

# Reconfigurable Computing

**Prof. Dr. Christophe Bobda**  
**University of Potsdam**  
**Institute for Computer Science**  
**Chair for Computer Engineering**

---

---

# Chapter 2

## Design and Integration

# Agenda

---

1. In System Integration
2. Design Flow

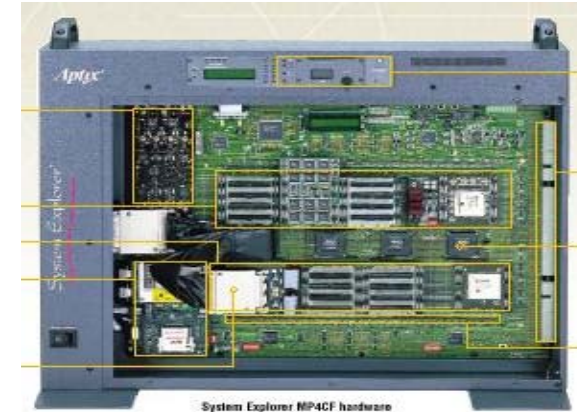
---

# In System Integration

# 1. Rapid Prototyping

- 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 ASIC. The emulation process allows for testing the correctness of the circuit, sometimes under real operating conditions before production.

The APTIX-System Explorer and the ITALTEL Flexbench systems are two examples of emulation platforms.



APTIX System Explorer



ITALTEL FLEXBENCH

# 1. Non Frequent reconfiguration

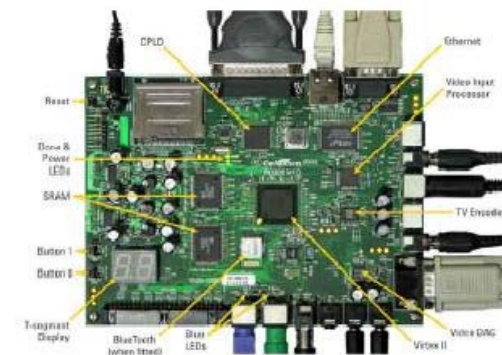
2. **Non-frequently reconfigurable systems:** The RD is used as application specific device similar to ASIC. However the possibility of upgrading the system by means of reconfiguration is given.

Such systems are used as prototyping platform, but can be used as running environment as well.

Examples are:  
The RABBIT System, the celoxica RC100, RC200, RC300, the Nallatech BenADIC.



The Nallatech BenADIC



The Celoxica RC200

# 1. Frequent reconfiguration

### 3. Frequently reconfigurable systems:

Usually coupled with a processor, the RD is used as an accelerator for time-critical parts of applications.

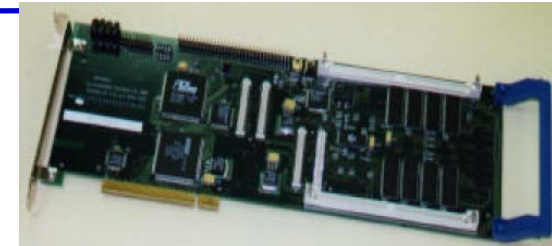
The processor accesses the RD using function calls.

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 Raptor 2000, the celoxica RC1000 and RC2000, the Nallatech Ballynuey.

More and more stand-alone frequently reconfigurable systems are appearing.



The Celoxica RC1000



The Raptor 2000



The Nallatech Ballynuey

# 1. Static and dynamic Reconfiguration

---

- The three ways of using a reconfigurable systems can be classified in two
- big categories:
  1. **Static reconfigurable systems.**

The computation and reconfiguration is defined once at compile time.

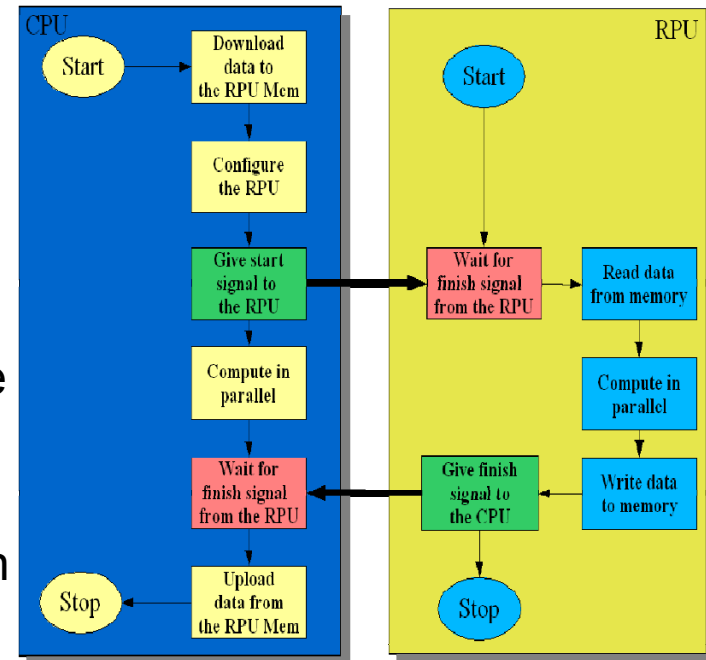
This category encounters the rapid prototyping systems, the non-frequently reconfigurable systems as well as some frequently reconfigurable systems.
  2. **Dynamic or run-time reconfigurable systems.**

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.



