Reconfigurable Systems and their Influence on Mobile and Multimedia Applications

Valery Sklyarov, Iouliia Skliarova

University of Aveiro, Portugal
Department of Electronics, Telecommunications and Informatics / IEETA
Part 1. Origin, History and Evolution of Reconfigurable Systems

Part 2. Design, Implementation, Practical Applications and Future of Reconfigurable Systems

Part 3. Reconfigurable Systems in Engineering, Research and Education
Origin and History of Reconfigurable Systems and Programmable Logic Devices
The earliest work on reconfigurable computer architecture was done at the University of California at Los Angeles (UCLA)

**Fixed + Variable structure computer**

- A standard processor $F$ augmented by an array of reconfigurable hardware $V$
- A new way to evolve higher performance computing from any general purpose computer
- Flexibility of software + the speed of hardware


<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1947</td>
<td>Shockley, et al. – the first transistor at Bell Labs;</td>
</tr>
<tr>
<td>1958</td>
<td>Jack Kilby – the first integrated circuit;</td>
</tr>
<tr>
<td>1962</td>
<td>Hofstein, et al. - metal-oxide semiconductor field-effect transistor (MOSFET);</td>
</tr>
<tr>
<td>1970</td>
<td>Intel – the first 1024-bit DRAM;</td>
</tr>
<tr>
<td>1970</td>
<td>Fairchild – the first 256-bit SRAM;</td>
</tr>
<tr>
<td>1970</td>
<td>Ron Cline (Signetics) – the first PLD (PLA);</td>
</tr>
<tr>
<td>1971</td>
<td>Intel – the first microprocessor, 4004;</td>
</tr>
<tr>
<td>1978</td>
<td>Monolithic Memories Inc. – the first PAL;</td>
</tr>
<tr>
<td>1984</td>
<td>Altera – the first CPLD based on a combination of CMOS and EPROM technologies;</td>
</tr>
<tr>
<td>1985</td>
<td>Xilinx - the first FPGA, the XC2064™ - a radical new form of programmable logic;</td>
</tr>
<tr>
<td>1991</td>
<td>The world's first commercial reconfigurable computer, the Algotronix CHS2X4</td>
</tr>
</tbody>
</table>
FPGA and PLD market

The market for FPGAs and other PLD is expected to grow from $3.2 billion in 2005 to $6.7 billion in 2010, according to Gartner Dataquest [1].

The FPGA/PLD market is dominated by Xilinx Inc. and Altera Corp. (84% of the market in 2005 [1]).

Gartner's numbers contrast with those of In-Stat predicting the FPGA market to grow from $1.9 billion in 2005 to $2.75 billion in 2010 [1].

In-Stat's forecast covers only FPGAs, while the Garter forecast includes other PLDs.

Some interesting facts from [1]:

→ The largest FPGA consumers will be communications and industrial, whose combine market share of the FPGA market will increase from 73.8% in 2005, to 76.8% by 2010.

→ FPGAs find numerous applications in the scope of electronic system design, from glue logic to high-complexity application specific (ASIC-type) devices.

→ Pioneering products such as Xilinx’s Virtex or Altera’s Stratix families will find their main applications in the development of high-volume products.

Top PLD manufactures (2004) [1]

Xilinx – 51%, Altera – 32%, Lattice – 8%, Actel – 6%, QuickLogic – 2%, Others – 1%.

Top FPGA manufacturers (2004) [1]

Xilinx – 59%, All others – 41%.

Xilinx history

1985. Xilinx introduces its first product, the FPGA XC2064™.
1991. The XC4000™ family of FPGAs is introduced.
1999. New low power and lower cost CPLD products.
2001. Virtex™-II family – the first platform FPGA.
2003. Spartan™-3 family of products is introduced - the world's first 90nm FPGA.
2004. Virtex™-4 family of products is introduced (multi-platform FPGA family).
2006. Virtex™-5 family of products is introduced - the world's first 65nm FPGA.

In January, 2003, Xilinx was named number four on Fortune magazine's "100 Best Companies to Work For in America" list.

Xilinx Inc. took top honors for the third consecutive year in the FPGA EDA tool vendor survey conducted by EE Times in 2006 [3].

1. History of Xilinx. Available at: http://direct.xilinx.com/company/history.htm
Moore’s law has allowed vendors to produce new FPGAs every two years with generally double the density, better performance, and improved features” [4]

Evolution of Reconfigurable Systems
Outline

• Programmable Logic Devices - PLD;
• Field Programmable Gate Arrays - FPGA;
• FPGA architectures;
• Platform FPGA;
• Multi-platform FPGA;
• System-on-chip;
• Network-on-chip;
• Reconfigurable computing;
• Reconfigurable systems.
PLD

SPLD

PROM

PLA

PAL

CPLD

Some others
<table>
<thead>
<tr>
<th>a b c d</th>
<th>y_1 y_2 y_3 y_4 y_5 y_6 y_7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 1 1 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>1 1 1 1 0 0 1</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1 0 1 1 0 1 1</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>1 0 1 1 1 1 1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1 1 1 0 0 0 0</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 1 1 1 0 1 1</td>
</tr>
</tbody>
</table>

\[ y_5 = \overline{a} \overline{b} \overline{c} \overline{d} \lor \overline{a} \overline{b} c \overline{d} \lor \overline{a} b c \overline{d} \lor a \overline{b} \overline{c} \overline{d}; \]

