RE-HASE: Regular-Expressions Hardware Synthesis Engine

Mohamed El-Hadedy
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
hadedy@illinois.edu

Xinfei Guo
Department of ECE
University of Virginia
Charlottesville, VA 22903, USA
xg2dt@virginia.edu

Xiaoping Huang
Northwestern Polytechnical University
Xi'an, Shaanxi 710065, China
huangxp@nwpu.edu.cn

Martin Margala
Department of ECE
University of Massachusetts Lowell
Lowell, MA 01854, USA
martin_margala@uml.edu

ABSTRACT

The regular expressions (RegEx) provide a concise and flexible means for matching strings of text, such as particular characters, words or patterns of characters. In general, there are two types of RegEx - Deterministic Finite Automaton (DFA) that can be only in and transition to one state at a time and Non-deterministic Finite Automaton (NFA), which support multiple states transitions. RegEx matching hardware accelerators are costly mainly because they need to support programmability of regular expressions. Typical networking applications, however, do not have to update matching rules (including Regex definitions) on the fly. The existing tools for implementing RegEx in hardware are not able to cover various matching rules due to lack of the reconfigurability. In this paper, we propose a full tool which can translate the RegEx (NFA or DFA) to RTL and can automatically map regular expression rule set into FPGA or ASIC based on the matching length. The proposed tool refers the state machine module concept in Verilog for translation and has the flexibility of producing RegEx as a set of rules that are regrouped based on the critical path delay of each rule or the total number of cycles each rule could take. The tool works as following: firstly, it analyzes the matching length of each rule, and then an interval is defined, and the tool groups the common set of rules into subgroups. Each subgroup is treated as an independent regular expression rule set. For each sub-group, the tool first automatically translate the NFA table into a parallel logic circuit, then stitch each logic circuit into sub-modules. Each sub-module can be placed and routed independently. The final output of the tool could be implemented on the distributed-memory systems or shared-memory architecture with partially-reconfigurable style. The proposed tool can benefit from the coherency interfaces, which are provided by IBM (CAPI) and Intel (HARP, QPI) to store the subgroup outputs in the cache memory. To exercise the tool, we implement widely-used RegEX benchmarks on both FPGA and ASIC platforms with 28/32nm technology node, and the synthesis results show that the design achieves better performance and lower total power compared to previous implementations of RegEx on hardware.

H2RC'17 November 17, Denver, CO, USA

DOI: XXXXX

Keywords

Regular expression; ASIC; FPGA; Synthesis

1. INTRODUCTION

Field-programmable gate array (FPGA) is an integrated circuit that can be configured after manufacturing. It can be used to implement any logic function that could be performed by a hardwired circuit, but has the great advantage that its functionality can be updated after shipping. Compared to applications running on a CPU, applications implemented on an FPGA can benefit from massive parallelism, fast on-chip communication, and Application-specific hardware designs, potentially allowing for superior performance. Nevertheless, developing applications for FPGAs is considerably more difficult and time-consuming than writing software for a general purpose processor.

Regular expressions can be processed by using a finite automaton, which makes them suitable for FPGAs, since state machines can be implemented directly in hardware not requiring external memory. Implementing the regular expressions on FPGA can be benefited from the parallelism of the FPGA. There are many benchmarks that address the complexity of the regular expression such as backdoor, dotstar and spyaware. In this paper, we implemented the above three regular expressions by using the NFA and the DFA approaches. The input stream of any regular expression basically is an ASCII code (8-bit). In the paper, The DFA

implementation relies on using the table based architecture, which means each state consists of a table of the 256 possibilities of which state should be based upon the input value (8-bit). Each of regular expression benchmark we used consists of sets of rules. We implemented the rule sets separately and makes them running in parallel benefiting from the FPGA parallelism architecture. Each rule sets can be translated by the ISE as a big decoder describes by number of states and each state has another decoder describes the transition to the states based on the input stream (8-bit). The maximum frequency of each rule set is effected by the size of the decoder because the decoder naturally is a combinational circuit, which can increase the critical path between the input and the output. We also implemented the regular expressions on the FPGA by using the NFA approach, which relies on using register (LUT) architecture by describing each rule set as a couple of registers are handling the data from the input and the output through a couple of comparator. This implementation shows a better performance compare to the DFA. The reason is the DFA relied on decoder and NFA divided the critical path between the input and the output by partitioning to registers (LUT). The same RTL is also implemented in ASIC to show that the approach is efficient and fast when building the custom regular expression hardware.

