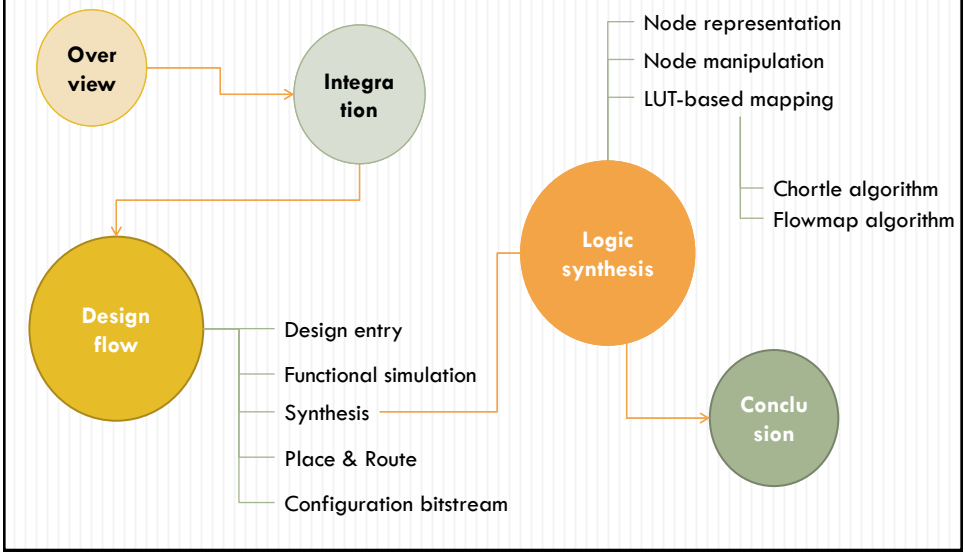
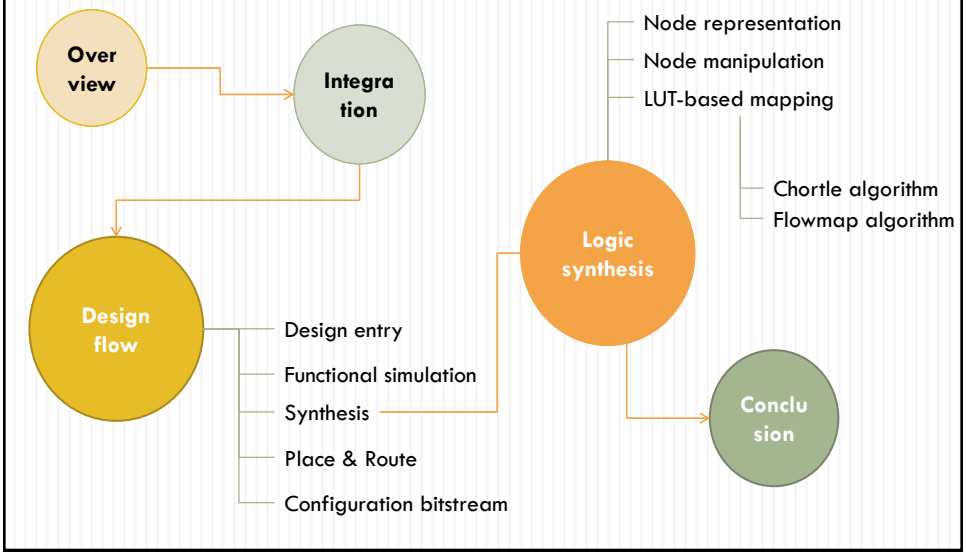
IMPLEMENTATION DESIGN FLOW

Hà Minh Trần Hạnh – Nguyễn Duy Thái
Course: Reconfigurable Computing

Outline



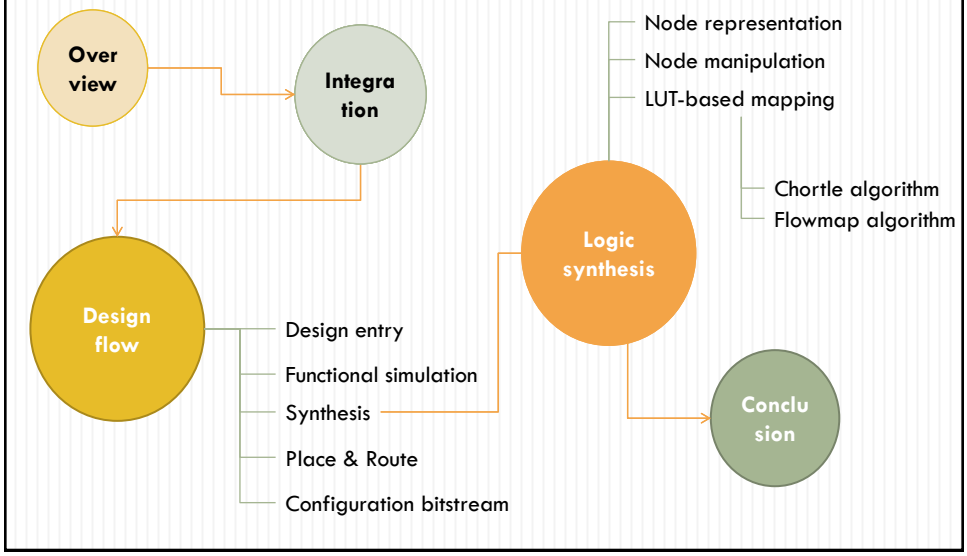
Outline



Overview

- The steps required to implement an application.
- FPGA design flow:
  - Not aim at generating a set of instructions but hardware components that will be loaded at different time on available resources.
  - Technology mapping targets look-up tables rather than NAND gates.