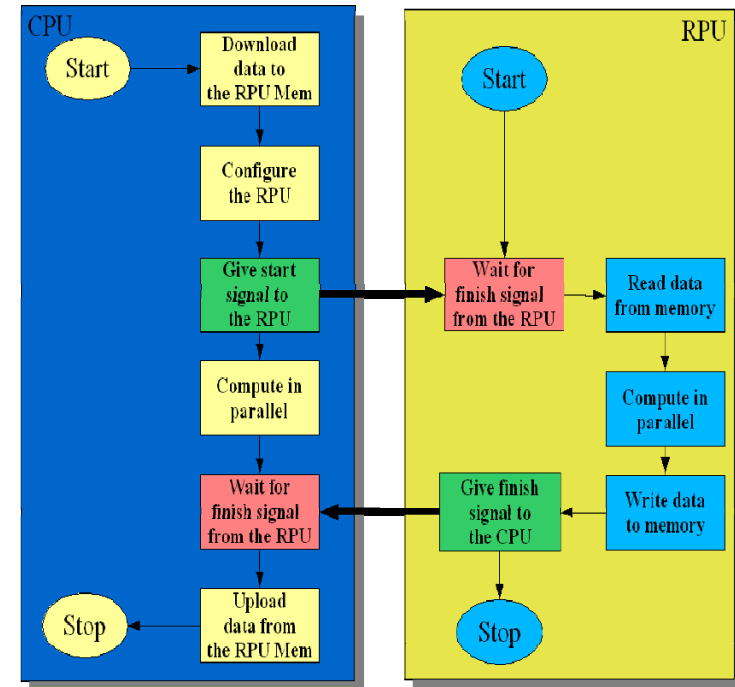
# 1. Computation flow

- The computation in a reconfigurable system is usually done according to the figure aside. The processor controls the complete system.
- 1) It first downloads data to be computed by the RD memory 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.



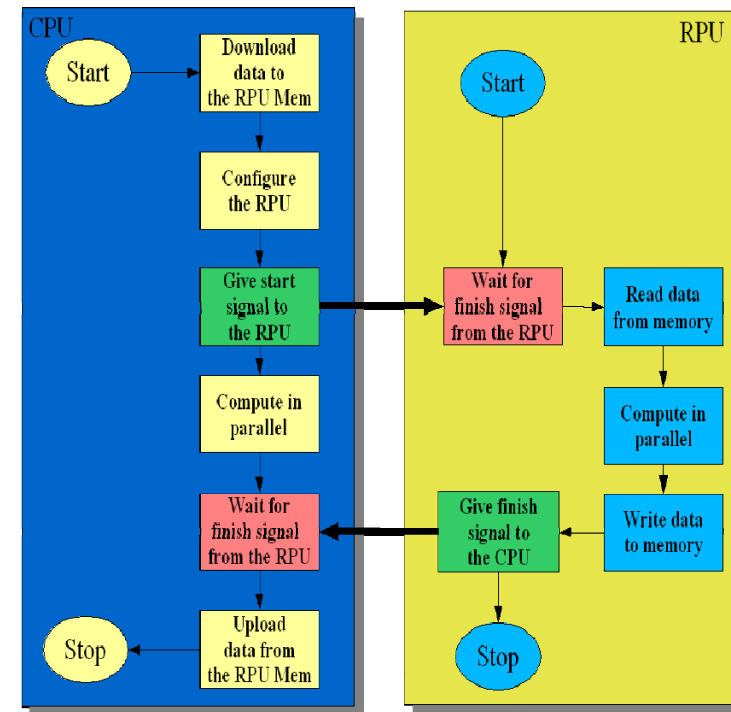
# 1. Computation flow

- 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 need.
  - 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.



# 1. Computation flow

- Devices like the Xilinx Virtex II/II-Pro and the Altera Excalibur feature one or more soft or hard-macro processors.
  - the complete system can be integrated in only one device.
- The reconfiguration process can be:
  - **Full:** The complete device have to be reconfigured. (Operation interruption occurs)
  - **Partial:** Only part of the device is configured while the rest keeps running.

