FPGA & Reconfigurable Computing
Lecture-6

By

Dr. Swagata Mandal

Assistant Professor, Electronics and Communication

Jalpaiguri Government Engineering College

Node Manipulation

- The goal of the BDD is to generate an optimized boolean network with either a less amount of gates or the lowest depth of the gates.

- A given number of transformations can be applied to the network for this purpose

  - **Decomposition**: The decomposition takes a single Boolean function and replaces it with a collection of new expressions.

  - We say that a Boolean function f(X) is decomposable if we can find a function g(X) such that f(X) = f′(g(X),X).

  - Example: The function $f = a\bar{b}\bar{c} + a\bar{b}d + \bar{a}c\bar{d} + bc\bar{d}$ for example operates on 12 literals. Using decomposition, f can be written as: $f = a\bar{b}(\bar{c} + d) + (\bar{a} + b)c\bar{d}$ = $a\bar{b}(\bar{c} + d) + \overline{a\bar{b}}\left(\overline{\bar{c} + d}\right)$ = $XY + \overline{X}\overline{Y}$ with X = $a\bar{b}$ and Y = $\bar{c} + d$. This decomposition reduces f to only 8 literals.

Node Manipulation(cont'd)

- **Extraction:** The extraction is used to identify common intermediate sub-functions of a set of given functions in order to avoid redundancy.

- Example: $f = (a + bc)d + e$ and $g = (a + bc)\bar{e}$ can be rewritten as $f = xd + e$ and $g = x\bar{e}$ with $x = a + bc$. The output x of the common part of the circuit will be used as common input for f and g.

- **Factoring:** Factoring is the transformation of SOP-expressions in factored form. For example the function $f = ac + ad + bc + bd + e$ can be rewritten as $f = (a + b)(c + d) + e$

- **Substitution:** Substitution replace an expression e within a function f with the value of an equivalent function g(X) = e. For example the function $f = (a + bc)(d + e)$ can be rewritten as$f = g(d + e)$, with $g = a + bc$

- **Collapsing:** Also called **elimination**, collapsing is the reverse operation of the substitution. Collapsing is used to eliminate levels in order to meet the timing constraints. The function $ga + g\bar{b}$ for example will be replaced by $f = ac + ad + b\bar{c}\bar{d}$ with $g = c + d$.

LUT-based Technology Mapping

- The technology independent optimization phase ends with a reduced Boolean network in which the fan-in and fan-out of gates vary.
- The next step consists of allocating LUTs, which are the FPGA library elements for the implementation of the different nodes. Several goals might be followed here.
- If the goal is to minimize the chip area used by the circuit, then the mapping will try to allocate the less possible amount of LUTs.
- If the goal is to minimize the delay of the final circuit, then the mapping will try to minimize the depth of the LUTs used.
- Other goals like testability and low power might also be followed.
- Several LUT technology mapping algorithms have been developed in the past. Depending on their optimization goals, those algorithms can be classified in three categories

LUT-based Technology Mapping(cont'd)

- The first category contains the algorithms, whose goal is to minimize the area.
- This category includes *Chortle-crf*, *MIS-fpga* and the *Xmap*. Due to its popularity and the proof of area optimality for LUTs with less than 5 inputs, we will discuss the *Chortle* algorithm.
- The algorithms in the second category target the delay minimization. Algorithms in this category include *FlowMap*, *Chortle-d*, *DAG-map* and the *MIS-pga-delay*.
- The third category contains algorithms which focus on maximizing the routability. This category includes the Bhat and Hill work as well as the Schlag, Kong and Chang approach.
- Problem Definition: ***Given a Boolean network representing a function f and an integer k. Find an implementation of f using only k-inputs LUTs, such that the amount of LUTs use is minimal or the delay of the resulting circuit is minimal***

LUT-based Technology Mapping(cont'd)

- The LUT-based technology mapping is the problem of covering a given graph (the Boolean network) with a set of k-input LUTs.
- The area of the final implementation is defined through the amount of LUT used, the first condition is equivalent to having a covering with a minimal amount of LUTs.
- The delay in an FPGA is influenced by two main factors: The delay in LUTs and the interconnection delay.
- While the delay in LUT is known, the interconnection delay can only be accurately known after the place and route phase.
- Delay estimation at this stage is done using the depth of LUTs in the design, thus assuming the interconnection delay to be one for each wire.
- The more LUTs are available on a path from the input to the outputs the higher the delay in the circuit.

