Partial Reconfiguration And Specialized Circuitry for Flexible FPGA-based Packet Processing

Sven Hager Daniel Bendyk Björn Scheuermann

Computer Engineering Group
Humboldt University of Berlin, Germany
Email: {hagersve, bendyk, scheuermann}@informatik.hu-berlin.de

***Abstract*—In order to process network packets at high rates, network functions like routing or firewalling require specialized hardware like TCAMs (Ternary Content Addressable Memories), ASICs (Application Specific Integrated Circuits), or GPUs (Graphics Processing Units). Such hardware must be fast enough to process packets at line rate, and furthermore, it must be programmable in order to update the packet processing policy (e. g., a forwarding table or firewall rule set). From a fundamental point of view, though, these goals are conflicting because a generic programmable circuit must provide sufficient resources to support a wide range of policies, which can lead to unused circuitry and low clock rates. In addition, it misses logic optimization opportunities with regard to the structure of the installed policy. In this work, we investigate the optimization potential of automatically generated network processing circuits that are tailor-made for a specific policy. Using the example of router forwarding information bases (FIBs), we demonstrate that circuits which are partially evaluated with respect to the implemented FIB need more than one order of magnitude less logic resources than an equivalent generic forwarding circuit. We combine this approach with the partial reconfiguration capability of FPGAs in order to obtain an efficient low-latency forwarding engine whose matching circuitry can be replaced on demand.**

***Index Terms*—Circuit Generation; Packet Forwarding; Partial Evaluation**

I. Introduction

Transmitted messages in packet-switched networks are subject to a variety of important control functions, such as IP forwarding or firewalling. Although these functions may serve different purposes, e. g., guiding a network packet to a suitable next hop or implementing a security policy, they all share two major requirements: first, they must be fast enough to process packets at the respective line rate in order to avoid performance bottlenecks in the network. Second, they must be programmable in order to reflect changes in the implemented policy, such as route updates or new rules in access control lists [1], [2]. Especially the first requirement is paramount in demanding environments such as backbone networks where link speeds of 40 Gbps or more are the rule rather than the exception [3]. Assuming worst-case traffic consisting of 64-byte Ethernet frames and a link rate of 40 Gbps, incoming network packets must be processed in less than 13 ns. These real-time constraints and the fact that routing tables and firewall rule sets tend to grow rather than to shrink [4] underline the need for hardware-accelerated packet processing engines.

Accordingly, many sophisticated packet processing schemes have been proposed which are based on special-purpose hardware providing massive true parallelism, such as ASICs [5], [6], GPUs [7], [8], and FPGAs [9]–[12]. Although these approaches differ greatly in detail and with respect to the underlying hardware platform, their architectures are mostly based on a strict separation between a generic packet processing algorithm and a corresponding configuration memory that stores the employed packet processing policy. For instance, an IP router runs a longest prefix matching algorithm in order to forward network packets according to the currently installed forwarding information base (FIB), while a firewall employs a generic packet classification algorithm whose operation is guided by a rule set stored in memory. As a result, these architectures have to process two input parameters at runtime: (1) the network packets, and (2) the in-memory data structure which represents the currently active packet processing policy.

In this work, we take a fundamentally different approach that merges the packet processing algorithm and the corresponding policy into a single unified circuit description. That way, we eliminate the need to interpret the policy data structure at runtime by partially evaluating the employed algorithm with respect to the policy S—that is, instead of utilizing a generic configurable circuit C, we replace all configurable variables by constants based on S, which yields a new circuit C_S whose operation is fixed on the policy S, as sketched in Figure 1. The advantages of this technique over generic memory-based approaches are threefold: first, due to the fact that specialized circuits incorporate the policy in their implementation, no configuration memories and thus less hardware resources are required. Second, as the specialized circuitry only needs to

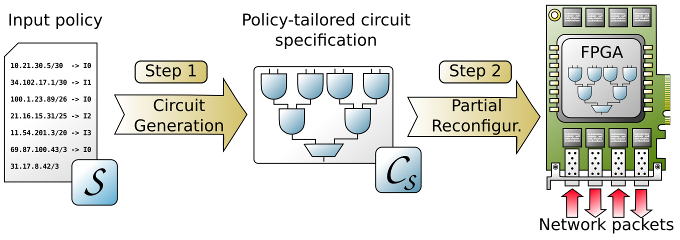


Fig. 1: Partial reconfiguration replaces policy-specific circuitry.

implement one specific policy S, it is generally much smaller than an equivalent generic circuit where S is not known at design time. Third, the specialized circuits can be logic-optimized with regard to the structure of the policy S because S is integrated into the circuitry itself.