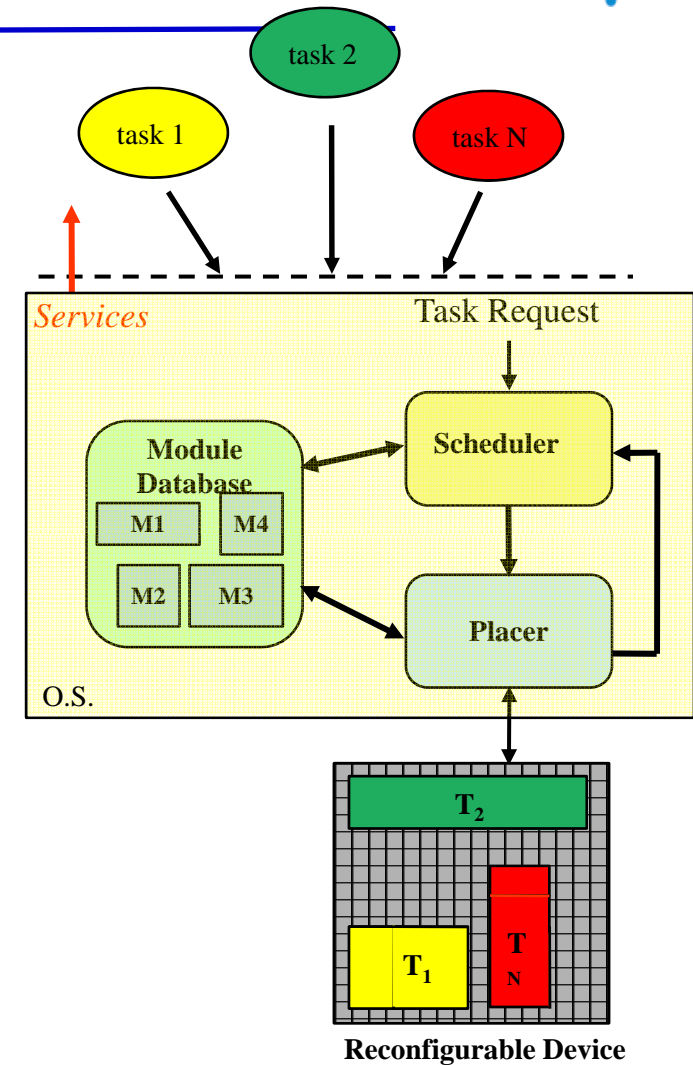


# 1. Computation flow

- Full reconfiguration devices
  - function to be downloaded at run-time are developed and store in a database.
  - No geometrical constraints restriction are required for the function.

- Partial reconfiguration capabilities
  - modules represented as rectangular boxes, are pre-computed and stored in a data base.
  - With relocation, the modules are assigned to a position on the device at run-time

- In the two cases, modules to be downloaded at run-time are digital circuit modules
  - development is done according to digital circuit design rules