Outline



Integration

Usage

Rapid prototyping

- Emulator for another digital devices such as ASIC.

Non-frequently reconfigurable system

- For testing and initialization at startup and for upgrading purpose.
- No reconfiguration happens during operation.

Frequently reconfigurable system

- Usually coupled with a host processor to reconfigure device and control complete system.

Integration

Reconfiguration time

Compile-time reconfiguration

Never change during computation.

More interesting for only fully-reconfigurable devices.

Run-time reconfiguration

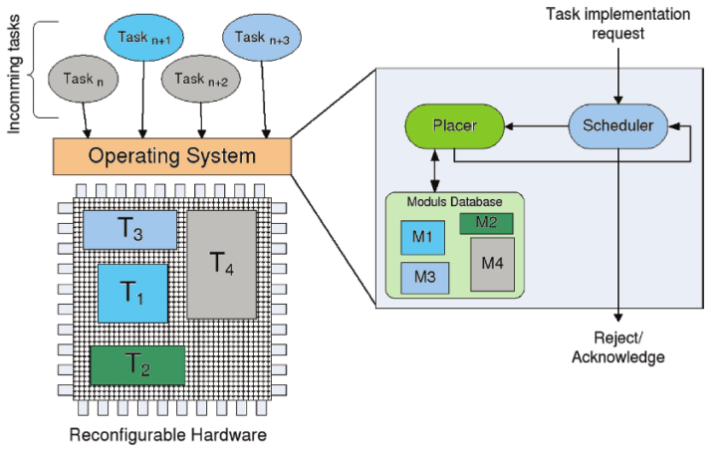
Not known at compile time. Given task is known at run time.

Reconfiguration process exchanged parts of devices to accommodate the system to change operational and environmental conditions.

Integration

Management of Reconfigurable device

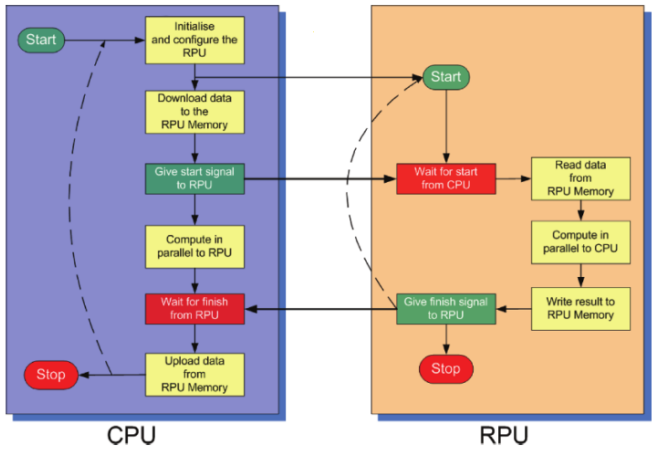
Common architecture



Integration

Management of Reconfigurable device

Common operation flow

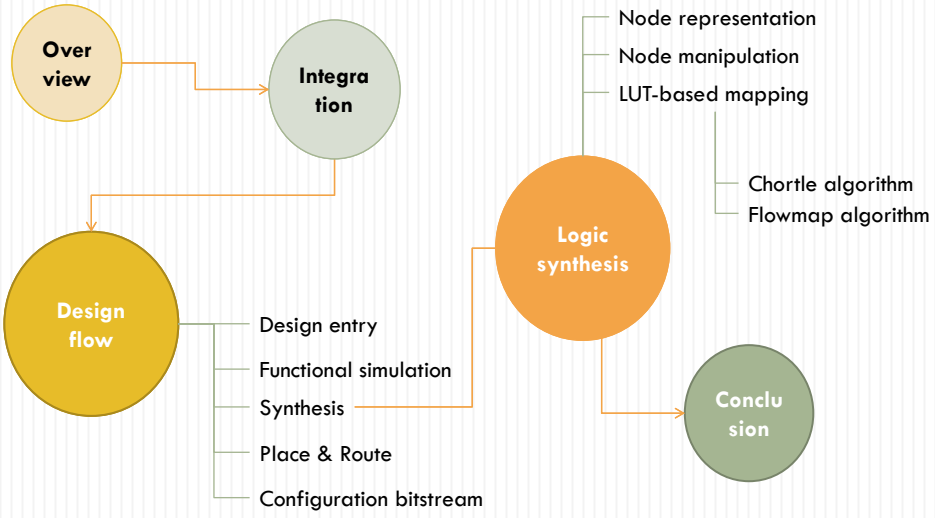


Integration

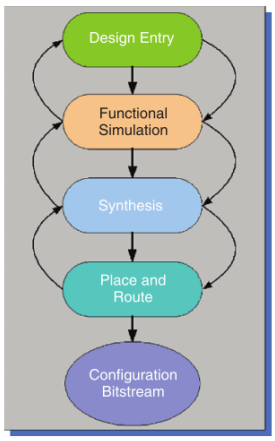
Design flow of dynamic reconfigurable system

