

# Handouts

## FPGA-related documents

1. ***Introduction to Verilog***, P. M. Nyasulu and J. Knight, Carleton University, 2003 (Ottawa, Canada).
2. ***Quick Reference for Verilog HDL***, R. Madhavan, AMBIT Design Systems, Inc, Automata Publishing Company, 1995 (San Jose, CA).

## Project-related documents

3. ***Project Guidelines*** and ***Project Specifications***.  
*(I'll hand these out in lab)*

# Introduction to FPGAs

## **Outline:**

### **1. What's an FPGA ?**

→ logic element “fabric”, i.e. logic gates + memory + clock trigger handling.

### **2. What's so good about FPGAs ?**

→ FPGA applications and capabilities  
→ FPGAs for physicists

### **3. How do you program an FPGA ?**

→ Intro to *Quartus II*  
→ Schematic design  
→ Verilog HDL design

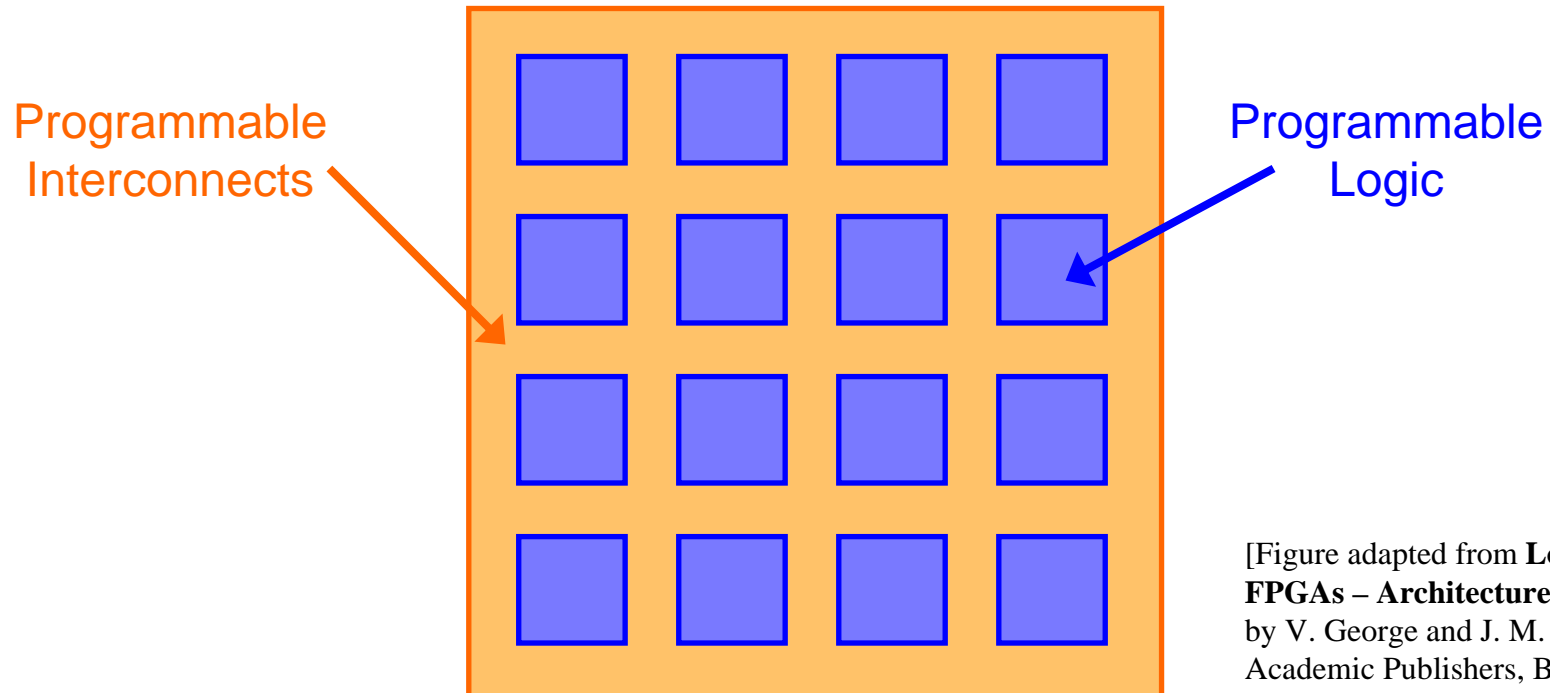
# What's an FPGA

**An FPGA is:**

- a Field Programmable Gate Array.
- a programmable breadboard for digital circuits on chip.

The FPGA consists of:

- programmable **Logic Elements** (LEs).
- programmable **interconnects**.
- custom circuitry (i.e. multipliers, phase-lock loops (PLL), memory, etc ...).