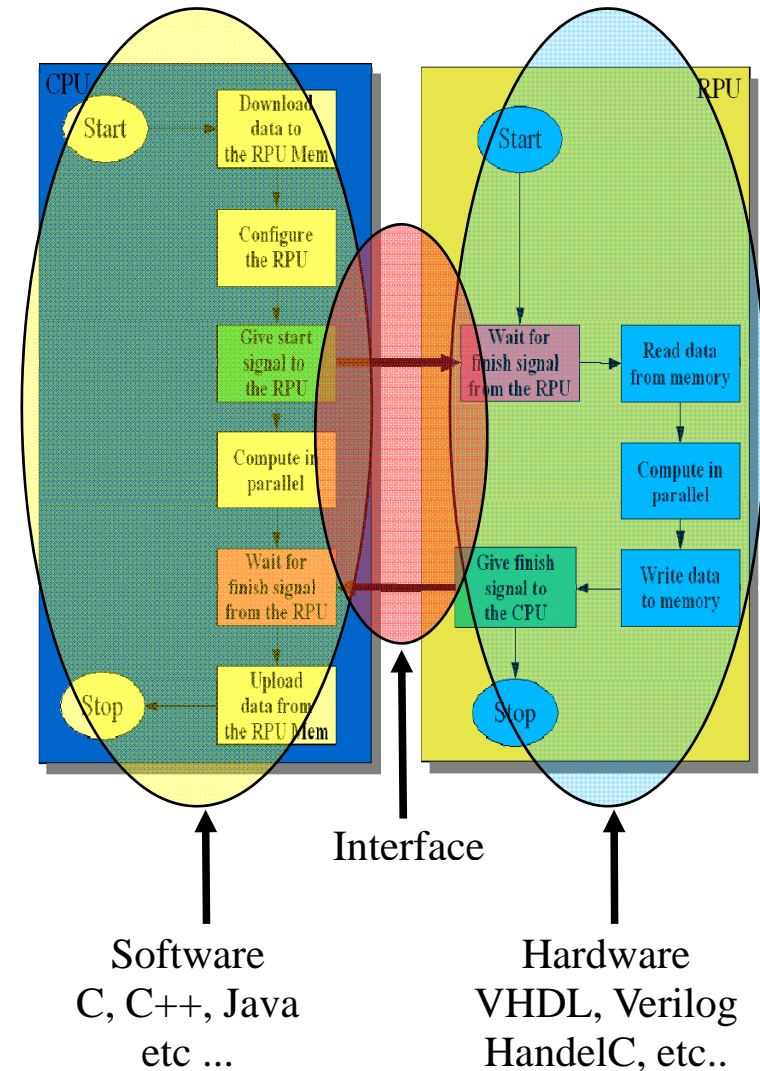


---

# Design Flow

## 2. Hardware/software partitioning

- 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 software language with common 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 target of this section.
  - The interface between software and hardware. This part is not in the scope of this course.



## 2. Coarse-grained RC

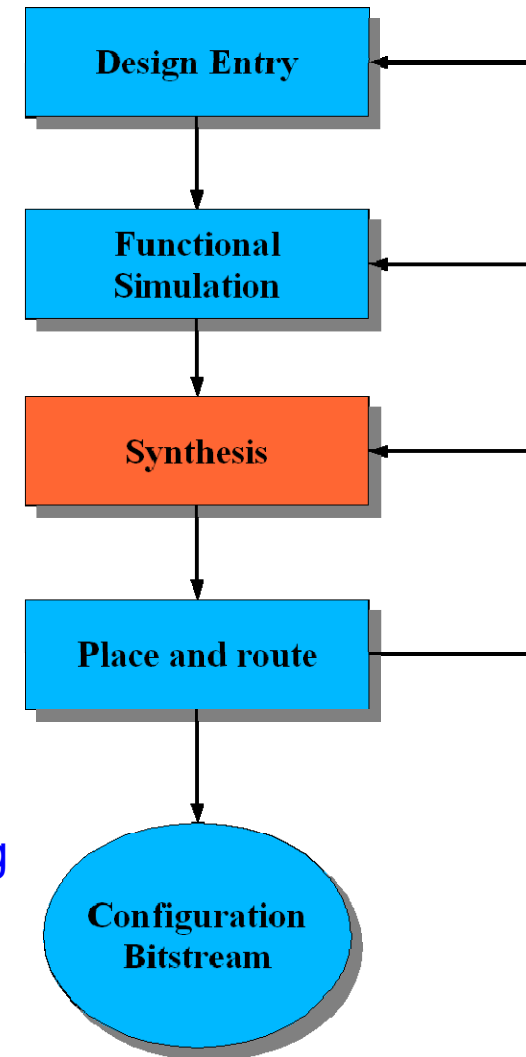
---

- The implementation of a coarse-grained RD is done with vendor specific language and tools.
  - 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.

<b>Manufacturer</b>	<b>Language</b>	<b>Tool</b>	<b>Description</b>
<b>PACT-XPP</b>	NML (Structural)	XPP-VC	C -> NML -> configuration format
<b>Quicksilver ACM</b>	Silver C	InSpire SDK	C -> SilverC -> configuration format
<b>NEC DRP</b>	C	DRP Compiler	C -> configuration format
<b>IPFLEX DAP/DNA</b>	C/Matlab	DAP/DNA FW	C/Matlab -> configuration format
<b>PicoChip</b>	C	PicoChip Toolchain	C -> configuration format

## 2. FPGA Design Flow

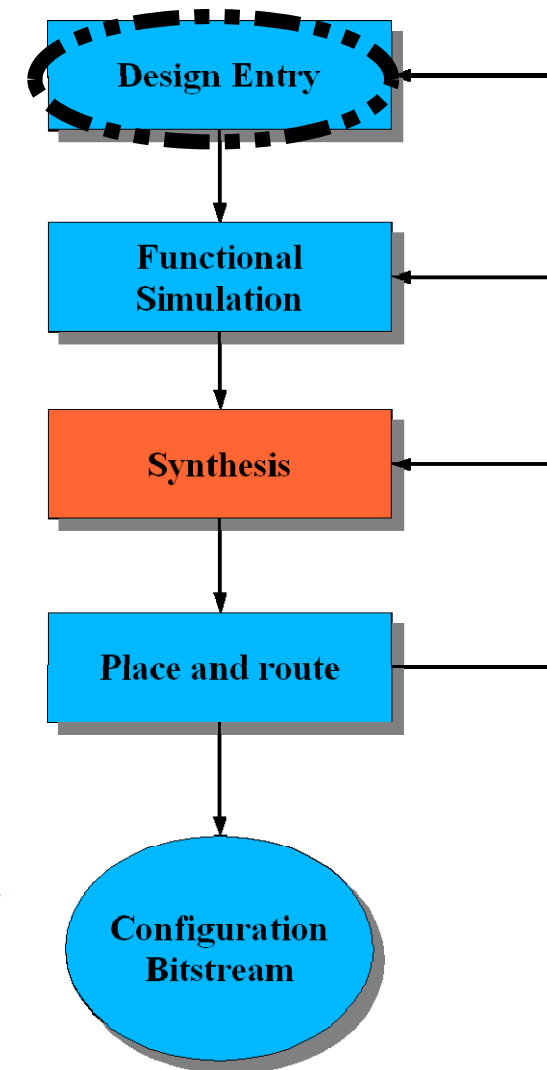
- The implementation flow of an FPGA designs is shown on the figure aside. It is a modifiedASIC design implementation divided in 5 steps.
- The steps (design entry, functional simulation, place and route) are the same for almost all digital circuits.
  - Therefore, they will be presented only briefly.
- FPGA-synthesis differs from other synthesis process in the technology mapping step
  - We will therefore consider some details of FPGA-synthesis, in particular the LUT-technology mapping which is proper to the FPGAs.





