# *AutoMapper* - An automated tool for optimal hardware resource allocation for networking applications on FPGA (under review) *

Swapnil Haria
BITS, Pilani
Rajasthan, India
swapnilster@gmail.com

Viktor Prasanna
Ming Hsieh Dept. of Electrical Engineering
University of Southern California
Los Angeles, CA 90089
prasanna@usc.edu

## ABSTRACT

It has now become imperative for routers to support complicated lookup schemes, based on the specific function of the networking hardware. We have developed an automated tool, AutoMapper, which can map lookup schemes onto a particular target architecture optimally, thereby providing a superior alternative to the time-consuming and resource-inefficient technique of manual conversion. It is based on an Integer Linear Programming (ILP) formulation that is able to allocate the limited hardware resources for a single lookup scheme, while optimizing either of the three performance metrics of latency, throughput or power consumption. We demonstrate the operation of the developed tool, by successfully mapping complex real world lookup schemes onto a state-of-the art FPGA device, with execution times being under a second on a dual-core computer with 4 GB of RAM, running at 2.40 GHz.

## Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Packet-switching networks*

## Keywords

Resource Allocation, Networking, High-level Tool

## 1. INTRODUCTION

Ethernet/IP based packet forwarding now comprises of complex sequences of lookup operations. A router is required to support complicated packet forwarding technologies (e.g. MAC-in-MAC [9], Q-in-Q [4], etc). It is no longer possible to ensure an optimal resource utilization using manual organization techniques due to the increasing complexity of lookup schemes, as well as the large number of potential implementation choices. This has necessitated the development of a automated mapping tool, to solve the mapping problem for complicated lookup schemes on hardware devices to accelerate the production of high-performance networking solutions. Field Programmable Gate Arrays (FPGAs) are the ideal choice of hardware platform for high-performance networking applications, due to the inherent parallelism, reconfigurability and the abundant on-chip resources available [11, 1, 7]. The FPGA fabric supports multiple algorithmic solutions for the efficient implementation of packet lookup engines. These various solutions excel with regards to some performance metrics at the cost of others. It has become exceedingly difficult to efficiently organize packet lookup schemes on hardware, using manual methods as the complexity of the lookup schemes as well as the variety of the search techniques and storage locations (on-chip/off-chip) increases. As a result, we have developed an automated tool which is able to solve the problem of converting complex lookup schemes into an FPGA-based pipelined architecture under a tight resource budget. Additionally the tool offers the flexibility of generating an optimal implementation, with the aim of either minimizing latency, conserving power or improving throughput. To summarize, we make the following contributions in the paper:

- Development of an automated mapping tool to
  - Convert complex lookup schemes into hardware implementations suitable for a particular target device (FPGA),
  - Optimize the performance metric of either latency, throughput or power in the generated mapping.

## 2. BACKGROUND

### 2.1 Lookup Scheme Representation

A collection of various lookup operations in a particular sequence is referred to as a lookup scheme. A graph is a convenient structure to represent the order of operations and the dependencies existing between the lookup schemes. This form of lookup scheme representation has been referred to as a Lookup Flow Graph (LFG) in related literature [3]. A LFG is a sequential arrangement of field nodes, with each individual field node depicting a distinct lookup operation. Information about the lookup operation as well as a list of alternative hardware implementations is associated with each

*(*) Supported by U.S. National Science Foundation under grant XXXX.