- Hardware/software partitioning process:
  - The part of the code to be executed on the host processor is determined.
  - The parts of the code to be executed on the reconfigurable device are identified.
  - The interface between the processor and the reconfigurable device is implemented.

Outline



Design flow



Design entry
- Make description of function.
- Tools: HDL, schematic, FSM editor, …

Functional simulation
- Simulate function to check correctness.

Synthesis
- Compile and optimize.
- Implement function with available modules in function library of target device to create netlist.

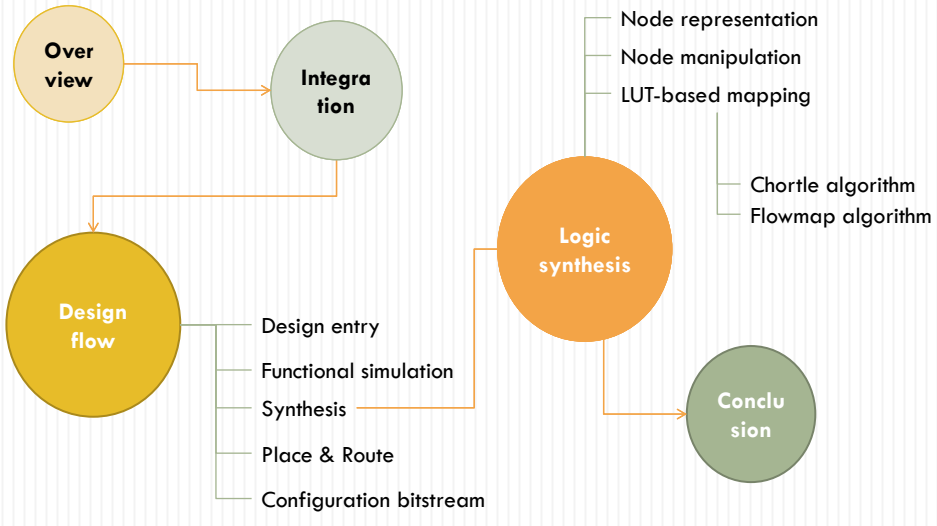
Place & route
- Place operators (mux, LUTs, FF, …) and connect together through routing on FPGA device.
- Normally achieve by CAD tools to generate a bitstream to download to device later.

Design flow

Design tools

Manufacturer	Tool	Description
Synopsys	FPGA Compiler	Synthesis
Mentor	FPGA	Synthesis, place and route
Synplicity	Sinplify	Synthesis, place and route
Xilinx	ISE	Synthesis, place and route (only Xilinx products)
Altera	Quartus II	Synthesis, place and route (only Altera products)
Actel	Libero	Synthesis, place and route (only Actel products)
Atmel	Figaro	Synthesis, place and route (only Atmel products)

Outline



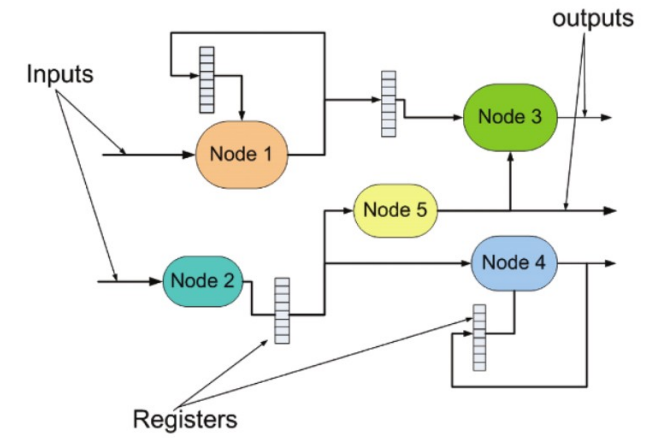
Logic synthesis

Overview

- Function is described as a digital structured system (register transfer description):
  - Combinational logic blocks. are nodes.
  - Registers to store results of combinational logic blocks.
- Goals is producing an optimal implementation of system on given hardware platform. In case of FPGA, it's generating configuration data.