Of course, the concept of specialized packet processing circuits requires an implementation platform whose circuitry can be adapted in case of a policy change. Therefore, we evaluate the proposed approach at the example of an IPv4 packet forwarding engine on a NetFPGA 10G board. By utilizing the partial reconfiguration ability of FPGAs, we can replace the matching circuitry at runtime without the need to power down the device, as sketched in Figure 1.

In short, the key contributions of this paper are:

- We present a 40 Gbps packet forwarding architecture on the NetFPGA 10G which leverages partial reconfiguration to replace the matching circuits at runtime.
- In contrast to previous works in this direction [13], [14], we compare the efficiency of specialized circuits to the state-of-the-art generic approach based on TCAMs. We find in our experiments that the FIB-specific circuitry requires up to 7x less lookup tables (LUTs) and up to 23x less flip-flops (FFs) than a TCAM-based architecture.
- We provide detailed measurements how the policy structure (in our case, the IPv4 routing table) influences the resource usage of the generated specialized circuitry.

The remainder of this paper is structured as follows: first, we discuss related work in this field of research in Section II. Subsequently, we briefly review the longest prefix match problem in the context of IP forwarding as well as the architecture of TCAMs, which serve as the baseline for our comparative evaluation, in Section III. In Section IV, we introduce the concept of specialized matching circuits and present the architecture of our partially reconfigurable proof-of-concept forwarding engine. Next, we evaluate the efficiency of our tailor-made circuits in Section V. Finally, we conclude this paper in Section VI.

II. Related Work

In the past decades, the research community has proposed a wide variety of algorithmic approaches for network packet classification in general [2], [4], [15]–[17] and longest prefix matching in particular [1], [18]–[20]. These techniques transform the packet processing policy such as a firewall rule set or an IP forwarding table into a data structure which is subsequently used by a corresponding search algorithm to perform lookups. Examples for such data structures are multi-dimensional decision trees [4], [15], hash tables [16], [19], bit vectors [17], or prefix trees [18], [20]. Due to their hardware-agnostic design and the strict separation between search algorithm and configuration memory, most of these approaches can be implemented on a diverse range of underlying hardware platforms, such as general purpose CPUs, GPUs [7], [8], or FPGAs [10]–[12], [21]. However, there is a price to be paid for this flexibility: because the algorithm and the corresponding configuration memory must be general enough to process an arbitrarily shaped policy, they cannot exploit the specific structure of the policy, as illustrated in Section I.

The concept of specialized packet processing circuits for routers and firewalls has been proposed previously in [13] and [14], respectively. In [13], the authors propose to translate routing tables to circuit descriptions based on binary decision diagrams, while in [14] parallel matching circuitry for firewall applications is generated. Although these works mention the potential of logic optimization, they do not answer the following important questions: (1) how much gain in terms of LUT/FF resources can be achieved in comparison to a programmable generic architecture, and (2) which architecture can be used to replace the specialized circuitry at runtime, without the need to shut down the device? In order to answer these questions, we demonstrate the applicability of partial reconfiguration to replace the specialized circuitry in order to perform policy updates at runtime. Furthermore, we will show in our evaluation that policy-tailored forwarding circuitry requires over an order of magnitude less LUT/FF resources than an equivalent TCAM-based circuit on an FPGA.

III. Background

As we demonstrate the potential of specialized packet processing circuitry at the example of an IPv4 forwarding engine, we briefly review the concept of *longest prefix matching* as well as the common TCAM-based implementation technique.

