VLSI Design Automation Lecture-4

By

Dr. Swagata Mandal

Assistant Professor, Electronics and Communication

Jalpaiguri Government Engineering College

Run-Time Reconfiguration

- The computation and configuration sequences are not known at compile-time.
- Request to implement a given task is known at run-time and should be handled dynamically.
- The reconfiguration process exchange part of the device to accommodate the system based on the operational and environmental conditions.
- Run-time reconfiguration is a difficult process that must handle side effect factors like defragmentation of the device and communication between newly placed modules.
- The management of the reconfigurable device is usually done by a scheduler and a placer that can be implemented as part of an operating system running on a processor .
- The processor can either resides inside or outside the reconfigurable chip.

Run-Time Reconfiguration

- The scheduler manages the tasks and decides when a task should be executed.
- The tasks, which are available as configuration data in a database are characterized through their bounding box and their run-time.

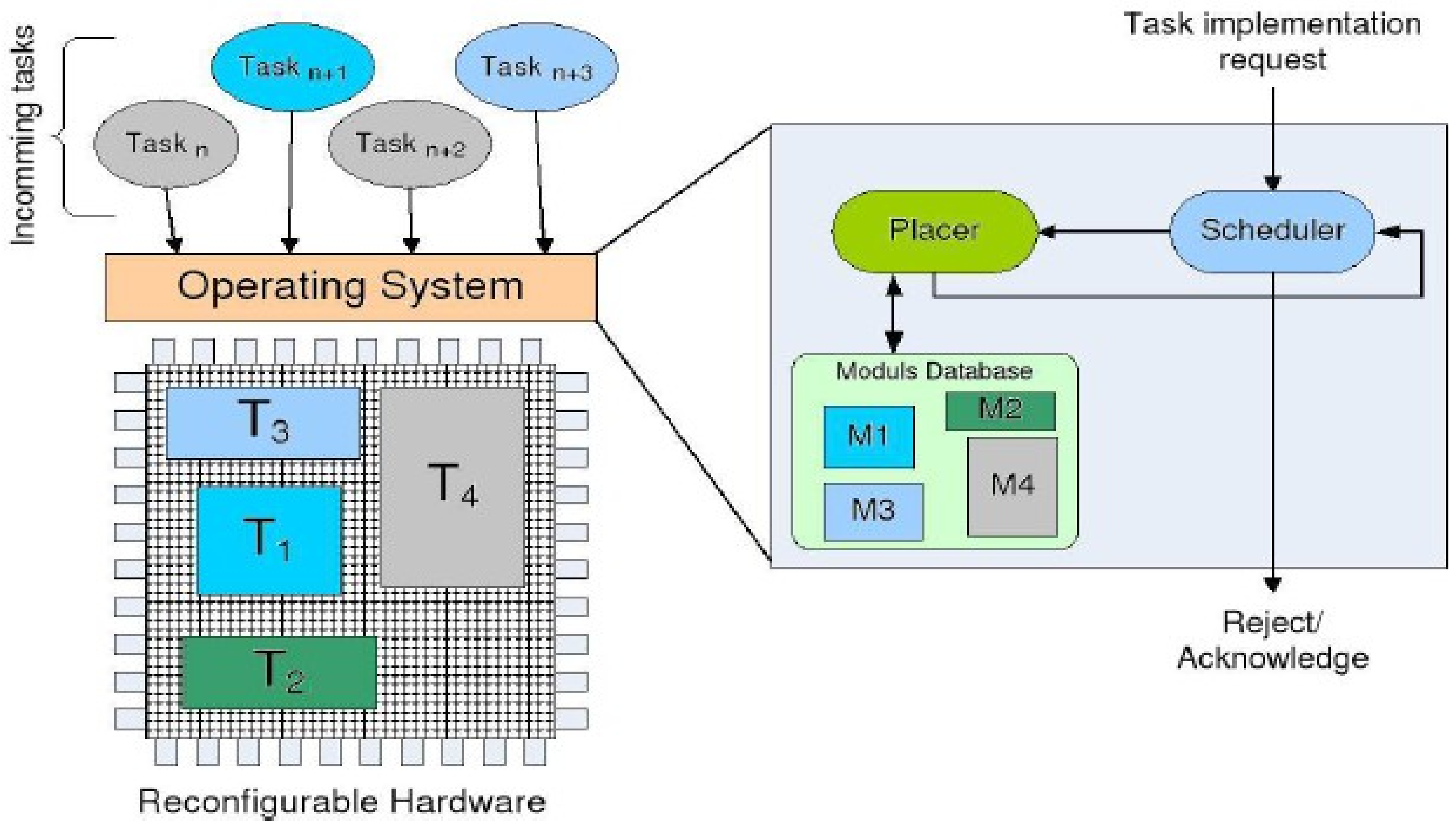


Fig 1: Architecture of a run-time reconfigurable system

Run-Time Reconfiguration(cont’d)

- The bounding box defines the area that a task occupies on the device.
- The management of task execution at run-time is therefore a ***temporal placement problem***.
- The scheduler determines which task should be executed on the RPU and then gives the task to the placer which will try to place it on the device, i.e. allocate a set of resources for the implementation of that task.