[Figure adapted from **Low Energy FPGAs – Architecture and Design**, by V. George and J. M. Rabaey, Kluwer Academic Publishers, Boston (2001).]

# Logic Element (LE)

An FPGA consists of a giant array of interconnected **logic elements (LEs)**. The LEs are identical and consist of **inputs**, a **Look-Up Table (LUT)**, a little bit of **memory**, some **clock** trigger handling circuitry, and **output** wires.

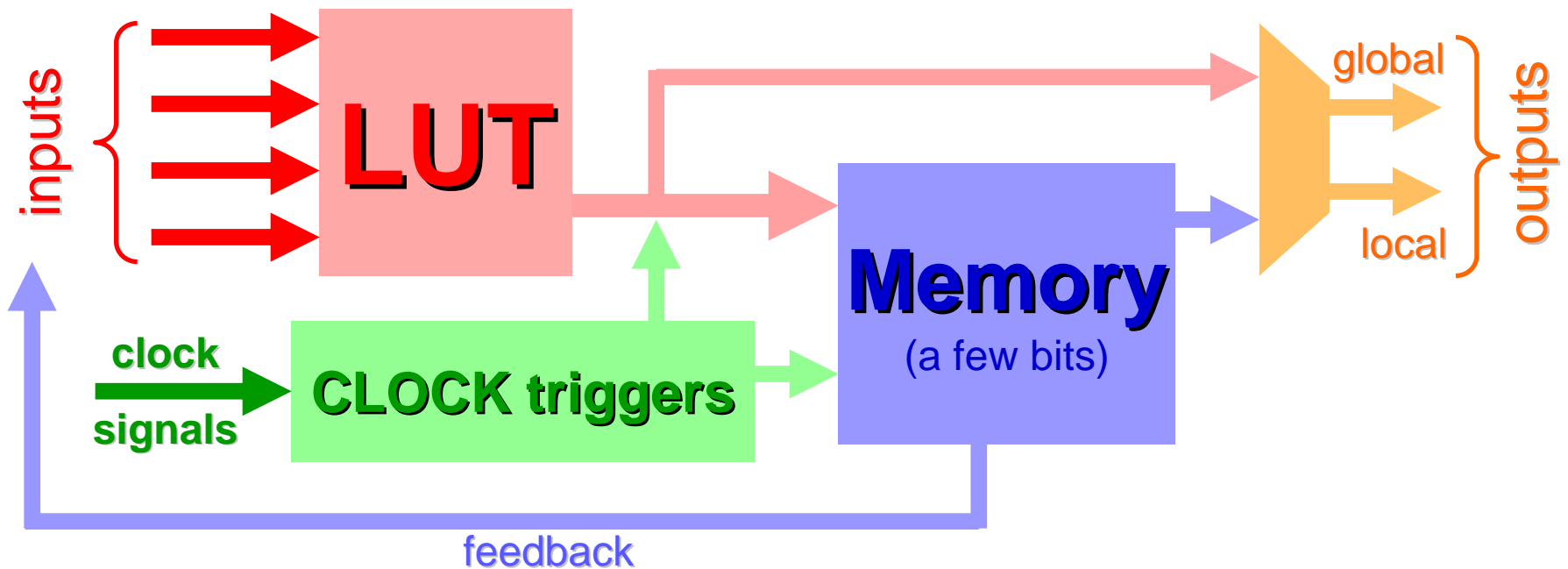
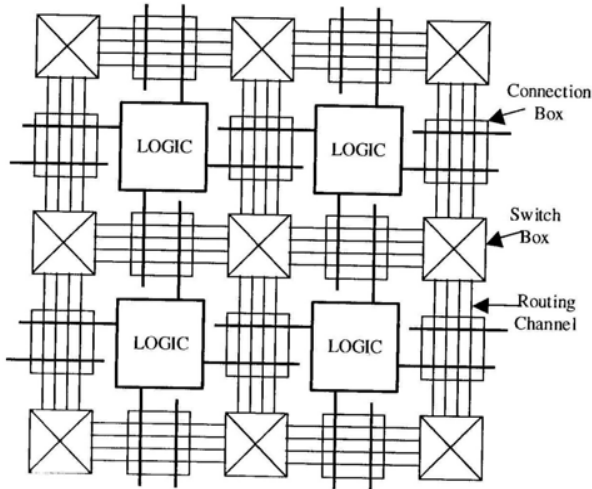
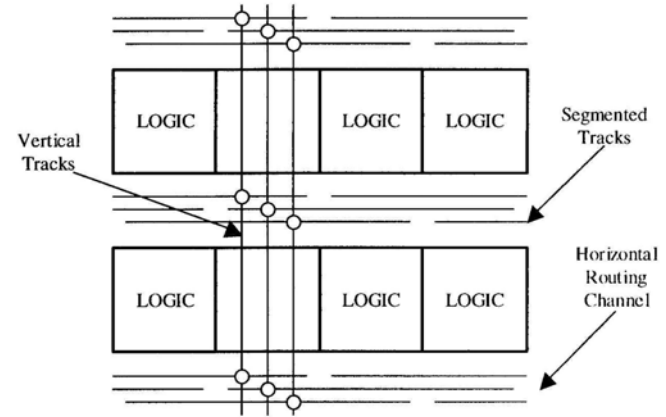