A. Longest Prefix Matching

In order to route an IP packet over a path of multiple hops, each intermediate router has to forward the packet to the next hop until the destination is reached. Therefore, a hop-specific *routing table* provides the outgoing port with respect to the destination address of an incoming packet. In the case of IP routing, the routing table represents the above-mentioned packet processing policy. As it is impractical to separately store the outgoing port for every possible destination address (which would require 2^{32} or 2^{128} entries for IPv4 or IPv6, respectively), addresses are aggregated to *subnets*. A subnet is a set of IP addresses whose most significant k bits are equal. In order to forward an incoming network packet, a router performs a *longest prefix match* in its *forwarding information base (FIB)* (a representation of the routing table that is designed for fast lookups). That is, packets are always forwarded towards the most specific matching subnet in the FIB. As a simple example, consider the routing table for 8-bit addresses shown in Table I. Although a packet with the destination address 10110001 matches on subnets no. 1, 3, and 4, it would be forwarded to subnet no. 1, as it represents the longest matching prefix.

TABLE I: A routing table for 8-bit addresses

Subnet no.	Prefix	Address	Mask	Outgoing port
1	1011000*	10110000	11111110	2
2	01110*	01110000	11111000	1
3	101*	10100000	11100000	3
4	*	00000000	00000000	0

The difficulty of longest prefix matching arises from the fact that routers potentially have to forward millions of packets per second on the basis of steadily growing routing tables [4]. In order to process such huge amounts of data at line rate, dedicated packet processing hardware is required, as described in the following sections. For the remainder of this paper, we will use the terms FIB and routing table interchangeably.

B. TCAMs

Ternary Content-Addressable Memories (TCAMs) are a common way to implement FIBs in high-speed routers [9], [22]. An n-element TCAM consists of n parallel match lines L_i which compare a specified k-bit search word X with the contents of a data register D_i with respect to a mask register M_i. Hence, each match line L_i implements the function

$$L_i = \bigwedge_{j=0}^{k-1} \overline{\left(M_i^j \wedge D_i^j\right) \text{xor} \left(M_i^j \wedge X^j\right)} \,, \qquad (1)$$

which is basically an equality check for the bits selected by the specified mask. In contrast to algorithmic packet forwarding schemes which are based on sophisticated data structures [18], [19], the TCAM approach is rather straightforward: the destination address DA of an incoming packet is matched against every single FIB entry in parallel [6]. This operation yields a multi-match result vector V of size n, whose ith bit is set to 1 iff the ith prefix in the FIB matches DA, otherwise it is set to 0. In order to extract the longest matching prefix from the set of matching prefixes, a priority encoder is used to find the index of the first set bit in V. Of course, this requires the FIB entries to be sorted with respect the prefix lengths. Figure 2 shows the circuit structure of a 4-element TCAM which is configured to perform longest prefix matching according to the routing table in Table I.

The central advantage of TCAM-based packet forwarding is that it requires constant time to process incoming packets, regardless of the structure and size of the routing table. However, TCAMs are significantly more expensive than conventional

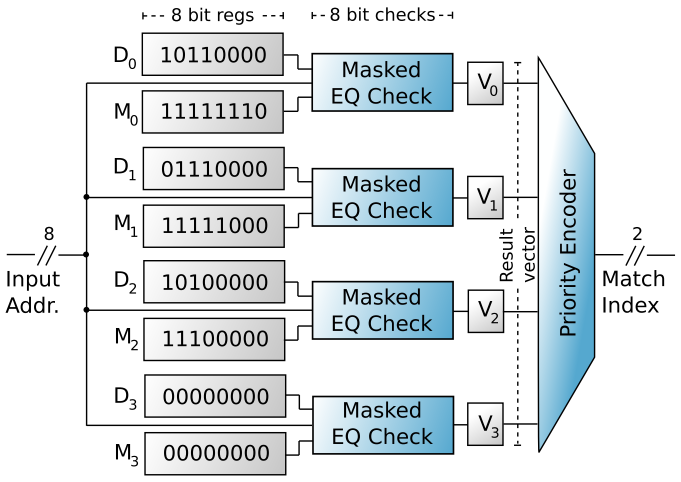


Fig. 2: A 4-element TCAM for 8-bit addresses.

