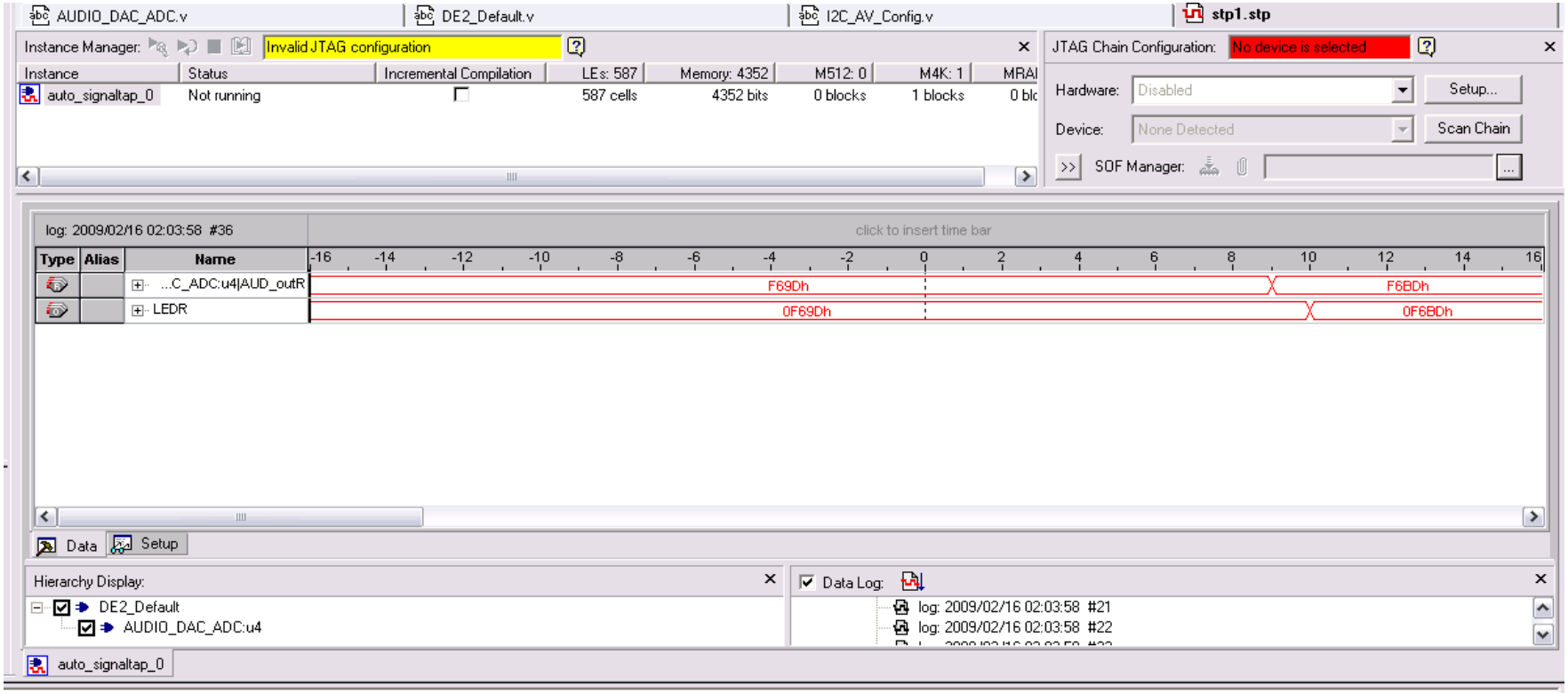
Clock definition

Trigger definition

The signal you want to "see"

Tools – Internal Logic Analyser

The logic analyser will collect data from the registers and output it through the JTAG programming interface

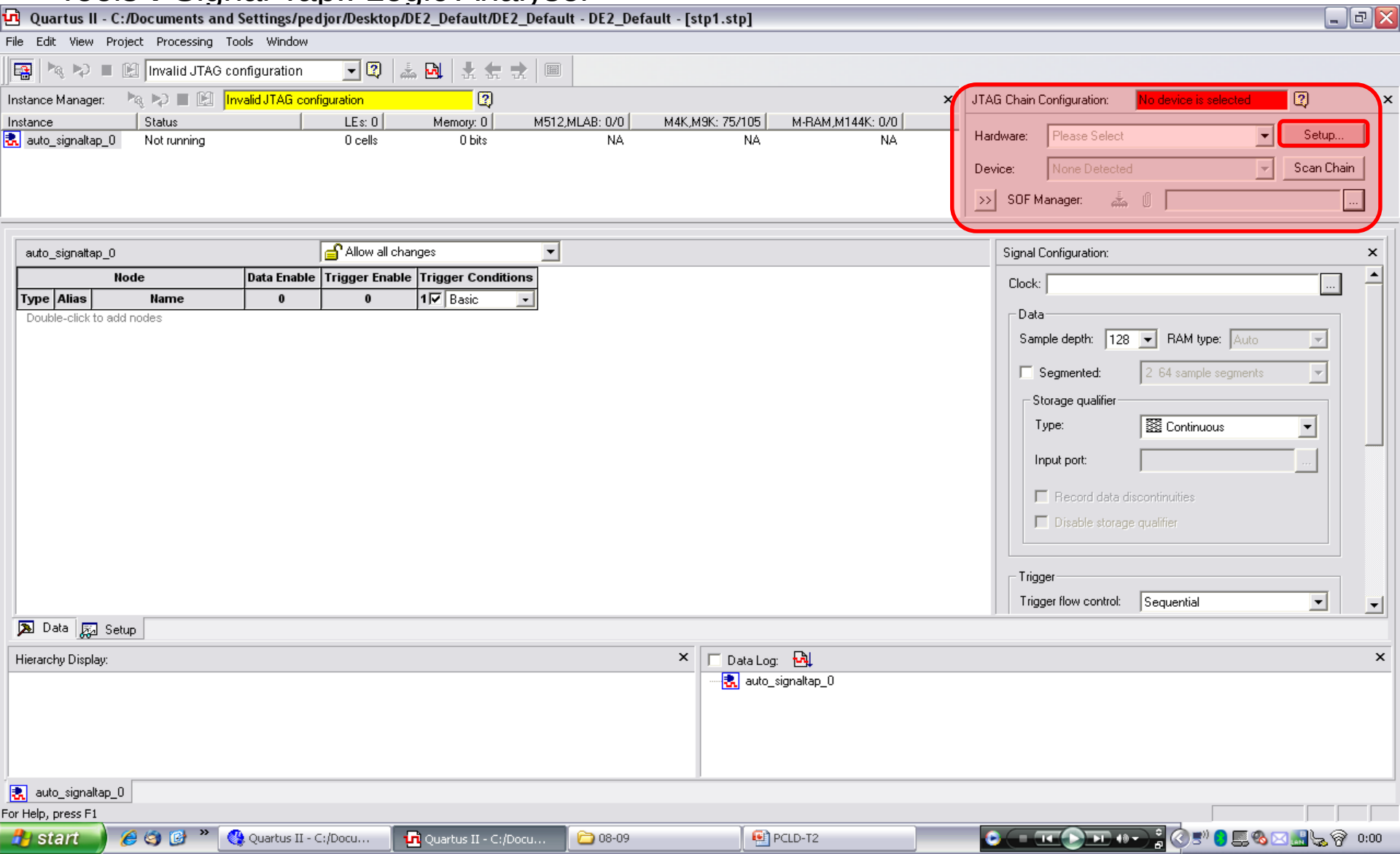


Tools – Signal probe

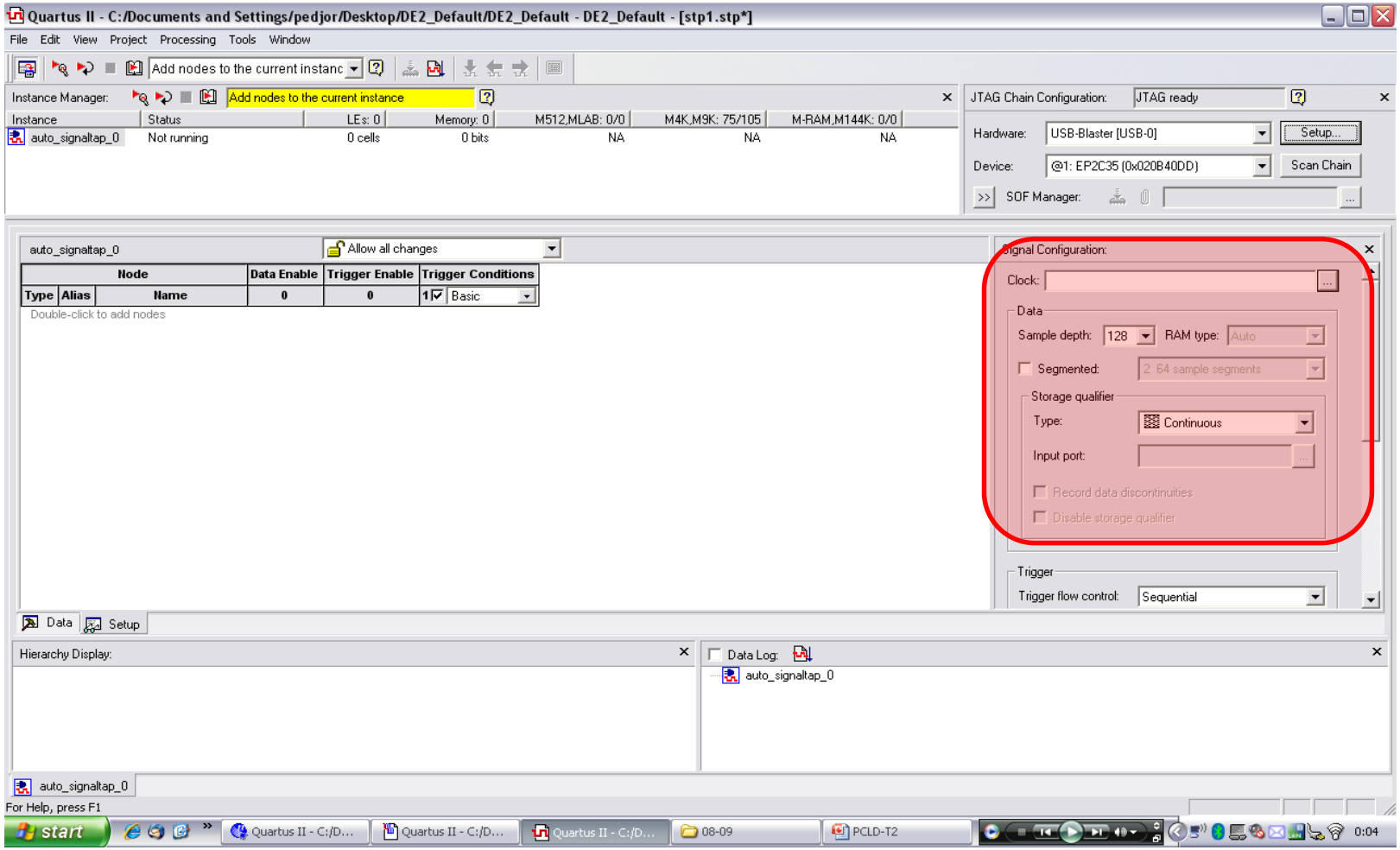
Internal signals can be extracted to output pins and connected to na external logic analyser. Signals can be exchanged easily…

