Outline

1. Introduction
   1.1. Basic Terms
   1.2. Reconfigurable Computing Platforms
   1.3. Why “Application-Specific”? 
2. Architecture Studies
   2.1. ASIF
   2.2. ASTRA
3. Applications
   3.1. Interleaving
   3.2. Reconfigurable (De)Interleaver
4. Tools
   4.1. Template-Based Design
   4.2. VTR
   4.3. Archimed and Pythagor
   4.4. CustArD
5. Summary and Conclusions
1. Introduction
What Does “Reconfigurable Computing” Mean?

Reconfigurable device (reconfigurable processing unit, RPU) is a hardware device able to adapt to the application.

Reconfigurable computing is defined as the study of computation using reconfigurable devices.

Configuration is the process of changing the structure of a reconfigurable device at start-up-time.

Reconfiguration is the process of changing the structure of a reconfigurable device at run-time.

FPGA is the most common type of RPU.

**Gate-Array:** Logic (transistors) is pre-fabricated, interconnect is added later to implement customer-specific functionality. Both steps are done in the fab. NRE cost reduction, since master wafers fabrication costs are shared among many customers.

**FPGA:** Logic (look-up-tables) and interconnect are pre-fabricated but not configured, the customer gets an “empty” device and can determine its functionality by configuring it according to the own requirements (hence “field-programmable”).

**Related terms:** Custom Computing Machines (CCM), Reconfigurable Logic (RL), Field-Programmable Logic (FPL), …
1. Introduction

2. Architecture Studies

3. Applications

4. Tools

5. Summary and Conclusions

1.1. Basic Terms

FPGA: Basic Concepts

Logic block: universal logic module, in most cases a Look-up-Table (LUT)

Connection block, CB: configurable interconnect (logic to routing channel)

Switch box: configurable interconnect (routing channel to routing channel)
1. Introduction

1.1. Basic Terms

FPGA: Illustrating the Principle

- different logic functions using the same hardware
- functionality is changed by reconfiguration: restructuring the hardware, not changing the software
1.1. Basic Terms

How Do We Reconfigure the Device?

Configuration memory (SRAM)

Decoder

LUT

Routing channel

Switch box

Connection block

Routing channel

Configuration

c = a \oplus b

0 1 1 0

2:1

Switch box

to LUT

0 1 0 0

0 1 0 0

0 1 0 0

0 1 0 0

0 1 0 0

0 1 0 0

0 1 0 0

0 1 0 0

0 1 0 0

0 1 0 0

0 1 0 0

0 1 0 0

0 1 0 0

0 1 0 0

0 1 0 0
1. Introduction
2. Architecture Studies
3. Applications
4. Tools
5. Summary and Conclusions

1.1. Basic Terms

How Do We Reconfigure the Device?

Configuration memory (SRAM)

Decoder

LUT

Routing channel

Switch box

Connection block

Routing channel

Configuration

to LUT

\[ c = a \lor b \]
How Do We Reconfigure the Device?

Configuration memory (SRAM)

Decoder

$c = a \lor b$

Switch box

Connection block

Routing channel

LUT

Routing channel to LUT
How Do We Reconfigure the Device?

Configuration memory (SRAM)

Decoder

Switch box

Routing channel

LUT

Connection block

Routing channel

Configuration

c = a ∨ b
Modern Devices Are Much More Sophisticated

- hierarchical interconnect (LUTs are grouped into clusters, fast local interconnect, slower inter-cluster-interconnect, several clustering levels)
- fracturable LUTs
- embedded memories
- pre-fabricated IP modules
  - transmitter
  - (multi)processor cores
  - PLL, DLL
  - specialized digital signal processing (DSP) logic blocks
  - standard interface modules (PCIe, USB, ...)
  - ...