RAMs with the same capacity [6], are less power efficient [22], and can only be operated at relatively low clock rates [9]. Furthermore, the memory resources offered by TCAMs are often underutilized due to the TCAM's regular structure. Figure 2 shows that the four-element TCAM needs to provide two 8-bit registers for every stored prefix in order to support an arbitrarily shaped routing table of at most four prefixes. Thus, even if a prefix is shorter than 8 bit, as it is the case for every prefix in Table I, it still occupies two full registers.

IV. Architecture

In the previous section, we argued that the general structure of a TCAM can lead to underutilized logic resources. Therefore, we propose to specialize the matching circuitry for a particular routing table before it is used to forward packets. That way, we can keep the deterministic matching performance of TCAMs and eliminate the drawback of wasted resources.

A. Specialized Forwarding Circuits

As we target an FPGA as the underlying hardware platform, we can specify an arbitrary Boolean function f which is implemented by the FPGA at runtime (of course, we must obey the restriction of logic resources available on the FPGA during the definition of f). Here, we exploit the fact that a routing table R can be represented as a Boolean function $f_R : \{0,1\}^k \rightarrow \{0,1\}^m$, where k is the number of address bits and m is the number of bits used to encode the output port. Accordingly, we can generate an HDL representation of f_R which implements a full parallel longest prefix match, just as a TCAM does. However, in contrast to a TCAM, f_R specifies only the minimal logic resources which are required to implement the specific routing table R. For example, Listing 1 shows the Verilog code which implements the four parallel matches on the prefixes of Table I. The prefixes are written as literal values into the generated code, which has the important implication that no flip-flops will be inferred for the prefixes during synthesis. Instead, they are stored in lookup tables as part of the function f_R. As a consequence, the synthesis tool is able to perform logic optimization based on the concrete prefixes of the implemented FIB. Hence, the generated specialized circuit tends to be smaller than an equivalent generic TCAM due to the compact representation of the prefixes as part of the logic function and the logic optimization performed during synthesis, as we will demonstrate in our evaluation. Figure 3 illustrates the generated matching circuitry together with the (also generated) priority encoder.

B. Partial Reconfiguration in the NetFPGA 10G Pipeline

In order to evaluate the applicability of partial reconfiguration to replace the specialized matching circuits at runtime, we

```
V_0 <= (addr[7:1] == 'b1011000);
V_1 <= (addr[7:3] == 'b01110);
V_2 <= (addr[7:5] == 'b101);
V_3 <= 1;
```

Listing 1: Verilog code that implements prefix matches.

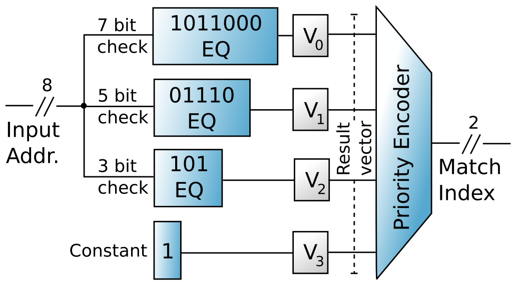


Fig. 3: A tailor-made FIB circuit according to Table I.

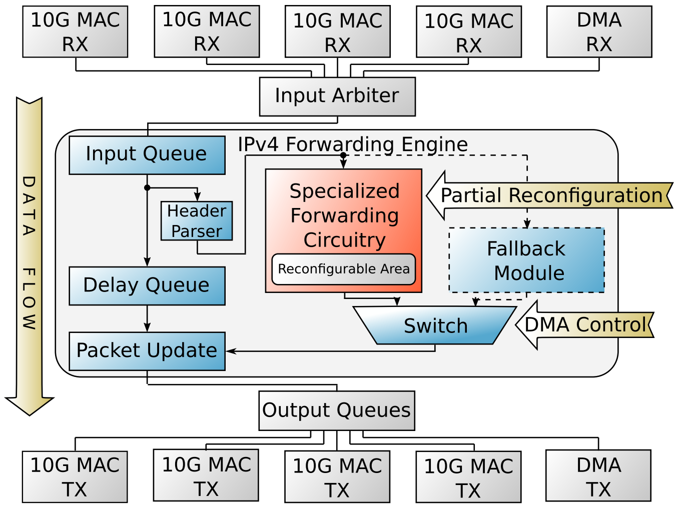


Fig. 4: Overview of the reconfigurable IPv4 forwarding engine.

