Reconfigurable Computing

Design and Implementation

Chapter 4.1

Prof. Dr.-Ing. Jürgen Teich
Lehrstuhl für Hardware-Software-Co-Design
In System Integration
Reconfigurable devices (RD) are usually used in three different ways:

1. **Rapid Prototyping**: The RD is used as emulator for a circuit to be produced later as an ASIC. The emulation process allows for testing the correctness of the circuit, sometimes under real operating conditions before production.

The APTIX-System Explorer, the EVE ZeBu-Server and the Synopsys CHIPit System are three examples of emulation platforms.
2. Non-frequently reconfigurable systems: The RD is used as application-specific device similar to an ASIC. However, the possibility of upgrading the system by means of reconfiguration is given.

Such systems are used as prototyping platforms, but can be used in a running environment as well.
2. Non-frequently reconfigurable systems: The RD is used as application-specific device similar to an ASIC. However, the possibility of upgrading the system by means of reconfiguration is given. Such systems are used as prototyping platforms, but can be used in a running environment as well.

Source: Xilinx, Inc.
3. Frequently reconfigurable systems:
   Usually coupled with a processor, the RD is used as an accelerator for time-critical parts of applications. The processor accesses the RD using function calls. The reconfigurable part is usually a PCI-board attached to the PCI-bus. The communication is useful for configuration and data exchange.

Examples are the Digilent XUPV5 Board, the Raptor 2000, Xilinx ML605 Board, the Nallatech BenOne PCIe. More and more stand-alone frequently reconfigurable systems are appearing.
The three ways of using a reconfigurable systems can be classified in two big categories:

1. **Statically reconfigurable systems.** The computation and reconfiguration is defined once at compile-time. This category includes rapid prototyping systems, non-frequently reconfigurable systems as well as some frequently reconfigurable systems.

2. **Dynamically or run-time reconfigurable systems.** The computation and reconfiguration sequences are not known at compile-time. The system reacts dynamically at run-time to computation and therefore, to reconfiguration requests. Some non-frequently reconfigurable systems as well as most frequently reconfigurable systems belong to this category.
The computation in a reconfigurable system is usually done according to the figure aside. The processor controls the complete system.

1) First, the required input data for the RD is downloaded to the RD memory.

2) Then, the RD is configured to perform a given function over a period of time.

3) The start signal is given to the RD to start computation. At this time, the processor also computes its data segment in parallel to the RD.
4) Upon completion, the RD acknowledges the processor.

5) The processor collects the computed data from the RD memory. If many reconfigurations have to be done, then some of the steps from 1) to 5) should be reiterated according to the application's needs. A barrier synchronisation mechanism is usually used between the processor and the RD. Blocking access should also be used for the memory access between the two devices.
Devices like the Xilinx Virtex II/II-Pro and the Altera Excalibur feature one or more soft or hard-macro processors. Therefore, the complete system can be integrated in only one device.

The reconfiguration process can be:

- **Full**: The complete device has to be reconfigured (operation interrupt occurs).
- **Partial**: Only part of the device is configured while the rest keeps running.
For a dynamically reconfigurable system with only *full reconfiguration* capabilities, functions to be downloaded at run-time are developed and stored in a database. No geometrical constraint restrictions are required for the function.

For a stand alone system with *partial reconfiguration* capabilities, modules represented as rectangular boxes are pre-computed and stored in memory. During relocation, the modules are assigned to a position on the device at run-time.

In both cases, modules to be downloaded at run-time are *digital circuit modules* which are developed according to digital circuit design rules.
Design Flow
The implementation of a reconfigurable system is a Hardware/Software Co-Design process which determines:

- The software part, that is the code segment to be executed on the processor. The development is done in a programming language using common compiler tools. We will not pay much attention to this part.

- The hardware part, that is the part to be executed on the RD. This is the focus of this section.

- The interface between software and hardware.
Design Flow – Coarse-grained RC

- The implementation of a coarse-grained RD is done using vendor-specific languages and tools. This is usually a C-like language with the corresponding behavioral or structural compilers.

- For the coarse-grained architectures presented in the previous chapter, the languages and tools are summarised in the table below.