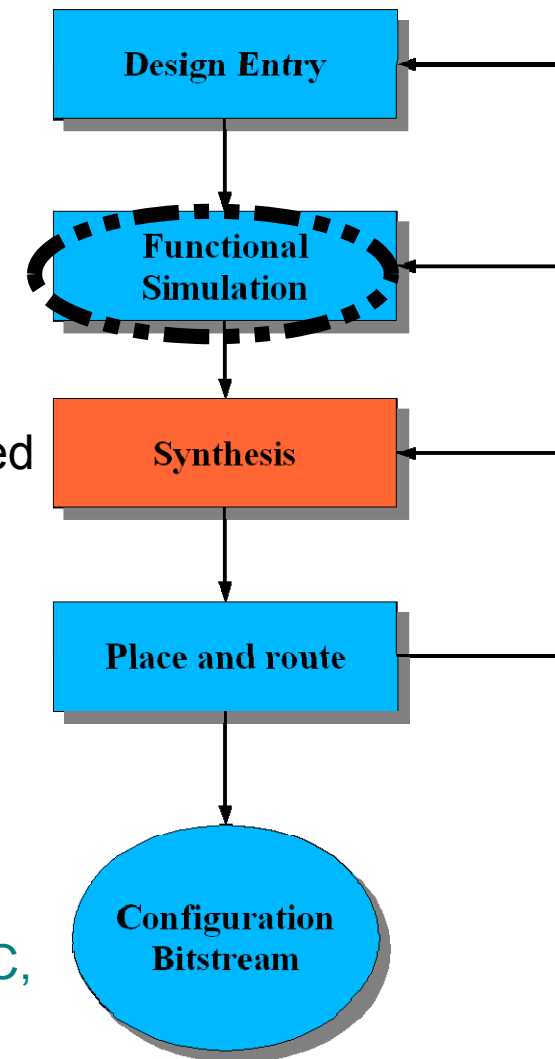
## 2. FPGA Design Flow - Design entry

- The design entry can be done with
  - A schematic editor:
    - Selection components from a (target device) and graphically connecting them together to build complex modules.
      - Finite State machine (FSM) can also be entered by drawing the states and transitions either as graph or as table.
  - A Hardware Description Language (HDL)
    - structural as well as behavioral description of complex circuits.
    - Behavioral description: useful for designs containing loops, Bit-vectors, ADT, FSMs.
    - Structural description: emphasizes the hierarchy in a given design.



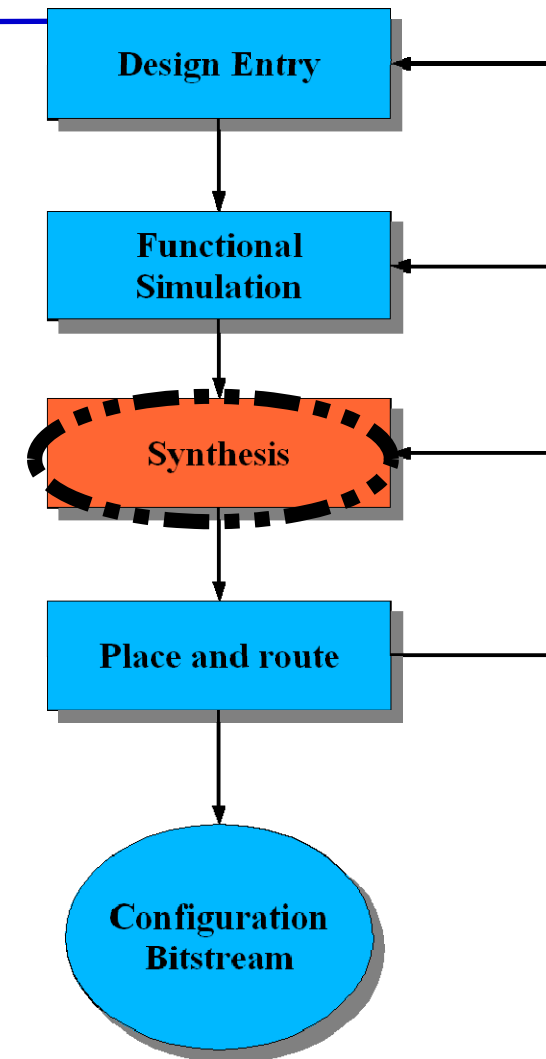
## 2. FPGA Design Flow – Functional Simulation