created a proof-of-concept implementation on top of the NetFPGA 10G board which contains a Xilinx Virtex-5 TX240T FPGA as well as four 10 Gbps Ethernet SFP+ ports. We modified the NetFPGA reference packet processing pipeline, which implements a basic network interface controller [23], and integrated an IPv4 forwarding module into the pipeline, as sketched in Figure 4. The forwarding module consists of a header parser to extract the destination IPv4 address of incoming packets, a set of queues for store-and-forward processing, as well as a reconfigurable area that contains the FIB-specialized matching circuitry and priority encoder. During operation, network packet data enter the forwarding engine over an input queue filled by the input arbiter which merges the incoming data streams into a single stream. Subsequently, each packet in the input queue is buffered in the delay queue while its destination address is classified by the specialized matching circuitry located in the reconfigurable block. The longest prefix match itself is performed in a pipelined fashion and can emit one match result per clock cycle in the same manner as a TCAM. Finally, the buffered packets are updated with respect to the computed destination MAC address of the next hop and written to the output queues for further transmission.

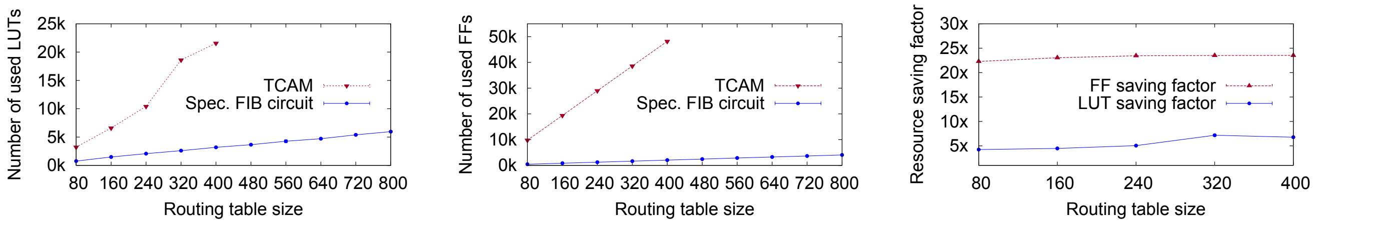
An important detail of this architecture is the *switch* block which controls the communication between the matching circuitry and the packet update block. In order to keep the system in a consistent state during a partial reconfiguration of the matching circuitry (which is in the order of milliseconds), it must be ensured that no invalid data is sent from the reconfigurable area to the packet updater. Our current implementation suspends forwarding operations while the matching circuitry is updated by toggling a DMA-controlled register which prohibits any data flows through the switch. However, the system can be constantly kept operational through a simple change in the switch and the addition of a fallback module, which could be a redirector to a software-based routing module or a small TCAM for an intermediate routing policy, or even another reconfigurable area. Furthermore, the fallback module could be used to handle incremental updates to the routing table, as changes of the implemented policy require the generation of a new partial bitfile for the specialized matching circuitry, which takes about an hour in our current implementation. Thus, a practical implementation could generate a specialized forwarding circuit for the slowly changing parts of a routing table, which may not change for days or even weeks [24]. In fact, according to [24], the majority of routing table updates are caused by few prefixes, which could be handled by the fallback module. The integration of such a fallback module into our reconfigurable architecture is sketched in Figure 4.

Our forwarding engine operates on data words of 256 bit width with a clock frequency of 160 MHz, just as the NetFGPA 10G reference pipeline, and can process incoming packets at full line rate of 40 Gbps. Of course, the proposed architecture size capacity is limited by both the number of LUTs/FFs available on the FPGA as well as the timing constraints imposed by the static and generated parts of the design. However, our evaluation shows that specialized forwarding circuits require less logic resources and result in more timing-friendly designs than a TCAM-based forwarding circuit.