- If the placer is not able to find a site for the new task, then it will be sent back to the scheduler which can then decide to send it later and to send another task to the placer.
- The host CPU is used for device configuration and data transfer.
- Usually, the reconfigurable device and the host processor communicates via a bus that is used for the data transfer between the processor and the reconfigurable device.

Run-Time Reconfiguration(cont'd)

- In supercomputing, systems are made upon a high speed processor from Intel or AMD and an FPGA-board attached to a bus like the PCI.
- In embedded systems, the processors are more and more integrated in the reconfigurable devices and are heavily used for management purpose rather than for computation.
- The RPU acts like a coprocessor with varying instruction sets accessible by the processor in a function call.
- The computation flow can be summarized as shown in algorithm 1.
- At the beginning of a computation the host processor configures the reconfigurable device.
- Then it downloads the segment of the data to be processed by the RPU and give the start signal to the RPU.

CPU-RPU communication

- **Algorithm1: CPU-RPU configuration and computation steps**

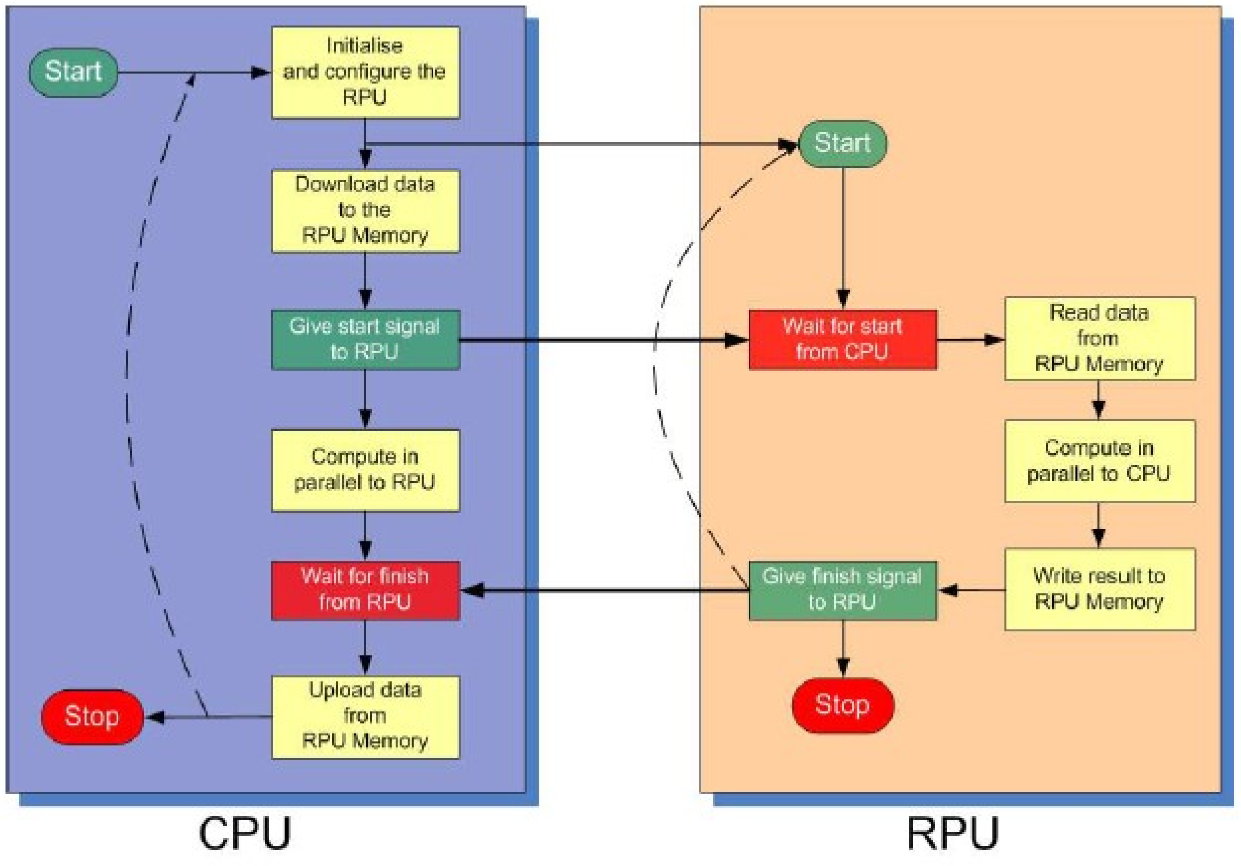
```
1: Start
2: Initialize the RPU
3: while (1) do
4:     Configure the RPU to implement a new task
5:     Download Data for RPU computation into RPU-memory
6:     Computes in parallel with the RPU if necessary
7:     Upload the data computed by the RPU from the RPU-memory
8: end while
9: Stop
```