SignalTap step by step

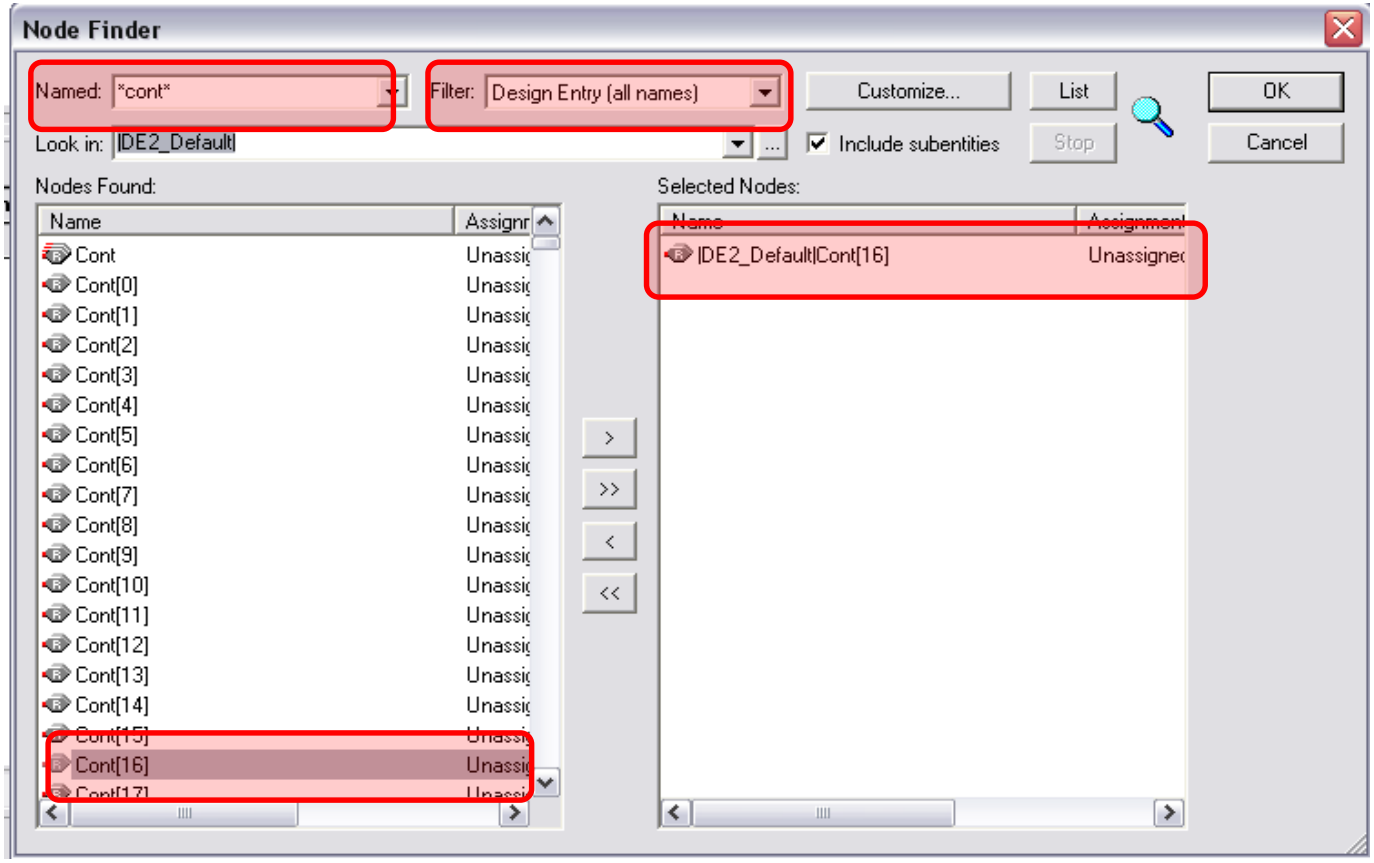
- Open DE2 default and compile it!
- Program DE2
- Tools→Signal TapII Logic Analyser



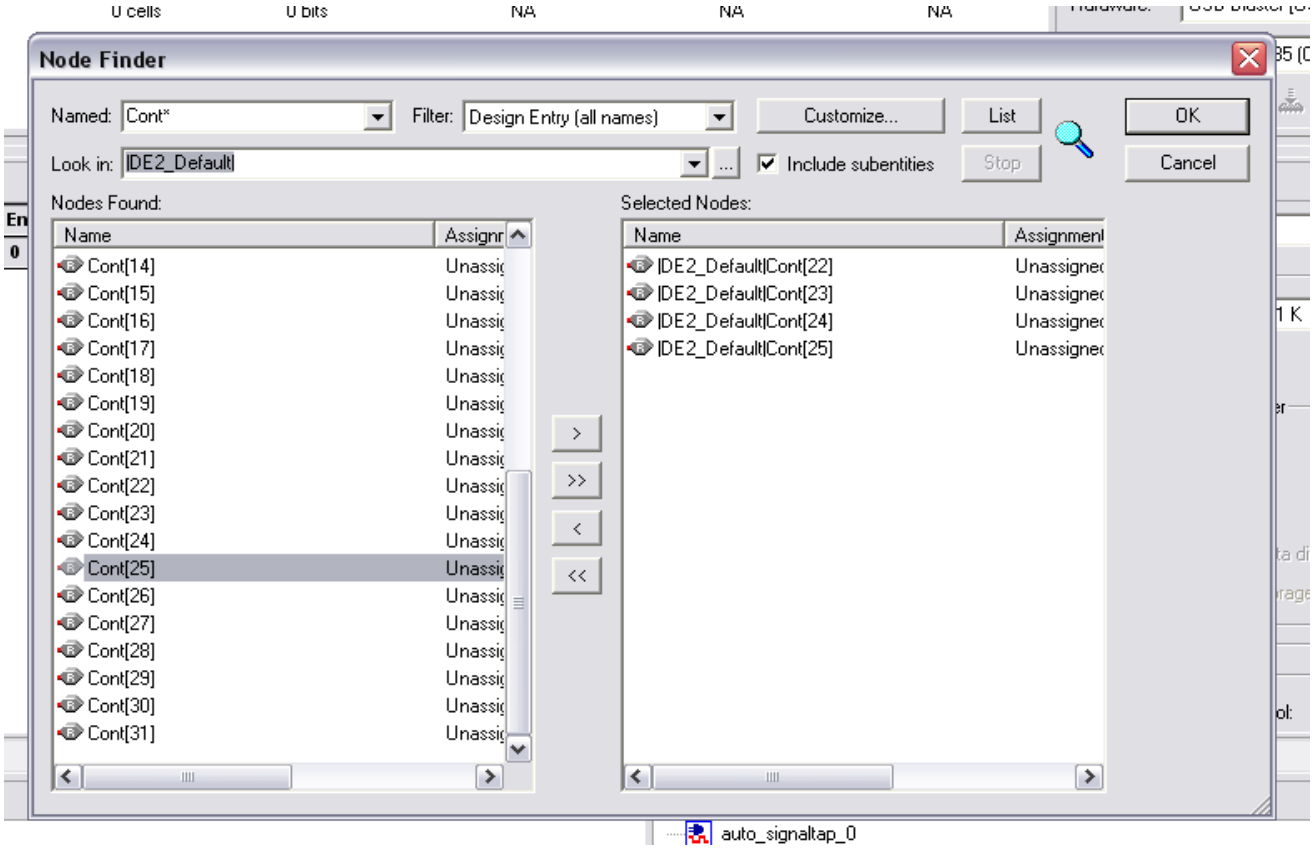
•Setup the hardware (choose the USB blaster)



•Define the clock

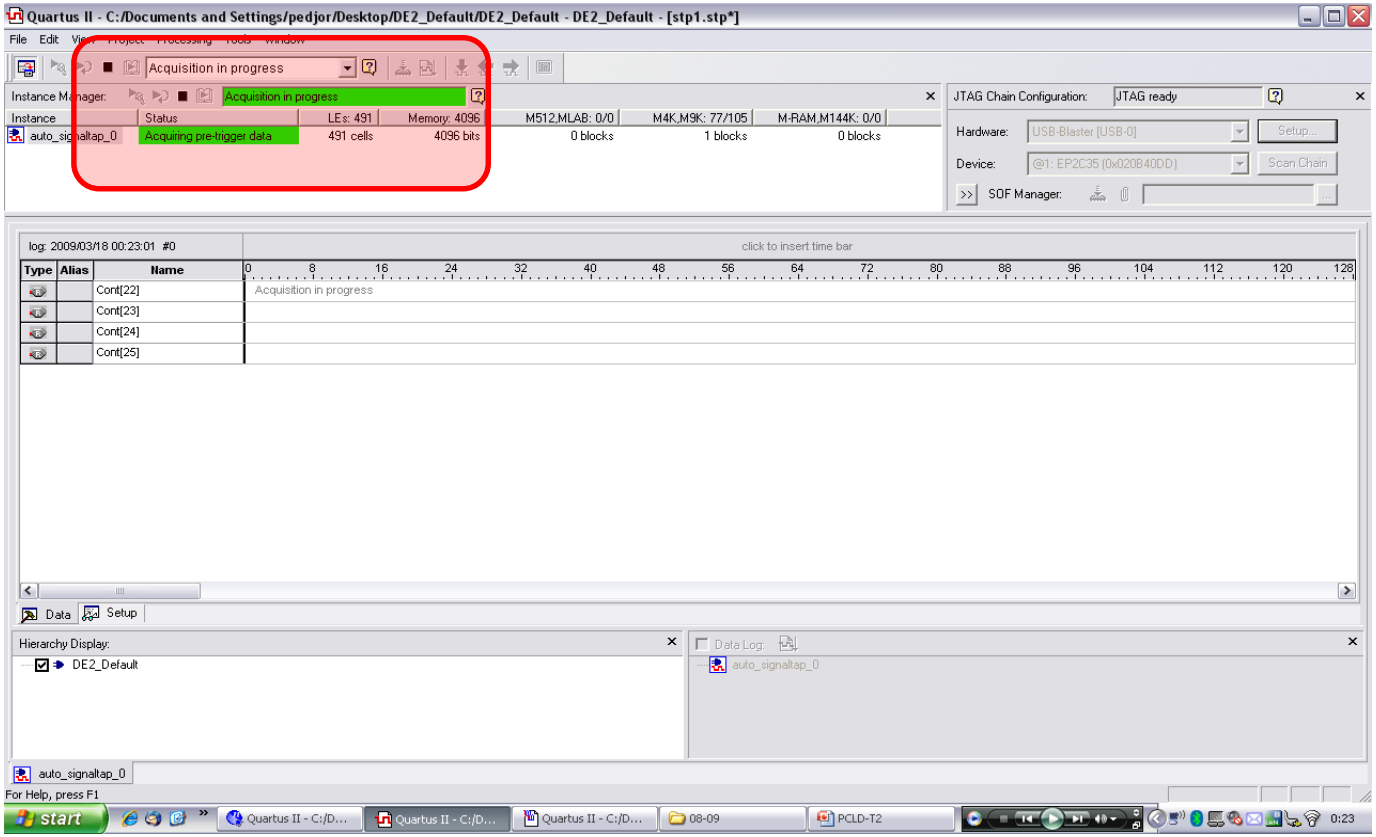

Node Finder
Named: *cont*
Filter: Design Entry (all names)
Customize...
List
OK
Look in: IDE2_Default
Include subentities
Stop
Cancel
Nodes Found:
Name
Assignr
Cont
Cont[0]
Cont[1]
Cont[2]
Cont[3]
Cont[4]
Cont[5]
Cont[6]
Cont[7]
Cont[8]
Cont[9]
Cont[10]
Cont[11]
Cont[12]
Cont[13]
Cont[14]
Cont[16]
Selected Nodes:
IDE2_Default|Cont[16]
Unassigned
>
>>
<
<<

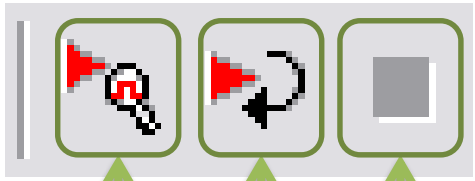
Choose the signals to observe. Choose Cont 25,24,23,22



•Compile the project! You may need to save some files and answer some questions
•Program the board