*FPGA* '12 California USA

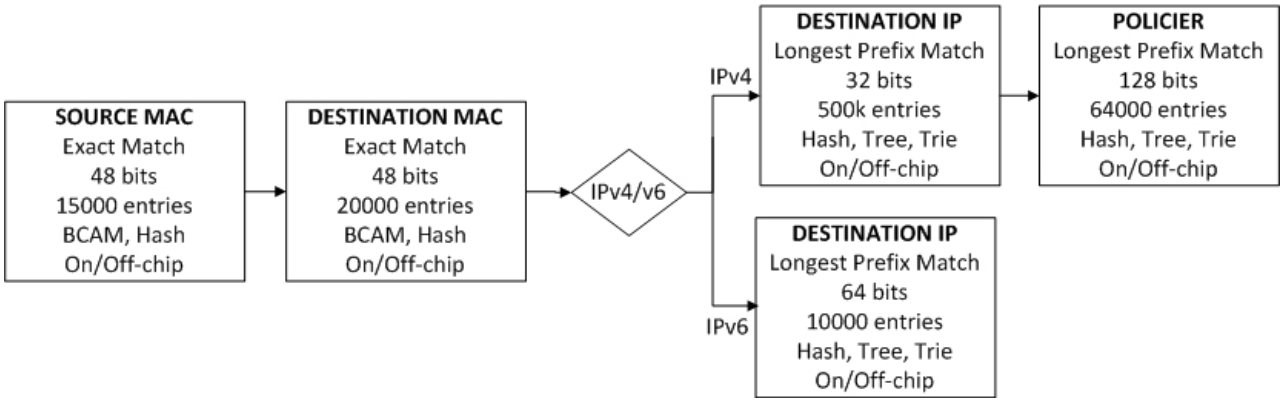


**Figure 1: Sample LFG**

node. The specific header field under consideration, width of the field, the type of match required along with the size of the corresponding lookup table constitutes the information content of a LFG field node. There may be multiple paths existing in the LFG, depending on the presence of decision nodes. A decision node enforces a condition on the properties of the packet being processed. The choice of path after the decision node depends on the outcome of the condition. A typical LFG representation is shown in Figure 1, containing 5 field nodes and 1 decision node.

## 2.2 Mapping Problem

The mapping problem can be fundamentally defined as the generation of a linear pipeline architecture which incorporates the lookup operations specified by a LFG, which can be accommodated on a given target device while optimizing one of the three performance metrics of packet lookup engines: 1) power efficiency, 2) throughput, and 3) latency.

The mathematical formulation can be stated as follows: given an input LFG, a collection of desired implementation choices for each field node of the LFG, the resource constraints of the target device, to select the appropriate implementation choice for each node such that the overall hardware implementation fits onto the target device, while seeking to optimize the chosen performance metric.

The problem of mapping a LFG to a linear pipeline architecture is made complex due to the multiple possible paths available to an incoming packet. This is resolved by inclusion of stages, corresponding to every field node in the LFG, in the linear pipeline. This calls for the inclusion of additional logic, to determine the actual pipeline stages in which lookup operations occur for a particular packet. Also there are other hardware-level considerations which obscure the high-level view of the problem. Hence assumptions such as considering the latency only in terms of the number of pipeline stages involved to eliminate the dependence on the actual operating frequency, are made to facilitate a a high-level definition. Detailed problem definition can be found in [3, 2].

An abstract model is constructed for the target device so as to neglect extraneous information and concentrate only on the resources critical to the mapping task. FPGA is the target device considered, and the relevant resources on which the model is based are I/O pins and on-chip memory.

# 3. TOOL DETAILS

A GUI-based tool has been developed in the Java programming language, to provide an optimum solution to the problem of hardware resource allocation in the context of networking applications. It is capable of generating high-level hardware mappings for any feasible combination of complex lookup schemes as well as target device specifications. A screenshot of the input screen of the software can be seen in Figure 2.

## 3.1 User Inputs

The tool initially asks the user to provide the total number of nodes in the input LFG. This is used to dynamically generate the layout for the GUI, to enable easy interaction of the user with the software. The user inputs required for each node are the name of the lookup operation, the number of the previous node in the LFG, the field width as well as the number of entries in the lookup table. 4 commonly used hardware implementation schemes for lookup operations, 1) Hash, 2) Tree, 3) Trie, 4)CAM, are provided as possible options for the physical mapping for the lookup operation at each node [6, 8, 11, 1, 7]. A maximum of three of these implementation choices may be disabled for a node depending on the user's requirements. The tool is sophisticated enough to require minimum user input, and is capable of computing the resource requirements for each of the implementation choices selected for a LFG field node. The specifications of the FPGA device selected as the target for the implementation, specifically the on-chip memory and the number of I/O pins, are also required as input by the software. Finally, the software can operate in one of three different modes of operation as elaborated in Section 3.2 and the user must select the preferred mode for the current mapping.

## 3.2 Modes of Operation

The Automapper functions in one of three modes of operation; the high-level mapping generated by the tool can be optimized for any of the following metrics: latency, throughput or power. In each of the modes, the software displays the most optimum choice of hardware implementation for all the LFG nodes, the overall latency of the physical mapping of the set of lookup schemes depicted by the input LFG as well as the FPGA resources consumed by the implementation.

