Investigating the Feasibility of FPGA-based Network Switches

Jiuxi Meng*, Nadeen Gebara*, Ho-Cheung Ng*, Paolo Costa†, and Wayne Luk*
*Imperial College London
Email: {jiuxi.meng16, n.gebara17, h.ng16, w.luk}@imperial.ac.uk
†Microsoft Research
Email: paolo.costa@microsoft.com

***Abstract*—FPGAs are being increasingly used on network interface cards (NICs) as offload units to accelerate packet processing tasks. The rationale is that by customizing the NIC logic it is possible to achieve higher performance for the most critical tasks while eliminating unnecessary logic, thus improving overall efficiency.**

In this paper, we aim to investigate if similar benefits can also be extended to network switches. We compare different switch architectures and analyze their suitability to an FPGA implementation. We discuss several optimization techniques to overcome the challenges of limited FPGA resources and assess the scalability of our designs up to 10, 25, and 50 Gb/s throughput per port.

I. Introduction

FPGAs are often used in today's data centers to support compute-intensive workloads such as data analytics and machine learning as well as to accelerate network processing functions at the end hosts. For example, Microsoft has shown that using FPGAs as a bump-in-the-wire between the CPU and the NIC can accelerate Bing search, machine learning inference tasks, and packet processing [1]–[3]. In this paper, we explore whether FPGAs can be used to implement and customize network switches.

While modern network switches support some limited form of programmability, e.g. [4], most of their operations are implemented as fixed logic. This poses two main challenges. First, data center operators can not arbitrarily modify this logic and implement custom network functions. For example, if a new packet scheduling policy is to be implemented, the only option is to engage with the switch manufacturer (if at all possible) and wait for the next chip tape out, which typically requires 18 months or longer. Second, today's cloud providers only use a fraction of all functionality offered by network switches. This leads to a waste of silicon area as well and potentially higher costs and power consumption than necessary. An FPGA-based switch in contrast provides more efficient solutions as they allow the flexibility of only implementing the switch functions needed (including custom ones).

However, it remains unclear whether an FPGA-based switch can match the performance of commodity switches used today. If we just consider the total transceiver bandwidth provided by FPGAs today, the gap between FPGAs and commodity switches is shrinking. The latest generation Intel Stratix 10 TX FPGA has an aggregate transceiver bandwidth of 8Tb/s [5], which is almost two thirds of the bandwidth provided by top-end data center switches (12.8 Tb/s). This, however, only reflects the *IO* capability and does not indicate whether the FPGA has sufficient memory and compute capabilities to switch packets at such rates.

This paper makes a first step towards determining this by focusing on the core functionality required by any switch, i.e., the ability of forwarding packets from its input ports to its output ports. Such functionality can be provided by either a crossbar-based fabric along with some buffering and an appropriate scheduling algorithm, or a memory based fabric that operates at a rate that is proportional to the number of switch ports (see Section II). Since the memory speed is limited by the AC and DC characteristics of the FPGA, our work is limited to crossbar-based switch designs. In this paper, we explore the trade-offs provided by different architectures for implementing the building blocks (buffers and scheduling logic) of a crossbar-based switch, and reveal the challenges in scaling such architectures on FPGAs. We make the following contributions:

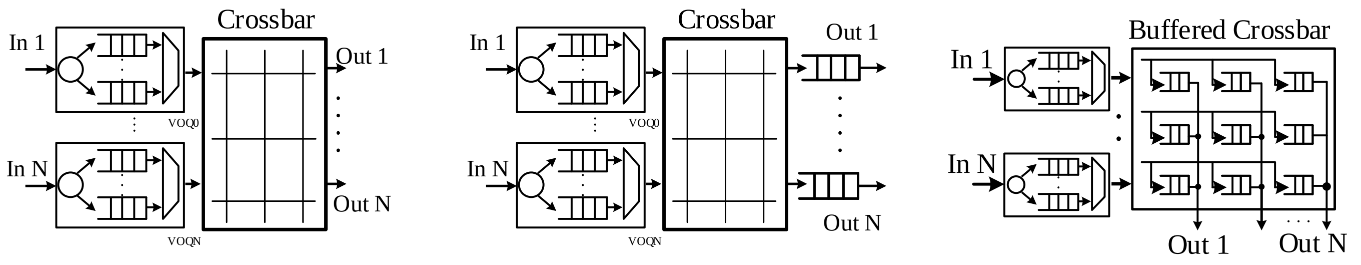
1) The development of an efficient buffer-sharing technique that minimizes memory resource usage on FPGAs for input buffered crossbar switch (Section III).
2) A feasibility study that shows the performance and resource utilization trade-offs of considered switch architectures (Section IV).
3) A technology-independent model to predict the performance and resource requirements of the implemented architectures at various scales (Section V).