- The host and the RPU can then process in parallel on their segments of data.
- At the end of its computation the host reads the finish signal of the RPU.
- At this point the data (computation result) can be collected from the RPU memory by the processor.

Run-Time Reconfiguration(cont'd)

- The RPU can also send the data directly to an external sink. During the computation process RPU is configured only once.

- If the RPU has to be configured more than once, then the body of the while loop must be run again according to the number of reconfigurations to be done before.



Start
Initialise and configure the RPU
Download data to the RPU Memory
Give start signal to RPU
Compute in parallel to RPU
Wait for finish from RPU
Upload data from RPU Memory
Stop
CPU
Start
Wait for start from CPU
Read data from RPU Memory
Compute in parallel to CPU
Write result to RPU Memory
Give finish signal to RPU
Stop
RPU

Figure: A CPU-RPU configuration and computation step

Run-Time Reconfiguration(cont'd)

- The design flow of a dynamic reconfigurable system is primary a hardware/software partitioning process in which:
  - The part of the code to be executed on the host processor is determined.
  - This part is usually control-dominated.
  - The parts of the code to be executed on the reconfigurable device are identified.
  - Those are usually data-dominated parts for which efficient dataflow computation modules are required.
  - The interface between the processor and the reconfigurable device is implemented.

FPGA Design Flow

Design Entry:

- The description of the function is made using either a schematic editor, a hardware description language (HDL), or a finite state machine (FSM) editor.

- A schematic description is made by selecting components from a given library and connecting them together to build the function circuitry.

- This process has the advantage of providing a visual environment that facilitates a direct mapping of the design functions to selected computing blocks.

- The final circuit is built in a structural way. However, designs with very large amount of function will not be easy to manage graphically.