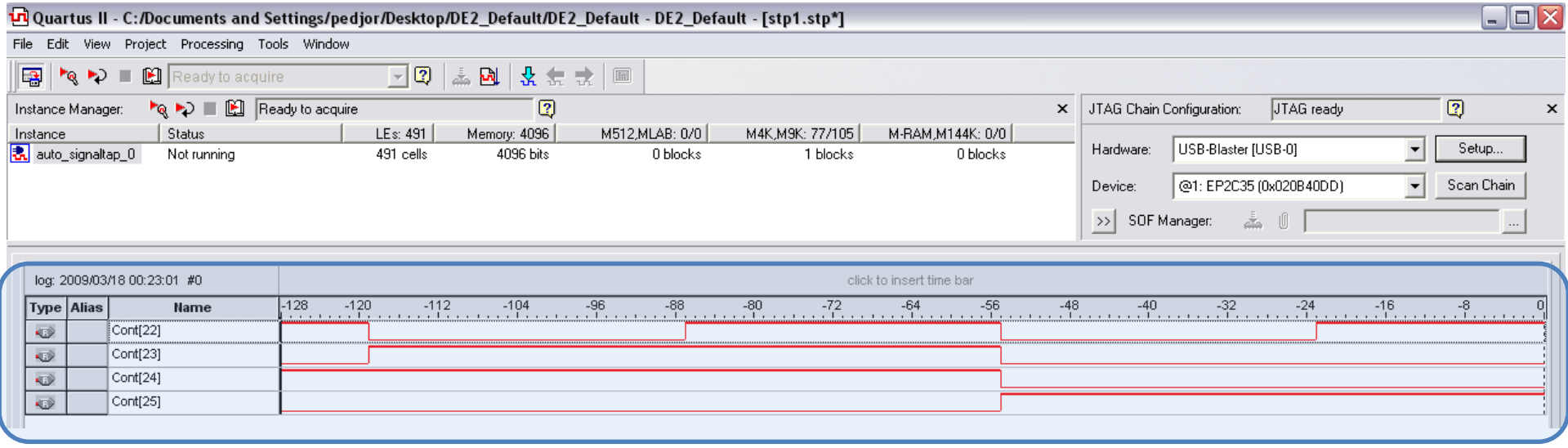
Run Analysis→Green (acquisition in progress, acquiring data



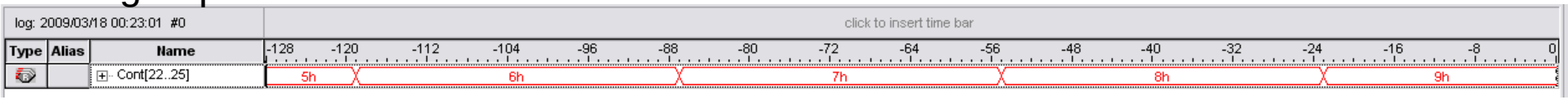


Run Analysis (1 time) Run Analysis (continuous) Stop

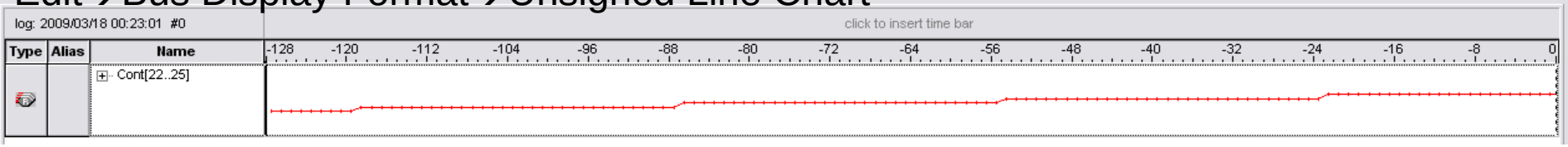
Finally: data!!



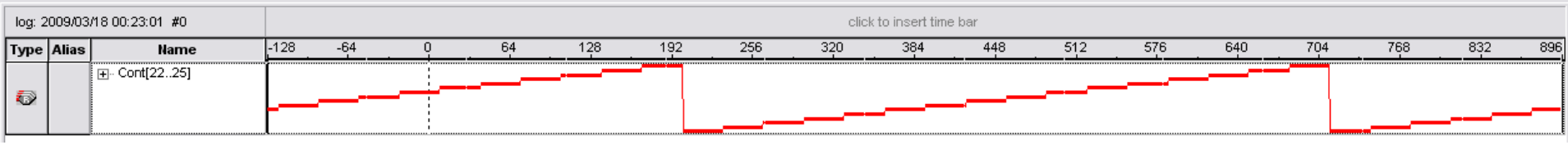
- Select the four signals;
- Edit→group



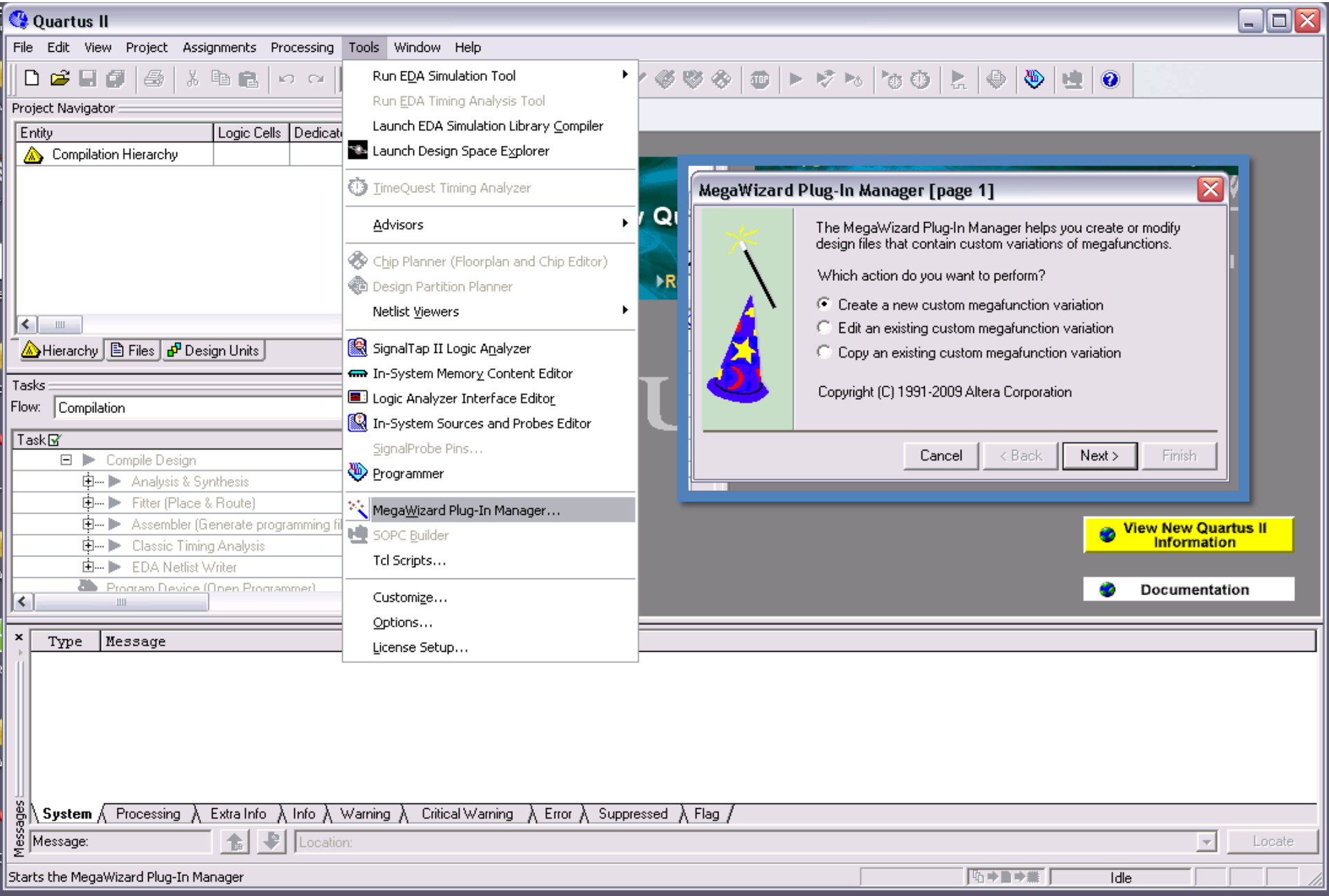
- Select the group
- Edit→Bus Display Format→Unsigned Line Chart