4 variables
<table>
<thead>
<tr>
<th>a b c d</th>
<th>y1 y2 y3 y4 y5 y6 y7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 1 1 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>1 1 1 0 0 0 1</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1 0 1 1 0 1 1</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>1 0 1 1 1 1 1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1 1 0 0 0 0 0</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 1 1 1 0 1 1</td>
</tr>
</tbody>
</table>
Some others

PLD

SPLD

CPLD

PROM

PLA

PAL

PLA and PAL
\[ y_5 = \overline{a} \overline{b} \overline{c} \overline{d} \lor \overline{a} \overline{b} \overline{c} \overline{d} \lor \overline{a} \overline{b} \overline{c} \overline{d} \lor \overline{a} \overline{b} \overline{c} \overline{d}; \]

\[ y_5 = \overline{b} \overline{c} \overline{d} \lor \overline{a} \overline{c} \overline{d}; \]
A trivial algorithm that enables us to count the number of ones in a binary vector.

```
\begin{tabular}{|c|c|c|c|c|}
\hline
a_0 & 00 & a_1 & 01 & 1 & D_2 \\
\hline
y_1 & 01 & a_0 & 00 & x_1 & - \\
y_2 & a_2 & 10 & \overline{x_1x_2} & D_1 \\
\hline
a_2 & 10 & a_0 & 00 & x_1 & - \\
y_3 & a_2 & 10 & \overline{x_1x_2} & D_1 \\
a_3 & 11 & a_0 & 00 & x_1 & - \\
\hline
y_3 & a_2 & 10 & \overline{x_1x_2} & D_1D_2 \\
y_4 & a_3 & 11 & \overline{x_1x_2} & D_1D_2 \\
\hline
\end{tabular}
```
CPLD were introduced by Altera in 1984

Programmable interconnect matrix

EP300 PLDs introduced—world's first reprogrammable PLD and first commercial programmable logic device patent. 20 years of Altera innovations. Available at: http://media.corporate-ir.net/media_files/irol/83/83265/2003ar.pdf
Example of XPLA3 architecture [1]

The design software hides the CPLD resources, which enables end users to work with higher level constructs and to abstract from the architectural details.

16 Mcells per Logic block

2-24 Logic blocks per device

Each logic block – 36 x 48 PLA

CPLD applications [1]

Configurable Logic Block

Programmable Interconnect
Features of Xilinx CLBs

- Number and functionality of CLBs vary from device to device;
- Each CLB consists of a configurable switch matrix with 4 or 6 inputs, flip-flops and selection circuits;
- The switch matrix is very flexible and can be configured to handle combinatorial logic, shift registers, or RAM.

Features of Xilinx PI

- PI makes routes between CLBs (IOBs) allowing to construct very complicated circuits and to provide an external interface with the circuit;
- The design software takes care about all the details and makes the interconnect routing task hidden to users.

Getting Started with FPGAs. Available at: http://www.xilinx.com/company/gettingstarted/index.htm
1. Eliminates skew;
2. Provides four phases of the clock;
3. Doubles the clock;
4. Divides the clock by 1.5, 2, 2.5, 3, 4, 5, 8 or 16

Platform FPGAs

Standard FPGAs, which are targeted at multiple applications such as embedded systems, digital signal processing, etc. [1]

Multi-Platform FPGAs

Multiple FPGA platforms make possible a specific platform to be optimized specifically for a certain domain of applications – such as logic, connectivity, DSP and embedded processing – to meet application requirements previously delivered only by ASICs while remaining programmable at heart [2]

Spartan-3 Architecture Layout [1]

Spartan-3 CLB = 4 slices

Each slice includes:
- Two RAM-based logic function generators (two Look-Up Tables – LUTs);
- Two storage elements (D-type flip-flop or a level-sensitive latch);
- wide-function multiplexers;
- carry logic;
- arithmetic gates.

LUT can be configured as distributed RAM or a 16-bit shift register.

There are four types of interconnect, which passes signals among the functional elements. They are optimized for different clock frequencies, provide higher or smaller routing flexibilities, etc.

The DCM supports three major functions [1]:

- Clock-skew Elimination;
- Frequency Synthesis;
- Phase Shifting.

From 4 to 104 18Kbit configurable, synchronous dual-port RAM blocks

From 4 to 104 embedded multipliers. Each multiplier takes two 18-bit operands and produces a 36-bit product

From 2 to 4 DCMs

The DCM supports three major functions [1]:

Virtex-4 Multi-platform FPGA Family [2]

Advanced Silicon Modular Block (ASMBL™) columnar architecture [1,2]

Virtex-4 Multi-platform FPGA Family [1]

• LX: High-performance logic applications;

• SX: High-performance solution for digital signal processing (DSP);

• FX: High-performance, full-featured solution for embedded platform applications.

LX: high-performance logic; SX: High-performance DSP; FX: Embedded applications
Virtex-5 Multi-platform FPGA Family [1]

- LX: High-performance logic applications;
- LXT: High-performance logic with advanced serial connectivity;
- SXT: High-performance signal processing applications;
- FXT: High-performance embedded systems.

Recent FPGAs of Altera

The Cyclon II FPGA family (90 nm) [1]

- Phase-Locked Loops (PLL).
- Embedded Multipliers.
- Logic Array.
- M4K Memory Blocks.
- I/O Elements.

Recent FPGAs of Altera

The Stratix III FPGA family (65 nm) [1]

L – Balanced logic, memory and DSP for general-purpose applications;
E – Enhanced memory and DSP for DSP intensive applications;
GX – Integrated transceivers for high bandwidth interface applications.