The implementation chosen in the latency-optimized mode has the minimum latency amongst all hardware implemen-


**Figure 2: Screenshot of the Automapper software**

tations of the input LFG which can be accommodated on the target FPGA, but there is no optimization of the resource usage. The throughput-optimized mode is configured to generate a hardware mapping which leads to the most number of pipelines on the target FPGA. All the pipelines will be identical, and act as independent lookup engines. This mode leads to the software preferring implementation choices which consume lesser device resources, but may exhibit a higher latency. The low-power mapping mode aims to minimize the dynamic power consumed by the hardware mapping, and generally results in a higher latency of the pipeline.

## 3.3 Tool Implementation

The tool is able to understand the structure of complex LFGs using the previous node number parameter associated with each node. It uses this to identify the distinct paths existing in the complex LFG, and also identify the location of the decision nodes, if any. The software uses a parameterised approach to compute the resource costs of any selected implementation choice, and these parameters are chosen appropriately to reflect real-world observations. The software operates by first creating a MILP formulation for the given input conditions, and correspondingly making a optimization model using the IBM ILOG CPLEX 9.0 Callable Library [5]. The optimal solution provided by CPLEX is then interpreted by the tool to generate the appropriate high-level mapping.

The algorithm driving the tool considers the latency as the overall latency of the pipeline, as measured by the number of stages involved in the entire lookup operation. The number of stages is considered in order to make the latency independent of the actual frequency of operation. The latency-optimized mode seeks to minimize the number of stages involved in the lookup engine, while ensuring that the mapping generated can be accommodated on the target device. The throughput-optimized mode of operation requires a repeated application of the above process to compute the maximum number of lookup engines that can fit on the target device. This is caused by the restrictions imposed by the MILP formulation. The low-power mapping mode also first generates a latency-optimized mapping, whose power footprint is calculated by using a suitable algorithm. The algorithm uses the memory requirements of each implementation choice in addition to a scaling parameter differing for different implementation choices. The tool then tries to generate a mapping whose power footprint is the least amongst the various feasible mappings.

## 3.4 ILP Formulation

The mapping problem is solved by the tool by converting it to a corresponding ILP formulation, and solving it using the CPLEX library. The objective function of the ILP formulation seeks to optimize the chosen performance metric. The constraint equations deal with the restrictions imposed by the resources (I/O pins and on-chip memory) available on the FPGA. Also, an additional constraint is added to ensure that only one implementation choice can be selected for every decision node for every potential solution identified. For each input case, the software generates the appropriate objective function, as well as the constraint equations, solves the developed ILP formulation, and then checks the validity of the results.Detailed ILP formulation can be found in [2]. An alternate solution using back tracking has been in proposed in [3].

# 4. PERFORMANCE EVALUATION

The developed tool is tested by mapping a set of complex lookup schemes onto a state-of-the-art FPGA platform. We considered Xilinx Virtex-6 XC6VLX760 [10], with 33 Mb of on-chip memory and 1200 external I/O pins to be the target device. We realistically assumed that 650 external I/O pins and 25 Mb of on-chip memory were available for the actual implementation of the lookup engine, with the rest being dedicated for the purposes of input/output and other tasks. The software was executed on a dual-core laptop with 4 GB of RAM, running at 2.40 GHz.

To test the functionality of the software, we developed five realistic lookup schemes based on actual lookup schemes used by network equipment vendors. The relevant details of the schemes are reported briefly in Table 1, to conserve space. The lookup schemes are numbered from I-V, in increasing orders of complexity. The lookup schemes developed have a great amount of variation in terms of the individual lookup operations involved, as well as the sequence of operations to test the robustness of the tool. The lookup schemes created for testing purposes were converted into an LFG initially, to simplify the input process.

The results of the mapping process are compiled in Table 2. The overall latency obtainable for implementations generated by the latency-optimized mapping mode are much smaller than the latency of other resulting implementations. This is caused by the fact that implementation choices with lesser number of pipeline stages, such as CAM and hashing, are preferred over other choices, subject to the resource constraints of the FPGA.