Figure: Architecture of a single Logic Element

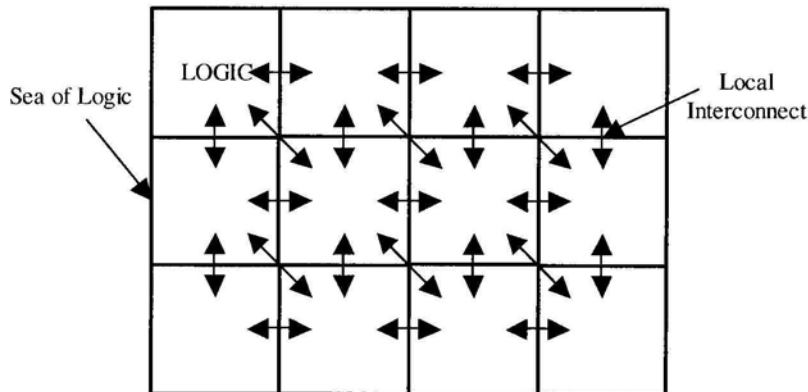
# Interconnect Architectures



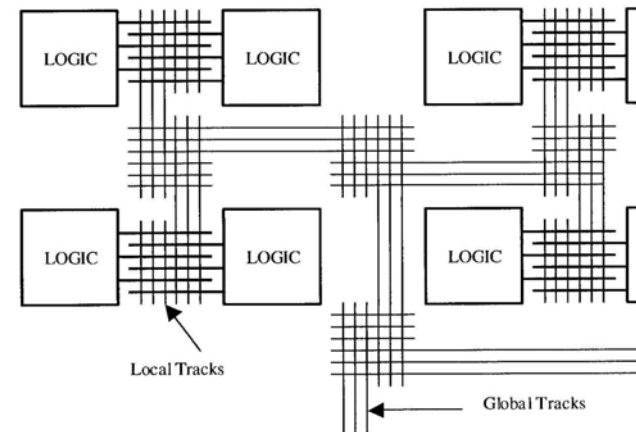
Island Style Architecture



Row-Column Architecture



Sea-of-Gates Architecture



Hierarchical Architecture

# FPGA devices (I)

## 2 primary manufacturers:

1. **Xilinx** (founded by Ross Freeman, original inventor of FPGAs in 1984).
2. **Altera**: we will use an Altera Cyclone II FPGA and associated design software, Quartus II, in the course.

## Other “specialty” FPGA manufacturers:

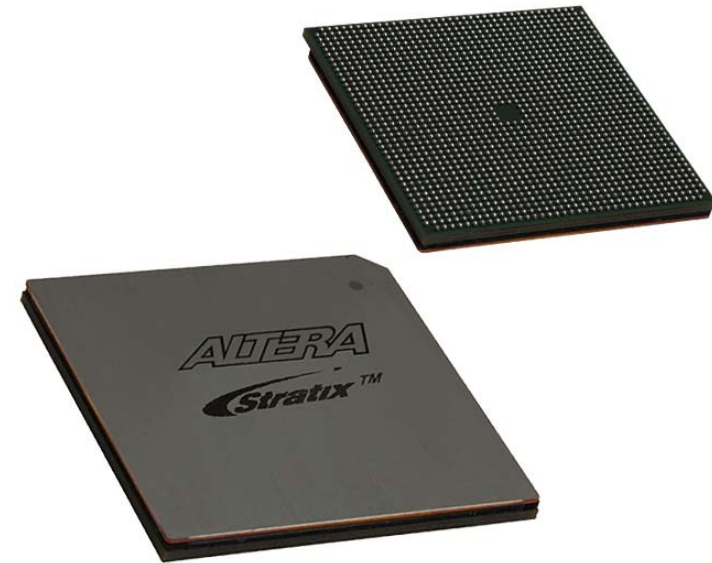
Achronix, Actel, Atmel, Cypress, Lattice Semiconductor, MathStar, QuickLogic, etc ...



100-pin TQFP package  
[[www.digikey.com](http://www.digikey.com)]



240-pin PQFP package  
[[www.21control.com](http://www.21control.com)]



1508-pin BGA package  
[[www.digikey.com](http://www.digikey.com)]

# FPGA devices (II)

- 4k – 300k LEs.
- Clock speeds: 100-500 MHz (1 GHz in the works).
  - Most FPGA circuit implementations will run slower than the maximum clockspeed.
- Memory: 10 Kbytes – 10 Mbytes.
- i/o pins: up to 1200.
- Price range: \$10 - \$7k



[figure from [www.fys.uio.no](http://www.fys.uio.no)]

# Applications

## ➤ **Low-cost customizable digital circuitry**

- Can be used to make any type of digital circuit.
- Rapid product development with design software.
- Upgradable.
- Sort of like “soft-hardware” [R. G. Shoup].

## ➤ **High-performance computing**

- Complex algorithms are off-loaded to an FPGA co-processor.
- Application-specific hardware.
- FPGAs are inherently parallel and can have very efficient hardware algorithms: typical speed increase is x10 - x100.

## ➤ **Evolvable hardware**

- Hardware can change its own circuitry.
- Neural Networks.

## ➤ **Digital Signal Processing**

- Reconfigurable DSP hardware.
- In principle, DSP can simulate any analog circuit in combination with DACs and ADCs (still requires amplifiers, though).