**FPGA Evolution**

**XC4000 FPGAs** – system clock up to 80 MHz [1]

The fastest FPGAs of Achronix Semiconductor – up to 1.93 GHz [2]

**Spartan FPGAs** – Maximum system gates – 40K = 40 000 [1]

**Spartan-3 FPGA** – Maximum system gates – 5M = 5 000 000 [3]
FPGA Evolution

Max System Gates

Max CLBs
FPGA Evolution

System Performance

Clock (MHz)

Spartan Spartan-II Spartan-IIIE Spartan-3 Virtex-5
“Structured ASICs leverage standard-cell technology and the most advanced semiconductor processes to embed logic and hard functions—such as memory, phase-locked loops (PLLs), clock networks, and power bussing—into pre-engineered, pre-verified base layers. The structured ASIC is customized using just the top metal layers” [1].
Structured ASICs

Design flow [1]

1. RTL design.
2. Selection FPGA and HardCopy devices.
3. Using synthesis tools.
4. Place and route for FPGA and HardCopy devices.
5. Verifying timing constraints.
6. Verifying the complete design using FPGA.
7. Submitting the project to HardCopy design center.
8. Receive a HardCopy structured ASIC (10 weeks).

1. Bob Zeidman, The Death of the Structured ASIC. My list of the worst semiconductor products would include the Structured ASIC, a device that is not fictitious and is not a joke, but deserves to be one. Chip Design Magazine. April/May 2006. Available at: http://www.chipdesignmag.com/display.php?articleId=434


<table>
<thead>
<tr>
<th>Platform</th>
<th>FPGA</th>
<th>Structured ASIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification</td>
<td>14-24 pm</td>
<td>55% less [2]</td>
</tr>
<tr>
<td>Implementation</td>
<td>6-12 pm</td>
<td></td>
</tr>
<tr>
<td>Verification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototyping</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evaluation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Malachy Devlin. Multi-FPGA systems for High Performance Computing applications Available at: http://www.ieee.org/oncomms/sector/electronics/Articles/Object/23057E18-0825-84BD-6C1F87BD5B887FF9
FPGAs [1]:

- Can be seen as ‘Soft’ ASICs;
- Introduce a new computing paradigm;
- Eliminate the necessity for von Neumann architecture although such architecture can be used if required;
- Enable the designers to implement algorithms directly in silicon;
- Make parallelism a key feature;
- Permit any required external interface to be established, etc.
(When) will FPGAs kill ASICs?
The 38th conference on Design automation (2001)

“FPGA have grown to become multi-million gate devices with system-level features that can implement complete system on a chip. More and more ASIC designers are using FPGAs while the number of designs moving from FPGAs to ASICs is reducing simultaneously”

“ASICs are already dead; they just don’t know it yet”

Will Power Kill FPGAs? FPGA 2006 Panel

“Power consumption is now a limitation in many FPGA user applications”

Did FPGAs kill Structured ASICs? [3]

Programmable SoCs- the end of ASICs? [4]
System-on-a-chip (SoC) is a single microchip integrating components for a complex electronic device with typical applications in the scope of embedded systems, telecommunications, multimedia and consumer electronics.

SoC includes such components as:

- Microcontroller(s), microprocessor(s), DSP(s);
- Memory;
- Industry standard digital interfaces;
- Analog interfaces;
- Etc.

The **Network on Chip (NoC)** employs networking method for communication between sub-systems available on a chip and makes significant improvements over bus systems.

To meet the communication requirements of large SoCs, a network-on-chip (NoC) paradigm is emerging as a new design methodology [1].

FPGAs can be seen as a very adequate and promising base for programmable SoCs and NoCs, because of the following:

- Platform FPGAs contain the majority of the required SoC/NoC components;
- Multi-platform FPGAs can be retargeted to the desired scope of SoC/NoC, such as embedded systems, telecommunications, multimedia, etc.;
- FPGA-based design has numerous advantages over other known design techniques.

---

Reconfigurable Computing [1]  
“A methodology of using programmable logic devices in a system design such that the hardware-based logic can be changed to perform various tasks”

Reconfigurable computing [2] “is computer processing with highly flexible computing fabrics. The principal difference when compared to using ordinary microprocessors is the ability to make substantial changes to the data path itself in addition to the control flow”.

The earliest work on reconfigurable computer architecture was done at the University of California at Los Angeles (UCLA) [3].

The world's first commercial reconfigurable computer, the Algotronix CHS2X4, was completed in 1991 [2].

Reconfigurable system is a system based on reprogrammable logic devices allowing to implement different functionality within the scope of predefined constraints

“The reconfigurable system concept enables different standards and protocols to be tried, tested, and put on road trials; and if you don’t find them to be suitable, you can just load in another bus interface and try it out until you find the best” [1]

“A solution that designers are turning to is reconfigurable systems based on FPGAs that can be reprogrammed to accommodate changing standards and protocols late in the design process and even in production” [1]

Design, Implementation, Practical Applications and Future of Reconfigurable Systems
Outline

• Molecular electronics;
• Static and dynamic reconfiguration;
• Design flow;
• Applications in mobile computing and multimedia;
• Future of reconfigurable systems.
Moore's Law

Transistor density of integrated circuits, with respect to minimal component cost, doubles every two years.

Served as a goal for industry for more than 40 years!

Data from Intel
Kurzweil projects that a continuation of Moore's Law until 2019 will result in transistor features just a few atoms in width.

Then, a new type of technology will replace current integrated circuit technology.