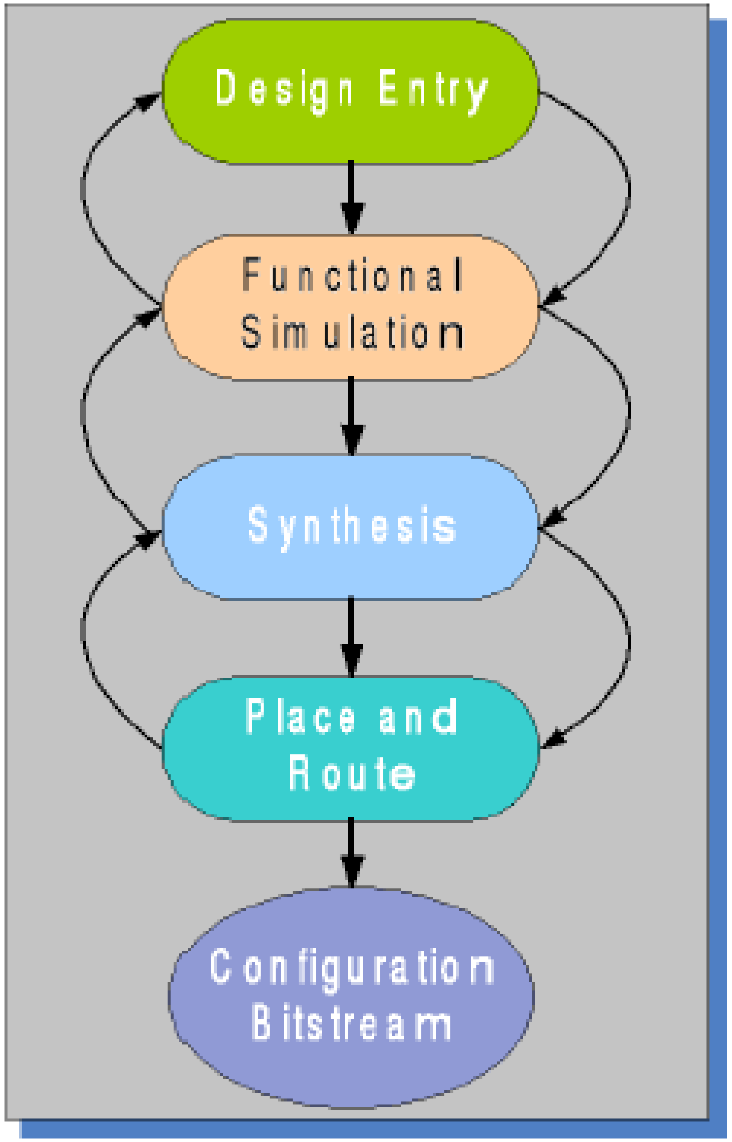


Fig: FPGA Design Flow

FPGA Design Flow (cont'd)

- Instead, a Hardware Description language (HDL) may be used to capture the design either in a structural or in a behavioural way.

- Besides VHDL and Verilog, which are the most established HDLs several C-like languages, are also available like ***Handel-C***, ***ImpulseC***, ***SystemC***.

- **Functional Simulation:**

- After the design entry step, the designer can simulate the design to check the correctness of the functionality.
- This is done by providing test patterns to the inputs of the design and observing the outputs.
- The simulation is done in software by tools which emulate the behaviour of the components used in the design.
- During the simulation, the inputs and outputs of the design are usually shown on a graphical interface, which describes the signal evolution in time.

FPGA Design Flow (cont'd)

- **Logic Synthesis:**
- After the design description and the functional simulation, the design can be compiled and optimized. It is first translated into a set of Boolean equations.
- Technology mapping is then used to implement the functions with the available modules in function library of the target architecture.
- In case of FPGAs, this step is called LUT-based technology mapping, because LUTs are the modules used in the FPGA to implement the boolean operators.
- The result of the logic synthesis is called the *netlist*. A *netlist* describes the modules used to implement the functions as well as their interconnections.
- There exist different netlist formats to help exchange data between different tools.
- The most known are the *Electronic Design Interchange Format* (EDIF). Some FPGA manufacturers provide proprietary formats. This is the case the *Xilinx Netlist Format* (XNF) for the Xilinx FPGAs.

FPGA Design Flow (cont'd)

- **Place and Route:**
- For the netlist generated in the logic synthesis process, operators (LUTs, Flip-Flopss, Multiplexers, etc...) should be placed on the FPGA and connected together via routing.

- Those two steps are normally achieved by CAD tools provided by the FPGA vendors.

- After the placement and routing of a *netlist*, the CAD tools generate a file called a *bitstream*.

- A bitstream provides the description of all the bits used to configure the LUTs, the interconnect matrices, the state of the multiplexer and I/O of the FPGA.

- The full and partial bitstreams can now be stored in a database for downloading into the configuration memory.