# FPGAs for physicists

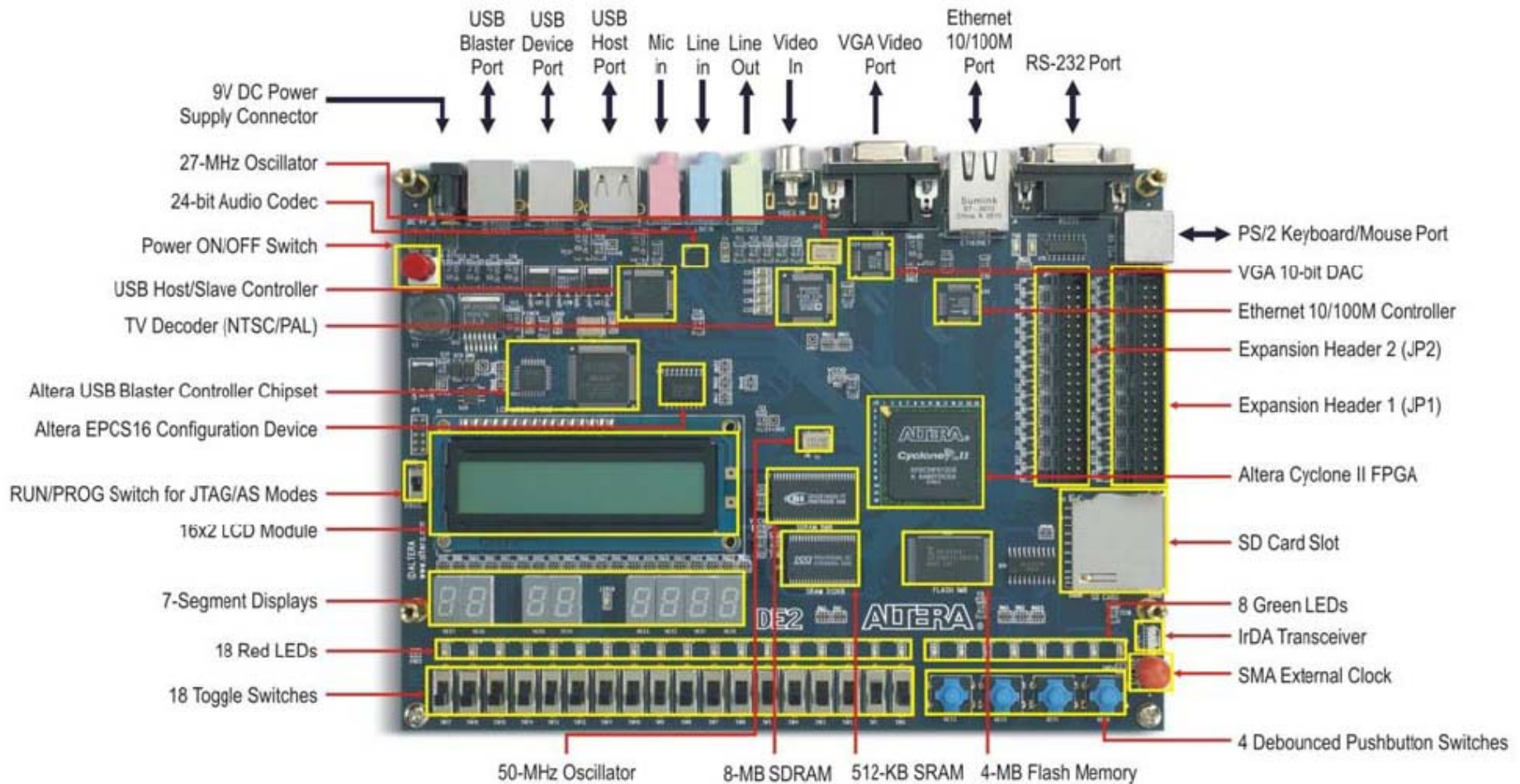
Physicists use FPGAs in the following applications:

- Coincidence triggering (particle physics & quantum optics).
- DSP circuits.
- Specialty filters.
- Customizable feedback loops (Atomic Physics).
- Lock-in amplifiers (Atomic and Solid State Physics).
- Multi-channel analyzers (Particle, Nuclear, & Atomic Physics).
- etc ...