- Reconfigurable Systems-on-Chip (SoC) or even Multi-Processor System-on-Chip (MPSoc)
1.2. Reconfigurable Computing Platforms

**Hierarchical FPGA**

- extendable to more than 2 hierarchy levels
- interconnect gets slower with every additional hierarchy level (fast local vs. slower global)
Fracturable LUT (Altera/Intel)

ALM stands for “Adaptive Logic Module”
1.2. Reconfigurable Computing Platforms

Xilinx DSP48E2 Block

[Diagram of Xilinx DSP48E2 Block with labels for inputs and outputs such as B, A, D, C, CARRYIN, OPMODE, CARRYINSEL, BCIN*, ACIN*, PCIN*, CARRYCASCIN*, MULTISIGNIN*, CARRYCASCOUT*, MULTISIGNOUT*, X, Y, Z, INMODE, CARRYIN, W, ALUMODE, RND, XOR OUT, P, 17-Bit Shift, PATTERNDETECT, PATTERNBDETECT, CARRYINSEL, CARRYIN, MULT 27x18, Dual A, D, and Pre-adder, Dual B Register, etc.]
1. Introduction

1.2. Reconfigurable Computing Platforms

Xilinx Zynq UltraScale+ RFSoc (www.xilinx.com)

- **Processing System**
  - **Application Processing Unit**
    - Arm® Cortex®-A53
    - NEON™ Floating Point Unit
    - 32KB I-Cache w/Parity
    - 32KB D-Cache w/ECC
    - Memory Management Unit
    - Embedded Trace Macrocell
    - GIC-400
    - SCU
    - CCI/SMMU
    - 1MB L2 w/ECC
  - **Real-Time Processing Unit**
    - Arm® Cortex®-R5
    - Vector Floating Point Unit
    - 128KB TCM w/ECC
    - 32KB I-Cache w/ECC
    - 32KB D-Cache w/ECC
    - GIC

- **DDR Controller**
  - DDR4/3/3LLPDDR4/3 ECC Support
  - 256KB OCM with ECC

- **System Control**
  - DMA
  - Timers & WDT
  - Resets
  - Clocking
  - Debug

- **Security**
  - Config
  - AES Decryption
  - Authentication
  - Secure Boot
  - TrustZone
  - Voltage/Temp Monitor

- **Platform Management Unit**
  - Power
  - System Management

- **High-Speed Connectivity**
  - DisplayPort
  - USB 3.0
  - SATA 3.0
  - PCIe Gen2
  - PS-GTR

- **General Connectivity**
  - GigE
  - CAN
  - UART
  - SPI
  - Quad SPI NOR
  - NAND
  - SD/eMMC

- **Programmable Logic**
  - RF Signal Chain
    - Up to 5GSPS RF-ADCs
    - SD-FEC
  - High-Speed Connectivity
    - 33G SerDes
    - 100G Ethernet MAC
    - 100G Interlaken
    - PCIe® Gen4
  - General-Purpose I/O
    - High-Performance HPIO
    - High-Density HDIO
  - Storage & Signal Processing
    - Block RAM & UltraRAM
    - DSP

- **System Monitor**
1.3. Why “Application Specific”?

Standardization vs. Specialization

General-purpose FPGAs suite almost any application domain but imply a significant overhead. This issue is addressed by the manufacturers:

- different product families
  - memory-oriented
  - logic-oriented
  - DSP-oriented
  - processor-centered
  - communication-centered
  - AI-support

- different device complexity within one family:
  - more or less logic blocks, memory, DSP blocks etc.
  - different technology nodes for the same device architecture

Still, commercially available FPGAs remain general-purpose computing engines
Case Study 1: Xilinx Inc.

- CPLD device family
  - CoolRunner-II
- 4 FPGA device families
  - Spartan-6 and -7 (low-cost)
  - Artix-7 (low-cost)
  - Kintex-7, Kintex UltraScale, Kintex UltraScale+ (mid-range)
  - Virtex-5, -6 and -7, Virtex UltraScale, Virtex UltraScale+ (high-end)
- SoC and MPSoc
  - Zynq-7000
  - Zynq UltraScale+
- Adaptive Compute Acceleration Platform (ACAP)
  - Versal

All product names are registered trademarks of Xilinx Inc.
1.3. Why “Application Specific”?

Case Study 2: Intel

- 3 FPGA/CPLD device families
  - MAX II (low-cost)
  - MAX V (low-cost)
  - MAX 10 (non-volatile)

- 4 FPGA device families
  - Cyclone III, IV, V and 10 (low-cost)
  - Agilex F (general purpose), I (interface) and M (memory)
  - Arria I, II and V (mid-range)
  - Stratix III, IV, V and 10 (high-end)

- SoC and MPSoc
  - Cyclone V
  - Arria V and 10
  - Stratix 10
  - all Agilex product lines

All product names are registered trademarks of Intel Corporation
Mainstream vs. Application-Specific

Mainstream Reconfigurable Computing is application-specific by definition:

- RPUs can be configured to suit the needs of almost any application
- but at high price, since a lot of overhead is involved to make them generally applicable

Application-Specific Reconfigurable Computing goes one step further:

- stick to a few (or even a single) application domain (or even just a couple of applications)
- reduce the overhead as far as possible
2. Architecture Studies
2.1. ASIF

Application-Specific Inflexible FPGA

Basic approach:

- use hierarchical FPGA architecture with variable number of levels
- optimize interconnect to route a predefined set of netlists only
- replace reconfigurable logic blocks by hard-macros (if possible and useful)
- reconfigure the ASIF for each netlist individually (time-multiplexing)

70 % area reduction compared to the general-purpose hierarchical FPGA implemented in the same technology

Farooq, Marrakchi, Mehrez, *Tree-based Heterogeneous FPGA Architectures*, Springer 2012
A 4-Level Hierarchical FPGA
A 4-Level Hierarchical FPGA
A 4-Level Hierarchical FPGA
A 4-Level Hierarchical FPGA
A 4-Level Hierarchical FPGA
Heterogeneous Tree-Based Architecture

Switchbox (reduced)

Switchbox

Switchbox

Switchbox

Switchbox (reduced)

Switchbox

Switchbox

Switchbox

ADD DSP


Switchbox

Switchbox

Switchbox

Switchbox (reduced)

Switchbox

Switchbox

Switchbox

Switchbox

Cluster level 1

Cluster level 2

Cluster level 3

Cluster level 4
2.2. ASTRA

Advanced Space-Time Reconfigurable Architecture

Basic approach:

- keep the classical island-style architecture but
- separate data flow from control flow
- make logic blocks operating on words instead of single bits
- implement global interconnect exclusively for control
- allow data transfers to adjacent blocks only (reduces the interconnect overhead dramatically)

- implement additional registers to switch between parallel/serial computation within the block (hence "space-time reconfigurable")

2.2. ASTRA

Top View
### 2.2. ASTRA

#### Some Benchmarks

<table>
<thead>
<tr>
<th>Application</th>
<th>ASTRA-2, temporal Area/mm²</th>
<th>ASTRA-2, spatial Area/mm²</th>
<th>ASIC, spatial mm²</th>
</tr>
</thead>
<tbody>
<tr>
<td>VITERBI-Decoder (64 states, 4-bit input)</td>
<td>1.54</td>
<td>2.25</td>
<td>0.20</td>
</tr>
<tr>
<td>8-pt FFT (8-bit input)</td>
<td>1.27</td>
<td>3.30</td>
<td>0.13</td>
</tr>
<tr>
<td>FIR filter (16 tap, 8-bit input)</td>
<td>0.45</td>
<td>1.12</td>
<td>0.09</td>
</tr>
</tbody>
</table>

11–25× silicon area of an equivalent ASIC implementation (for commercial FPGAs this ratio is ≥ 35–40× if no special blocks are used)
3. Applications
Interleaving in Digital Communication Systems

Interleaving schemes

- Convolutional interleaving
- Block interleaving
  - Matrix interleaving
  - Random interleaving
  - Algebraic interleaving

- Present in almost any relevant standard family: IEEE 802.11, DAB, DVB, LTE, ...
- Used to achieve several goals: Improve the quality of forward error correction, better use of frequency diversity, ...
- Top-view architecture: Memory which is read and written using different address sequences ➔ address generation is the key
- Most address generation schemes can be implemented using only three basic operations: permutation, transposition and bit rotation
Universal Reconfigurable (De)Interleaver

Case study for DAB, DVB, IEEE 802.11a/g and UMTS (HSDPA):

- Does not have much in common with traditional FPGA architectures, still it is (application-specific) reconfigurable computing

### The Same Study Mapped to ASTRA

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Universal deinterleaver (last slide)</th>
<th>ASTRA run-time reconfigurable (36 LB)</th>
<th>ASTRA statically reconfigurable (100 LB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration vector</td>
<td>905 bit</td>
<td>5,2 Kbit</td>
<td>14 Kbit</td>
</tr>
<tr>
<td>External configuration memory size</td>
<td>94 Kbit</td>
<td>542 Kbit</td>
<td>1,5 Mbit</td>
</tr>
<tr>
<td>(104 configurations)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State register size</td>
<td>928 Bit</td>
<td>576 Bit</td>
<td>1,6 KBit</td>
</tr>
<tr>
<td>Area (CMOS090)</td>
<td>0,2 mm$^2$</td>
<td>0,2 mm$^2$</td>
<td>0,6 mm$^2$</td>
</tr>
<tr>
<td>Configuration loading time</td>
<td>5 clock cycles</td>
<td>variable</td>
<td>variable</td>
</tr>
<tr>
<td>Pipeline depth</td>
<td>9 stages</td>
<td>4 stages (variable)</td>
<td>4 stages (variable)</td>
</tr>
</tbody>
</table>
4. Tools
4.1. Template-Based Design