Boolean Network

- ***Given a boolean network G, we define the following:***
  - A **primary input (PI)** node is a node without any predecessor.
  - A **primary output (PO)** node is a node without any successor.
  - The level(*l(v)*) of a node *v* is the length of the longest path from the primary inputs to *v*.
  - The **depth of a network** G is the largest level of a node in G.
  - The fan-in of a node *v* is the set of gates whose outputs are inputs of *v*.
  - The fan-out of *v* is 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*.

- ***A tree or fan-out-free circuit is one in which each node has a maximal fan-out of one.***

- ***A leaf-DAG is a combinational circuit in which the only gates with a fan-in greater than one are the primary inputs.***

K-Bounded network

- **K-Bounded network:** Given a Boolean network G and a subgraph H of G
  - With *input*(*H*), we denote the set of all nodes not included in H, which are predecessors of some nodes in H.
  - G is K-bounded if $|input(v)| \leq$ K for all nodes of G.

- A K-bounded Boolean network can be directly mapped to a set of k-inputs LUT by assigning a LUT for each node. However this straightforward approach may not produce the expected optimal results.

Cone at a node: Given a Boolean network G

- A cone at a node *v* is the tree with root *v* which spans from *v* to its primary inputs.
- The cone is K-feasible if:
- Any path connecting two nodes in to *v* lies entirely in

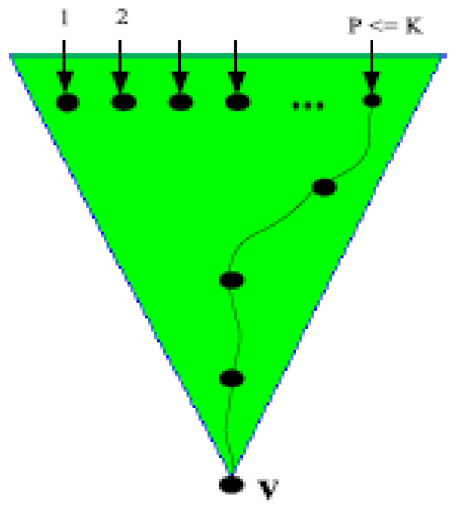


Figure: Example of a *K-feasible* cone at a node *v*

K-Bounded network(cont'd)

- With the previous definition of *K-feasible* cones, the LUT technology mapping becomes the problem of covering the graph with a set of *K-feasible* cones that are allowed to overlap.

- The technology mapping results in a new DAG in which nodes are *K-feasible* cones and edges represent communication among the cones.

  - Figure shows the covering of a graph with 3-feasible cones and the resulting LUT-mapping to 3-input LUTs.