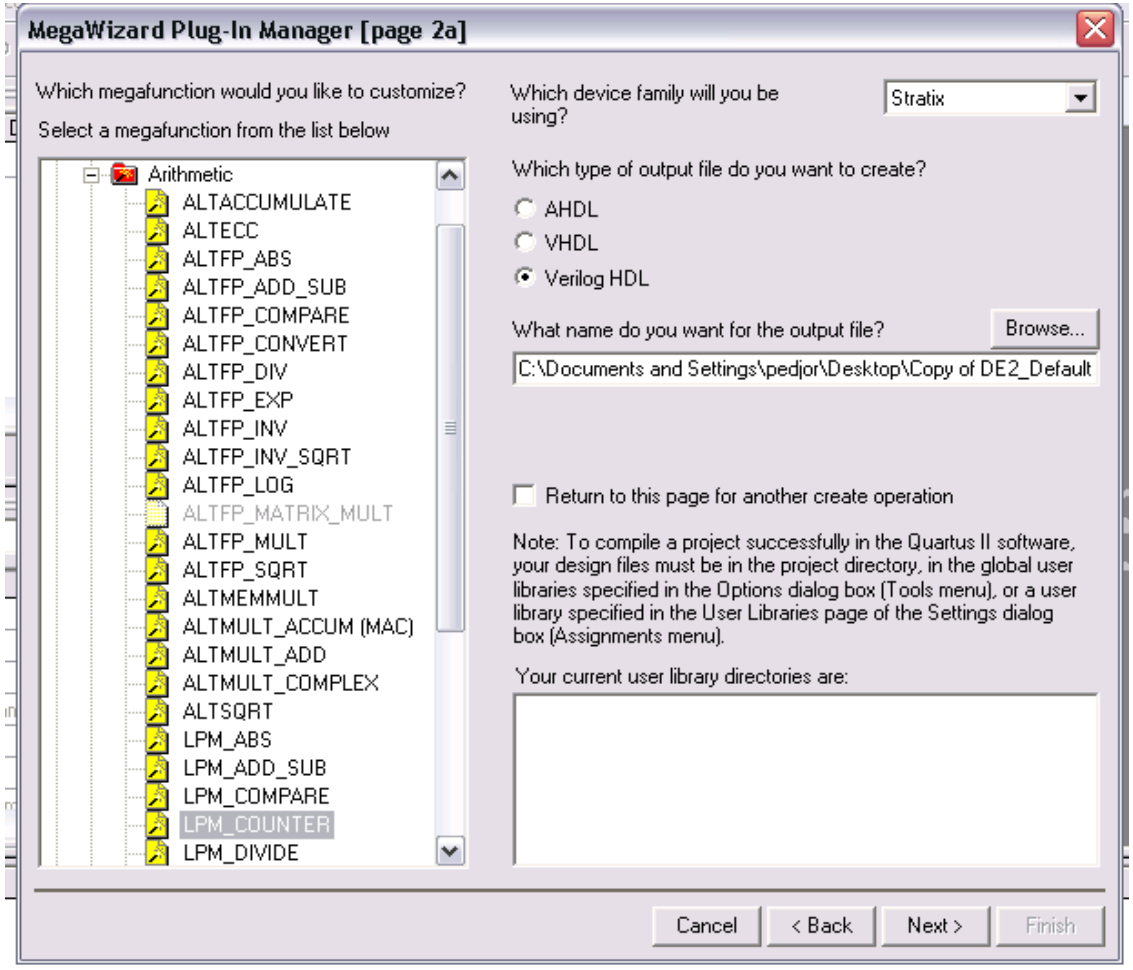


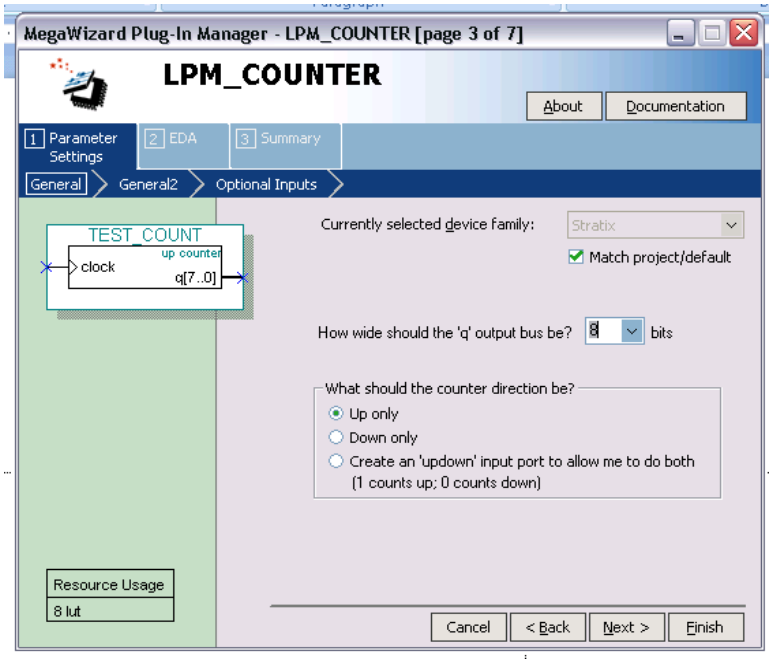
- Unzoom

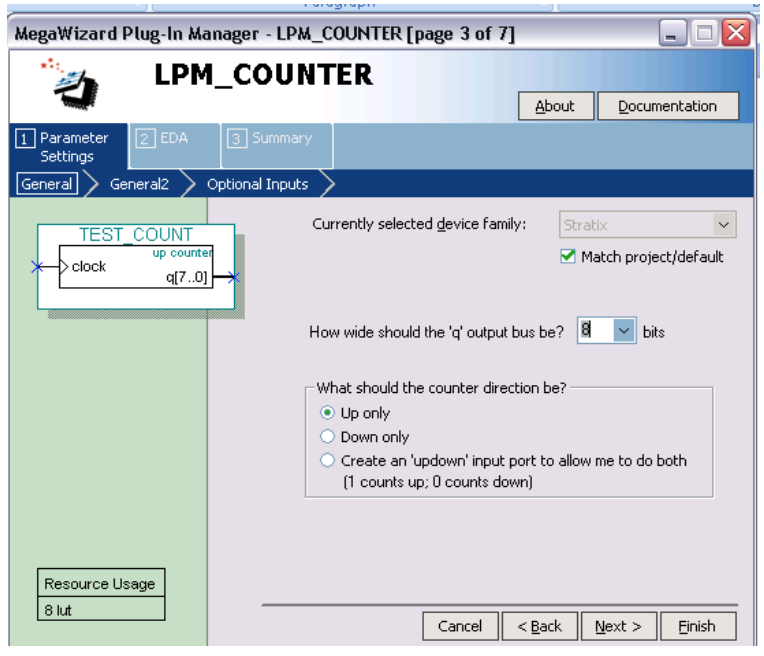


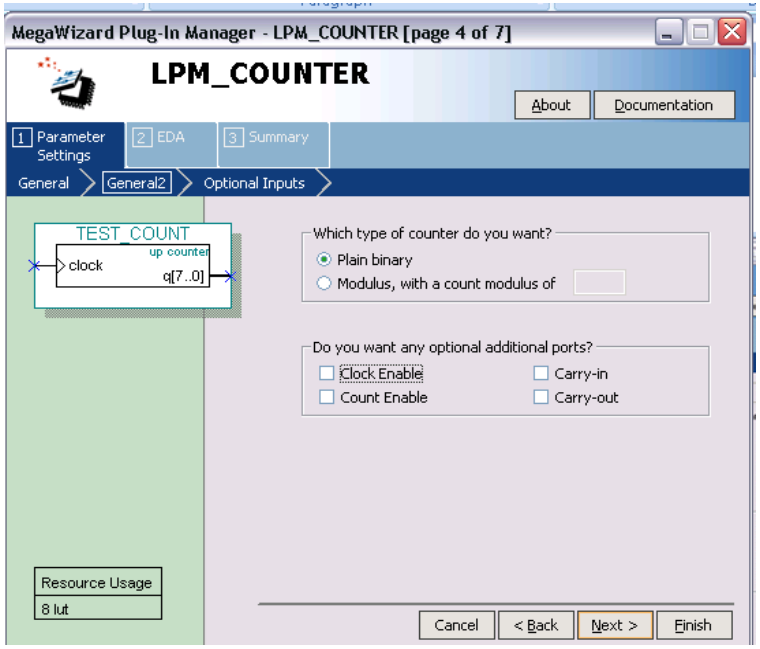
Tools – Mega Wizard

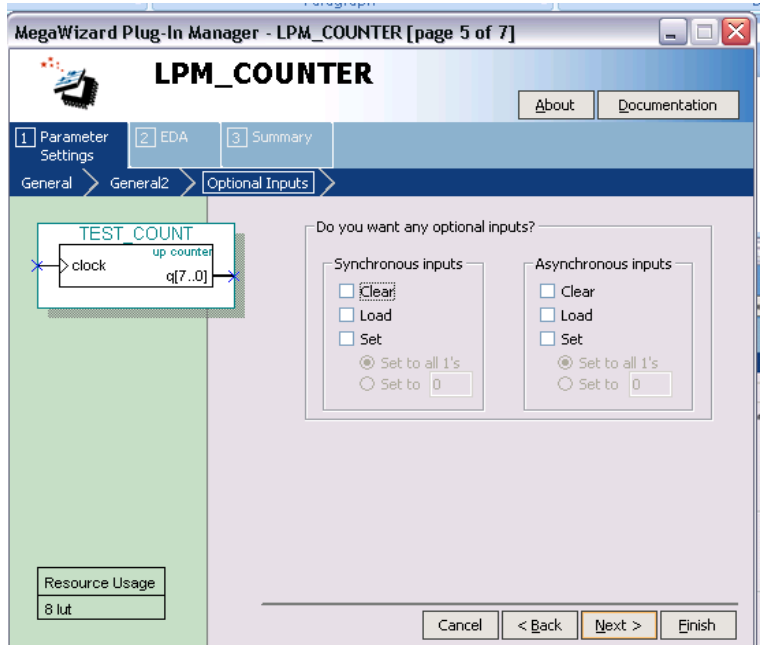


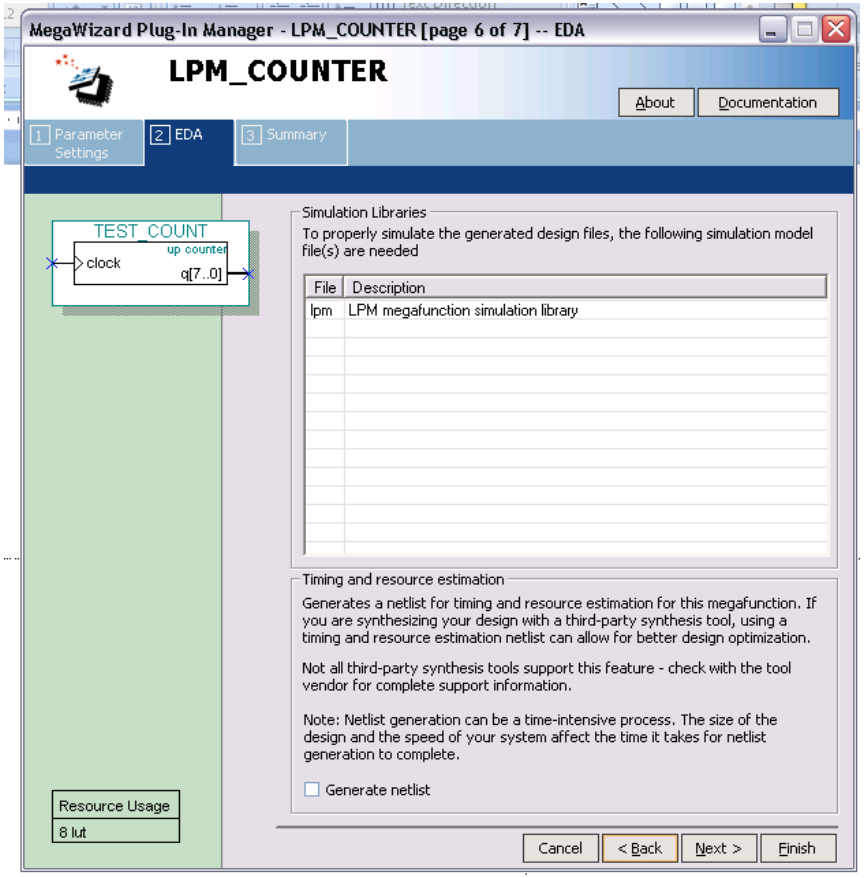

MegaWizard Plug-In Manager [page 2a]
Which megafunction would you like to customize?
Select a megafunction from the list below
Arithmetic
ALTACCUMULATE
ALTECC
ALTFP_ABS
ALTFP_ADD_SUB
ALTFP_COMPARE
ALTFP_CONVERT
ALTFP_DIV
ALTFP_EXP
ALTFP_INV
ALTFP_INV_SQRT
ALTFP_LOG
ALTFP_MATRIX_MULT
ALTFP_MULT
ALTFP_SQRT
ALTMEMMULT
ALTMULT_ACCUM (MAC)
ALTMULT_ADD
ALTMULT_COMPLEX
ALTSQRT
LPM_ABS
LPM_ADD_SUB
LPM_COMPARE
LPM_COUNTER
LPM_DIVIDE
Which device family will you be using?
Stratix
Which type of output file do you want to create?
AHDL
VHDL
Verilog HDL
What name do you want for the output file?
Browse...
C:\Documents and Settings\pedjor\Desktop\Copy of DE2_Default
Return to this page for another create operation
Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in the global user libraries specified in the Options dialog box (Tools menu), or a user library specified in the User Libraries page of the Settings dialog box (Assignments menu).
Your current user library directories are:
Cancel
< Back
Next >
Finish