V. Evaluation

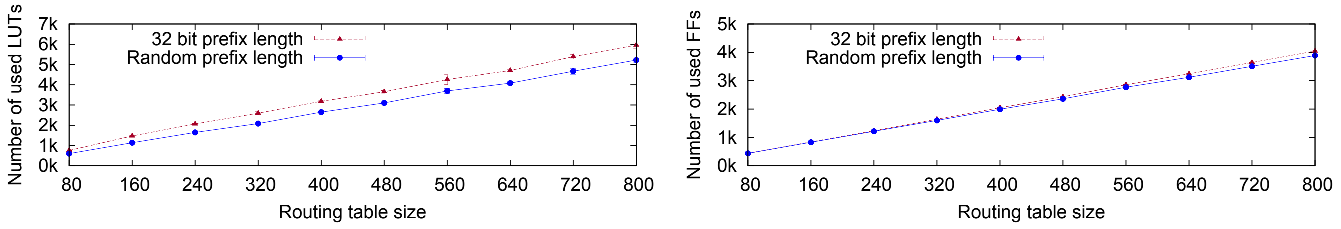
We put our main focus on the evaluation of specialized forwarding circuits on two aspects: first, we compare the size of the generated circuitry in terms of LUTs/FFs with the size of a generic programmable TCAM implementation with an equivalent degree of parallelism, latency, and throughput. Second, we evaluate the potential of logic optimization that takes place during the synthesis of the specialized circuitry with regard to the structure of the specified routing table. Therefore, we generated a large body of routing tables which were translated into Verilog descriptions of specialized forwarding circuits, as described in Section IV-A. Subsequently, partial bitfiles were generated from these circuit descriptions using the Xilinx `ngdbuild`, `map`, `par`, and `bitgen` tools. The number of FFs and LUTs used for the partial designs were extracted from the implementation logfiles after the build process. Also, we generated bitfiles from Verilog TCAM descriptions of different capacities using the same workflow. The NetFPGA 10G board served as the implementation platform for both the specialized forwarding circuits and the generic TCAMs.

In our first experiment, we investigate the scalability of both the specialized forwarding circuitry and the TCAMs with respect to the size of the routing table. To this end, we generated routing tables consisting of random 32 bit prefixes.



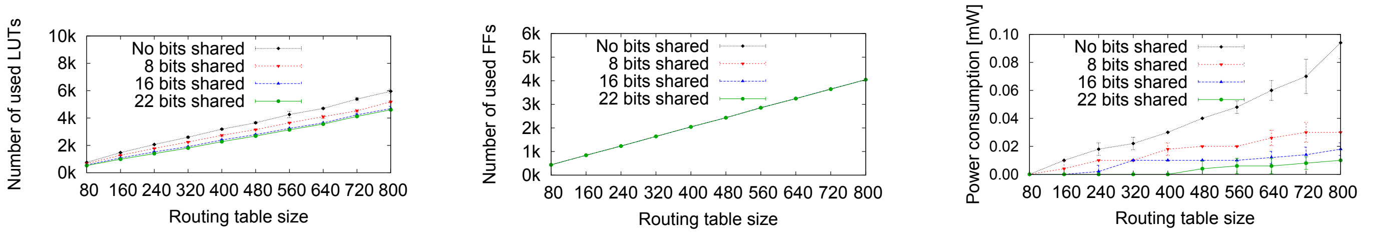
(a) Matching circuit LUT usage. (b) Matching circuit FF usage. (c) Resource savings with specialized circuits.

Fig. 5: Scalability of matching circuits with respect to routing table size/capacity.



(a) Matching circuit LUT usage. (b) Matching circuit FF usage.

Fig. 6: LUT/FF resource consumption of specialized FIB circuits: 32 bit prefixes vs. random prefix lengths.



(a) Matching circuit LUT usage. (b) Matching circuit FF usage. (c) Matching circuit power consumption.

Fig. 7: LUT/FF/Power consumption of specialized FIB circuits with shared logic functionality.

We started with 80 entries and increased the routing table size in steps of 80 to a maximum of 800 entries. This process was repeated five times with different random seeds. Also, we generated descriptions for TCAMs with the respective capacities up to 400 entries. Higher capacities for TCAMs were impossible due to space restrictions on the FPGA. Figures 5a and 5b show the average LUT and FF consumption as well as the standard deviation. It can be seen that the usage of LUTs and FFs for the specialized circuitry increases linearly with the routing table size, as larger routing tables lead to larger logic functions. Accordingly, more LUTs are required to implement these functions. Also, the growing size of the generated priority encoder requires more FFs due to its pipelined operation. However, it can also be observed that the TCAM circuits consume more LUTs and FFs, up to a factor of 7 for the LUTs and 23 for the FFs, as revealed by Figure 5c. The reasons are twofold: first, the TCAM assigns two registers per routing table entry and thus requires more FFs. Second, the TCAM's matching logic is generally larger than that of the specialized FIB circuits and thus requires more LUTs.

