Lecture 10: CAD 3: FPGA Routing (Advanced)*

www.eecs.ucf.edu/~mingjie/EEL5722

Prof. Mingjie Lin

* Some slides adopted from UMN EE5301 by Kia Bazargan & NWU EECS357 lectures
Overview

• Recap + Feedbacks for the 2nd lab

• FPGA Routing
  – FPGA Routing Problem Formulation
    • Difference from conventional VLSI routing problem
  – Steiner Tree and MST Algorithms
  – Practical FPGA Routing Algorithms
FPGA Routing Problem Formulation
FPGA Routing Problem

- Routing represent the final step in that CAD system
- The router tool allocates the FPGA’s routing resources to interconnect the placed logic cells
- The router must ensure that all interconnections are formed
- Other constraints might be applied, such as maximizing the speed performance of timing-critical connections
INTRODUCTION

• A classic approach in solving the routing problem for FPGAs was to adopt a “Divide-and-Conquer” strategy:

1. Partition routing resources to routing areas
2. Perform Global Routing
3. Perform Detailed Routing

• Another approach is to perform one-step Detailed Routing
FPGA Routing Resources

- Prefabricated wire segments
  - Routing constraints: Sharing of a wire segment by different nets is not possible
- Limited Routability
  - High RC delays
  - Large area of switches
FPGA Routing
Routing Graph $G_r (V_r, E_r)$

- $V_r$: I/O pins of logic modules, wire segments
- $E_r$: feasible connections between the nodes
- Routing problem: Find vertex disjoint trees $T=\{T_1, \ldots, T_n\}$
Maze Routing

- Will find shortest path for a single wire, if such a path exists.
- **Two phases:**
  - Label nodes with distance, radiating from source
  - Use distances to trace from sink to source, choosing a path that always decreases distance to source
- **Use different cost functions to achieve routing objective**
  - Timing-driven
  - Routibility-driven
Steiner Tree & Basic Algorithms
Spanning Tree
Building Minimal Spanning Trees

• Prim’s algorithm: simple variation of Dijkstra’s SSSP algorithm
  – Change Dijkstra’s algorithm so the priority of bridge \((f \rightarrow n)\) is \(\text{length}(f, n)\) rather than \(\text{minDistance}(f) + \text{length}(f, n)\)
  – Intuition: Starts with any node. Keep adding smallest border edge to expand this component.

• Algorithm produces minimal spanning tree!
Prim’s MST algorithm

Tree MST = empty tree;
Heap h = new Heap();
// any node can be the root of the MST
h.put((dummyRoot → anyNode), 0);
while (h is not empty) {
    get minimum priority (= length) edge (t→f);
    if (f is not lifted) {
        add (t→f) to MST; // grow MST
        make f a lifted node;
        for each edge (f→n)
            if (n is not lifted)
                h.put((f→n), length(f,n));
    }
}

Minimum Spanning Tree is EASY!
The Minimum Steiner Tree Problem:

Given an undirected graph \( G \) with nonnegative edge costs and whose vertices are partitioned into two sets, **required** vertices and **Steiner** vertices, find a minimum cost tree in \( G \) that contain all the required vertices and any subset of the Steiner vertices.
Minimum Steiner Tree is HARD!

- Special cases:
  - Shortest path problem
  - Minimum spanning tree problem
  - But the general problem is NP-complete
  - Lucky for us:
    - Simple heuristic can easily achieve 2-optimal solution
    - Treat all Steiner nodes as required, run a MST algorithm
Timing-Driven FPGA Routing Problem & Practical Algorithms*

*Ismail 2006
ALGORITHM #1

PathFinder: A Negotiation-based Performance-driven Router for FPGAs
(Ebeling C. et. Al. FPGA 1995)
Ideas

• Given
  – Independent of the FPGA target arch. Routing resources as a Directed Graph $G(V,E)$
  – Given a signal $i$ in a FPGA-mapped circuit, $Net \ Ni = \{s_1, \ldots, t_i\}$, $N_i \in V$
  – Each nodes is tagged with a constant delay $d_n$ and a congestion cost $c_n$
    • $c_n = (b_n + h_n) \times p_n$ $\leftarrow c_n$: cost of a node $n$ in a route, $b_n$: basic cost of using $n$; $h_n$: related to the history of congestion on $n$ during previous iterations of the global router; $p_n$: related to the number of other signals presently using $n$