- After the design entry, the 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 description.
  - However only part of the code which is used for simulation can be synthesized later.
  - The most used HDLs are:
    - VHDL (behavioral, structural)
    - Verilog (behavioral, structural)
    - Some C/C++-like languages (SystemC, HandelC, etc...)



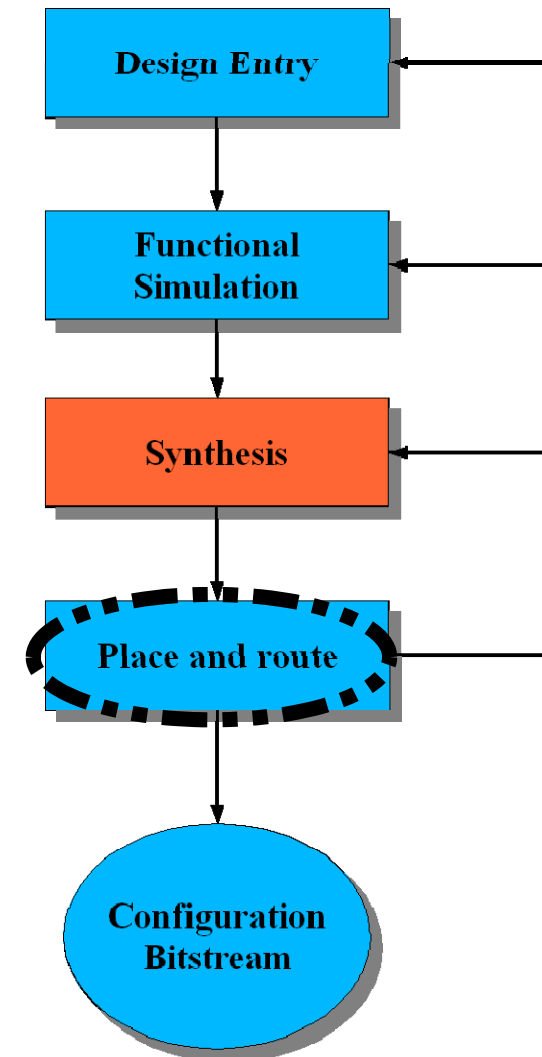
## 2. FPGA Design Flow – Synthesis

- 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 is used to assign the 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)**.
  - Vendor specific netlist also exists. Example: XNF (Xilinx Netlist Format)



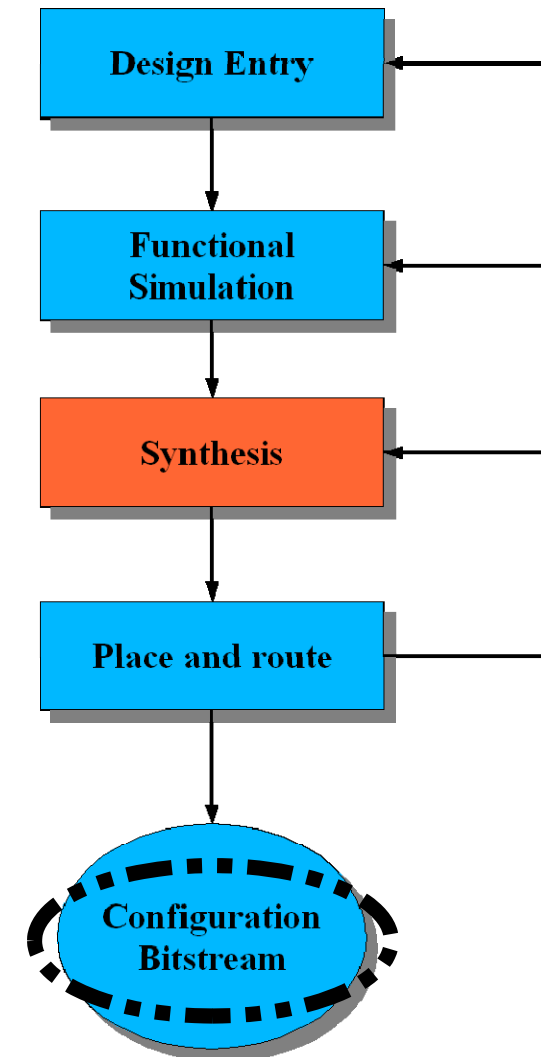
## 2. FPGA Design Flow – Place and route

- The Netlist provides only informations about the components and their interconnections in a given design. The place and route must be used to:
  - Assign locations to the components.
  - Provide communication paths to the interconnections.
- The place and route steps are optimization problems for which some cost must be minimized.
- The most important factors are:
  - The clock frequency.
  - The signal latency.
  - The routing congestion.
  - Etc...