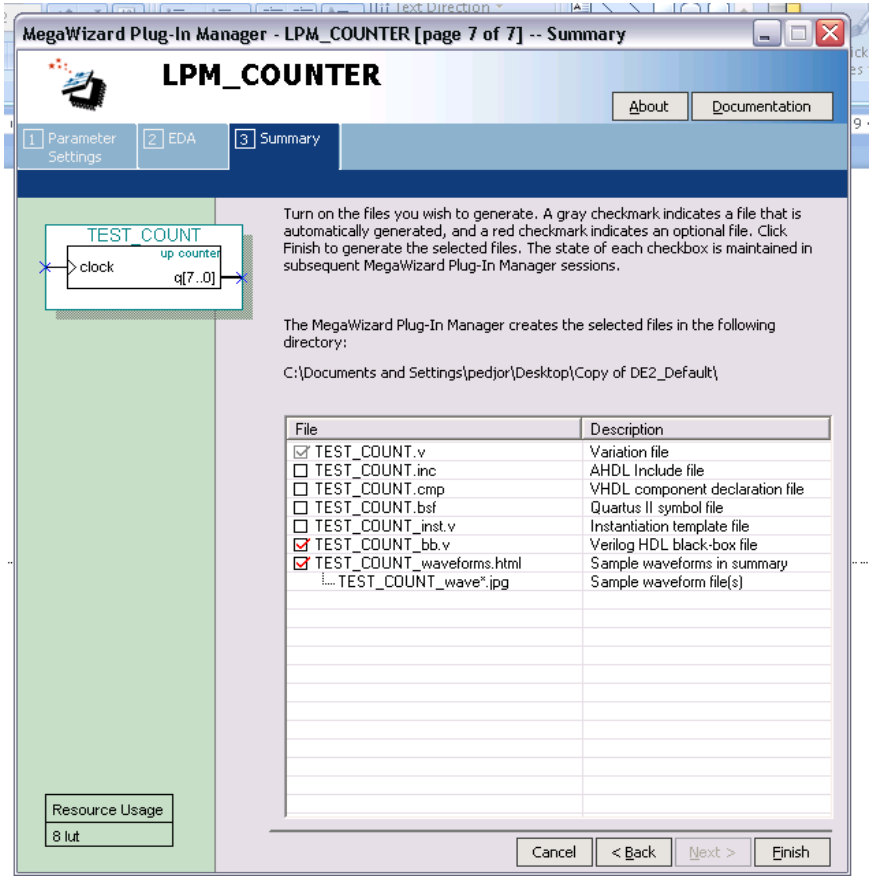

MegaWizard Plug-In Manager - LPM_COUNTER [page 3 of 7]
LPM_COUNTER
About
Documentation
1 Parameter Settings
2 EDA
3 Summary
General
General2
Optional Inputs
TEST_COUNT
up counter
clock
q[7..0]
Currently selected device family:
Stratix
Match project/default
How wide should the 'q' output bus be?
8
bits
What should the counter direction be?
Up only
Down only
Create an 'updown' input port to allow me to do both
(1 counts up; 0 counts down)
Resource Usage
8 lut
Cancel
< Back
Next >
Finish


MegaWizard Plug-In Manager - LPM_COUNTER [page 3 of 7]
LPM_COUNTER
About
Documentation
1 Parameter Settings
2 EDA
3 Summary
General
General2
Optional Inputs
TEST_COUNT
up counter
clock
q[7..0]
Currently selected device family:
Stratix
Match project/default
How wide should the 'q' output bus be?
8
bits
What should the counter direction be?
Up only
Down only
Create an 'updown' input port to allow me to do both
(1 counts up; 0 counts down)
Resource Usage
8 lut
Cancel
< Back
Next >
Finish

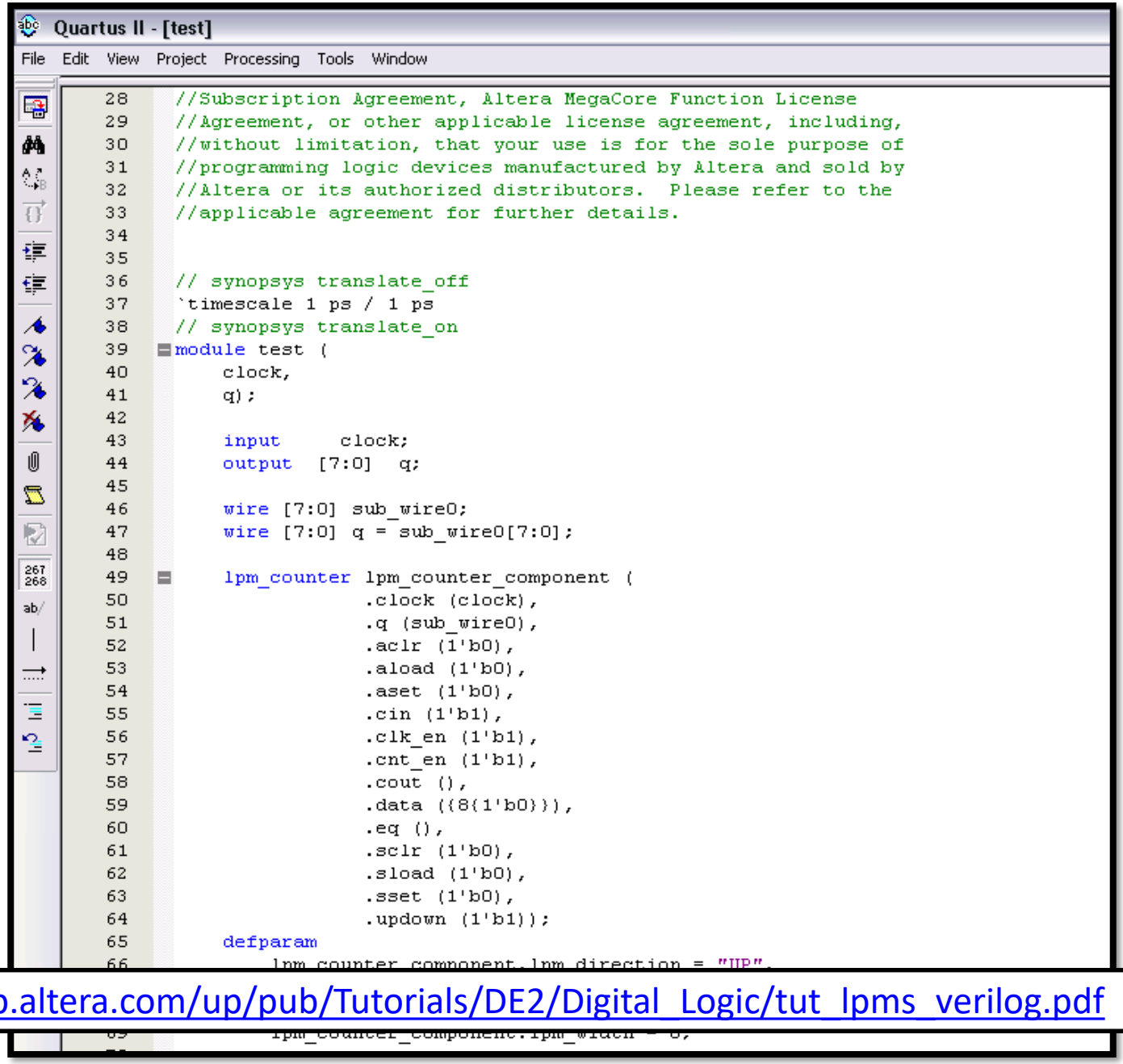

MegaWizard Plug-In Manager - LPM_COUNTER [page 4 of 7]
LPM_COUNTER
About
Documentation
1 Parameter Settings
2 EDA
3 Summary
General
General2
Optional Inputs
TEST_COUNT
up counter
clock
q[7..0]
Which type of counter do you want?
Plain binary
Modulus, with a count modulus of
Do you want any optional additional ports?
Clock Enable
Count Enable
Carry-in
Carry-out
Resource Usage
8 lut
Cancel
< Back
Next >
Finish


MegaWizard Plug-In Manager - LPM_COUNTER [page 5 of 7]
LPM_COUNTER
About
Documentation
1 Parameter Settings
2 EDA
3 Summary
General
General2
Optional Inputs
TEST_COUNT
up counter
clock
q[7..0]
Do you want any optional inputs?
Synchronous inputs
Clear
Load
Set
Set to all 1's
Set to
0
Asynchronous inputs
Clear
Load
Set
Set to all 1's
Set to
0
Resource Usage
8 lut
Cancel
< Back
Next >
Finish


MegaWizard Plug-In Manager - LPM_COUNTER [page 6 of 7] -- EDA
LPM_COUNTER
About
Documentation
1 Parameter Settings
2 EDA
3 Summary
TEST_COUNT
up counter
clock
q[7..0]
Simulation Libraries
To properly simulate the generated design files, the following simulation model file(s) are needed
File
Description
lpm
LPM megafunction simulation library
Timing and resource estimation
Generates a netlist for timing and resource estimation for this megafunction. If you are synthesizing your design with a third-party synthesis tool, using a timing and resource estimation netlist can allow for better design optimization.
Not all third-party synthesis tools support this feature - check with the tool vendor for complete support information.
Note: Netlist generation can be a time-intensive process. The size of the design and the speed of your system affect the time it takes for netlist generation to complete.
Generate netlist
Resource Usage
8 lut
Cancel
< Back
Next >
Finish


MegaWizard Plug-In Manager - LPM_COUNTER [page 7 of 7] -- Summary
LPM_COUNTER
About
Documentation
1 Parameter Settings
2 EDA
3 Summary
TEST_COUNT
up counter
clock
q[7..0]
Turn on the files you wish to generate. A gray checkmark indicates a file that is automatically generated, and a red checkmark indicates an optional file. Click Finish to generate the selected files. The state of each checkbox is maintained in subsequent MegaWizard Plug-In Manager sessions.
The MegaWizard Plug-In Manager creates the selected files in the following directory:
C:\Documents and Settings\pedjor\Desktop\Copy of DE2_Default\
File
Description
TEST_COUNT.v
Variation file
TEST_COUNT.inc
AHDL Include file
TEST_COUNT.cmp
VHDL component declaration file
TEST_COUNT.bsf
Quartus II symbol file
TEST_COUNT_inst.v
Instantiation template file
TEST_COUNT_bb.v
Verilog HDL black-box file
TEST_COUNT_waveforms.html
Sample waveforms in summary
TEST_COUNT_wave*.jpg
Sample waveform file(s)
Resource Usage
8 lut
Cancel
< Back
Next >
Finish

Mega Wizard – What you get



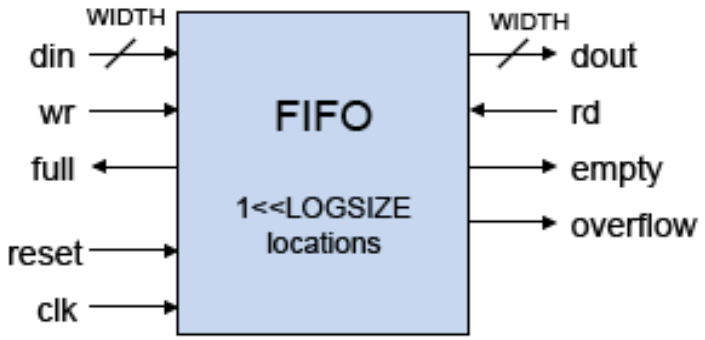
ftp://ftp.altera.com/up/pub/Tutorials/DE2/Digital_Logic/tut_lpms_verilog.pdf

Memories in Verilog

- `reg bit;  // a single register`
- `reg [31:0] word;   // a 32-bit register`
- `reg [31:0] array[15:0];  // 16 32-bit regs`

-
```
wire [31:0] read_data,write_data;
wire [3:0] index;

// combinational (asynch) read
assign read_data = array[index];

// clocked (synchronous) write
always @(posedge clock)
    array[index] <= write_data;
```

FIFOs




```
// a simple synchronous FIFO (first-in first-out) buffer
// Parameters:
//    LOGSIZE  (parameter) FIFO has 1<<LOGSIZE elements
//    WIDTH    (parameter) each element has WIDTH bits
// Ports:
//    clk      (input) all actions triggered on rising edge
//    reset    (input) synchronously empties fifo
//    din      (input, WIDTH bits) data to be stored
//    wr       (input) when asserted, store new data
//    full     (output) asserted when FIFO is full
//    dout     (output, WIDTH bits) data read from FIFO
//    rd       (input) when asserted, removes first element
//    empty    (output) asserted when fifo is empty
//    overflow (output) asserted when WR but no room, cleared on next RD
module fifo #(parameter LOGSIZE = 2,   // default size is 4 elements
                        WIDTH = 4)     // default width is 4 bits
            (input clk,reset,wr,rd, input [WIDTH-1:0] din,
             output full,empty,overflow, output [WIDTH-1:0] dout);
…
```

And finally… Microprocessors

Nios® II

Lots of tools and tutorials…
e.g. NIOS IDE (Integrated Development Environment)
DE2 demonstrations

Tools – SOPC builder

http://www.altera.com/education/demonstrations/sopc-builder/sopc-builder-demo.html