"Moore’s Law cannot continue unabated for more than 600 years for any technological civilization" ¹

Challenges

International Technology Roadmap for Semiconductors, Emerging Research Devices

Short term: “enhanced CMOS” – heterogeneous integration of new technologies with the CMOS platform

Long term: Inventing fundamentally new approaches to information and signal processing
New Technologies

1D structures (carbon nanotubes, nanowires)

Resonant tunneling devices

SET

Ferromagnetic logic

Spin transistor

Molecular

Number of articles in technical journals that appeared in the Science Citation Index database for July 2003 – July 2005

Data from ITRS’2005
“The concept of molecular electronic devices is based on electronic properties of individual molecules tailored to perform logic operations and on the assembly of a large number of these functional building blocks into molecular circuits.” ¹

¹ International Technology Roadmap for Semiconductors, 2005, Emerging Research Devices

“Organic molecules exist which have two mechanically distinct parts, such as a ring and a rod or interlocking rings. Applying a programming voltage across the molecule adds or subtracts an electron, shifting the ring and changing the molecule’s conductivity. It functions as a non-volatile programmable molecular switch.” ²


“Molecular switches have no gain, so signals must be restored between switch array by devices with gain, such as CNW-FETs.” ²
“Carbon nanotube is a molecular sheet a single atom thick, which has wrapped around into a tube, as little as one nanometer wide” ¹


“Nanowires can be grown to hundreds of microns or perhaps millimeters in length. However, at this high length to diameter ratio, they become highly susceptible to bending and ultimately breaking.” ²

Arrays of modest-size crossed nanowires with switchable diodes at crosspoints can be used for constructing memory cores and programmable crossbar interconnect arrays.

Each NW has to be addresses uniquely in order to program individual crosspoints allowing to implement a given logic function and routing and to avoid defective NW.

It was shown that fully programmable, post-fabrication configurable, FPGA can be built with all logic and restoration occurring at the nanoscale.

In current technologies, FPGA area is greatly determined by the area of its programmable switches.

Molecular-scale programmable switches are estimated to be two orders of magnitude smaller than lithographic crosspoints.

Although molecular FPGA will have overhead for lithographic support and defect tolerance, it is expected that molecular FPGA will surpass lithographic CMOS ASICs in density in the early 2010s. ¹

Very high device densities may be achieved;

Switching energy may be significantly reduced (comparing to CMOS) and may approach the thermodynamic limit;

The speed of molecular devices is limited and is expected to be much lower than that of CMOS;

Novel architectures will have to be more defect-tolerant relative to current CMOS devices;

Since FPGA are based on very regular crossbar structures they “offer the best chance for eventually fabricating nanocomputers with more than $10^{10}$ devices on a chip.”

Cost per gate in molecular devices is estimated to be much lower than for any other kind of device in the year 2016 (varying from $1E-12$ to $1E-10$).

---

1 International Technology Roadmap for Semiconductors, 2005, Emerging Research Devices
Reconfiguration modes

- **Static**
  - Problem instance
  - Configuration
  - Execution
  - Result

- **Dynamic**
  - Problem instance
  - (modify) configuration
  - Execution
  - Result

  Run-time reconfiguration

  Run-time reconfiguration
Static reconfiguration

Application

Configuration

Reconfiguration
Global dynamic reconfiguration

Application

A

B

C

virtual hardware
Partial dynamic reconfiguration
Reconfigurable models

new configuration

new configuration

new configuration
Partial Reconfiguration

- Ability to reconfigure selected FPGA areas anytime after its initial configuration

**Active partial reconfiguration** – reconfiguration is done while the design is operational and the device is active

**Static partial reconfiguration** – reconfiguration is done while the device is inactive

**Advantages of partial reconfiguration:**

- Adaptability of hardware algorithms;
- Ability to share hardware between applications;
  - Increased resource utilization, reduced device count, power consumption and overall costs;
- Shorter reconfiguration times;
- Ability to provide continuous hardware servicing;
- Possibility to upgrade hardware remotely.
Examples: Virtex™-4, Spartan™-3/E

Module-based partial reconfiguration uses modular design concepts to reconfigure large blocks of logic

Difference-based partial reconfiguration allows small changes to be made in an FPGA design and then bitstream is generated based on only the differences in the two designs.
FPGA Design Flow

- Specification
  - behavioral simulation
  - functional simulation
- Synthesis
- Implementation
  - timing simulation
  - in-circuit verification
- Verification
- FPGA
Specification and Synthesis

- Schematic entry
- Hardware description languages
- System-level specification languages
- General-purpose languages

```vhdl
entity full_adder is
  Port ( a, b, cin : in std_logic;
        sum, cout : out std_logic);
end full_adder;

architecture Behavioral of full_adder is
begin
  sum <= a xor b xor cin;
  cout <= (a and b) or (a and cin) or (cin and b);
end Behavioral;
```

```c
void full_adder(unsigned int a, 
                unsigned int b, 
                unsigned int cin)
{
    sum = a ^ b ^ cin;
    cout = (a & b) | (a & cin) | (cin & b);
}
```
Example