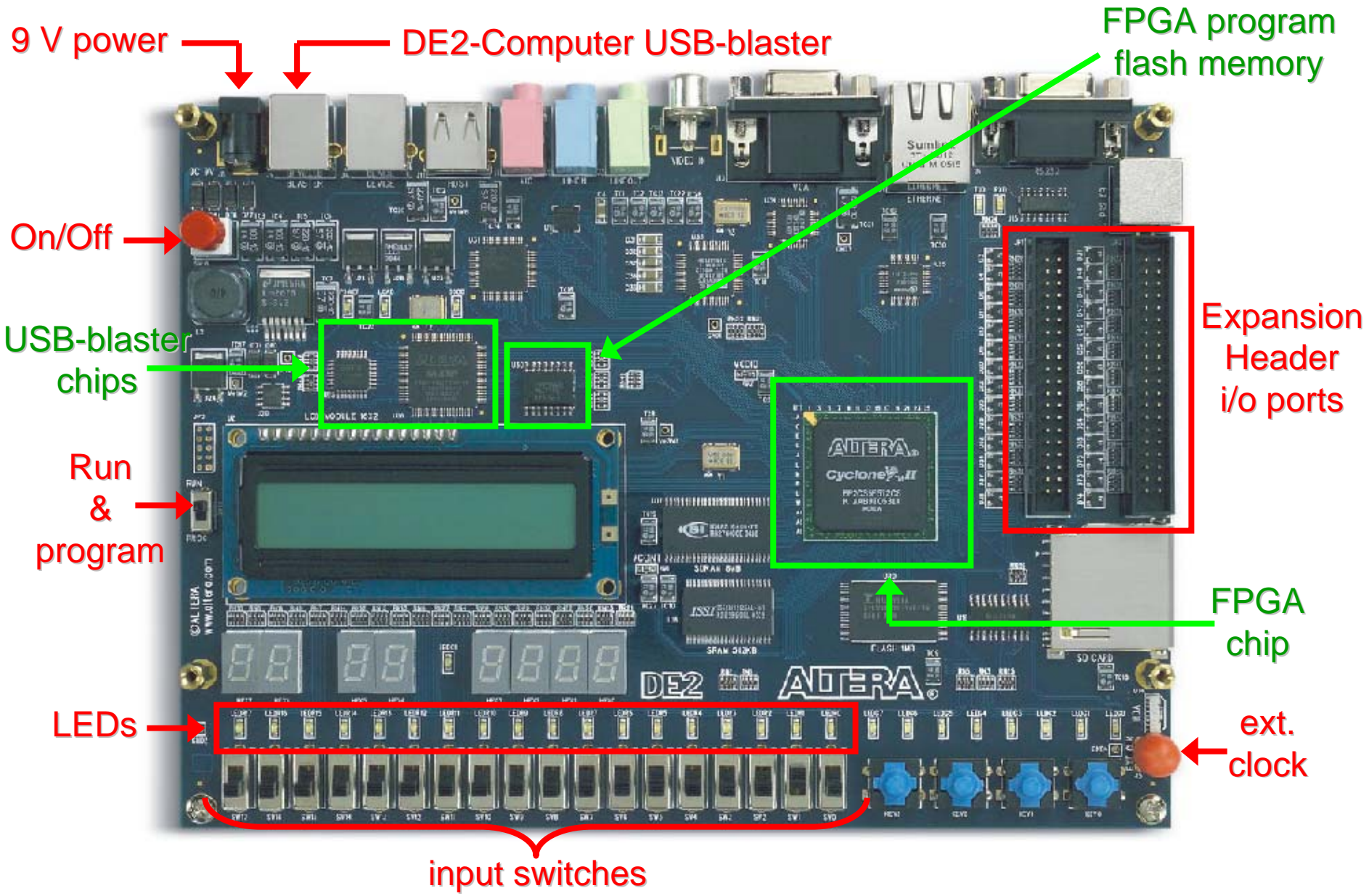
**Digital Advantage:** *Once a signal is digital, processing does not add any noise (the ADCs and DACs do add noise).*