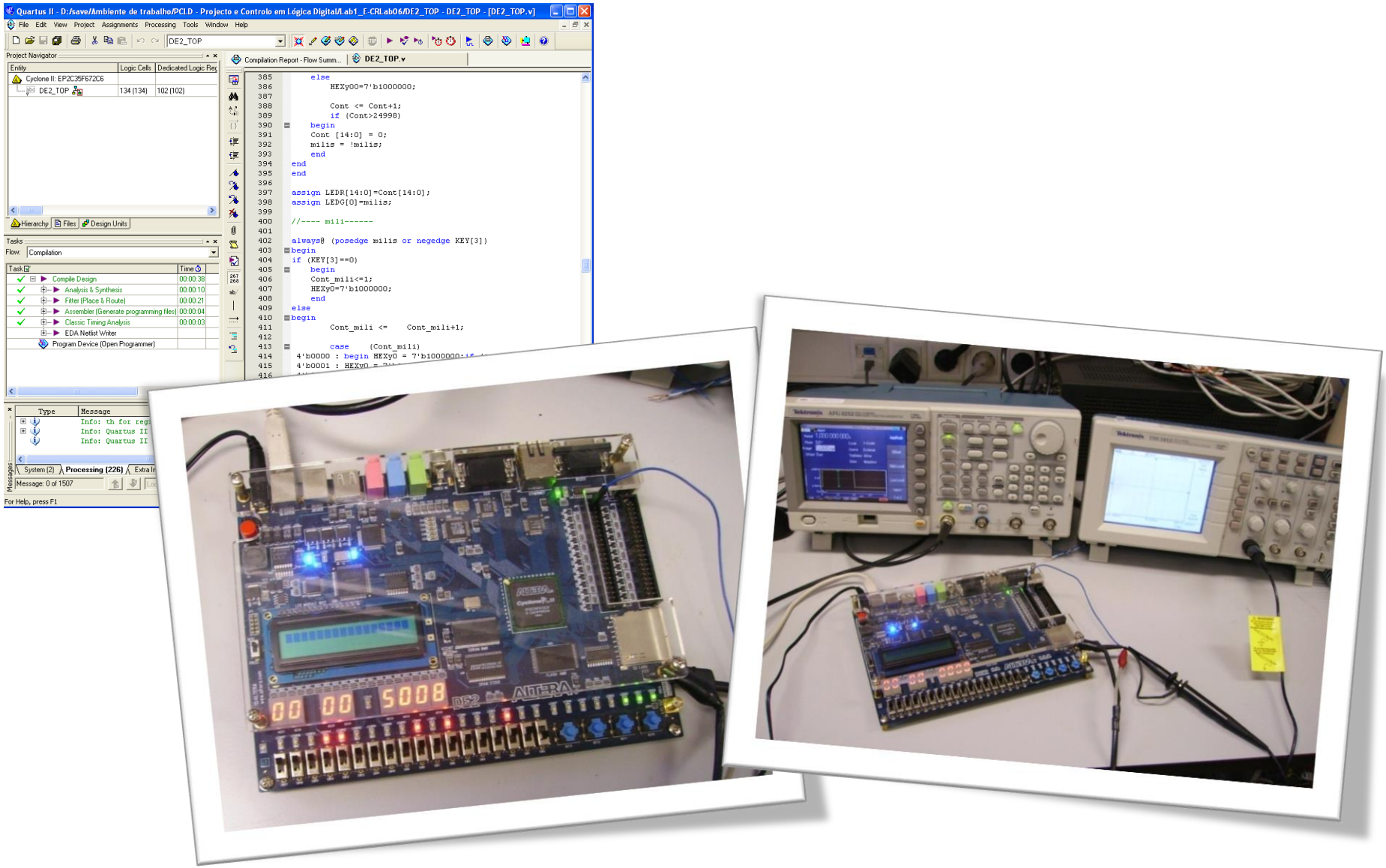
ftp://ftp.altera.com/up/pub/Tutorials/DE2/Computer_Organization/tut_sopc_introduction_verilog.pdf

The SOPC Builder is a tool used in conjuction with the Quartus II CAD software. It allows the user to easily create a system based on the Nios II processor, by simply selecting the desired functional units and specifying their parameters.

There are other choices of µ-processors to implements. E.g.: mi
Operating systems can be used. E.g. µ-Clinux™

The lab exercise

cronograph



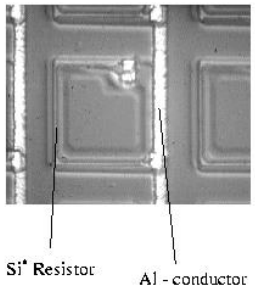
****

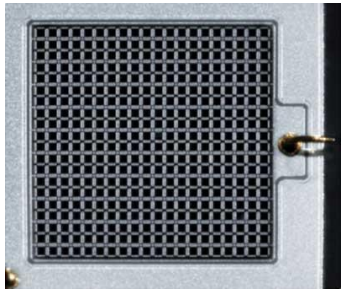
****

SiPMs …

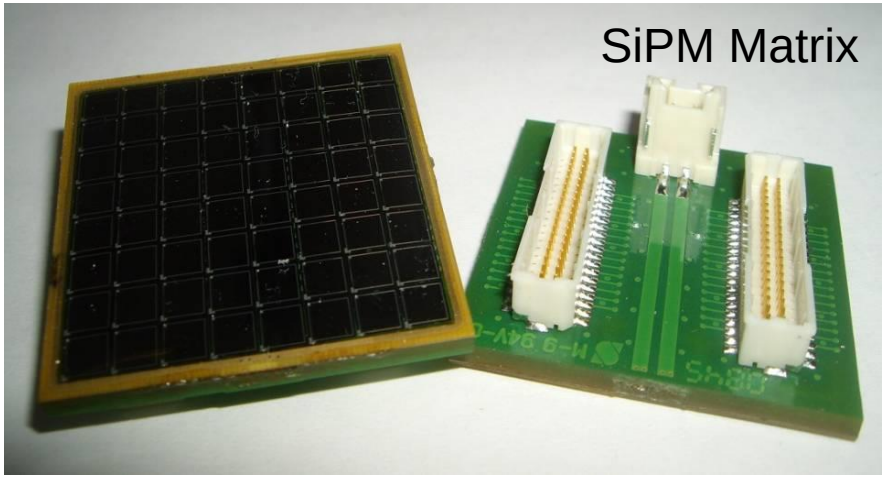
SiPM – Silicon PhotoMultiplier

Foto-Diodo de avalanche
em modo Geiger com
resistência Quenching Resitor





SiPM pixel

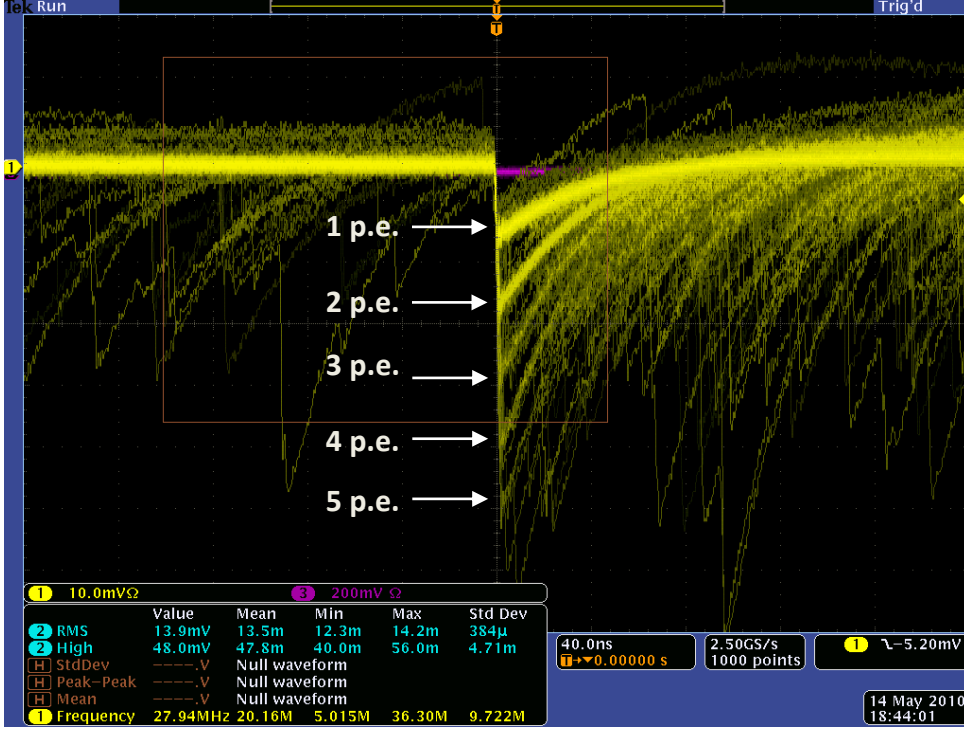


SiPMs – CMOS binário
1 célula tem sempre o mesmo sinal
Saída: Soma de várias células
Sinais “digitalizados”
Ideiais para “Single Photon Counting”
Eficientes
Podem ser expostos a luz
Tensões baixas (<100v)

Problemas:
Dependência com a temperatura;
Dependência com a tensão
Ruído
Crosstalk

No LIP…

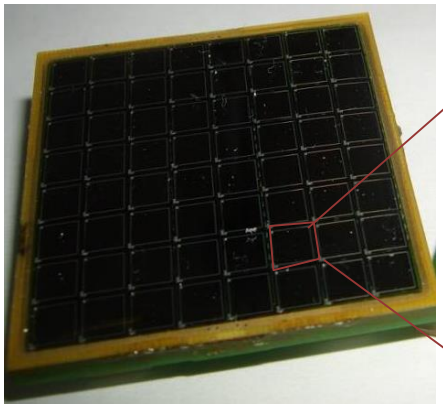
Caracterização dos SiPM
Sistema controlo temperatura (~ -20ºC)
Sistema leitura 64 canais

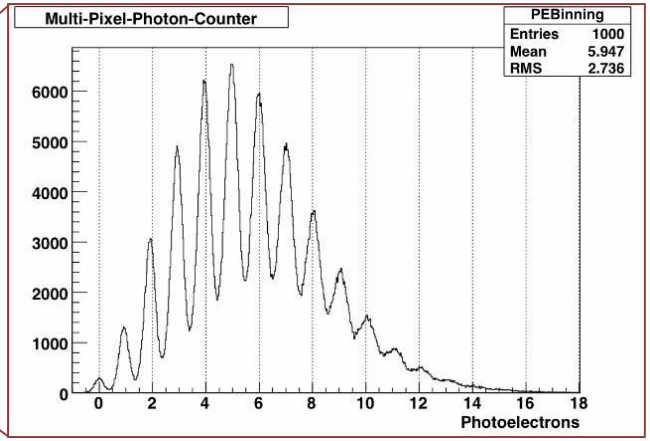


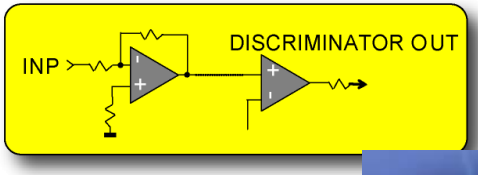
Substituir uma câmara de Auger:

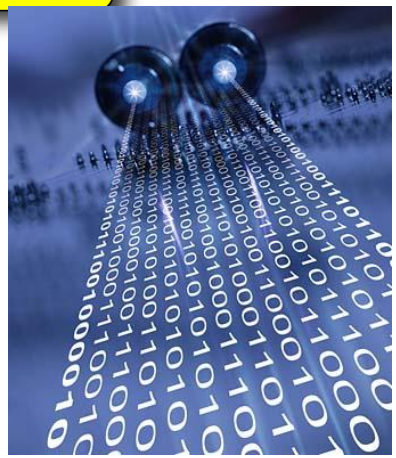
Auger: 800 mm x 800 mm = 6.4 x 10^5 mm^2
1 SiPM: 3 mm x 3 mm = 9 mm^2
Nº canais da ordem de 7 x 10^4
Ganhos:
Resolução
Eficiência do detector