Next, we evaluate the impact of different prefix lengths on the resource consumption of the specialized FIB circuitry. As mentioned in Section III-B, the TCAM resource consumption is agnostic to the prefix lengths of the currently installed FIB, as every entry always requires the same number of registers. However, this is not the case for the specialized FIB circuits because a k-bit prefix is implemented by a k-bit logic function, which should lead to resource savings if $k < 32$. In fact, real routing tables contain many different prefix lengths [25], so we generated routing tables of the same sizes as above, but with uniformly random prefix lengths between 1 and 32. Again, we extracted the LUT/FF consumption of the partial bitfiles for the corresponding FIB circuits and repeated this experiment five times. Figures 6a and 6b show the average LUT/FF requirements as well as the standard deviation in comparison to the LUT/FF consumption of the specialized circuitry for the full 32 bit routing tables from the first experiment. Figure 6a confirms that the circuitry with random prefix lengths need a significantly smaller number of LUTs than the circuits for the 32 bit prefixes. In contrast, Figure 6b reveals that the number of used FFs does not change significantly. The reason for this behaviour is that the matching logic is implemented by LUTs and not by FFs, which are only needed for the priority encoder.

In our final experiment, we examine the effect of logic optimization, which takes place in the different stages of bitfile generation, on the generated circuitry. Because we translate a specified routing table into a corresponding logic function, the synthesis tools can take advantage of bitwise redundancies of the prefixes in the table. In order to investigate the impact of this optimization, we generated four routing tables consisting of 32 bit prefixes, but the prefixes in each table differ only in the least significant 10, 16, 24, and 32 bits, respectively. This means that the LUT implementation of the matching logic for the most significant 22, 16, 8, and 0 bits can be shared between the different prefix checks, respectively. Again, we repeated this experiment five times and show the averaged results and the standard deviations in Figures 7a to 7c. It can be seen in Figure 7a that the amount of used LUTs indeed decreases with an increasing number of shared prefix check bits, as a result of the logic optimization. However, as the logic optimization does not reduce the total number of prefix checks, the priority encoder which resolves the longest prefix match has the same size for each of the four routing table classes. Accordingly, the number of used FFs does not change in this experiment, as shown in Figure 7b.

Finally, we used the Xilinx XPower Analyzer in order to estimate the power consumption of the generated circuitry. Figure 7c reveals that the matching circuit's power consumption decreases with an increase in the shared prefix bits, since more shared prefix bits lead to smaller matching circuits. In comparison, even the smallest TCAM that we synthesized (80 entries) already consumed $0.38\,\mathrm{mW}$, which is about four times higher than the largest specialized circuits (800 prefixes) with no shared prefix bits. The power measurements show that the matching circuits (both specialized and TCAM) make up only a small fraction of the entire design's power consumption (ca. $5.1\,\mathrm{W}$) and therefore should be taken with a grain of salt. However, they still indicate that specialized matching circuitry is more power-efficient than their generic TCAM counterparts.

VI. Conclusion

In this work, we investigated the optimization potential of specialized packet processing circuitry for FPGAs at the example of tailor-made IPv4 forwarding circuits. In contrast to state-of-the-art TCAM matchers whose functionality is controlled via runtime-programmable configuration memories, specialized forwarding circuitry has a much smaller footprint in terms of used FPGA resources. Our evaluation results reveal that specialized forwarding circuits require up to 7x less lookup tables and up to 23x less flip-flops than an equivalent generic matching architecture with equivalent degrees of matching performance, latency, and parallelism. As a consequence, the FPGA can implement larger forwarding tables in specialized circuits, which still provide TCAM-equivalent matching performance. This technique can also be applied to other policy-based packet processing applications which often rely on TCAMs, such as firewalling [14]. In order to enable policy updates without the need to shutdown the device, we devised an architecture which is amenable for partial reconfiguration of the used matching circuitry. That way, we can combine the advantages of efficient specialized matching circuits with the flexibility needed for practical applications.