• The heart of the PathFinder algorithm
  – Negotiated Congestion (NC) Algorithm
Negotiated Congestion Algorithm

- Derived from an iterative scheme for global routing of custom IC’s
- Uses a delay/congestion trade-off for routed nets
  - Signals are allowed to share routing resources initially, but subsequently they must negotiate with other signals to determine which one needs the shared resource most
- PathFinder is composed of two steps:
  - A Signal Router: which routes one signal at a time using a Shortest Path (SP) algorithm with Breadth-First Search (BFS) approach.
  - A Global Router: which calls the signal router to route all signals, adjusting the resource costs to achieve full routing
Negotiated Congestion Algorithm

- During first iteration, Global router is run without any cost/penalty for sharing routing resources
- During subsequent iterations:
  - Cost/penalty is gradually increased
  - Nets in effect would negotiate for resources
  - Sharing of resources can still occur with sub-iterations
  - The global router reroutes signals using the signal router until no more resources are shared
- The Shortest Path w/ Rip-up and reroute approach is performed with a timing-analysis-based scheme
  - Calculated “Slack Ratio” used for negotiating
  - Slack ratio determines which net has the highest priority over a routing resource
Slack Ratio

• Slack Ratio for net (i) (SR\(_i\))
  – Delay of longest path of i / Delay of critical path
  – Symbolizes the relative contribution of each connection in the circuit (i.e. source-sink pair) to the overall delay of the circuit

• SR is used to calculate the weights of the routed nets
  – Nets with SR ~ 1 will be assigned a greater weight by the signal router. Hence, they will have a bigger chance of being routed directly
  – For nets with SR ~ 0, will yield smaller weights and more attention is paid to just routing them while avoiding congestion

• The net with the highest weight will gain access to the routing resource
Performance of Path-Finder

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CLBs</th>
<th>Logic Levels</th>
<th>APR delay (ns)</th>
<th>PathFinder delay (ns)</th>
<th>% over APR</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>116</td>
<td>8</td>
<td>113.5 (fails)</td>
<td>91.1</td>
<td>-19.7%</td>
</tr>
<tr>
<td>9sym</td>
<td>76</td>
<td>11</td>
<td>134.8</td>
<td>124.1</td>
<td>-7.9%</td>
</tr>
<tr>
<td>9symml</td>
<td>99</td>
<td>18</td>
<td>197.9</td>
<td>182.9</td>
<td>-7.6%</td>
</tr>
<tr>
<td>alu2</td>
<td>123</td>
<td>18</td>
<td>270.8 (fails)</td>
<td>243.3</td>
<td>-10.1%</td>
</tr>
<tr>
<td>dk16</td>
<td>128</td>
<td>7</td>
<td>78.5</td>
<td>79.5</td>
<td>+1.3%</td>
</tr>
<tr>
<td>duke2</td>
<td>99</td>
<td>7</td>
<td>141.5</td>
<td>106.5</td>
<td>-24.7%</td>
</tr>
<tr>
<td>ex1</td>
<td>75</td>
<td>7</td>
<td>120.3</td>
<td>117.9</td>
<td>-1.9%</td>
</tr>
<tr>
<td>kevb</td>
<td>74</td>
<td>8</td>
<td>112.7</td>
<td>86.6</td>
<td>-23.2%</td>
</tr>
<tr>
<td>planet</td>
<td>151</td>
<td>7</td>
<td>194.4 (fails)</td>
<td>179.9</td>
<td>-7.5%</td>
</tr>
</tbody>
</table>
Pros & Cons of Path-Finder

- **Advantages:**
  - Full routability achieved by allowing the overuse of routing resources at an initial stage
  - Keeps track of the routed nets to determine an optimal solution

- **Disadvantages:**
  - Routing and rerouting approach leads to a decrease in the routing process speed
  - Can cause problems for nets with high fanouts since they can span most of the FPGA space
ALGORITHM #2

A min-cost flow-based detailed routing algorithm
(Lee S. et. Al. ICCAD 2003)
Motivations

• Authors argue that most FPGA detailed routers route one net at a time
• The net-ordering problem is another major issue
• They present a routability-driven detailed routing algorithm, which also considers routing delay