Design Tools

- The design entry, the functional simulation and the logic synthesis are done using the CAD tools from Xilinx, Synopsys, Synplicity, Cadence, ALTERA and Mentor Graphics. The place and route as well as the generation of configuration data is done by the corresponding vendors tools

Manufacturer	Tool	Description
Synopsis	DC Compiler	RTL Synthesis
Mentor Graphics	Precision Synthesis	RTL Synthesis
Mentor Graphics	ModelSim	Simulation and verification
Synplicity	Sinplify	Synthesis, place and route
Xilinx	ISE/Vivado	Synthesis, place and route
Altera	Quartus II	Synthesis, place and route
Microsemi	Libero	Synthesis, place and route
Atmel	Figaro	Synthesis, place and route

Logic Synthesis

- A function, assigned to the hardware in the hardware/software co-design process, can be described as a digital structured system.

- As shown in Figure, such a digital structured system consists of a set of combinatorial logic modules (the nodes), memory (the registers), inputs and outputs.

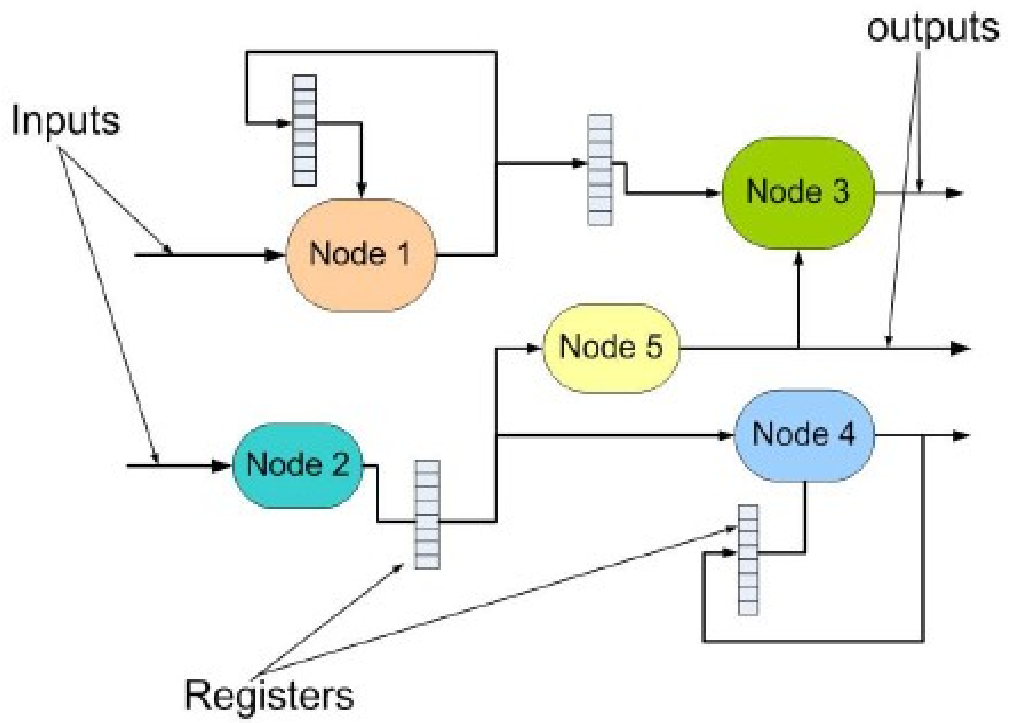


Figure: A structured digital system

Logic Synthesis(cont'd)

- The inputs provide data to the system while the outputs carry data out of the system.
- Computation is performed in the combinatorial parts and the results might be temporally stored in registers that are placed between the combinatorial blocks.
- A clock is used to synchronize the transfer of data from register to register via combinatorial parts.
- The description of a design at this level is usually called ***register transfer*** description, due to the register to register operation mode previously described.
- For such a digital system, the goal of the logic synthesis is to produce an optimal implementation of the system on a given hardware platform.
- In the case of FPGA, the goal is the generation of configuration data which satisfies a set of given constraints like the maximal speed, the minimum area, the minimum power consumption, etc.

Logic Synthesis(cont'd)

- In a structured system, each combinatorial block is a node that can be represented as a two-level function or as multi-level function.

- ***Two-Level Logic Synthesis*:**
- Two-level synthesis deals with the synthesis of designs represented in two-level logic.

- Those are representations in which the longest path from input to output, in term of number of gates crossed on the path, is two.

- Two-level logic is the natural and straightforward approach to implement a Boolean function, because each Boolean function can be represented as a sum of product terms.

- In the first level, the products are built using the AND-primitives.

