University of Konstanz
Distributed Systems Laboratory
Prof. Dr. Marcel Waldvogel
Marc Kramis

Seminar: FPGA & VHDL
SS 08
October 31, 2011

# FPGAs as Computing Platform

Thomas Zink

**Abstract**

Since the discovery of Moore's Law advances in hardware have shown to fulfill it accurately. However, maintaining that fact seems to be increasingly difficult[1]. The von Neumann architecture suffers from drawbacks that have become known as the *von Neumann Syndrome*[2]. Inefficient data transmission through the memory wall and instruction-stream-based sequencing are the source of its illness. To reduce the symptoms the number of computing cores and the cache sizes are increased. But higher parallelism demands greater programming skills. *Reconfigurable Computing* promises to overcome those problems by performing a paradigm shift towards data-stream-based, instruction-less machines. *FPGA*s present a low-cost reconfigurable computing platform. This abstract gives a short overview of how *FPGA*s are used as computing platforms today.

## 1 Introduction

Before the age of reconfigurable computing (*RC*) two different kinds of algorithm execution platforms existed. There are *ASIC*s which are application specific and inflexible but provide the best performance. And there are *microprocessors* ($\mu p$), which are programmable to execute all kinds of applications but have performance problems. *RC* tries to bridge this gap by utilizing configurable hardware and combining the flexibility of software with the high performance of hardware.

A wide variety of reconfigurable hardware devices exist which can not be reviewed here. For an extensive overview see [3]. Reconfigurable devices are classified by the grain of their architecture which influences the performance for certain applications, the power consumption and the rate of reconfiguration. *FPGA*s have a fine-grained architecture with a 1-bit path-width which makes them optimal for implementing algorithms that operate at bit level but which leads to overhead when implementing designs that operate at a higher path-width since the routing space occupied will be very large, resulting in longer path ways and higher power consumption. Coarse-grained architectures like *DPA*s (Data Path Arrays) have broader path-widths. Thus they are more suitable for designs like processors but wasteful when operating on data smaller than word size. There are also *Data Path FPGA*s (*DPFPGA*) which have a mixed fine-/coarse-grained architecture to combine the advantages of both worlds.

*FPGA*s can be reconfigured at deployment, between execution phases or even during execution. Parts of the chip remain operational while other parts are reconfigured. This allows adaptive and on the fly hardware generation.

The number of gates available in *FPGA*s have grown to several millions recently and development as well as end-user costs are shrinking. Thus, despite their comparatively inefficiency regarding processor designs their popularity in academic as well as commercial usage has grown continuously.

Two application domains for *FPGA*s as computing platforms can be identified

which will be discussed in the next sections. They are *application acceleration* (*AA*) and *high performance computing* (*HPC*).

## 2 Reconfigurable Computing Origins

Most *RC* systems in use today are based on an architecture proposed 1960 by Gerald Estrin in [4]. His "fixed plus variable structure computer" consists of a standard processor connected to an array of reconfigurable hardware, or fabric coprocessor modules (*FCM*), that can be set up to perform specific tasks in parallel at the speed of hardware. His idea didn't draw

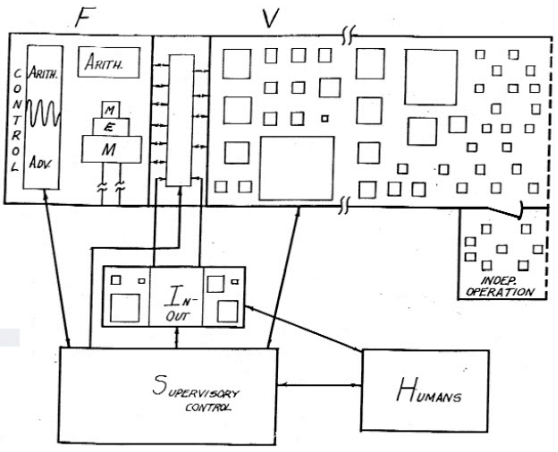


**Figure 1:** *Fixed plus variable structure computer[5]*

too much attention since at that time $\mu$ps were rather cheap compared to reconfigurable hardware and powerful enough with exponential growth in performance. But things changed when affordable *FPGA*s hit the market and the design flaws of the von-Neumann computer showed effect especially in *HPC*. Today a number of vendors provide systems based on Estrin's design.

The *FCM* array can be connected to the CPU in three different ways.