Photon counting



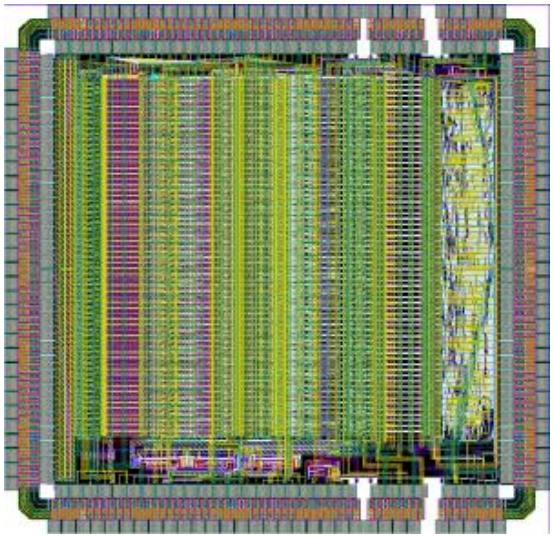




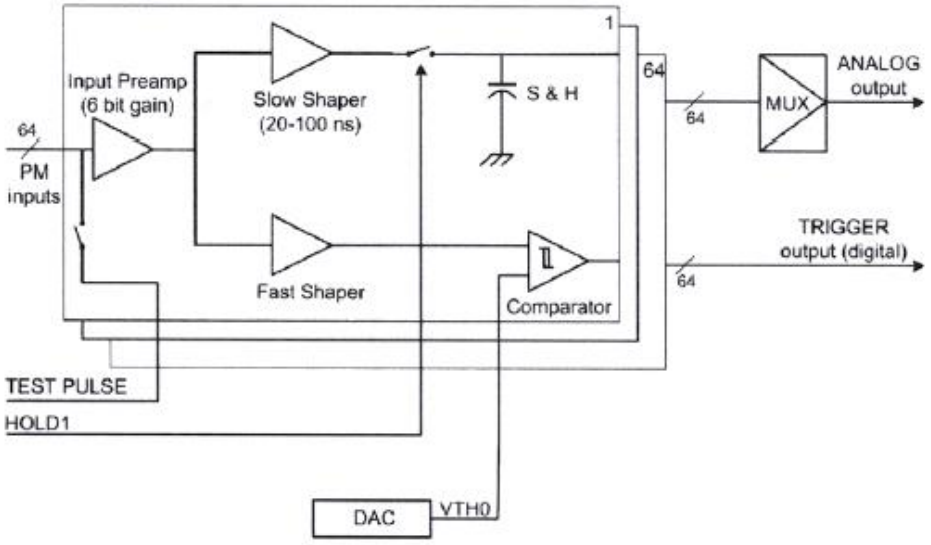


Threshold simples: Digital
Vários Thresholds: Aumentar gama dinâmica
Pode ser implementado num ASIC: Muitos Canais

O ASIC de front-end

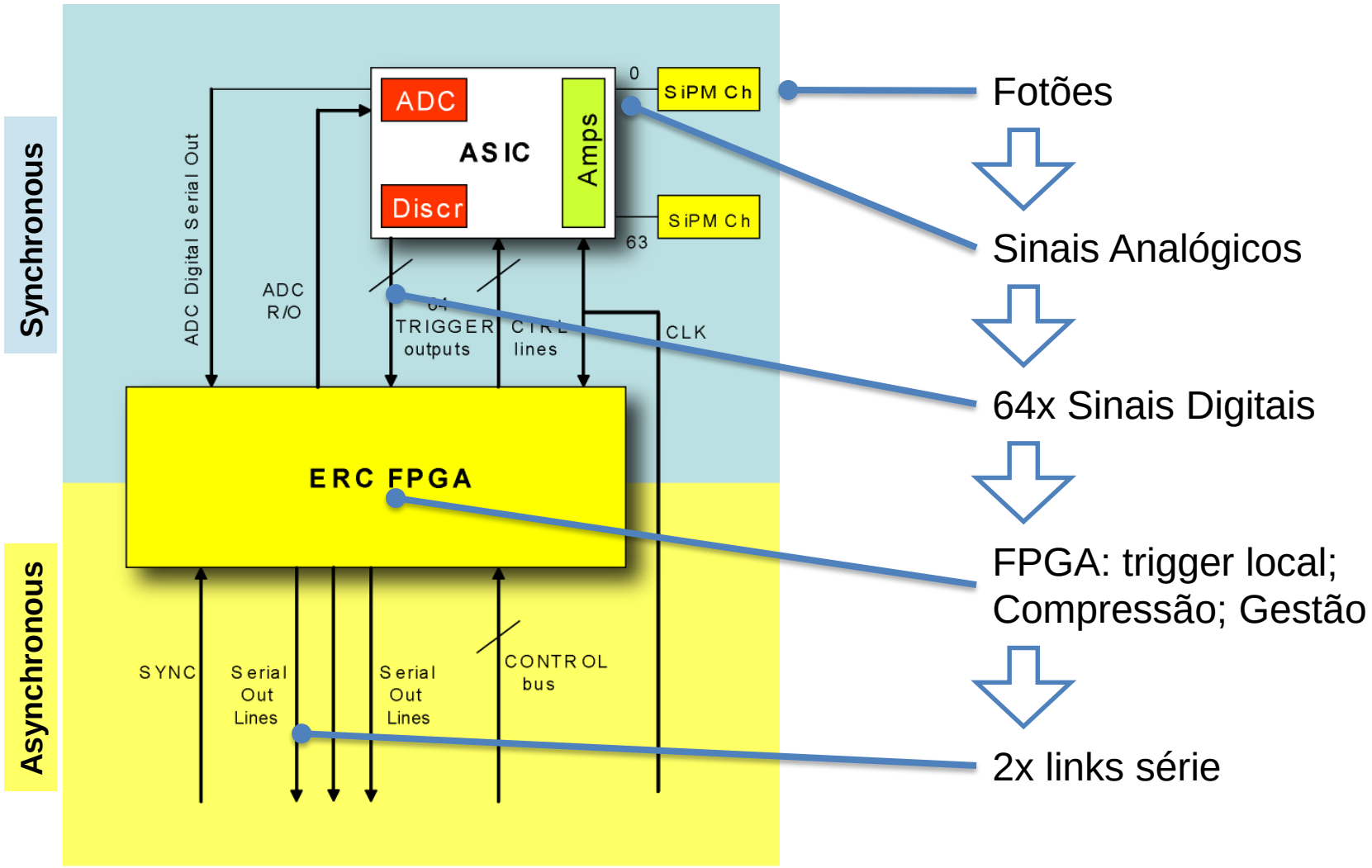


Baseline Option
ASIC MAROC3

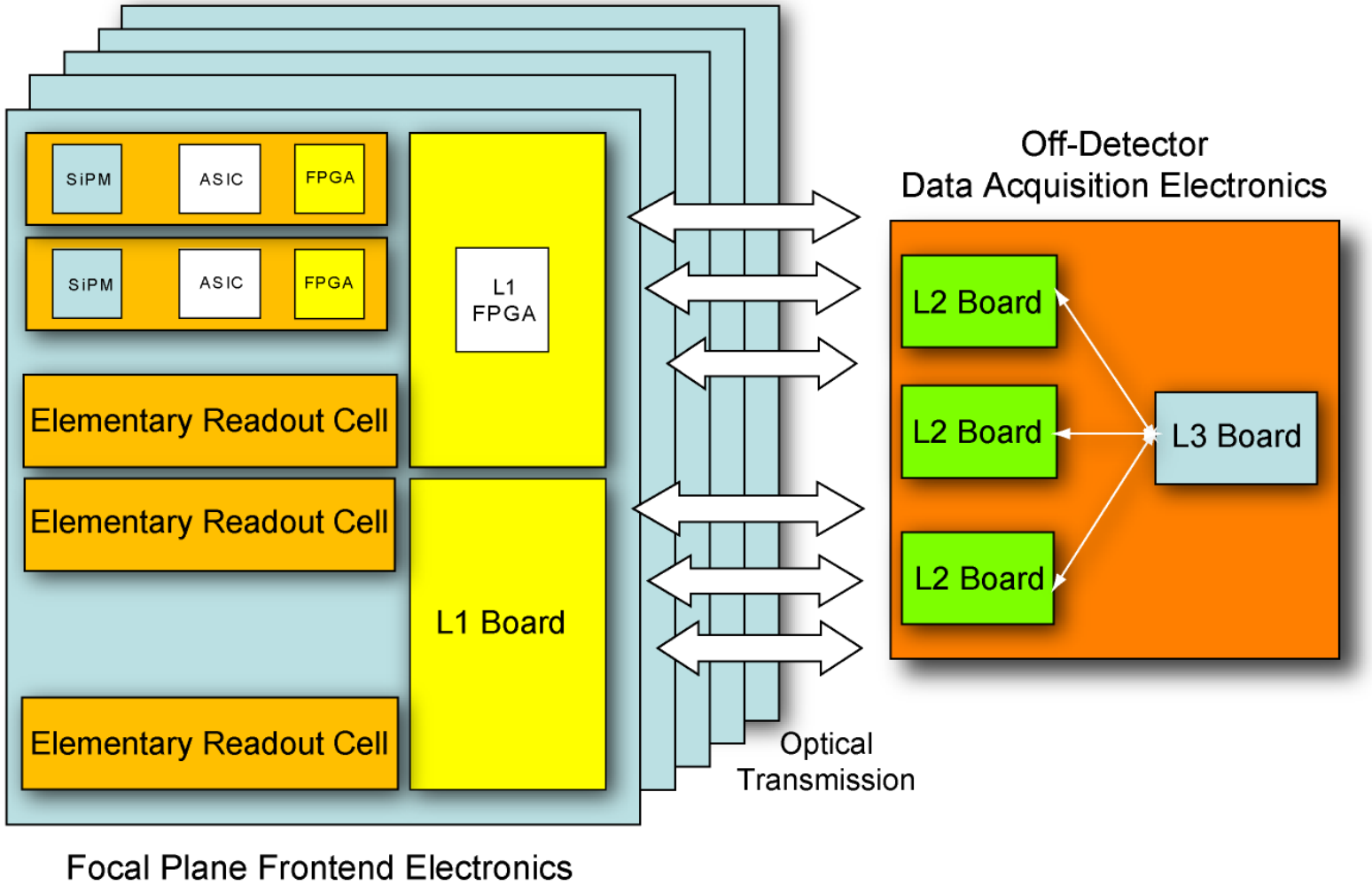


64 pré-amplificadores
Ganho programável por canal
64 sinais digitais de saída
ADC 12 bits

ERC - Elementary Readout Cell



Aquisição de Dados em pirâmide…



L2/L3 boards

Solução Comercial:
Placas MicroTCA (industria Telecomunicações)
Vários links opticos por placa
Podem ser instalados numa crate com bus
FPGAs Rápidas
Reprogramáveis

Altera® Stratix® IV GX FPGA
16 x 6.25 GHz SerDes transceivers
Optical Links (3.125 Gbps) or SFP+ (6.25 Gbps)
2 x 1 GB SDRAM

Scalable solution

PC Optical Link Board

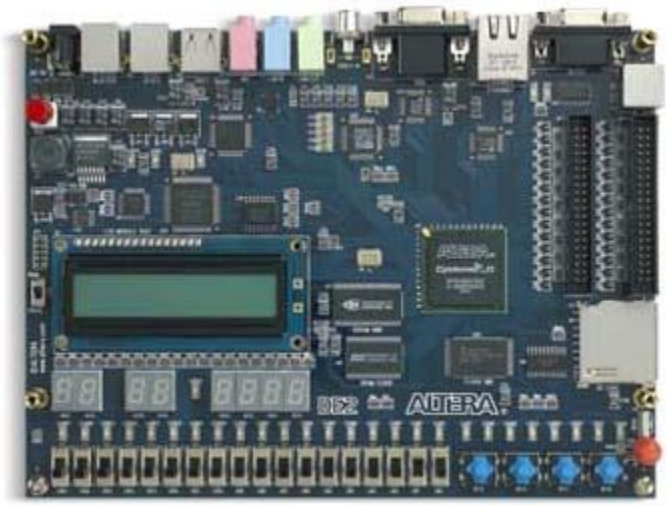
As Placas podem ser programadas como placas de trigger de nível L2 ou L3