Logic synthesis

Structured digital system



Logic synthesis

Synthesis Approaches

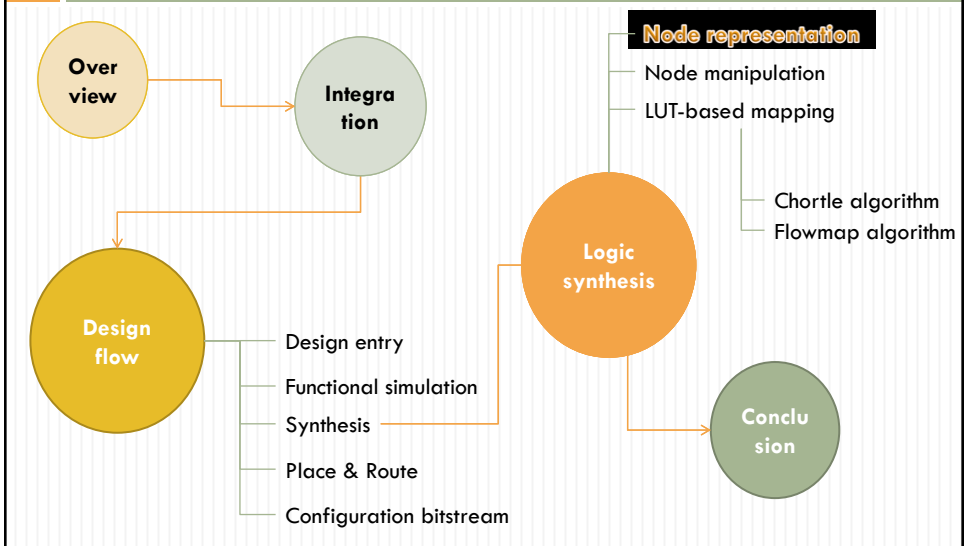
Two-level logic synthesis

- Natural and straightforward method to implement Boolean functions.
- All functions can represented as a sum of product terms.
- Longest path is two.
- Most appropriate for PAL and PLA.

Multi-level logic synthesis

- Present as multi-level logic.
- Longest path is greater than two.
- Smaller, faster and less power in most case.
- Used for standard cells, mask-programmable or field-programmable devices.

Outline



Node representation

Technology dependent

- Only valid gates chosen from target library are used.
- Final implementation matches the node presentation.

Technology independent

- Design is not tied to any library.
- 2 steps:
  - Optimizing Boolean equations.
  - Mapping parts of Boolean network to set of LUTs.

Node representation

- Efficiently use memory and easy to manipulate.
- Technology-independent synthesis method is most used for easier optimization.
- In FPGA, synthesis follows 2 steps:
  - Minimize Boolean equations.
  - Map Boolean networks to a set of LUTs.

Node representation

Representation approaches

SOP form
- Representing as Sum Of Products.
- Being well understood, easy to manipulate.
- Non-representativity of logic complexity.

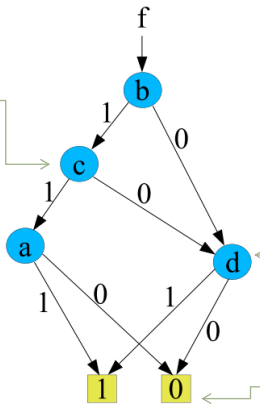
Factored form
- c [a+b(d+e)].
- Representative of the logic complexity, easy to estimate comlexity of logic implementation.
- Lack of manipulation algorithms.

BDD form
- Representing as Binary Decision Diagram.
- Manipulations of BDDs are well defined and efficient.

Node representation

Binary Decision Diagram

F(c) = c.F(a) + (~c).F(d)

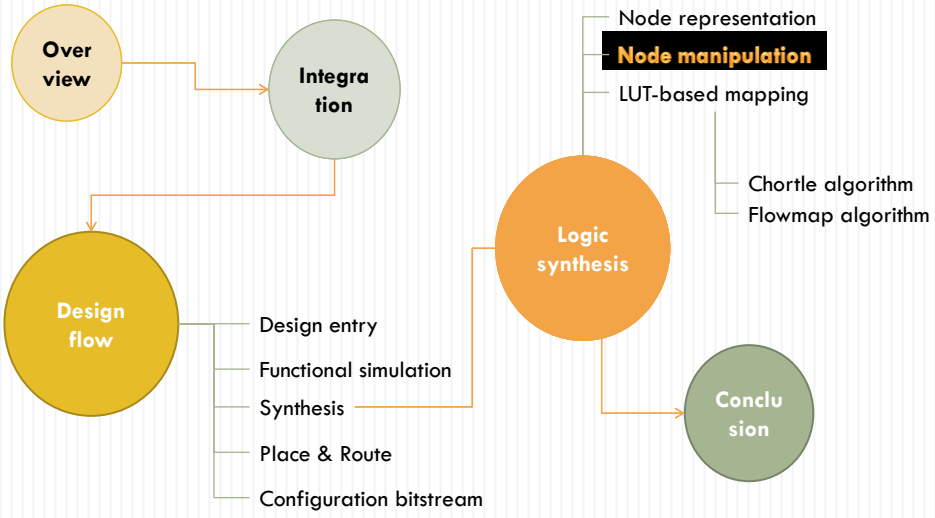


Variable node: index = {0 .. N}

Constant node: value = {0; 1}

$f = abc + \bar{b}d + b\bar{c}d$

Outline



Node manipulation

- Transformations to get optimized representation in previous section:

Decompose:

$$f = a\bar{b}\bar{c} + a\bar{b}d + \bar{a}c\bar{d} + bc\bar{d} = a\bar{b}(\bar{c}+d) + (\bar{a}+b)c\bar{d}$$
$$= a\bar{b}(\bar{c}+d) + \overline{a\bar{b}(\bar{c}+d)} = XY + \overline{XY}$$

Extract:

$$f = (a+bc)d + e = xd + e$$
$$g = (a+bc)\bar{e} = x\bar{e}$$

Factor:

$$f = ac + ad + bc + bd + e = (a+b)(c+d) + e$$

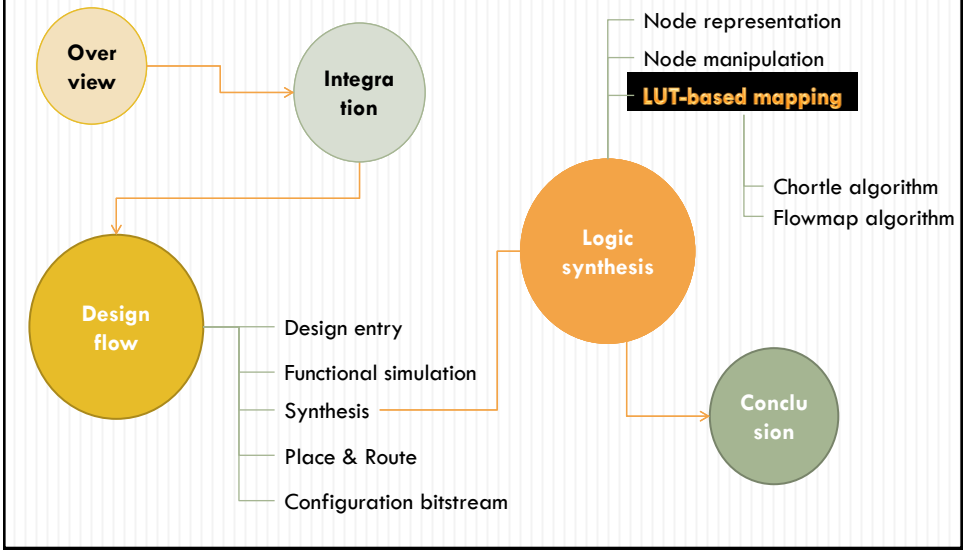
Substitute:

$$f = (a+bc)(d+e) = g(d+e)$$

Collapse:

$$f = ga + \bar{g}b = ac + ad + b\bar{c}\bar{d}$$
$$g = c + d$$

Outline



LUT-based mapping

Algs for minimizing the Area

- **Chortle-crf.**
- MIS-fpga
- Xmap.

Algs for minimizing the Delay

- **FlowMap.**
- Chortle-d.
- DAG-map.
- MIS-pga-delay.

Algs for maximizing the Routability

- Bhat and Hill.
- Schlag.
- Kong and Chang.

LUT-based mapping

Prerequisite definitions

- Primary input (PI) node: node without any predecessor.
- Primary output (PO) node: node without any successor.
- The level l(v) of a node: the length of the longest path from the primary inputs to that node.
- The depth of a network G: the largest level of a node in G.
- The fan-in of a node v: the set of gates whose outputs are inputs of v.
- The fan-out of v: the set of gates that use the output of v as input.
- Given a node $v \in G$, input(v) is defined as the set of node of G, which are fan-in of v, i.e. the set of predecessors of v.

LUT-based mapping

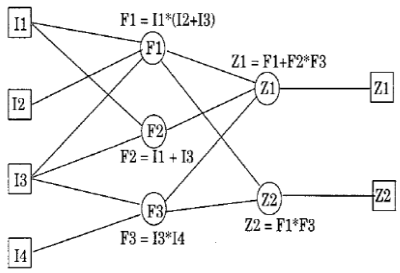
28

Modified tree-covering approaches

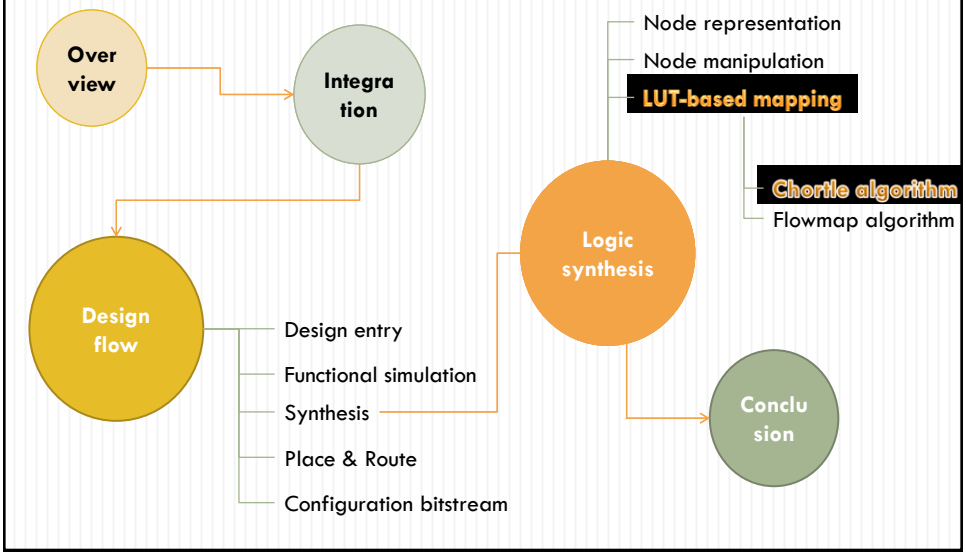
Begins with an AND/OR representation of the optimized Boolean network.