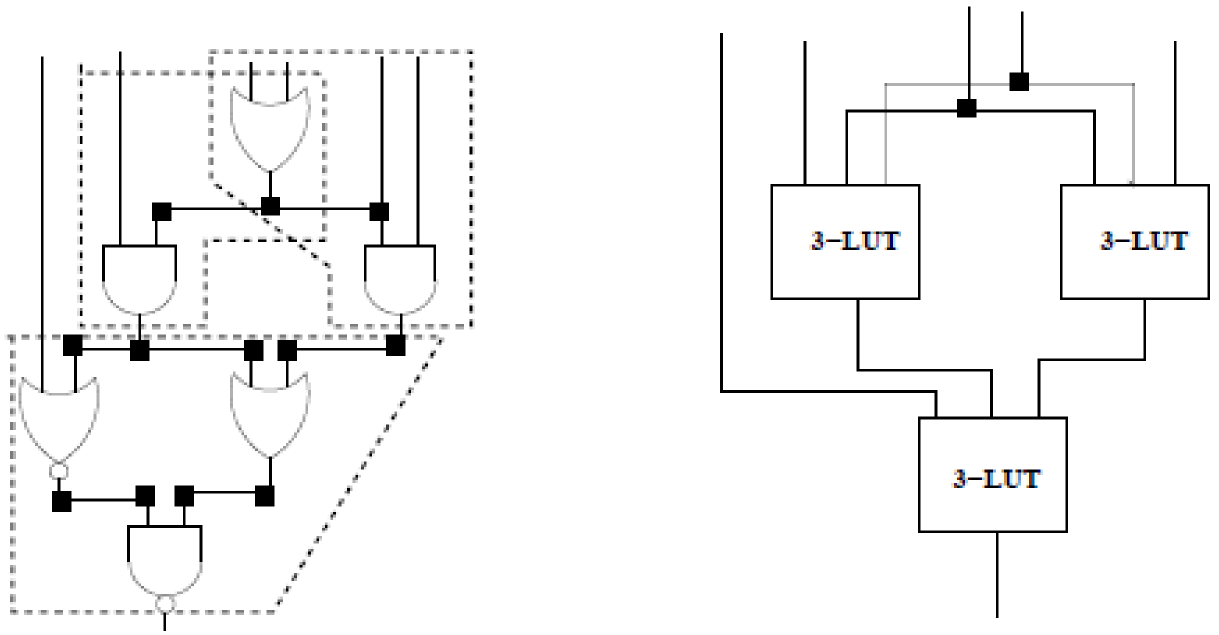


Figure : Example of a graph covering with K-feasible cone and the corresponding covering with LUTs

Chortle Algorithm

- The *chortle* algorithm was developed by Francis *et. al* at the University of Toronto in 1991 with the aim of minimizing the amount of LUTs in the implementation of a given circuit.
- It operates in two steps: In the first step, the original Boolean network is partitioned into a forest of trees that are then separately mapped into circuits of K-input LUTs.
- The second step assembles the circuits implementing the trees to produce the final circuit.
- The transformation of the original network into a forest is done by partitioning each fan-out node *v*.
- Therefore, sub-network rooted at *v* is duplicated for each input triggered by the fan-out nodes of *v*.
- The resulting sub-networks are either trees or leaf-DAGs.

Mapping the trees

- The strategy used by *Chortle* to map a tree is a combination of bin packing and dynamic programming.

- In the bin packing problem, items of different volumes must be packed into a finite number of bins or containers each of volume V in a way that minimizes the number of bins used.

- Each tree is traversed from the primary inputs to the primary outputs.

- At each node *v*, a circuit referred to as the ***best Circuit***, implementing the cone at *v* extending from the node to the primary inputs of the network is constructed.

- The ***best circuit*** is characterized by two main factors: The tree rooted at *v* and represented by a cone must contain the minimum number of LUTs and the output LUT (the root-LUT) implementing *v* should contain the maximum number of unused input pins.

Mapping the trees(cont'd)

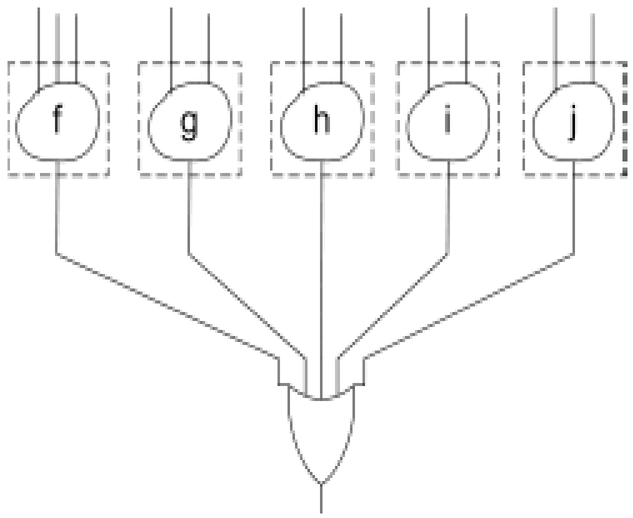
- For a primary input *p* the ***best circuit*** is a single LUT whose function is a buffer.
- Using the dynamic programming, the ***best circuit*** at a node *v* can be constructed from its fan-in nodes, because each of them is already optimally implemented.
- The procedure enforces the use of the minimum number of LUTs at a given node.
- The ***best circuit*** is then constructed from the minimum number of LUTs used to implement its fan-in nodes.
- The secondary goal is to minimize the number of unused inputs of the circuit rooted at node *v*.
- The construction of the tree is done in two steps. First a *two-level decomposition* of the cone at *v* is constructed and then this decomposition is converted into a *multi-level decomposition*.