Hardware

http://www.altera.com
http://www.terasic.com.tw/en/

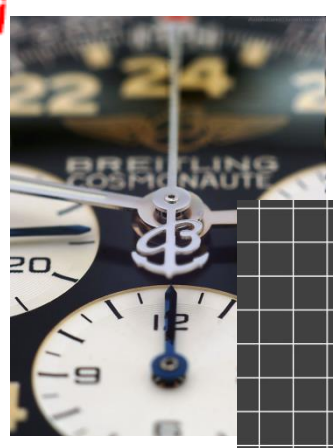
Development boards from ~$100

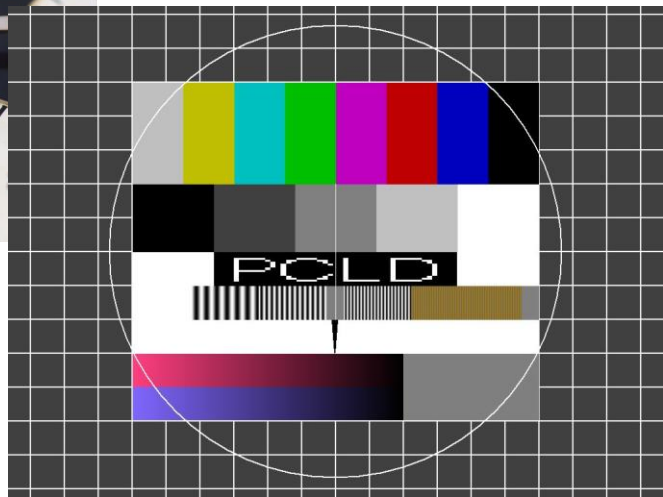
PCLD



3-4 semanas aulas
14 semanas laboratório

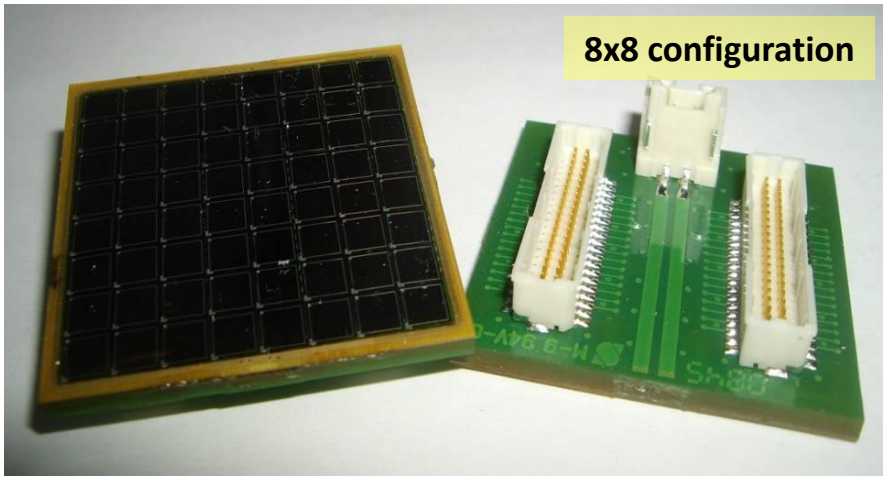
4 exercícios:
- "Olá Mundo"
- "Breitling"
- Mira Técnica
- Jogo





1 Trabalho final "negociado"
Caderno de encargos define requisitos

SiPM Multi-Pixel Arrays



Zecotek Photonics, MAPD3-N device

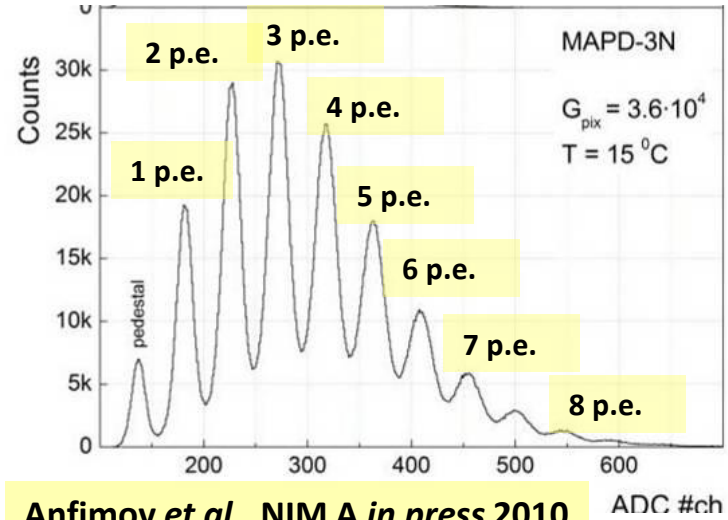
Gain: $7x10^4$ – $1x10^5$ (11-15 fC for 1 p.e.)

Dark current: 1-3 MHz per 3x3 mm^2 pixel

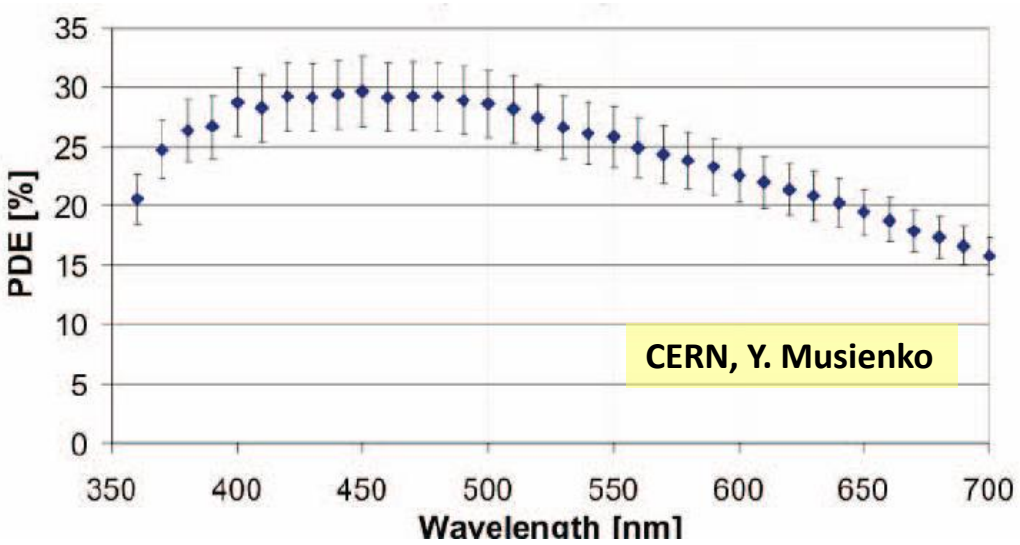
Temperature sensitivity (dM/dV) ~5%/ºC

Crosstalk: 5-15%

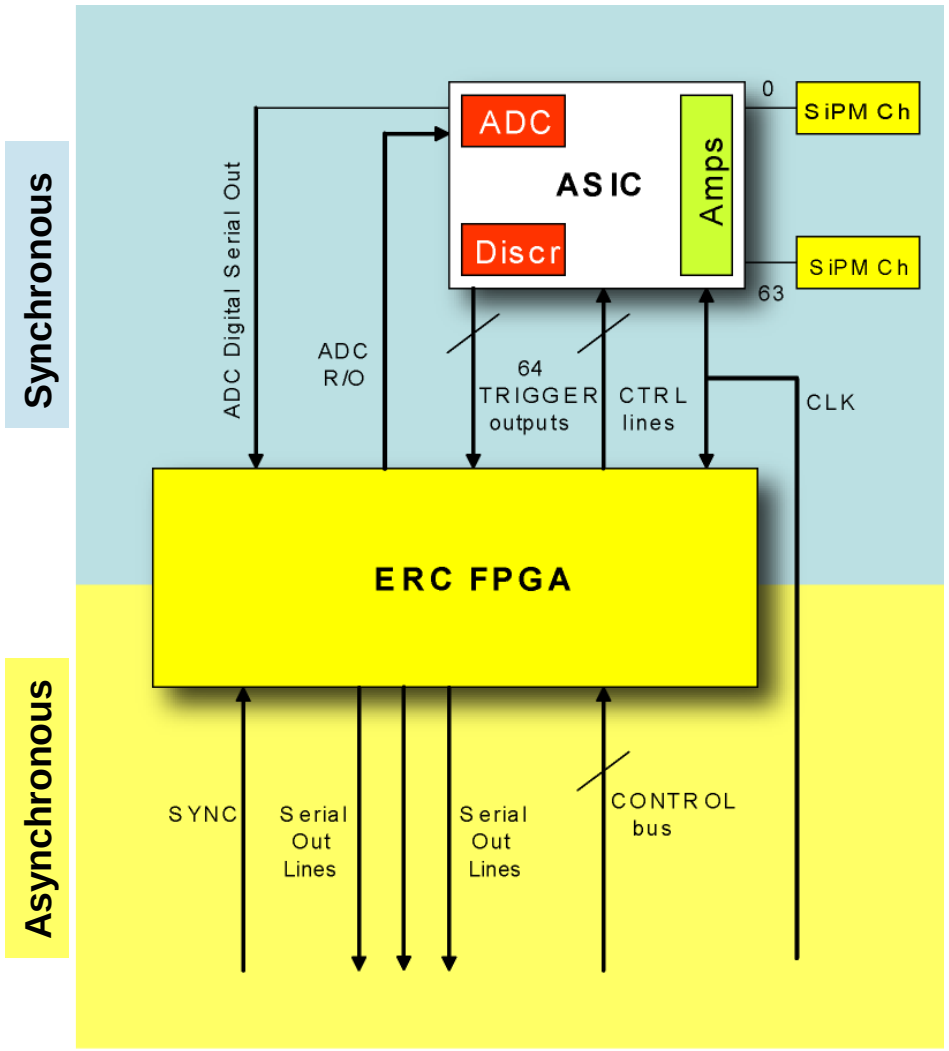
Bias supply: 90 V, common cathode



Anfimov *et al.*, NIM A *in press* 2010



ERC - Elementary Readout Cell



64 SiPM (8x8 configuration) array

64 input channel ASIC with 64 discriminators

Charge output available as option

Local clock at 40/80 MHz

Total bandwidth between ASIC and ERC FPGA: 2.56 Gbps at 40 MHz or 6.4 Gbps at 100 MHz

ERC FPGA performs zero suppression and local threshold

Output serial links (LVDS, min 100 MHz)

Typical payload: 64 bits + 40 bit Timestamp (clock counter) + overhead + trailer ~ 110-120b

2x100 Mbps LVDS link compatible with ~2 MHz event rate / 64 channels)

Main options

Signals Digitized and time-tagged "as soon as possible".
Data forward @ max bandwith using off-the-shelf links and boards.

- A clock is distributed only to front-end boards, for time marker assignment
- Each trigger primitive has a time marker assigned by the front-end board
- All trigger primitives from front-end boards sent asynchronously to trigger boards via data links running at their own clocks, for maximum bandwidth
- In an asynchronous system, each trigger level works at its own clock frequency
- No timing procedures are required

From Industry: Fast Data transfer standards. E.g.

- xTCA from Telecom Industry
- Inherent asynchronous (network switching)
- Data and slow-control data streams can be send over the same data lines, reducing the number of cables required

Bonus: DAQ architecture can re-use standard MA-PMTs