# The DE2 development board

We will use the DE2 development board from Altera for all of the FPGA labs



# DE2: important stuff





# DE2: important specs

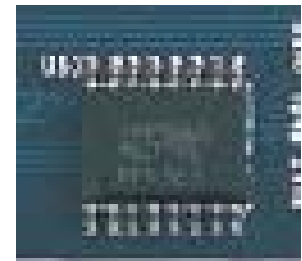
## FPGA chip:

- Cyclone II: EP2C35F672C6N
- 33,216 LEs.
- 60 Kbytes of on-chip memory.
- 35 18-bit x 18-bit multipliers.
- 4 Phase Lock Loops (PLLs).
- ~260 MHz DSP speed.
- 90 nm technology.
- 475 i/o lines.
- 672 BGA package.



## FPGA configuration chip:

- EPCS16 configuration device.
- 2.1 Mbytes of Flash memory.
- Stores the FPGA circuitry program when DE2 is off.
- Used for Active Serial (AS) programming.

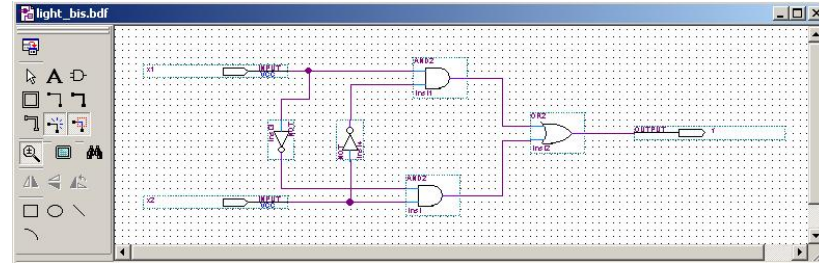


# FPGA programming



QUARTUS<sup>II</sup>

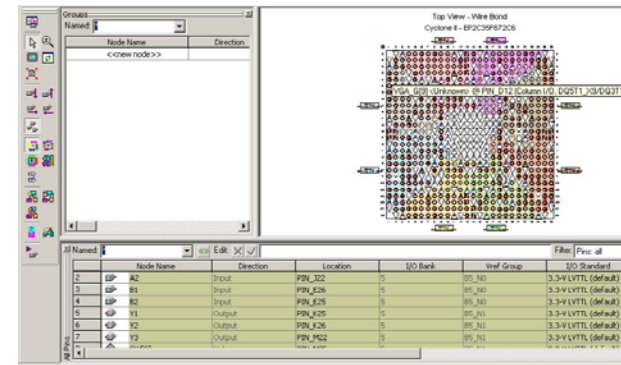
1. Start project in Quartus II.
2. Enter design via Schematic file or Verilog HDL program.



### 3. Compile.

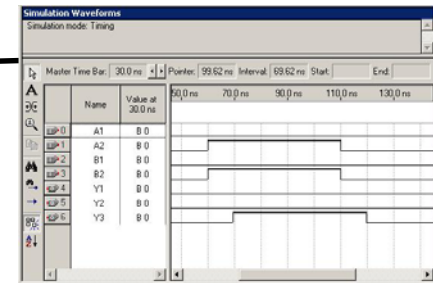
### 4. Check compilation in Technology Viewer.

5. Assign input and output variables to actual i/o pins.



### 6. Compile.

7. Simulate the circuit.



8. Load circuit into FPGA.



9. Test circuit.

# Verilog HDL

We will use Verilog HDL (Hardware Description Language) to program the FPGA.