Celoxica DK Design Suite ¹
- Specification in Handel-C
  - EDIF file

Xilinx ISE ²
- Specification in VHDL
  - XST
  - Implementation

Xilinx ISE Quartus II
Symplicity Pro
Precision Synthesis

¹ http://www.celoxica.com
² http://www.xilinx.com
Example

Specification in standard C/C++

Software/ Hardware partitioning

Generate synthesizable HDL code

Export to implementation tools

Algorithm simulation

Generate Software/Hardware interfaces

Export to implementation tools

FPGA

Verification

Synthesis

Implementation

Specification

Catapult Synthesis

CoDeveloper

1 http://www.celoxica.com
2 http://www.xilinx.com
Commercially-available Tools

- **Accelchip** provides high-level FPGA design tools based on Matlab and Simulink.
- **Agilent** provides an effective solution for in-circuit debug of FPGA and surrounding systems.
- **Aldec** provides FPGA design and verification tools, including a popular HDL simulator.
- **Altera** provides FPGA design, synthesis, implementation and verification tools.
- **Altium** provides FPGA synthesis and simulation tools, as well as royalty-free IP blocks that include processor cores.
- **Annapolis Micro Systems, Inc.** provides the CoreFire Design Suite for high-level design entry, and IP blocks for digital signal processing.
- **Celoxica** provides Handel-C tools for high-level design entry and verification.
- **Gedae** provides high-level FPGA and DSP programming tools allowing programmers to develop to heterogeneous systems and related optimization and verification tools.
- **Impulse** provides the Impulse C compiler and related optimization and verification tools.
- **LogicSim** provides a free Windows-based Verilog simulator and debugger, useful for FPGA RTL verification.
- **Lyrtech** provides hardware combining FPGA and DSP technologies plus high-level FPGA-DSP design tools based on Matlab, Simulink, Xilinx Sysgen and Texas Instruments CCS.
- **MathWorks** provides Simulink, a tool that is used as a high level design entry for FPGAs.
- **Mentor** provides the Catapult C high-level design tools, as well as a wide range of HDL synthesis and simulation tools.
- **National Instruments** provides FPGA development boards and LabVIEW FPGA, a high level graphical programming language with built-in IP blocks.
- **Synopsys** provides HDL synthesis tools, as well as a broad range of verification and high-level design tools.
- **Synplicity** provides HDL synthesis, FPGA physical synthesis and verification tools.
- **Xilinx** provides HDL design, synthesis, implementation and in-circuit verification tools.

... not complete!
“The days of designing FPGAs with schematics are gone.”

1 R. Goering, “FPGA users rank challenges, tasks”, EE Times, 07/31/2006
Design Productivity Gap

- inherited from ASIC

Design productivity is a real challenge for future systems.

Chip complexity (log scale)

Time

Moore’s Law

Platform FPGA

Design productivity (log scale)
Reuse intellectual property (IP) to improve design productivity

– design effort for the reused logic is only a portion of the effort needed for newly designed logic
– reuse rate for system-level design is expected to increase from 32% in 2005 to 55% in 2020 ¹

Design abstraction levels must be raised

– “higher levels of abstraction allow many forms of verification to be performed much earlier in the design process, reducing time to market and lowering cost” ¹

The level of automation must be increased

– reduce the number of iterations
– further improvement in design tools
– automated HW/SW partitioning in high-level design stages

¹ International Technology Roadmap for Semiconductors, 2005
Mobile and multimedia applications became an important driver for FPGA products and technologies

The life cycle of mobile consumer products is short, and will stay short in the future. Therefore, the design effort cannot be increased - it needs to stay at the current level for the foreseeable future. \(^1\) – need to shorten time to market and to lower system costs (NRE, design and test)

Portable devices are highly power-dependant – need to optimize processing per microwatt

Autonomous mobile devices used in applications (such as medical, space), where repair by human experts is for some reason impossible, may require to be either autonomously or remotely repaired – need for (remote) reconfigurability and adaptability, and also for simplifying recertification and preventing obsolescence issues

New design techniques: evolvable hardware – possibility to construct a circuit with a given specification, whose structure is previously unknown – need for self-reconfigurability

\(^1\) International Technology Roadmap for Semiconductors, 2005, System Drivers
- Software defined radios and cognitive radios

Feature demanded: adaptability
- to spectral environment;
- a waveform to compensate for channel fading;
- collaborate with multiple radios to receive a weak signal;
- accommodate damage to some of a radio’s processing resources by reconfiguring the remaining resources to support the most critical services;
- etc.

Partially reconfigurable platform FPGA have high processing throughput and can provide different levels of adaptability at low power and cost.

Examples

- Algorithm acceleration in Nallatech\textsuperscript{1,2}
  
  Biological sequencing
  
  Real-time video processing
  
  Gravity simulation

Remote mobile computing for gas and oil exploration
(Nallatech’s work with R. Associates) \textsuperscript{2}

Seismic processing
  
  - Kirchhoff algorithm
  
  - calculations in IEEE 754
  
  - 64 times faster than a 2GHz Pentium 4
  
  - 200 times less power consumption

performance improvement

reduced space

“The FPGA solution saves more than $10,000 in electricity bills per year per 64-processor 19” rack, based on seven cents per kilowatt hour”\textsuperscript{2}

\textsuperscript{1} M. Devlin, “Multi-FPGA systems for High Performance Computing applications”,
IEE Developers Forum, October 2005

\textsuperscript{2} “R. Associates Joins Nallatech’s Growing Channel Partner Program”,
FPGA and Structured ASIC Journal, August 2005
Examples

- Biotech applications

Smith-Waterman algorithm – a life sciences application for comparing DNA and amino acid sequences against known genes and proteins


2 http://www.cray.com
Examples

- Image processing
  - interactive gaming;
  - medical image scanning;
  - security systems (iris scanning, automatic number plate recognition);
  - etc.