Conventional Design Flow
Issues with Application-Specific Design

- While front-end (functional verification, synthesis) can be kept generic, the design flow starts to be vendor-specific starting with the technology mapping step.
- Traditional design flow optimizes the given application towards the underlying architecture (Intel, Xilinx, ...).
- Application-specific approach optimizes the architecture towards the given application (domain):
  1. Start with a more or less generic architecture
  2. Map your application and check the resource utilization
  3. Remove underutilized resources, optimize congested resources
  4. Iterate if needed
  5. Once finished, proceed with a final run using the optimized architecture instance
- Steps 1–4 are called “design space exploration and tuning”, step 5 is called “instance and test generation”.
- Template-based design
Two Phases of the Template-Based Design

...to be found in all following case studies!

Design space exploration and tuning (Phase 1)

Instance and test generation (Phase 2)

Shacham, Azizi, Wachs et al., Rethinking Digital Design: Why Design Must Change, IEEE Micro, Vol. 30(6), 2010
4.2. VTR

**Design Flow**

- open source (VTR = Verilog To Routing)
- based on standard FPGA architecture
- can handle most aspects of the modern devices
  - heterogeneous blocks
  - fracturable LUTs
  - complex logic blocks
  - special purpose cells (memories, DSP, ...)
- suitable for both architecture and algorithmic research

4.2. VTR

Architecture Description Example

```xml
<pb_type name="ble">
    <input name="in" num_pins="4"/>
    <output name="out" num_pins="1"/>
    <clock name="clk"/>
</pb_type>

<pb_type name="lut_4" blif_model=".names" num_pb="1" class="lut">
    <input name="in" num_pins="4" port_class="lut_in"/>
    <output name="out" num_pins="1" port_class="lut_out"/>
</pb_type>

<pb_type name="ff" blif_model=".latch" num_pb="1" class="flipflop">
    <input name="D" num_pins="1" port_class="D"/>
    <output name="Q" num_pins="1" port_class="Q"/>
    <clock name="clk" port_class="clock"/>
</pb_type>

<interconnect>
    <direct input="lut_4.out" output="ff.D"/>
    <direct input="ble.in" output="lut_4.in"/>
    <mux input="ff.Q lut_4.out" output="ble.out"/>
    <direct input="ble clk" output="ff clk"/>
</interconnect>
</pb_type>
```
4.3. Archimed and Pythagor

Design Flow

- former research project at Philips Research
- **not** based on standard FPGA architecture
  - any type of logic can be modeled
  - produces ready-for-manufacturing layout
  - was used for several real-world designs
  - includes test data generation
- requires external synthesis/technology mapping tools

4.3. Archimed and Pythagor

Pythagor in Action
4.4. CustArD

Design Flow

- template-based design methodology
- extremely flexible architecture template

Bostelmann, Sawitzki, A Heterogeneous Architecture Template for Application Domain Specific Reconfigurable Logic, Austrochip’2015.
Architecture Template

Grid representation

Tree representation

- Block is the only basic data structure which can be instantiated as core, grid or repeater
- Plug-in system with a simple interface (adding a new algorithm to the existing framework ➠ two methods in Python)
CustArD in Action

A placed and (partially) routed 4-bit counter circuit
5. Summary and Conclusions
To Round It Up . . .

- Application-specific reconfigurable computing is a promising approach to design digital systems
  - proven by numerous studies
  - well-known in the research community, less appreciated by the industry
- Is not suitable as a replacement for the mainstream, only useful if you need to squeeze the last bit out of your reconfigurable design
- Can be a nice solution, if you do not the full flexibility of the platform-FPGA
- Stable tools and design flows are still a big issue
  - Need to optimize architecture as well as application as well as design automation algorithms!
  - Template-based design is the solution
Thank you for your attention!
Image Credits