II. Background and Related Work

In this section, we provide the reader with the necessary context of different switching architecture and provide a summary of related literature work.

A. Network Switches on FPGAs

Various techniques for implementing high-performance, crossbar-based switches have been proposed in available literature. These techniques differ both in terms of the location of the buffers and the scheduling algorithms used to eliminate resource conflicts that can arise in a switch. Fig. 1 shows three



(a) Input Buffering **(b)** Combined Input and Output Buffering **(c)** Combined Input and Cross-point Buffering

Fig. 1: Buffering Techniques for Crossbar-based Switches

prominent buffering approaches. The strengths and weaknesses of each approach are described below.

(a) Input Buffering (IB): A limitation of placing one queue at each input port of a switch is Head of Line (HOL) blocking, which can significantly reduce the switch's maximum throughput [6]. This occurs when multiple input queues have packets destined to the same output. Those packets prevent subsequent packets destined to other output ports from traversing the switch, which ultimately leads to poor throughput.

To avoid HOL blocking, Virtual Output Queues (VOQs) are used in input buffered switches. The goal of VOQs is to have a queue for each output port at each input port such that packets no longer block each other when destined to different outputs as shown in Fig. 1a. Though VOQs eliminate HOL blocking, they introduce the need for simultaneous input and output conflict resolution. This means that the scheduler must not only determine input and output port matching, but also which VOQ to select from each input port. Therefore, for an N port switch, the scheduler must efficiently solve N^2 requests from VOQs, which greatly complicates the scheduling logic. This makes the choice of a scheduling algorithm that can achieve high throughput while still being simple enough to be implemented on an FPGA particularly challenging. Based on the outcome of our analysis of the complexity of implementing schedulers for input-buffered architectures on FPGAs [7], the scheduling algorithm iSLIP [8] is determined to be most suitable for an FPGA implementation. At a high level, iSLIP works as follows. At the beginning of each round, the input will send requests to the output grant arbiter based on the existence of packet in the corresponding VOQs. Then the output grant arbiter will take the request vector (vector consists of all the requests from the input) and grant one request with a round-robin fashion starting from the port next to the previously granted port. Finally, when the input accept arbiter receives the granted request from the grant arbiter, it will accept the requests using the same round-robin method as the grant arbiter.

Our simulation results indicate that iSLIP provides the best performance in terms of latency and throughput under uniform traffic patterns while achieving a good performance under non-uniform traffic patterns when compared with other simple scheduling techniques for input buffered switches. Therefore, we consider an input-buffered architecture with the scheduling algorithm iSLIP in this work and leave the exploration of more complex scheduling algorithms for future work.

(b) Combined Input and Output Buffering (CIOB): Placing buffers at the output provides the best ideal switch performance and eliminates the need for scheduling by having the buffers operating N times faster than the line-rate in an N-port switch. However, since the line-rate increases much faster than the memory bandwidth, output buffered switches scale poorly, and this is supported by available commercial products [9]. Combining input buffering and output buffering as shown in Fig. 1b allows the emulation of an output buffered switch using a broad class of packet scheduling algorithms while only requiring the scheduler, crossbar fabric, and output buffers to operate with a speed up of $2 - (1/N)$ as opposed to N [10]. However, operating the scheduling algorithm with such a speedup is still challenging on FPGAs especially when considering scaling to higher port counts (radix) and line rates. Therefore, we do not consider these architectures in our work.

(c) Combined Input and Cross-point Buffering (CICB): Buffers are located both at the inputs of the switch and are also made available for each input-output pair, as shown in Fig. 1c. This decouples the task of scheduling and allows the use of simpler scheduling algorithms at the switch's inputs and outputs, achieving a good performance in terms of both throughput and latency. However, the number of cross-point buffers scales quadratically with the number of ports and that makes cross-point buffering less attractive. To address this shortcoming, Hierarchical Crossbar switch with CIOB was proposed by Kim [11] in order to retain the advantages of the CICB while making it more FPGA-friendly as we discuss next.