- color space conversion;
- transforms (DCT, FFT);
- convolution;

Sundance image processing development platforms

1 http://www.sundance.com/
Examples

- Automotive infotainment
  - reduced system cost;
  - low power;
  - increased flexibility.

http://www.iee.org/oncomms/sector/electronics/Articles/Object/2301030F-AEFC-4F99-12D5DA60F20AA683
Examples

- Speech enhancement
  - speech recognition systems;
  - surveillance;

<table>
<thead>
<tr>
<th></th>
<th>GPP</th>
<th>DSP</th>
<th>FPGA (Cyclone 12 K)</th>
<th>ASIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design time</td>
<td>1-2 weeks</td>
<td>3-4 weeks</td>
<td>2-3 months</td>
<td>over a year</td>
</tr>
<tr>
<td>Power consumption</td>
<td>23 – 100 W</td>
<td>350 mW</td>
<td>183 mW</td>
<td>3.45 mW</td>
</tr>
<tr>
<td>Performance</td>
<td>not real-time</td>
<td>barely real-time</td>
<td>real-time</td>
<td>real-time</td>
</tr>
</tbody>
</table>
- Functions
  - searching;
  - sorting;
  - coding/decoding;
  - signal processing;
  - audio/video/image manipulation;
  - encryption;
  - error correction;
  - random number generation;
  - packet processing;
  - recovery from SEU;
  - etc.

- Applications
  - seismic processing;
  - acoustics;
  - astrophysics FFT;
  - adaptive optics;
  - cryptography;
  - graphics acceleration;
  - HDTV;
  - mobile radio;
  - car multimedia systems;
  - image recognition;
  - speech recognition;
  - security;
  - video format translation;
  - biotech applications;
  - vehicular traffic simulation;
  - financial modeling;
  - orbit, space, and extra-terrestrial applications
  - etc.
41% of developers overall are using one or more FPGA in their projects
35% of developers overall do not use any custom logic in their projects

---

"The growing system complexity will make it impossible to ship designs without errors in the future. Hence, it is essential to be able to fix errors after fabrication. In addition, reconfigurability increases reuse, since existing devices can be reprogrammed to fulfill new tasks." ¹

¹ International Technology Roadmap for Semiconductors, 2005, Design
Future Mobile Platforms

High processing performance  Low power consumption

Main processor

Main memory

Peripherals
The number of PE will grow exponentially.

1 International Technology Roadmap for Semiconductors, 2005, System Drivers
Reconfigurable systems in engineering, research activity and education
• Rapid prototyping;
• Reconfigurable systems in research with the following examples:
  – Experiments with alternative and competitive algorithms;
  – Implementation of computationally intensive algorithms;
• Reconfigurable systems in education.
There are many prototyping boards available for engineering and research (see, for example, [1])

Experience with the following prototyping boards (systems):

- **Annapolis FireFly™ (FPGA XC6216/6264)** - 1997-2001;
- **XESS XS40 (FPGA XC4010XL)** - 1998-2002;
- **Alpha Data ADM-XRC (FPGA Virtex-EM XCV812E)** - 2000-2003;
- **Alpha Data ADM-XPL (FPGA Virtex-II Pro XC2VP7)** - from 2003;
- **XESS XSA100 (FPGA Spartan-II XC2S100)** - from 2002;
- **Trenz TE-XC2Se (FPGAs Spartan-IIE XC2S300E/400E)** - from 2002;
- **Celoxica RC100 (FPGA Spartan-II XC2S200)** - from 2002;
- **Celoxica RC200 (FPGA Virtex-II XC2V1000)** - from 2003;
- **Celoxica RC10 (FPGA Spartan-3 XC3S1500L)** - from 2005;
- **DETIUA-S3 (FPGA Spartan-3 XC3S400)** - from 2005.

Designed at the department by Manuel Almeida
Software was designed by Bruno Pimentel

1. Xilinx prototyping board list. Available at: http://www.xilinx.com/ Development boards
Interface socket

Hardware designed by Manuel Almeida.

Software designed by Bruno Pimentel.
Hardware designed by Manuel Almeida
Software designed by Bruno Pimentel
Simulation of an execution unit in virtual mode on PC monitor screen

Interface with PC

Extension hardware for interactions with sensors and actuators

USB module

Extension hardware for interactions with sensors and actuators

Hardware designed by Manuel Almeida
Software designed by Bruno Pimentel
Simulation software designed by students
Extensions for different interfaces

Using devices designed by students

etc.
Hardware designed by Manuel Almeida
Software designed by Bruno Pimentel
Experiments with alternative and competitive algorithms

1) 17
2) 6
3) 18
4) 9
5) 5
6) 21
7) ...

17

root of a binary tree

6

left node because 6 < 17

18

right node because 18 > 17

5

left node of the node 6, because 5 < 17 and 5 < 6

9

right node of the node 6, because 9 < 17 and 9 > 6

21

right node of the node 18, because 21 > 17 and 21 > 18
// Function template to create a root node or add a node
template<class T> treenode<T> *addnode(treenode<T> *node, T value) {
    if(node==0) {
        // constructing a new node of type treenode<T>
        node = new treenode<T>;
        // exception handling
        node->val = value;    // Store new value in node
        node->count = 1;      // Only one to start. At the
        node->rnode = node->lnode = 0;  // beginning we have just one node
    } else if(value==node->val)
        node->count++;
    else if(value < node->val)
        // traverse the tree or construct the left node
        node->lnode = addnode(node->lnode,value);
    else
        // traverse the tree or construct the right node
        node->rnode = addnode(node->rnode,value);
    return node;  // return a pointer to the current node
}
It is known that recursion is an extremely powerful problem-solving technique that permits a problem to be decomposed into smaller sub-problems that are of exactly the same form as the original problem.