- The sums of the resulting products are built in the second level with the OR-primitives.

Multi Level Logic Synthesis

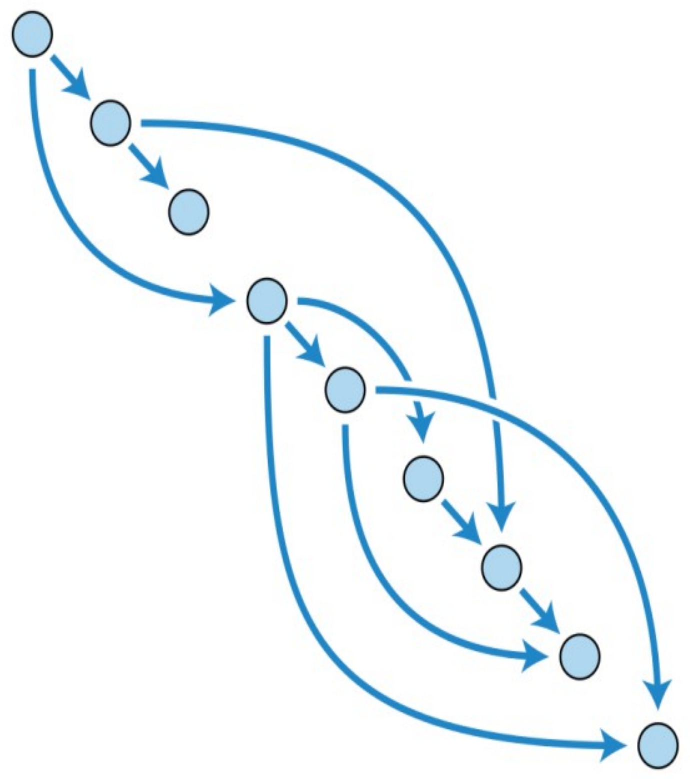
- In the multi-level synthesis, functions are represented using a multi-level logic.
- Those are circuits in which the longest path from input to output goes through more than two gates.
- Most of the circuits used in practice are implemented using multi-level logic.
- Multi-level circuits are smaller, faster in most cases and consume less power than two-level circuits.
- Two-level logic is most appropriate for PAL and PLA implementations while multi-level is used for standard cell, mask-programmable or field-programmable devices.
- We formally represent a node of the structured system as a *Boolean network*, i.e. a network of Boolean operators that reflects the structure and function of the nodes.

Directed Acyclic Graph

- A *Boolean networ*k is defined as a directed acyclic graph (DAG) in which a node represents an arbitrary Boolean function and an edge (i, j) represents the data dependency between the two nodes i and j of the network.

- Directed Acyclic Graph (DAG) consists of finitely many vertices and edges (also called arcs), with each edge directed from one vertex to another, such that there is no way to start at any vertex ***v*** and follow a consistently-directed sequence of edges that eventually loops back to ***v*** again.

- Equivalently, a DAG is a directed graph that has a topological ordering, a sequence of the vertices such that every edge is directed from earlier to later in the sequence.