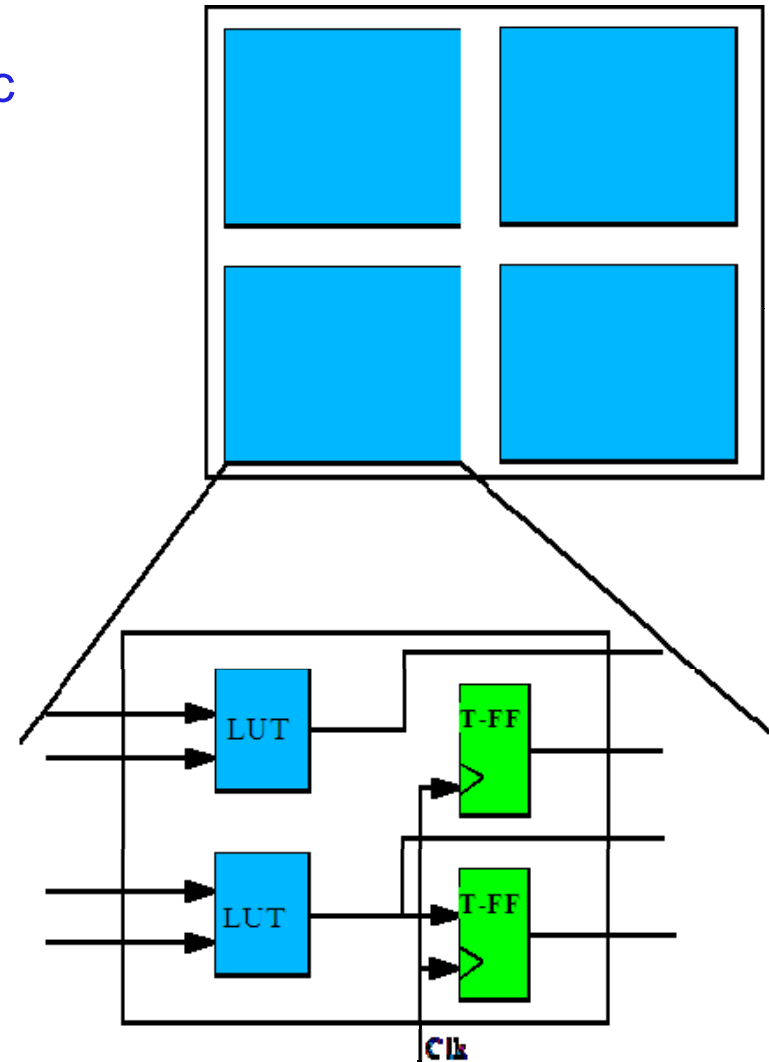
## 2. FPGA Design Flow – Configuration bitstream

- The Last step in the design process is the generation of the configuration stream also known as bitstream. A Bitstream describes:
  - The value of each LUT, that is the set of bits used to configure the function of a LUT.
  - The interconnections configuration describes:
    - The inputs and outputs of the LUTs
    - The value of the multiplexers
    - how the switches should be set in the interconnection matrix
- All informations regarding the functionality of LUTs, multiplexers and switches are available after the place and route step



## 2. FPGA Design Flow – Example

- Implements a Modulo 10-counter on a symmetrical array FPGA with a 2x2 Logic Block (LB) .
- The structure of the LBs is given in the picture aside. It consists of:
  - 2 2-inputs LUT.
  - 2 edge triggered T-Flipflop.
- Your result should minimize
  - The area
  - The latency



# 2. FPGA Design Flow – Example

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

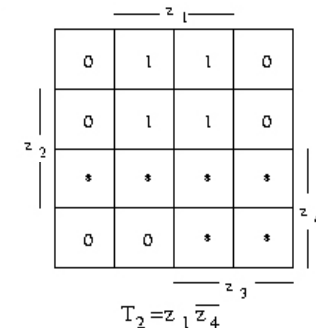
Karnaugh-minimization of the functions  $T_1$ ,  $T_2$ ,  $T_3$ , and  $T_4$

$z$	$z'$	$T_4$	$T_3$	$T_2$	$T_1$
0000	0001	0	0	0	1
0001	0010	0	0	1	1
0010	0011	0	0	0	1
0011	0100	0	1	1	1
0100	0101	0	0	0	1
0101	0110	0	0	1	1
0110	0111	0	0	0	1
0111	1000	1	1	1	1
1000	1001	0	0	0	1
1001	0000	1	0	0	1
1010	xxxx	x	x	x	x
1011	xxxx	x	x	x	x
1100	xxxx	x	x	x	x
1101	xxxx	x	x	x	x
1110	xxxx	x	x	x	x
1111	xxxx	x	x	x	x