2. RE-HASE FLOW

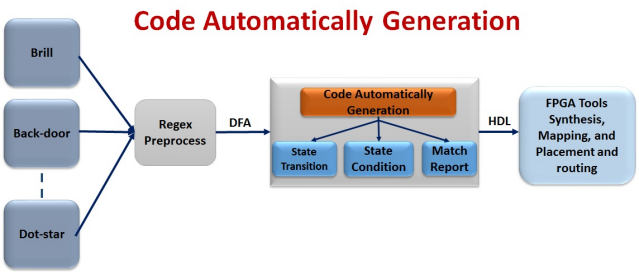

Figure 1: RE-HASE

We develop the tool called Re-HASE to automatically generate the HDL code from the xFA (NFA or DFA) transition table. Based on the xFA input file, we treat each transition table into an independent sub-module in HDL. So when there are how many transition table files in your working directory, there will be how many HDL code files would be generated. The generated HDL codes are independent; they can be assembled into a TOP HDL module. All of the generated HDL code are synthesizable.

In the tool, we refer the state machine module concept in Verilog design for translation, as shown in Fig. 1. We read transition table file line by line and parse each word in single line. According to the result, we calculate how many states would be inferred in the transition table; extract the transition conditions from one state to another state. When all the conditions are defined, we generate the state machine kernel. We precede each transition table file one by one.

The code automatically generation stage in the tool consists of three stages as in Fig. 1:

- **State Transition: construct the state machine State;**
- **State Condition: Extract the matching condition between different DFA states and drive the state to jump;**
- **Match Report: Collect the matching information and output onto the bus.**

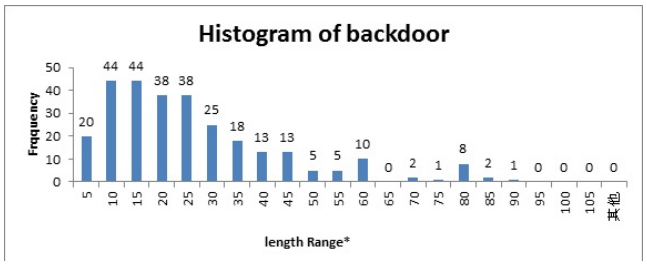

Figure 2: Histogram of backdoor

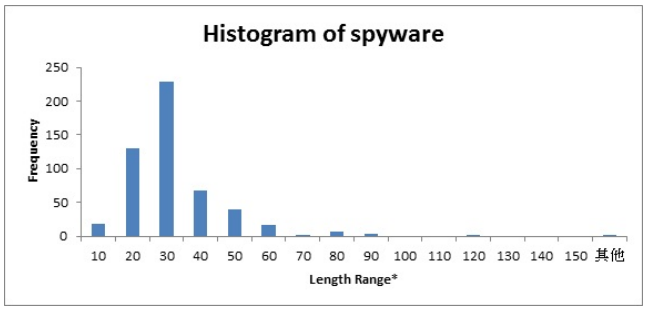

Figure 3: Histogram of spyware

The output of our tool is on HDL format, which can be synthesized, mapped, place and route by either FPGA tools or standard digital ASIC design flow.

Beside the basic functionality of the tool, which has been described above. The tool has a set of unique advantages such as:

- The tool analyzes the matching length of each rule. For example, based our analysis, the length of spyware or backdoor ranges from 5-100, as shown in fig. 2 and 3;
- We can define the interval and the tool can group the length in the interval into a separate subgroup. For each subgroup, we treat them as an independent regular expression rule set. In other words, Regular expression in each group almost has the same matching length;
- The tool could process each subgroup in sequence. For each subgroup, the tool first automatically translate the NFA translate table into parallel logic circuit, then stitch each logic circuit in to sub module. Each sub module can be placed and routed independently.