Boolean Network

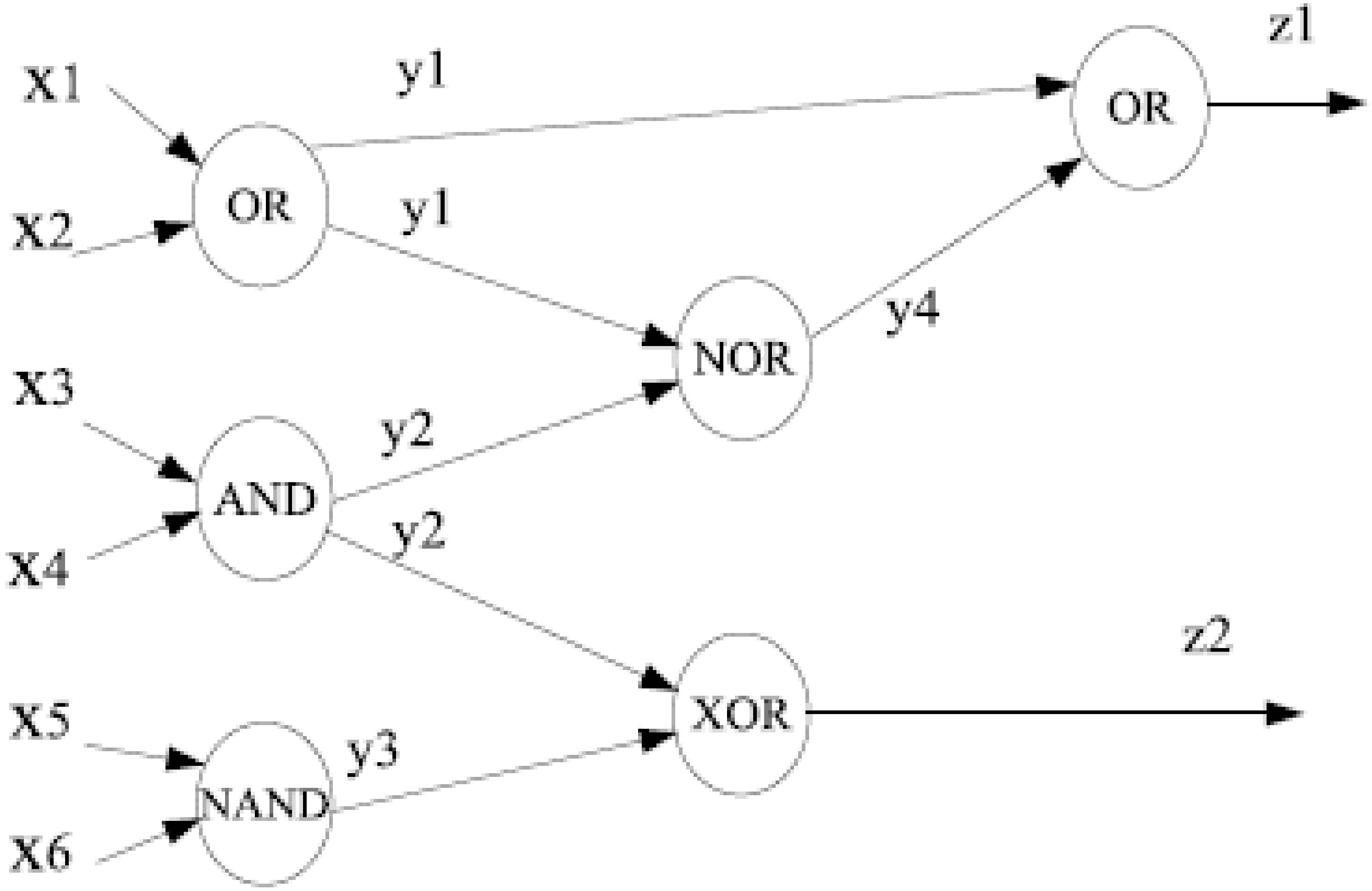


Figure: An example of boolean network

Logic Synthesis(cont'd)

- Logic synthesis can be done in two different approaches: the ***technology dependant synthesis*** and the ***technology independent***.
- In the first case, only valid gates chosen from the target library are used in the node representation.
- The final implementation matches the node representation.
- In the second case, the representation is technology independent, i.e the design is not tied to any library.
- A final mapping must be done to the final library in order to have an implementation.
- The technology independent method is most used, due to the large set of available optimization methods available.
- With a technology independent representation, synthesis for FPGA devices is done in two steps.

Logic Synthesis(cont'd)

- In the first step, all the Boolean equations are minimized, independent of the function generators used.
- In the second step, the technology mapping process maps the parts of the Boolean network to a set of LUTs.
- In general, the following choices are made for the representation of a node:
- **Sum of products form:** A Sum of Product (SOP) is the most trivial form to represent a Boolean function.
- It consists of a sum of product terms and it is well adapted for two-level logic implementation on PALs and PLAs. Example: $x\bar{y}z + \bar{x}yz + wx\bar{y}$
- This representation has the advantage that it is well understood and it is easy to manipulate.
- Many optimization algorithms are available (AND, OR, Tautology, two-level minimizers).