However this technique is not always appropriate, particularly when a clear efficient iterative solution exists.

This is primarily due to the large amount of states that are accumulated during deep recursive calls.

Besides in most high-level programming languages, a function call incurs a bookkeeping overhead. Recursive functions magnify this overhead because a single initial call to the function might generate a large number of recursive invocations of the function.

What about hardware implementation? How to implement recursive calls in hardware?
// Function template to sort and output a tree
template<class T> void treesort(treenode<T> *node) {
    if(node!=0) // if the node exists
    {
        treesort(node->lnode); // Sort left sub-tree
        // Display value after any hierarchical return
        cout << "value - " << node->val << "; repeated - " << node->count << endl;
        treesort(node->rnode); // Now sort right sub-tree
    }
}
Recursive calls (recursive) returns

Begin

\[ x_6 \]
\[ y_1, y_2, z_2 \]
\[ x_7 \]
\[ y_3 \]
\[ y_1, y_4, z_2 \]

End, \[ y_5 \]

17

18

3

12

10

13

15

14
Recursive algorithm for data sorting

Begin

$z_0$

Begin

$z_1$

$z_2$

End

$y_1, y_2, z_1$

$y_1, y_4, z_1$

$y_9$

$y_6, y_8$

$y_7, y_8$

End, $y_{10}$

Begin

$z_2$

$y_1, y_2, z_2$

$y_3$

$y_7, y_8$

End, $y_5$
Iterative algorithm for data sorting

Begin

End

sorting

construction of the tree
Hardware Implementation Details

- **Iterative implementation:**
  - VHDL: (using Finite State Machines - FSM);
  - **Handel-C:** (using constructions similar to C language).

- **Recursive and Modular implementation:**
  - VHDL: (using Hierarchical Finite State Machines - HFSM);
  - **Handel-C:** (using Hierarchical Finite State Machines - HFSM).
Experiments

Implemented algorithms for:

- (P1) Sorting
  - based on a binary tree (tree-based)
- (P2) finding a minimal row cover of a binary matrix
  - 128x128 matrices; approximate method (cyclic)
- (P3) the Knapsack problem
  - exact binary search method (tree-based)
- (P4) GCD computation
  - GCD of 2 unsigned integers (cyclic)
## Tools

<table>
<thead>
<tr>
<th>Language</th>
<th>VHDL</th>
<th>Handel-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problems</td>
<td>P1, P2, P3, P4</td>
<td>P1, P2</td>
</tr>
<tr>
<td>Synthesis</td>
<td>Xilinx ISE</td>
<td>Celoxica DK design suite</td>
</tr>
<tr>
<td>Implemen-tation</td>
<td>Xilinx ISE</td>
<td></td>
</tr>
<tr>
<td>FPGA</td>
<td>xc2s400e-6ft256 (Spartan-IIE family)</td>
<td>xc2v1000-4fg456 (Virtex-II family)</td>
</tr>
<tr>
<td>Board</td>
<td>TE-XC2Se (Trenz)</td>
<td>RC200 (Celoxica)</td>
</tr>
</tbody>
</table>
**VHDL Results**

**P1 – Sorting (tree-based)**

Legend:
- Red: Number of Slices of Recursive implementation
- Blue: Number of Slices of Iterative implementation
- Orange: Execution Time of Recursive implementation
- Purple: Execution Time of Iterative implementation

**P3 – Knapsack (tree-based)**

**P4 – GCD (cyclic)**
Handel-C Results

P1 – Sorting (tree-based)

P2 – Minimal Row Cover (cyclic)

P3 – Knapsack (tree-based)

P4 – GCD (cyclic)
Conclusions from the experiments

- Recursion can be implemented in hardware much more efficiently than in software;
- This is because any activation of a recursive subsequence of operations has been combined with the execution of the operations that are required by the respective algorithm. The same event takes place when any recursive subsequence is being terminated, i.e. when control has to be returned to the point after the last recursive call and an operation of the executing algorithm that follows the last recursive call has to be activated;
- The number of states required for the execution of a recursion in hardware can be reduced significantly compared with software but it is still greater than for iterative solutions;
- However such states are accumulated on stacks typically implemented on built-in memory blocks, which are relatively cheap;
- C/C++ implementations of iterative algorithms are faster comparing with recursive algorithms;
- VHDL-based projects for recursive algorithms give better results (resources and execution time) comparing with iterative algorithms;
- Handel-C based projects for iterative algorithms give better or similar results (resources and execution time) comparing with recursive algorithms.
Binary and ternary matrices

a b c d e f g h i j k l

1 1 0 0 1 1 1 0 1 0 1 1
1 2 0 1 0 0 1 1 1 1 0 1 0 0
1 3 0 0 1 1 0 0 0 1 0 0 0 1
1 4 1 0 1 0 1 1 1 0 0 1 0 0
1 5 1 0 0 0 1 0 1 1 0 1 1 1
1 6 1 1 1 1 0 0 0 0 1 0 1 0
1 7 0 0 1 0 1 1 1 0 1 0 0 1
1 8 1 0 1 1 1 1 1 1 1 1 1 1
1 9 1 1 1 1 1 1 1 1 1 1 1 1

Discrete matrix can be seen as a very adequate model for solving combinatorial problems in hardware.
The decision tree is constructed during the search process and it is traversed starting from the root.