**BUS connection** To exchange data between the CPU and the *FCM* array the data must be moved through a BUS that is clocked at much lower rates and suffers from arbitration. It usually takes multiple idle clock cycles to wait for data.

**I/O connection** A dedicated I/O port provides a simple data and control interface to connect the CPU to the *FCM*s. Clock rates are much higher than with a BUS and data movement has lower latency and higher rates.

**Instruction pipeline connection** In this model an auxiliary processing unit (*APU*)is directly connected to the CPU instruction pipeline. Instructions not recognized by the CPU are decoded by the *APU* and directed to the *FCM*s for execution. Shared memory is provided to exchange operands and results.

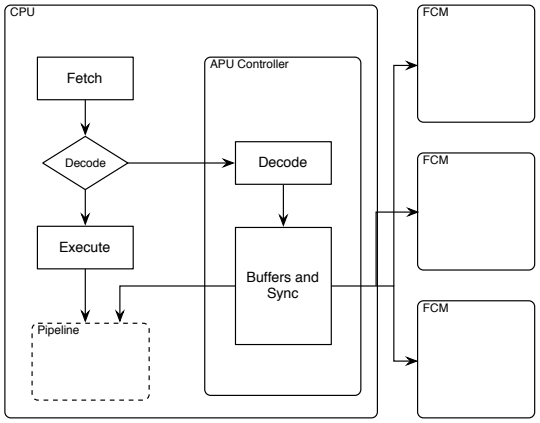


**Figure 2:** *Instruction pipeline connected* FCM*s[6]*

Using *FPGA*s it is possible to design complete processing systems on one chip, processor and custom coprocessors reside in the gate array. Alternatively Systems exist that feature a standard $\mu p$ with *APU* interface and an *FPGA* on one board, like the Xilinx Virtex-4 FX.

## 3 Application Domains

### 3.1 Application Acceleration

The major usage for *FPGA*s as computing platforms is *Application Acceleration*

($AA$). Computational expensive and parallel tasks that would normally require multiple $CPU$ instructions are offloaded to an array of $FCM$s. Thus $AA$ accelerates execution by reducing multiple instructions to a single instruction that is directly implemented in hardware and by introducing massive parallelism. $AA$ also aims at accelerating the hardware/software design by providing a higher level of abstraction. The system is designed using high level languages ($HLL$s) like ansi C/C++ or even Java. Profilers and optimizers identify bottlenecks and potential parallelism in the code which can be partitioned into parallel processes. The optimized source code is used to generate Hardware Description Language ($HDL$) files (usually Verilog or VHDL) and $HLL$ source files that define the software-to-hardware interfaces. The $HDL$ files are synthesized using the $FPGA$ provider's toolchains while the software sources can be compiled using standard compilers. Figure 3 depicts the design steps as shown in [7]. Optimizers exist that automatically perform all these steps, with poor results, however[7]. Usually the system is designed with parallelism and custom hardware in mind and the optimizers are used for support.

There are a number of proprietary and open source SDKs available, see table 1 for an overview of established products.

### 3.2 High Performance Computing

$RC$ for $HPC$ pushes $FPGA$-based solutions to the massively parallel world. A high number of $FPGA$s and multicore $\mu p$s are interconnected using a dedicated high speed bus to form a unified clustered system. Shared memory is connected to the bus to guarantee fast and consistent data exchange between all connected computing cores. Usually multiple $\mu p$s and $FPGA$s with an arbitrary number of memory reside on one board which interfaces with other boards via the bus. This architecture allows a mass of processes to execute in parallel, however, designing such systems is a challenging task.

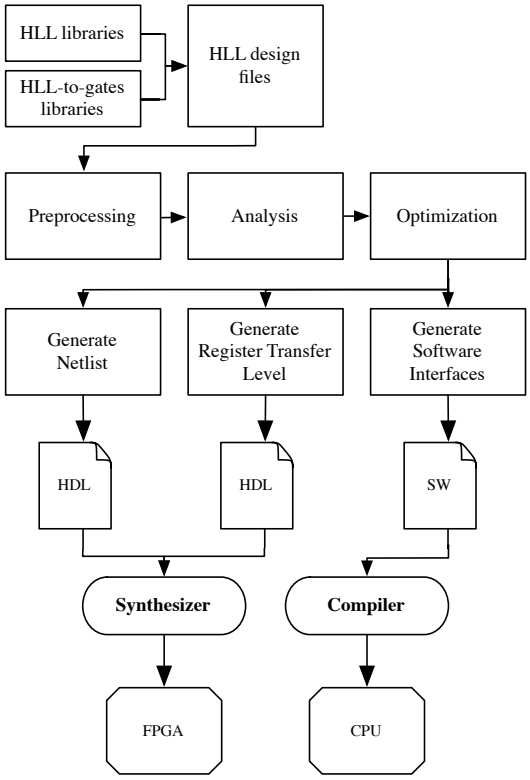


**Figure 3:** *Systems design using* HLL*s*

Most manufacturers providing reconfigurable super computers implement their own proprietary bus. Usually the boards come in some sort of blades which are stackable. Table 2 shows an overview of the most prominent $FPGA$ super computers. Numbers for $CPU$s and $FPGA$s are for a single blade.