3. FPGA & ASIC SYNTHESIZED RESULTS

`Snort` [2] is an open source Network Intrusion Detection System combining the benefits of signature, protocol and anomaly based inspection and is considered to be the most

Table 1: FPGA Synthesis Results for Benchmarks with Xilinx-V7 VC707 Board

Benchmark	Critical Path Delay (ns)	Logic latency (ns)	Route latency (ns)	Utilization (%)
dotstar005	1.257	0.256	1.001	2%
dotstar010	1.269	0.26	1.009	2%
dotstar020	1.294	0.269	1.025	2%
spyware_put	1.34	0.26	1.08	1.2%
backdoor	1.42	0.25	1.17	3.5%

Table 2: ASIC Synthesis Results for Benchmarks in 28/32nm technology

Benchmark	Critical Path Delay (ns)	Static Power (mW)	Dynamic Power (W)	Total Power (W)	Area (mm^2)	# of gate (K)
dotstar005	0.16	0.9582	0.284	0.2851	0.08653	176.7
dotstar010	0.2	0.9612	0.285	0.2860	0.08680	177.3
dotstar020	0.17	0.9513	0.282	0.2829	0.08588	175.4
spyware_put	0.28	0.2807	0.082	0.0822	0.02533	51.7
backdoor	0.24	0.1883	0.083	0.0831	0.01651	33.7

widely deployed IDS/IPS technology worldwide. We select `backdoor` [1] and `spyware-put` [3] rule sets from SNORT. In them, `Backdoor` rule sets is used to detect traffic generated by backdoor network connections, including those created by attackers using many rootkits and stealthy remote control applications (like subseven, netbus, and deepthroat).

`Dotstar` [4] is combined set of synthetic regular expressions for Becchi et al. containing all variations of the synthetic dotstar rules created from the backdoor snorts rules and the spyware rules used in the evaluation. We changed the possibility of the dotstar, because the dotstar is the bottleneck when processing regular expression.

Table 1 shows the FPGA implementation with the proposed RE-HASE generated RTL. The evaluation is done on Xilinx Vertex-7 development board. Shown in Table 2 are ASIC implementation results, where we synthesize the same benchmark suite with 28/32nm technology node using the standard synopsys design flow. The area is evaluated based on P&R results. It shows that all benchmarks can run at very high frequency while consuming a small amount of power. Due to that the designs are highly optimized, they take small area, which lead to low static power consumption. Overall, both evaluation results show that the RE-HASE flow is able to achieve low latency and low area (e.g. all designs take less than 5% of the whole FPGA fabric and less than $0.1mm^2$ on ASIC).

4. CONCLUSIONS

In this paper, we propose a full tool which can translate the RegEx (NFA or DFA) to RTL and can automatically map regular expression rule set into FPGA or ASIC based on the matching length. The proposed tool refers the state machine module concept in Verilog for translation and has the flexibility of producing RegEx as a set of rules that are regrouped based on the critical path delay of each rule or the total number of cycles each rule could take. In the future, we are going to use the cache-coherency interfaces as CAPI (IBM) and QPI (Intel) to implement the extra advantages of the tool, which is allowing the user to regroup the rules based on the number of cycles or the length of the critical path. The coherency interfaces will allow us to store the outputs of the regular-expression rules, which are regrouped in away the total execution-time will be reduced.

5. ACKNOWLEDGMENTS

This work was supported by Azure Research Hardware Program, Grant Award No. CRM0518510 and Intel Hardware Accelerator Research Program. Grant Award no. 085987.

6. REFERENCES

[1] Jiacun Wang, editor. Handbook of Finite State Based Models and Applications, crc press, 2013.

[2] Snort - Network Intrusion Detection & Prevention System https://www.snort.org/.

[3] Tech Tips: Spyware Surpasses Viruses, Worms and Unintended Deletions, http://www.washington.edu/doit/tech-tips-spyware-surpasses-viruses-worms-and-unintended-deletions.

[4] D. Pasetto, F. Petrini, and V. Agarwal. Dotstar: breaking the scalability and performance barriers inÂăparsing regular expressions. *Computer Science - Research and Development*, 25(1):93–104, May 2010.