• My take: sequential ➔ parallel
Ideas

• Routing all the nets connected to one common logic module simultaneously
  – Perfect for the LUT-based modules in the target FPGA architecture
  – The routing resources of an FPGA is modeled with a routing graph $G(V,E)$
  – A route of a net corresponds to a subtree in $G(V,E)$
  – The routing trees for all nets are vertex disjoint

Overall Problem: Given a routing resource graph $G$ for an FPGA, find vertex disjoint routing trees in $G$ for all the nets
Key Algorithm

Sub-Problem: The Routing for one LUT (ROL)

Given a routing resource graph $G(V,E)$ and a LUT, find routes for all the net segments connected to the LUT such that each edge and node is used at most once.

Algorithm FlowRoute
Input: $G_f, R, C, L$
Output: Routing trees for all nets in an FPGA
begin
1. Initialize $\lambda$
2. for each $l_k$ in $L$ do
3. Rip up all the nets connected to $l_k$
4. Call ROL_NF
5. Update costs and reset capacities
6. Update $\lambda$
7. Repeat Step 2-6 until no shared resource exists
end

Algorithm ROL_NF
Input: $G(V,E), R, C, l_k$
Output: routes for all net segments to $l_k$
begin
1. Construct the flow network $G_f(V_f,E_f)$
2. Assign costs and capacities
3. Run min-cost max-flow algorithm on $G_f(V,E)$
4. Derive the corresponding routes from the computed flow
end
Essence of Min-Cost Max-Flow (MCMF)

- The backbone of this work depend on the min-cost max-flow algorithm:
  - Cut: gaps between objects
  - Flow: actual number of wires across a cut
  - Capacity: max number of wires placed between a cut

- The min-cost max-flow algorithm assigns flows to all the cuts by observing the flow conservation rule
- Depth-first search technique is used
Example of MCMF
### Performance of MCMF

<table>
<thead>
<tr>
<th>circuit</th>
<th># of LUTs</th>
<th># of tracks</th>
<th>critical path delay (ns)</th>
<th>total wire length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>vpr</td>
<td>FlowRoute</td>
<td>improve</td>
</tr>
<tr>
<td>9symml</td>
<td>26</td>
<td>10</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>term1</td>
<td>32</td>
<td>13</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>apex7</td>
<td>63</td>
<td>13</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>example2</td>
<td>101</td>
<td>17</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>alu2</td>
<td>56</td>
<td>17</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>too-ltg</td>
<td>52</td>
<td>19</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>vda</td>
<td>114</td>
<td>23</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>alu4</td>
<td>390</td>
<td>33</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>s298</td>
<td>490</td>
<td>27</td>
<td>27</td>
<td></td>
</tr>
</tbody>
</table>

Remember for the PathFinder (Algorithm #1) **alu2**: CLBs = PathFinder Delay (ns) = 243.3
Pros & Cons of MCMF

• Advantages:
  – Multiple nets are routed simultaneously
  – Critical path delay improvement over VPR/PathFinder
  – Total Wire length improvement

• Disadvantages:
  – Assumes a symmetrical array-based FPGA
  – LUT-ordering problem is solved through iterative refinement which could be computationally expensive
REFERENCES


REFERENCES


HDL Tip of the Day

- Generate Blocks

```verilog
module generate_example();

reg read, write = 0;
reg [31:0] data_in = 0;
reg [3:0] address = 0;
wire [31:0] data_out;

initial begin
    $monitor ("%g read=%b write=%b address=%b data_in=%h data_out=%h", 
                $time, read, write, address, data_in, data_out);
    #1 read = 0; // why only for read
    #3 repeat (16) begin
        data_in = $random;
        write = 1;
        #1 address = address + 1;
    end
    write = 0;
    address = 0;
    #3 repeat (16) begin
        read = 1;
        #1 address = address + 1;
    end
end

always @ (*) begin
    if (write) mem[address] = data_in;
    end

always @ (read, address) begin
    if (read) data_out = mem[address];
end

endmodule
```
HDL Tip of the Day

• Difference between $monitor,$display & $strobe?
  – Syntax:
    – $display ("format_string", par_1, par_2, ... );
    – $strobe ("format_string", par_1, par_2, ... );
    – $monitor ("format_string", par_1, par_2, ... );
  – Differences?