(not to be confused with VHDL, another FPGA language)

A Verilog program describes how the LEs are configured and connected.  
*This is different from a regular program which is a series of sequential instructions to the CPU and some memory handling.*

- Advantages:**
- Sort of like *C programming*.
  - *You don't have to figure out the exact circuitry.*  
(the compiler does it for you)
  - *Easier and faster* to make more complex circuit designs.
  - You can use a vast programming libraries (**IP cores**).

**IMPORTANT:** Always comment your Verilog code.

# Verilog program

2 input 1-bit adder:

```
TwoInput1bitAdder_Verilog.v | Compilation Report - Flow Summary
1  module TwoInput1bitAdder_Verilog(a,b,result); // Note: Name of module should be the same as name of file.
2                                          // Note: Name of module/file should not start with a number.
3      input a,b; // declare input variables
4      output [1:0] result; // declare output variable 2-bit array (little endian format)
5
6      assign result[0] = a ^ b; // XOR operation for 1-bit addition
7      assign result[1] = a & b; // AND operation for 1-bit addition carry
8
9  endmodule // end of module command
10 // add this extra line so endmodule turns blue
11 // (otherwise code will not compile)
```

# Verilog program

2 input 1-bit adder:

```
TwoInput1bitAdder_Verilog.v | Compilation Report - Flow Summary
1  module TwoInput1bitAdder_Verilog(a,b,result);
2
3      input a,b;
4      output [1:0] result;
5
6      assign result[0] = a ^ b;
7      assign result[1] = a & b;
8
9  endmodule
10
11
```

```
// Note: Name of module should be the same as name of file.
// Note: Name of module/file should not start with a number.
// declare input variables
// declare output variable 2-bit array (little endian format)
// XOR operation for 1-bit addition
// AND operation for 1-bit addition carry
// end of module command
// add this extra line so endmodule turns blue
// (otherwise code will not compile)
```

comments

little-endian binary bit array (i.e. binary number)

assign: hardwires the input to the output.



# Verilog program

2 input 1-bit adder:

```
TwoInput1bitAdder_Verilog.v | Compilation Report - Flow Summary
1  module TwoInput1bitAdder_Verilog(a,b,result);
2
3      input a,b;
4      output [1:0] result;
5
6      assign result[0] = a ^ b;
7      assign result[1] = a & b;
8
9  endmodule
10
11
```

```
// Note: Name of module should be the same as name of file.
// Note: Name of module/file should not start with a number.
// declare input variables
// declare output variable 2-bit array (little endian format)
// XOR operation for 1-bit addition
// AND operation for 1-bit addition carry
// end of module command
// add this extra line so endmodule turns blue
// (otherwise code will not compile)
```

comments

little-endian binary bit array (i.e. binary number)

assign: hardwires the input to the output.

Same thing, but easier:

```
TwoInput1bitAdder_Verilog.v | Compilation Report - Flow Summary
1  module TwoInput1bitAdder_Verilog(a,b,result);
2
3      input a,b;
4      output [1:0] result;
5
6      assign result = a + b;
7
8  endmodule
9
10
```

```
// Note: Name of module should be the same as name of file.
// Note: Name of module/file should not start with a number.
// declare input variables
// declare output variable 2-bit array (little endian format)
// binary addition
// end of module command
// add this extra line so endmodule turns blue
// (otherwise code will not compile)
```

# Some Verilog tid-bits

```
Input [2:0] input1;           // 3-bit input array in little-endian format.  
Input [0:2] input2;         // 3-bit input array in big-endian format.
```

## Set input1 equal to “6” in binary.

```
Assign input1[0] = 0;  
Assign input1[1] = 1;  
Assign input1[2] = 1;
```

or

```
Assign input1 = 3'b110;
```

Set number width

Actual number

Set number type	
b=binary	d=decimal
O=octal	h=hexadecimal

## Set input2 equal to “6” in binary.

```
Assign input1[0] = 1;  
Assign input1[1] = 1;  
Assign input1[2] = 0;
```

or

```
Assign input1 = 3'b011;
```