Applying reduction rules:
- delete rows 8, 9; columns d, e, f, i, j, row 1; column k
- delete rows 2, 6; column b
- delete rows 2, 4, 5, 7; column g
- delete rows 2, 3, 5; column h

Applying reduction rules:
- delete columns a, h; row 7
- delete columns a, b, h, l;
- delete columns a, b, h, l;
- delete columns a, b, h, l;

The decision tree is built as follows:

- **Column b**: delete rows 2, 6; column b
  - Remaining: a, c, g, h, l
  - M = \[
    \begin{bmatrix}
    2 & 0 & 1 & 0 & 1 & 1 \\
    3 & 0 & 0 & 1 & 0 & 1 \\
    4 & 1 & 0 & 1 & 1 & 0 \\
    5 & 1 & 0 & 0 & 1 & 1 \\
    6 & 1 & 1 & 1 & 0 & 0 \\
    7 & 0 & 1 & 1 & 0 & 1
    \end{bmatrix}
  
- **Column g**: delete rows 2, 4, 5, 7; column g
  - Remaining: a, c, h, l
  - M = \[
    \begin{bmatrix}
    3 & 1 & 0 & 1 & 1 \\
    4 & 1 & 1 & 0 & 0 \\
    5 & 1 & 0 & 1 & 1 \\
    7 & 0 & 1 & 1 & 0
    \end{bmatrix}
  
- **Column h**: delete rows 2, 3, 5; column h
  - Remaining: a, b, c, g, l
  - M = \[
    \begin{bmatrix}
    4 & 1 & 0 & 1 & 1 & 0 \\
    6 & 1 & 1 & 1 & 0 & 0 \\
    7 & 0 & 0 & 1 & 1 & 1
    \end{bmatrix}
  
- **Column c**: delete columns a, h; row 7
  - Remaining: c, g
  - M = \[
    \begin{bmatrix}
    3 & 1 & 0 & 1 \\
    4 & 1 & 1 & 0 \\
    5 & 0 & 1 & 1
    \end{bmatrix}
  
- **Column l**: delete columns a, b, h, l;
  - Remaining: c, g
  - M = \[
    \begin{bmatrix}
    5 & 1 \\
    4 & 1 & 1 \\
    6 & 1
    \end{bmatrix}
  
Stop conditions:
- {c, g}
- {b, c, l}
- {b, l, ...}
Binary and ternary matrices

Orthogonality

\[
\begin{array}{cccccccc}
A & B & C & D & E & F & G \\
1 & 1 & - & - & 1 & - & - & - \\
2 & 1 & - & - & 0 & - & 1 & - \\
3 & 1 & - & - & 1 & - & 0 & - \\
4 & 1 & - & - & - & 1 & - & - \\
5 & 0 & - & 0 & - & 1 & - & - \\
6 & 0 & - & 0 & - & 0 & - & - \\
7 & 0 & - & 1 & 1 & 1 & - & - \\
8 & 0 & - & 1 & 0 & 1 & 0 & - \\
\end{array}
\]
General Purpose Computer

Software part for SAT solver

Data compression in software \( (T_s) \)

Transmitting non-compressed result

Hardware part for SAT solver

Data decompression in hardware \( (R_h, T_h) \)

FPGA

Transmitting compressed data \( (T_t) \)
Graph coloring

ICARA2006

Handel-C projects for Celoxica RC200 (FPGA Virtex-II XC2V1000)
Conclusions from the experiments

- Combinatorial search algorithms have two following distinctive features:
  - ✔ They require a huge number of different variants (feasible solutions) to be considered;
  - ✔ These variants can be ordered and examined with the aid of a decision tree that provides an efficient way for handling intermediate solutions.

- The decision tree is constructed during the search process and it is traversed starting from the root.
- Many combinatorial algorithms have to deal with a huge amount of data, which have to be transferred between a host computer and an FPGA-based accelerator. In many circumstances, due to the complexity, the problem cannot be completely solved just in an FPGA, and combined hardware/software solutions are employed. This involves multiple time consuming data transfers. To simplify the problem compression/decompression technique can be used.
Operations ≠ Operations

Problem solver Interface

SoC
FPGA gives numerous advantages for implementing computationally intensive combinatorial search algorithms, namely:

- Since any problem involves a huge number of similar operations, which are not the same for different combinatorial problems it is not easy to construct a universal combinatorial processor, i.e. different instructions have to be customized for a particular problem that is going to be solved. This can be trivially done with the aid of FPGA technology.

- Different practical applications require solving combinatorial tasks with varying complexity. However optimal results can be achieved in case if the size of operands permits any required operation to be performed in one clock cycle. Parameterizable circuits that provide such an opportunity can easily be implemented in FPGAs.

- FPGAs enable us to build on the same microchip any desired (customized) interface between a combinatorial accelerator and a general-purpose computational system (or any customized system that requires the accelerator).

- The complexity of recent FPGAs allows for the construction of a complete system-on-chip and a combinatorial accelerator can be implemented as an application-specific co-processor within this system.
Reconfigurable systems
in education

This part will
be considered in a
separate presentation
Conclusion

• FPGAs have proven their effectiveness in research and development.

• FPGA market is rapidly expanding its boundaries and this tendency will be kept in future.

• Available design tools allow for fast creation of highly optimized systems in vast varieties of application areas and at different levels of abstraction.

• The progress in FPGA technology is not sufficiently covered by the majority of university curricula.