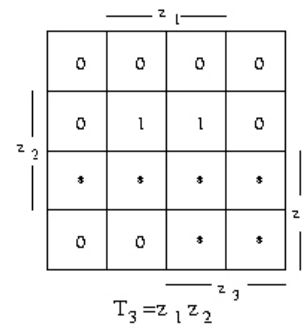
KV-Diagramme

$T_1$ )  $T_1=1$

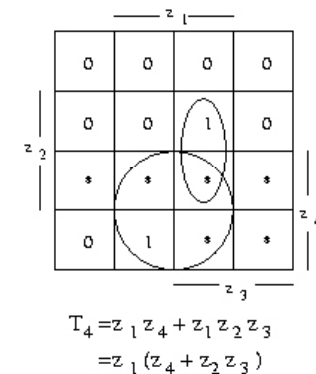
$T_2$ )



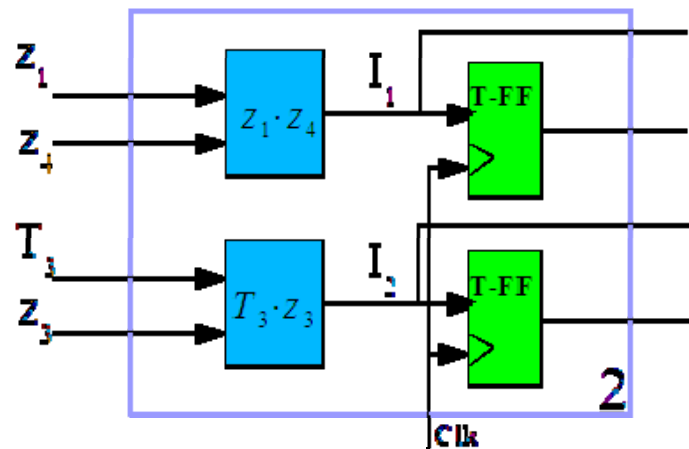
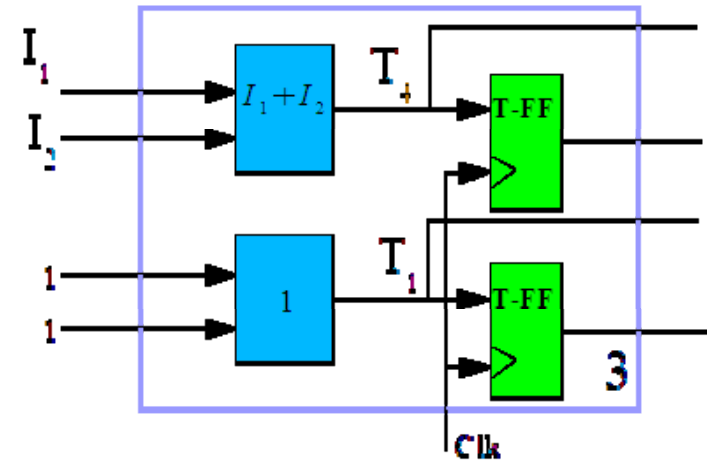
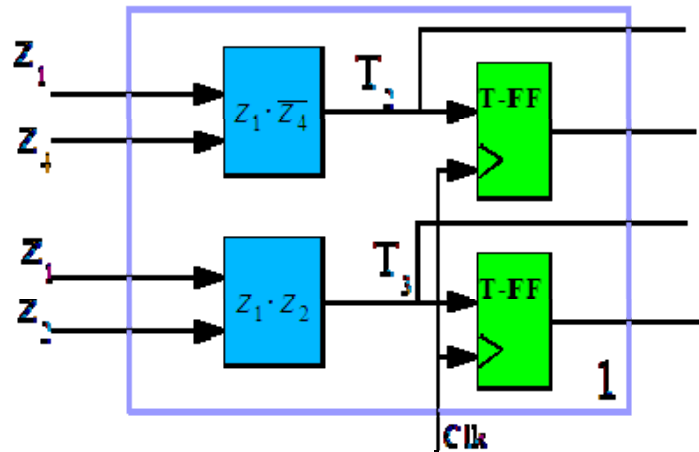
$T_3$ )



$T_4$ )



# 2. FPGA Design Flow – Example



Common product term

$$T_1 = 1$$

$$T_2 = z_1 \cdot \bar{z}_4$$

$$T_3 = z_1 \cdot z_2$$

$$T_4 = z_1 \cdot z_4 + z_1 \cdot z_2 \cdot z_3$$



# 2. FPGA Design Flow – Example