However, these are also much more resource-intensive and

**Table 1: Details about Sample Lookup Schemes**

| S.No. | Field Nodes | Decision Nodes | Field Width | Table Size |
|---|---|---|---|---|
| I | 4 | 0 | 16-48 | 64k-256k |
| II | 6 | 0 | 16-48 | 32k-100k |
| III | 6 | 1 | 20-128 | 64k-128k |
| IV | 7 | 1 | 16-128 | 4k-256k |
| V | 7 | 2 | 20-128 | 64k-128k |

**Table 2: Mapping results with latency being in terms of number of pipeline stages and power being in terms of comparative units**

| Lookup Scheme | Latency-Optimized | | | Throughput-Optimized | | Power-Optimized | | |
|---|---|---|---|---|---|---|---|---|
| | Latency | Power | Improvement(%) | Latency | Pipelines | Latency | Power | Improvement(%) |
| Scheme I | 5 | 135.2 | 92.42 | 38 | 7 | 66 | 10.34 | 92.35 |
| Scheme II | 7 | 131.3 | 92.63 | 69 | 5 | 95 | 10.92 | 91.68 |
| Scheme III | 8 | 104.45 | 95.53 | 135 | 4 | 179 | 16.2 | 84.49 |
| Scheme IV | 15 | 39 | 96.39 | 231 | 4 | 416 | 1.48 | 96.20 |
| Scheme V | 10 | 104.5 | 95.85 | 199 | 4 | 241 | 16.31 | 84.39 |

power-hungry as compared to other potential implementations such as the trie-based or BST-based techniques. Consequently, the power-optimized results employed these implementation choices, leading to a decrease in power consumption, but in the process resulting in a greater latency. The power consumption reported in the table is only based on an empirical method of calculating dynamic power consumption. The power results are not in any conventional unit of power. and should only be considered for the purposes of comparison. An average power reduction of 89.8% was found for this mode as compared to latency-optimized results. However, the latency-optimized mode results in an average improvement in overall pipeline latency of 94.57% as compared to the power-optimized mode.

The throughput-optimized mapping mode aimed at maximising the number of independent lookup engines that could fit onto a single target device. In the process, overall latency of the pipeline was sacrificed in order to reduce the resource footprint of a single lookup engine. All the mapped implementations had at least four pipelines fitting on the device, which translates approximately to a $4X$ increase in throughput. The execution time for all the test cases was about 100ms, excluding the time for user input.

# 5. CONCLUSION

In this work, we have developed an automated tool for mapping complex lookup schemes onto hardware devices, with the aim of effective resource utilization. The developed tool needs minimum user input, and is equipped with sophisticated and robust models in order to estimate the resource costs and performance of four common implementation alternatives for lookup operations. The tool offers the flexibility of optimizing the generated mapping with respect to latency, throughput or power consumption. The tool functions by creating a ILP formulation for the given input conditions, and using the IBM CPLEX libraries to solve this model optimally. A more comprehensive version of the tool is under development which will aim to provide an end to end solution for the mapping problem, by producing as output the synthesized implementation for the optimally mapped pipeline in Verilog, by interaction with specific CAD software.

# 6. REFERENCES


[1] M. Bando, Y.-L. Lin, and H. J. Chao. Flashtrie: Beyond 100-gb/s ip route lookup using hash-based prefix-compressed trie. *Networking, IEEE/ACM Transactions on*, PP(99):1, 2012.

[2] D. for Review.

[3] T. Ganegedara, V. Prasanna, and G. Brebner. Optimizing packet lookup in time and space on fpga. In *Field Programmable Logic and Applications (FPL), 2012 International Conference on*, 2012.

[4] Huawei. Q-in-q. `www.huawei.com/products/datacomm/pdf/view.do?f=556`.

[5] IBM. Ibm ilog cplex optimizer. `http://www-01.ibm.com/software/websphere/products/optimization/academic-initiative/index.html`.

[6] H. Le, T. Ganegedara, and V. Prasanna. Memory-efficient and scalable virtual routers using fpga. In *Field Programmable Gate Arrays (FPGA), 2011 International Symposium on*, march 2011.

[7] H. Le and V. Prasanna. Scalable high throughput and power efficient ip-lookup on fpga. In *Field Programmable Custom Computing Machines, 2009. FCCM '09. 17th IEEE Symposium on*, pages 167 –174, april 2009.

[8] H. Song and J. W. Lockwood. Efficient packet classification for network intrusion detection using fpga. In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, FPGA '05, pages 238–245, New York, NY, USA, 2005. ACM.

[9] Wikipedia. Ieee 802.1ah-2008. `http://en.wikipedia.org/wiki/IEEE_802.1ah-2008`.

[10] Xilinx. Virtex-6 lxt fpgas. `http://www.xilinx.com/products/silicon-devices/fpga/virtex-6/lxt.htm`.

[11] S. Yusuf, W. Luk, M. Sloman, N. Dulay, E. Lupu, and G. Brown. Reconfigurable architecture for network flow analysis. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(1):57 –65, jan.

2008.