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Language</th>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PACT-XPP</td>
<td>NML (Structural)</td>
<td>XPP-VC</td>
<td>C -&gt; NML -&gt; configuration format</td>
</tr>
<tr>
<td>Quicksilver ACM</td>
<td>Silver C</td>
<td>InSpire SDK</td>
<td>C -&gt; SilverC -&gt; configuration format</td>
</tr>
<tr>
<td>NEC DRP</td>
<td>C</td>
<td>DRP Compiler</td>
<td>C -&gt; configuration format</td>
</tr>
<tr>
<td>IPFLEX DAP/DNA</td>
<td>C/Matlab</td>
<td>DAP/DNA FW</td>
<td>C/Matlab -&gt; configuration format</td>
</tr>
<tr>
<td>PicoChip</td>
<td>C</td>
<td>PicoChip Toolchain</td>
<td>C -&gt; configuration format</td>
</tr>
<tr>
<td>TCPA (FAU)</td>
<td>C, Java, X10</td>
<td>PARO Compiler</td>
<td>C -&gt; PAULA -&gt; assembly code</td>
</tr>
</tbody>
</table>
The implementation flow of an FPGA design is shown. It is a modified ASIC design flow divided into 5 steps.

The steps (design entry, functional simulation, place and route) are the same for almost all digital circuits. Therefore, they will be presented only briefly.

In the synthesis step, the FPGA design flow differs from other synthesis processes. We will therefore consider some details of FPGA synthesis, in particular the LUT-technology mapping which is proper to FPGAs.
The design entry can be done either using:

- A schematic editor: Schematic description is done by selecting components from a (target device) and graphically connecting them together to build complex modules. Finite State Machines (FSM) can also be entered graphically or as a table. **Drawback:** Only structural description of circuits. Behavioral description is not possible.

- A Hardware Description Language (HDL): allows for structural as well as behavioral description of complex circuits.
  - The behavioral description is useful for designs containing loops, Bit-vectors, ADT, FSMs.
  - The structural description emphasizes the hierarchy in a given design.
After the design entry, functional simulation is used to logically test the functionality of the design.

- A testbench provides the design under test with inputs for which the reaction of the design is known.
- The outputs of the circuit are observed on a waveform and compared to the expected values.
- For simulation purpose, many operations can be used (mod, div, etc.) in the design.
- However, only part of the code which is used for simulation can be synthesized later.
- The most commonly used HDLs are:
  - VHDL (behavioral, structural)
  - Verilog (behavioral, structural)
  - Some C/C++-like languages (SystemC, HandelC, etc.)
The design is compiled and optimized. All non-synthesizable data types and operations must be replaced by equivalent synthesizable code.

- The design is first translated into a set of Boolean equations which are then minimized.
- Technology mapping then assigns functional modules to library elements. The technology mapping on FPGAs is called LUT-technology mapping.
- The result of the technology mapping is a netlist which provides a list of components used in the circuit as well as their interconnections.
- There exist many formats to describe a netlist. The most popular is the EDIF (Electronic Design Interchange Format).
The netlist provides only information about the components and their interconnections in a given design. *Place and route* tools are then used to:

- Assign locations to the components,
- Provide communication paths to realize the interconnect.

The place and route steps involve solving optimization problems to minimize a cost function. The most important cost functions are:

- Clock frequency,
- Latency,
- Cost (area used).
The last step in the design process is the generation of the configuration stream also known as *bitstream*. It describes:

- The value of each LUT, that is the set of bits used to configure the function of a LUT.

- The interconnection configuration describes:
  - The inputs and outputs of the LUTs,
  - The value of the multiplexers, and
  - how the switches should be set in the interconnection matrix.

All information regarding the functionality of LUTs, multiplexers and switches are available after the place and route step.
Exercise: Implement a Modulo 10-counter on a symmetrical FPGA with 2x2 Logic Blocks (LB).

The structure of a LB is given in the picture aside. It consists of:

- 2 2-inputs LUTs
- 2 edge-triggered T-Flipflops

The goal is to minimize

- area
- latency
Truth table of the modulo 10 counter. $z$ describe the states while $T$ describe the inputs of the T-FFs

<table>
<thead>
<tr>
<th>$z$</th>
<th>$z'$</th>
<th>$T_4$</th>
<th>$T_3$</th>
<th>$T_2$</th>
<th>$T_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0001</td>
<td>0010</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>0011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0011</td>
<td>0100</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0100</td>
<td>0101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0101</td>
<td>0110</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0110</td>
<td>0111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0111</td>
<td>1000</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1000</td>
<td>1001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1001</td>
<td>0000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1010</td>
<td>xxxx</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1011</td>
<td>xxxx</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1100</td>
<td>xxxx</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1101</td>
<td>xxxx</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1110</td>
<td>xxxx</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1111</td>
<td>xxxx</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Karnaugh-minimization of the functions $T_1$, $T_2$, $T_3$, and $T_4$.
Design Flow – FPGA – Example

Common product term

\[ T_1 = 1 \]
\[ T_2 = z_1 \cdot \overline{z}_4 \]
\[ T_3 = z_1 \cdot z_2 \]
\[ T_4 = z_1 \cdot z_4 + z_1 \cdot z_2 \cdot z_3 \]
Design Flow – FPGA – Example

Reconfigurable Computing

24