Two-level decomposition

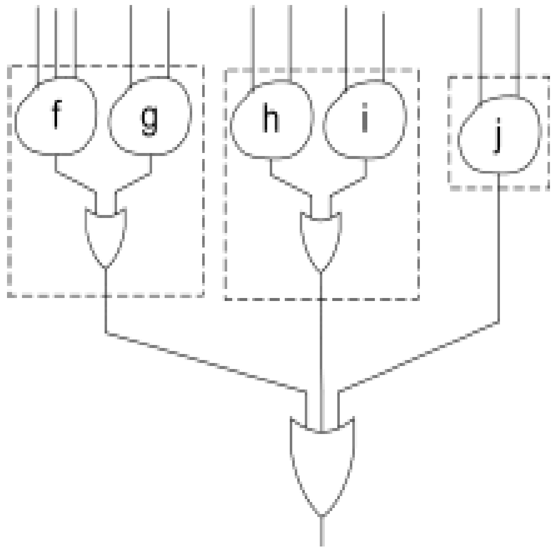
- The two-level decomposition consists of a single first level node and several two-level nodes.

- Each second level node implements the operation of the node being decomposed over a subset of the fan-in LUTs.

- The first level node is not implemented at this stage. This will be done in the second phase of the algorithm where the two-level representation is converted into a multi-



(a) Fan-in LUTs at a node



(b) The two-level decomposition

Fig: Chortle two-level decomposition

Two-level decomposition(cont'd)

- The two-level decomposition is constructed using a bin packing algorithm approach.
- In the traditional bin packing algorithm, the goal is to find the minimum number of bins with a given capacity, into which a set of boxes can be packed.
- In the *Chortle* algorithm, the bins are the second-level LUTs and the boxes are the fan-in LUTs.
- The capacity of each bin is the number k of LUT inputs.
- The packing at this stage consist of combining two fan-in LUTs, LUT1 that realizes the function f1 and LUT2 that realizes the function f2 into a new LUT (suppose LUTr) that implements the function f1 $\oslash$f2, where $\oslash$is the operation implemented by the fan-out node.

Two-level decomposition(cont'd)