- Decomposing network into a forest of trees.
- Optimal mapping of each tree into LUTs.
- Assembling together.

An example of Boolean network.



Outline



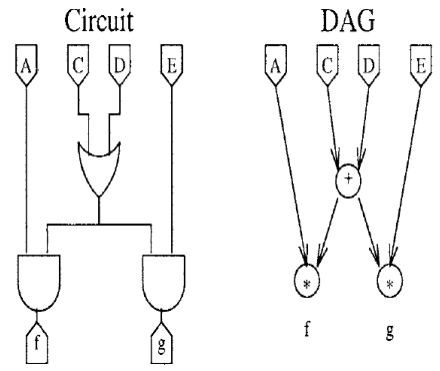
Chortle Algorithm

Concepts

- Specifically designed for TLU-based FPGAs.
- The input: Boolean network as a forest of directed acyclic graphs (DAGs):
  - Leaves : inputs.
  - Root : output
  - Internal nodes : logic functions AND/OR.
  - Edges : inverting or non-inverting signal paths.
- Goal : implementing the circuit using the fewest number of *K*-input CLBs in minimal running time.

Chortle Algorithm

Concepts



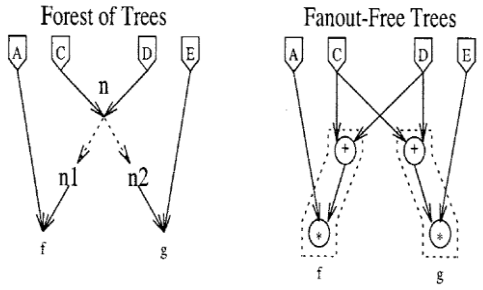
Boolean network and DAG representation.

Chortle Algorithm

Operation Flow

- Tree mapping
  - Converting the input graph to forest of fan-out-free trees, where each logic function has exactly one output.
  - Evaluating each sub-tree independently.

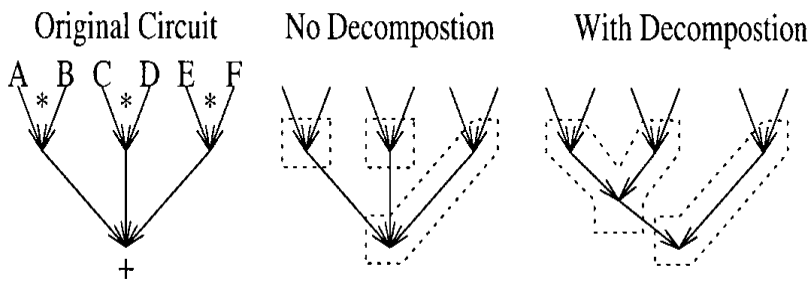
Forest of fan-out-free trees.



Chortle Algorithm

Operation Flow

- Decomposing



Chortle Algorithm

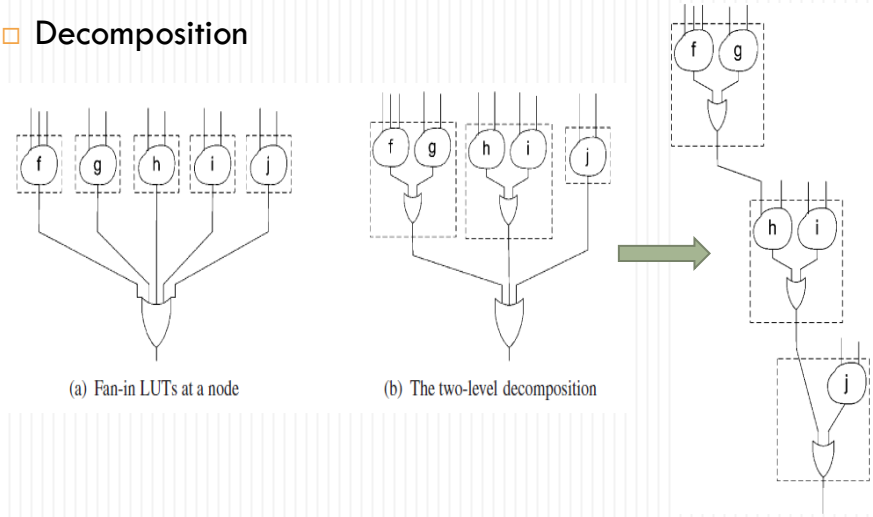
Operation Flow

- Decomposing
  - *Required* if the original circuit has a fan-in greater than *K*.
  - Two steps:
    1. Two-level decomposing.
    2. Converting into a multi-level decomposition.