Logic Synthesis(cont'd)

- The main disadvantage is the non representativity of the logic complexity.
- In fact, designs represented as sum of products are not easy to estimate as the complexity of the design decreases through manipulation.
- Therefore estimation of progress during logic minimization on SOPs is difficult.
- **Factored form:** A factored form is defined recursively either as a single literal or, as a product or a sum of two factored forms:
- $c(a+b(d+e))$ is a product of the factored forms c and $a + b(d + e)$, and $a + b(d + e)$ is a sum of the factored forms a and $b(d + e)$.
- Factored forms are representative of the logic complexity.
- factored forms are good estimation of complexity of the logic implementation.

Binary decision diagram (BDD)

- A binary decision diagram (BDD) is a rooted directed acyclic graph used to represent a boolean function.
- Two kinds of nodes exist in BDDs: variable and constant nodes.
- A *variable node v* is a non-terminal having as attribute its argument index *index(v)* $\in \{1, ..., n\}$ and its two children *low(v)* and *high(v)*.
- A *constant node v* is a terminal node with a value *value*(*v*) $\in \{0, 1\}$.
- A BDD in which an ordering relation among the nodes exists is called a Ordered BDD (OBDD).
- The non-terminal nodes are ordered from the root to the terminal nodes.
- Formally, for each non terminal node *v*, if *low*(*v*) is non-terminal, then *index(low(v))* < *index(v)*. Similarly, if *high*(*v*) is non-terminal, then *index(high(v))* < *index(v)*.

Binary decision diagram (cont'd)

- The correspondence between a BDD and a Boolean relation is defined as follows: A BDD with root v denotes a function f_v.

- If v is a terminal node, then if $value(v) = 1$, then $f_v = 1$, else ($value(v) = 0$) $f_v = 0$.

- If v is a non-terminal node with $index(v) = i$, the Shannon expansion theorem is used to express the function f_v as $f_v = \bar{x}_i f_{low(v)} + x_i f_{high(v)}$ where $f_{low(v)}$ and $f_{high(v)}$ denote the function rooted at v.

- The value of f_v for a given assignment is obtained by traversing the graph from the root terminal according to the assignment values of the nodes.

- A BDD G is a Reduced Ordered BDD (ROBDD) if the following holds:

- $low(v) \neq high(v), \forall v$ in G

Binary decision diagram (cont'd)

- $\forall v, v' \in G$, the sub-trees rooted at v and the sub-tree rooted at v' are not isomorphic (Two graphs which contain the same number of graph vertices connected in the same way are said to be isomorphic. Two graphs *G* and *H* with graph vertices $V_n = \{1,2,\ldots n\}$ are said to be isomorphic if there is a permutation *p* V_n such that {*u*,*v*} is in the set of graph edge *E*(*G*) iff {*p*(*u*),*p*(*v*)} is in the set of graph edges *E*(*H*)).

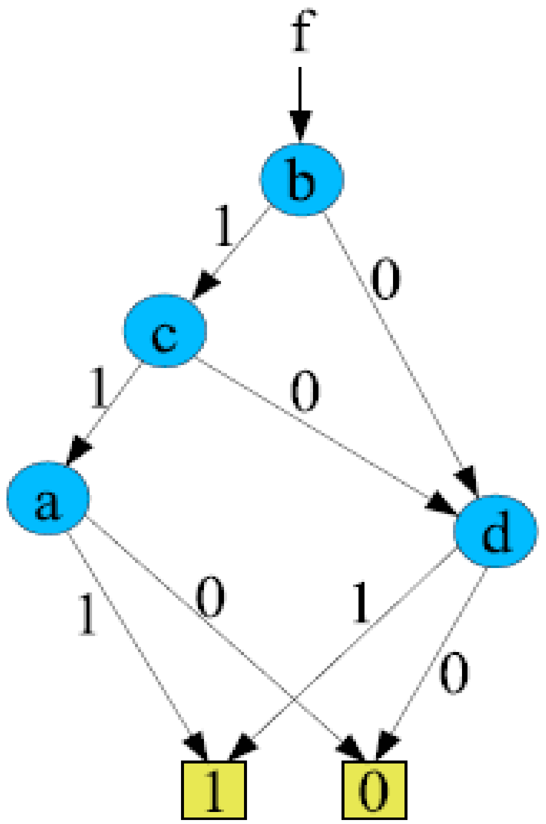


Figure: BDD-representation of the function
$f = abc + \bar{b}d + b\bar{c}d$

Thank You