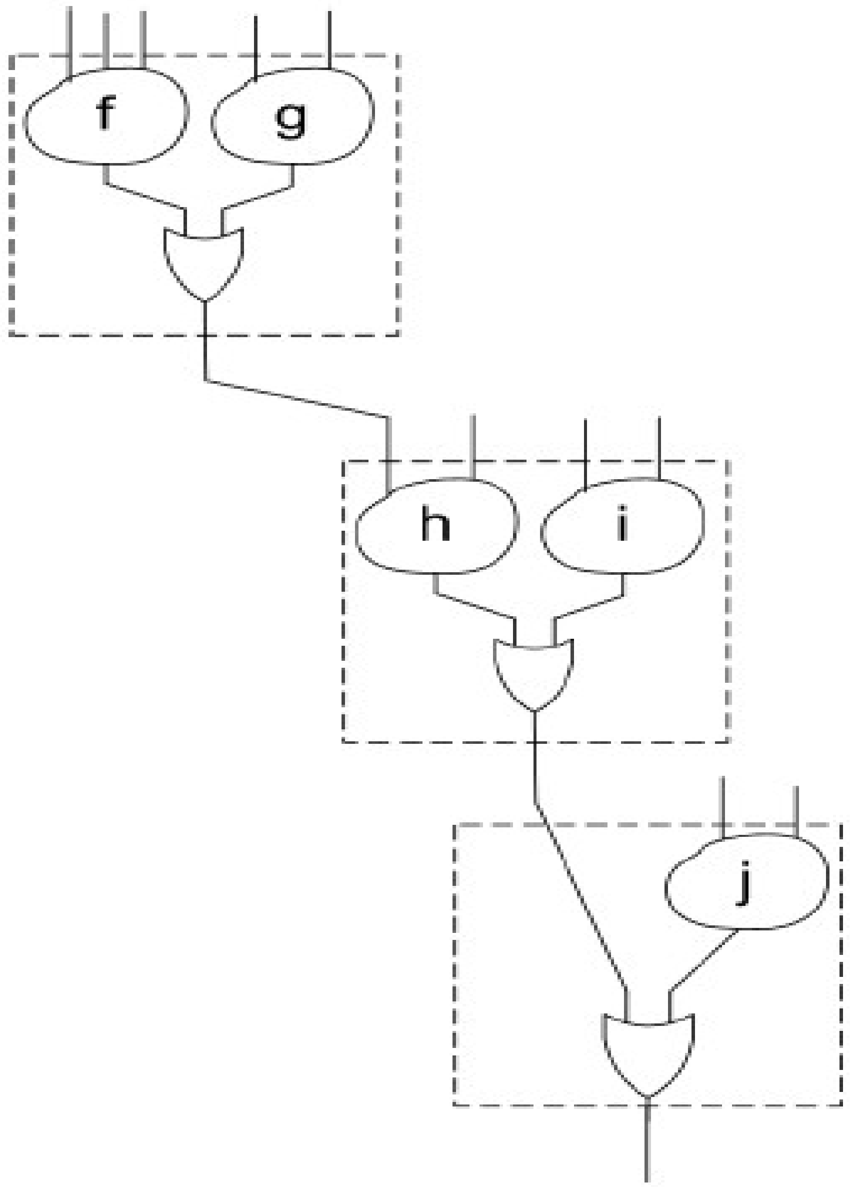
- In the previous figure (a) shows an original graph and its decomposition is shown in figure (b).
- In example, the ⊘is the OR function. The pseudo code of the two-level decomposition algorithm is given in below
- <u>Chortle's two-level decomposition</u>

```
Start with an empty list of LUTs
  While there are unpacked fan-in LUTs do
    If the largest unpacked fan-in LUT will not fit within any LUT in the list then
        create an empty LUT and add it to the end of the list
    End if
      pack the largest unpacked fan-in LUT into the first LUT it will fit within
  End while
```

Two-level decomposition(cont'd)

- This algorithm uses a *First-Fit-Decreasing (FFD)* method
- The FFD algorithm packs each number in order of non-increasing size into the first bin in which it fits, which places a fan-in LUT to be packed into the first LUT it will fit within.
- However, a *Best-Fit (BF)* approach that packed the fan-in LUTs into the LUT they best fit in, can also be used.
- **Multi-level Decomposition:** In the second step, the first-level nodes are implemented using a tree of LUTs.
- The number of LUTs used is minimized by using second level LUTs that have unused pins to implement a portion of the first-level three as shown in Figure

Multi-level Decomposition



Example of multi-level decomposition

Multi-level Decomposition(cont’d)

- The detailed procedure for converting two-level decomposition into a multi-level decomposition is given in algorithm

- <u> Chortle’s multi-level decomposition</u>

```
while there is more than one unconnected LUT do
    if there are no free inputs among the remaining unconnected LUTs then
        create an empty LUT and add it to the end of the LUT list
    end if
connect the most filled unconnected LUT to the next unconnected LUT with a free input
end while
```

Thank You