Acknowledgments

This work was funded by the BMWi (German Federal Ministry of Economics and Energy) in the context of the hardFIRE research project. We would like to thank Andreas Fießler, Wladislaw Gusew, Samuel Brack, and Frank Winkler for valuable discussions and feedback on this work.

References

[1] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *IEEE Network: The Magazine of Global Internetworking*, vol. 15, no. 2, Mar. 2001.

[2] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network: The Magazine of Global Internetworking*, vol. 15, no. 2, Mar. 2001.

[3] W. Jiang and V. Prasanna, "Data structure optimization for power-efficient IP lookup architectures," *IEEE Transactions on Computers*, vol. 62, no. 11, Nov. 2013.

[4] B. Vamanan, G. Voskuilen, and T. Vijaykumar, "Efficuts: Optimizing packet classification for memory and throughput," in *SIGCOMM '10*, Aug. 2010.

[5] P. Bosshart *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *SIGCOMM '13*, Aug. 2013.

[6] A. McAuley and P. Francis, "Fast routing table lookup using CAMs," in *INFOCOM '93*, Mar. 1993.

[7] M. Varvello, R. Laufer, F. Zhang, and T. Lakshman, "Multi-layer packet classification with graphics processing units," in *CoNEXT '14*, Dec. 2014.

[8] S. Han, K. Jang, K. Park, and S. Moon, "Packetshader: A GPU-accelerated software router," in *SIGCOMM '10*, Aug. 2010.

[9] H. Le, W. Jiang, and V. Prasanna, "Scalable high-throughput SRAM-based architecture for IP-lookup using FPGA," in *FPL '08*, Sep. 2008.

[10] D. Taylor, J. Lockwood, T. Sproull, J. Turner, and D. Parlour, "Scalable IP lookup for programmable routers," in *INFOCOM '02*, Jun. 2002.

[11] W. Jiang and V. Prasanna, "Scalable packet classification on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 9, Sept 2012.

[12] Y. Qu, S. Zhou, and V. Prasanna, "High-performance architecture for dynamically updatable packet classification on FPGA," in *ANCS '13*, Oct. 2013.

[13] R. Sangireddy and A. Somani, "High-speed IP routing with binary decision diagrams based hardware address lookup engine," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 4, May 2003.

[14] S. Hager, F. Winkler, B. Scheuermann, and K. Reinhardt, "MPFC: Massively parallel firewall circuits," in *LCN '14*, Sep. 2014.

[15] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," in *HOTI '99*, Aug. 1999, pp. 34–41.

[16] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," in *SIGCOMM '99*, Aug. 1999.

[17] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," in *SIGCOMM '98*, Aug. 1998, pp. 203–214.

[18] W. Doeringer, G. Karjoth, and M. Nassehi, "Routing on longest-matching prefixes," *IEEE/ACM Transactions on Networking*, vol. 4, no. 1, Feb. 1996.

[19] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," in *SIGCOMM '97*, Sep. 1997.

[20] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small forwarding tables for fast routing lookups," in *SIGCOMM '97*, Sep. 1997.

[21] T. Ganegedara and V. Prasanna, "StrideBV: Single chip 400g+ packet classification," in *HPSR'12*, Jun. 2012.

[22] D. Qunfeng, S. Banerjee, J. Wang, and D. Agrawal, "Wire speed packet classification without TCAMs: A few more registers (and a bit of logic) are enough," in *SIGMETRICS '07*, Jun. 2007.

[23] J. Naous, G. Gibb, S. Bolouki, and N. McKeown, "NetFPGA: Reusable router architecture for experimental research," in *PRESTO '08*, Aug. 2008.

[24] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, "BGP routing stability of popular destinations," in *Proceedings of the 2Nd ACM SIGCOMM Workshop on Internet Measurment*, Nov. 2002.

[25] S. Dharmapurikar, P. Krishnamurthy, and D. Taylor, "Longest prefix matching using Bloom filters," in *SIGCOMM '03*, Aug. 2003.