## 4 Discussion and Outlook

Expectations are high regarding $RC$. One can imagine self adapting and reconfiguring hardware similar to natural evolutionary processes. However, $RC$ is still in it's early childhood. Improvements in hardware/software design have been made by integrating both worlds into single SDKs at a high level of abstraction featuring well known $HLL$s. But results of automatic optimizers are too poor so developers still have to think in parallel hardware/software processes. None of the tools available make use of the $FPGA$'s capability

| language | extends | developer | licence | produces |
|---|---|---|---|---|
| DIMEtalk | ansi C | Nallatech | commercial | VHDL |
| Handel-C | ansi C | Celoxica | commercial | FPGA bitstream |
| Mitrion-C | ansi C | Mitrionics | commercial<br>free for personal use | FPGA bitstream |
| Impulse C | ansi C | Impulse | commercial | VHDL or Verilog |
| System C | ansi C++ | Open System C Initiative | open source | VHDL or Verilog |
| JHDL | Sun Java 1.3 | Brigham Young University | open source | FPGA bitstream |

**Table 1:** *HLL-to-gates development kits*

| Product | CPUs | FPGAs | Bus |
|---|---|---|---|
| SGI RASC RC 100 blade | None, must be coupled with host system | 2 Virtex 4 LX200 | SGI NUMAlink 4 Interconnect, 6.4GB/sec, latency $< 1\mu s$ |
| Cray XD1 | 12 AMD Opteron 64bit Single/Dual Core | 6 Xilinx Virtex-4 | RapidArray Interconnect, 4-8GB/sec, latency $< 1.7\mu s$ |
| FHPCA Maxwell | 1 2.8GHz Xeon | 2 Xilinx Virtex4 | on-board PCI-X, 600MB/s, rocketIO for blades, 2.5 Gbits/s |

**Table 2:** *FPGA based Super Computers*

of being reconfigurable during execution. Today, *RC* is limited to some sort of reconfigurable application specific system design. Nonetheless the potential is great and heavy research is done especially in the *HPC* area. It is expected that in the near future it will be possible to dynamically create hardware during execution. An example would be that instantiating a new object results in synthesizing new hardware. The long term goal of *RC* is to replace todays von-Neumann architecture with a No-Instrucion-Set-Computer, a machine that is completely data-stream driven. But this is not to be expected soon.

## References


[1] S. Park, B. Henz, and D. Shires. Reconfigurable computing for high performance computing computational science. In *HPCMP-UGC '07: Proceedings of the 2007 DoD High Performance Computing Modernization Program Users Group Conference*, pages 350–358, Washington, DC, USA, 2007. IEEE Computer Society.

[2] R. Hartenstein. Reconfigurable supercomputing: Hurdles and chances. invited article. In *ICS '06: Proceedings of the 20th annual international conference on Supercomputing*, 2006.

[3] R. Hartenstein. A decade of reconfigurable computing: a visionary retrospective. In *DATE '01: Proceedings of the conference on Design, automation and test in Europe*, pages 642–649, Piscataway, NJ, USA, 2001. IEEE Press.

[4] Gerald Estrin. Organization of computer systems - the fixed plus variable structure computer. *Proc. Western Joint Computer Conf.*, pages 33–40, 1960.

[5] Gerald Estrin. Reconfigurable computer origins: the ucla fixed-plus-variable (f+v) structure computer. *IEEE Ann. Hist. Comput.*, 24(4):3–9, 2002.

[6] D. Isaaca D. Pellerin G. Steiner, K. Shenoy. Algorithmic acceleration through automated generation of fpga coprocessors. *Embedded Solutions for Programmable Processing Designs*, 3:18–22, 2006.

[7] David Pellerin and Scott Thibault. *Practical FPGA Programming in C.* Prentice Hall, Upper Saddle River, NJ, USA, 2005.