Chortle Algorithm

Operation Flow

- Decomposition



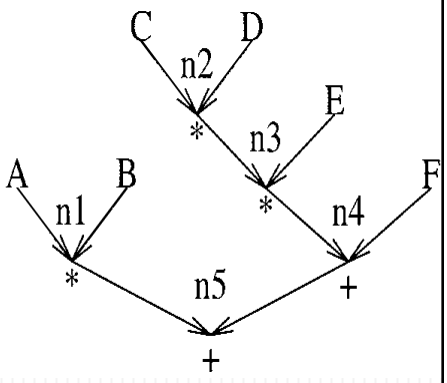
(a) Fan-in LUTs at a node

(b) The two-level decomposition

Chortle Algorithm

Example

For this example, we will assume that each CLB may have as many as four inputs (i.e., $K = 4$). The inputs, {A,B,C,D,E,F}, perform the logic function: A ∗ B + (C ∗ D) E + F.
In the postorder traversal n_1 is visited first, followed by n_2 … n_5. For n_1, there is only one possible mapping function, namely, $U = 2$, $\upsilon = \{1,1\}$. The same is true for n_2.

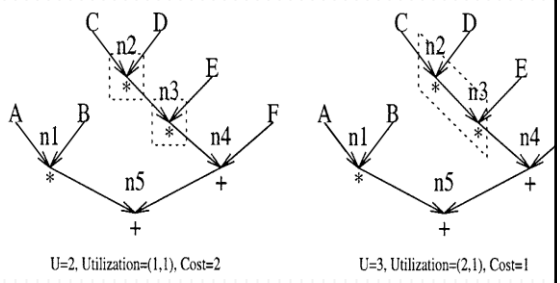


Chortle mapping example.

Chortle Algorithm

Example

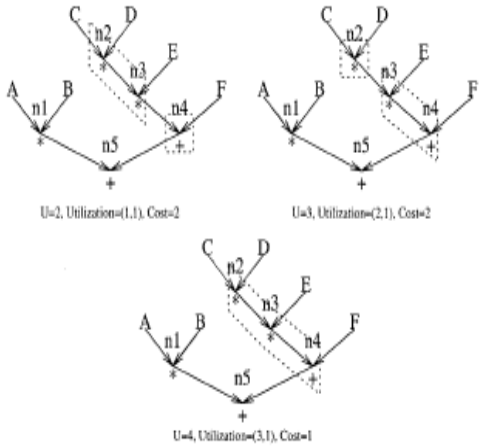
When *n*3 is evaluated, there are two possibilities, as illustrated First, the function could be implemented as a new CLB with two inputs (*U* = 2), driven from the outputs of *n*2 and E. This sub-graph would use two TLBs; thus, it would have a cost function of 2. For *U* = 3,only one utilization vector is possible, namely, υ = {2,1}. All three primary inputs C, D, and E are grouped into one CLB, thus producing a cost function of 1.



Mapping of node 3.

Chortle Algorithm

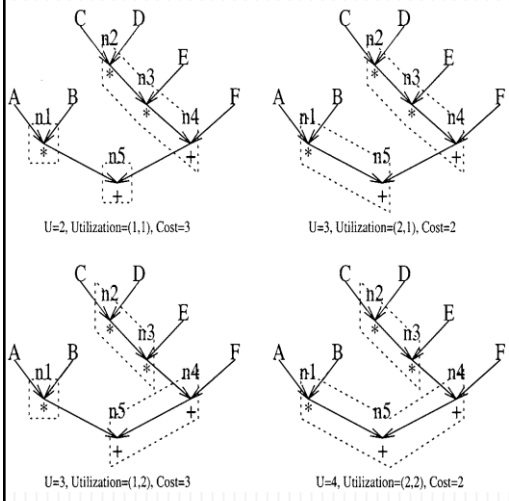
Example



Mapping of node 4.

Chortle Algorithm

Example

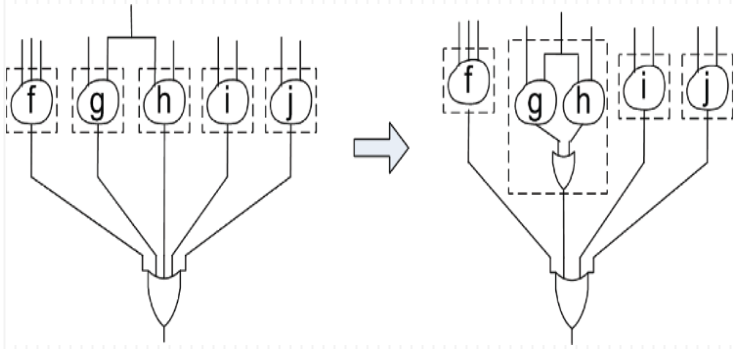


Mapping of node 5

Finally, we evaluate the output node, *n*5, as illustrated. We see that there are four possible mappings and, of those, two minimal mappings are possible. Chortle may return either of the mappings where two CLBs implement: $n5 = (A * B) + n3 + F$ and $n3 = (C * D) * E$.

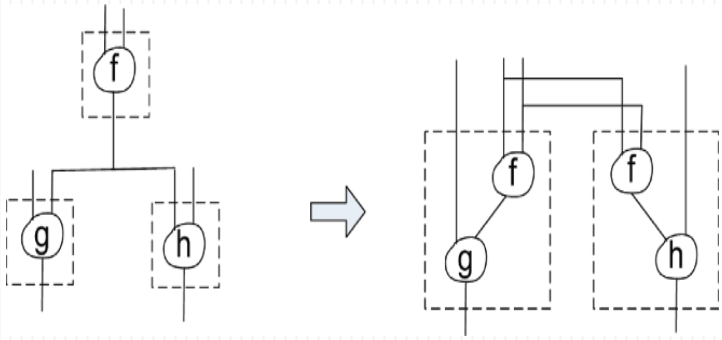
Improvement of the Chortle

1. Exploiting the reconvergent paths.

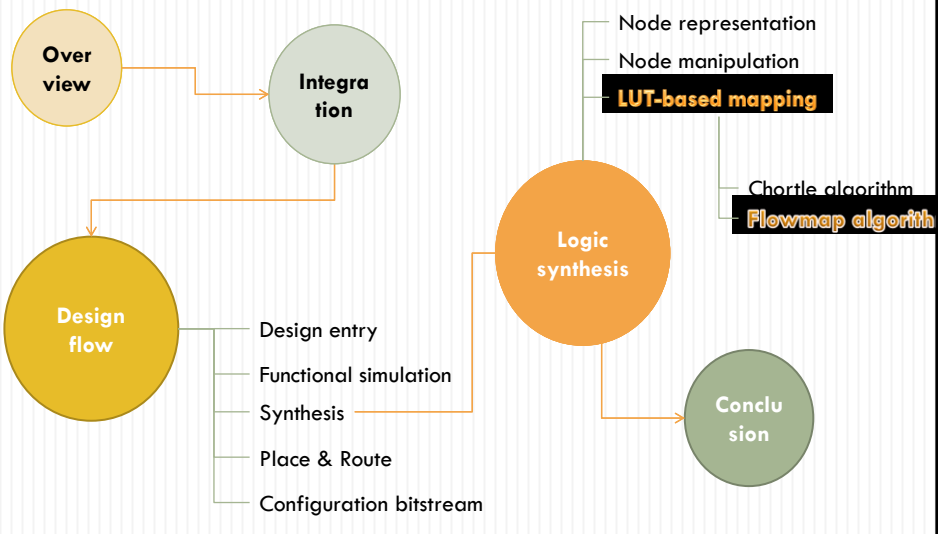


Improvement of the Chortle

2. Logic replication at fan-out LUTs.



Outline



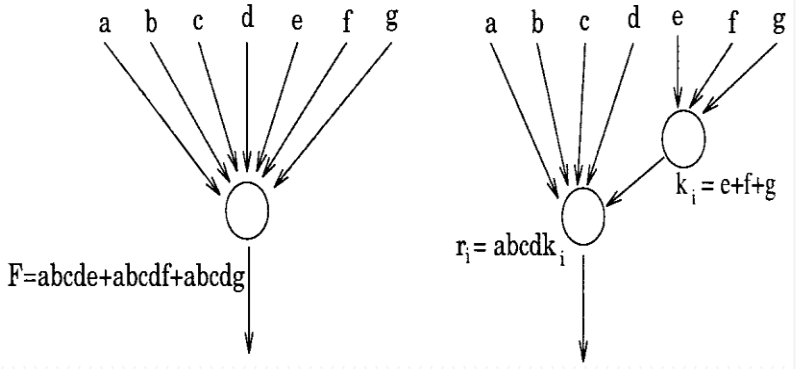
Flowmap

Concepts

- Optimal solution in polynomial time.
- Two steps:
  1. Decomposing:
     - For a given FPGA device, with a k-input TLU, all nodes of the network with more than k inputs must be decomposed.
     - Different methods decompose the network in different ways.
  2. Node Mapping.:
     - Example: Local Elimination: The operation used for local elimination is collapsing, which merges node ni into node nj whenever ni is a fan-in node to nj and the new node obtained is feasible.

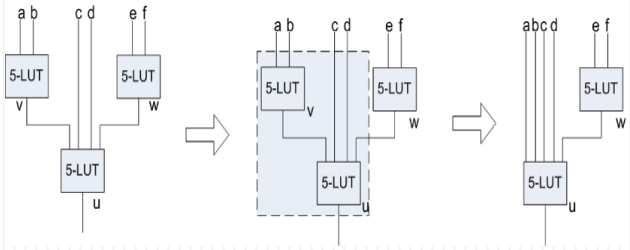
Flowmap

Decomposing



Flowmap

Node-mapping



Questions/Discussions

- Any question?


Thank you!