(d) Hierarchical Crossbar switch + CIOB (HC+CIOB): Kim et al. [11] proposes the use of a Hierarchical Crossbar architecture with CIOB to reduce the memory usage of cross-point buffered architectures as shown in Fig. 2. The author modified the CIOB architecture by replacing a $N \times N$ buffered crossbar with $(N/p)^2$ smaller $p \times p$ crossbars with CIOB. This modification reduces the memory usage by 40% compared to a fully buffered crossbar. The idea of using HC switch is particularly beneficial for FPGAs: dividing the large radix (number of ports) switch into smaller ones allows for simpler scheduling algorithms and simplifies the overall design. Dai and Zhu built on this idea and proposed a Grouped Crosspoint Queued (GCQ) switch targeting FPGAs [12]. The idea of the GCQ switch is to replace the CIOB sub-switches with small-

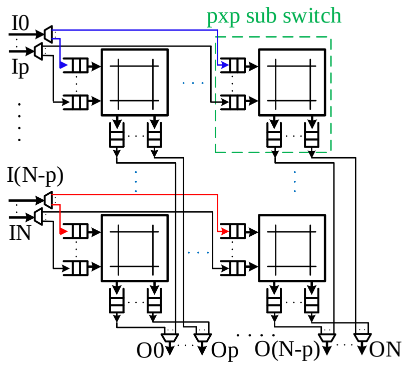


Fig. 2: Hierarchical crossbar switch with total radix of N and all sub-switches of radix p

radix memory based switches. Since BRAMs on the FPGA can run much faster than the fabric, the author chooses to speed up the block RAM S times faster than the fabric to emulate an S $\times$ S switch. We adopt a similar architecture in our analysis: we use a round-robin scheduler for conflict resolution and investigate its hardware resource usage for higher radix switches.

B. Related Work

Based on the above discussion, we found that switch architectures that make use of buffering approach **(a)** with the iSLIP scheduling algorithms and the buffering approach **(d)** that leverages GCQ with a simpler scheduling algorithm are most amenable for FPGA implementation due to their scalability and switching performance. Therefore, this paper focuses on their implementations and the corresponding investigation to demonstrate the feasibility of these two approaches. In particular, we study these approaches from a single-chip perspective and provide a comprehensive overview of their advantages and drawbacks.

To the best of our knowledge, this is the first work that provides a direct comparison of these two architectures in the context of FPGAs. Yoshigoe et al. [13] implement a 24 port 10-Gbps switch with CIOB structure. However, multiple FPGAs are used due to the limitation of the past FPGA technology resulting in a high latency due to packet transmission across FPGAs. NetFPGA [14] is an open source hardware and software platform for network developers. The latest NetFPGA-SUME board provides four 10Gb/s ports and is capable of achieving 100Gb/s aggregate bandwidth by further utilizing available FMC ports. It is further provided with a 4-port 10 Gb/s reference switch design that uses a CIOB fabric with a round-robin scheduler for conflict resolution. This results in limited scalability because the round-robin scheduler requires an internal data path that runs four times faster than the input interface to support all 4 ports. More recently, there has been a few proposals to use hardened NoCs [15], [16] for FPGA switch implementations. These papers, however, only demonstrate the possibility of leveraging NoCs for implementing low latency and area efficient network switches on the FPGAs through simulation since FPGAs with hardened NoCs are only recently made available [17]. The relatively small

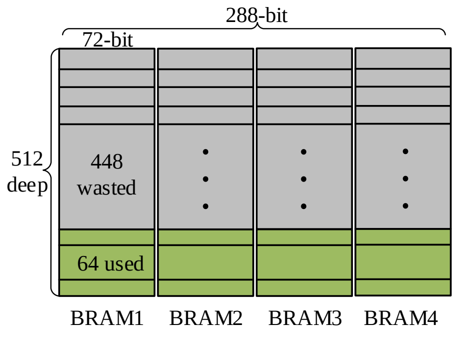


Fig. 3: BRAM implemented single queue for data width of 256 bit, larger data width results in higher BRAM usage as well as waste; A buffer depth of 64 provides zero packet loss as explained in section IV

number of master/slave NoC ports provided (28), however, is likely to limit their usage to medium/low radix switches.

III. Architecture

In this work, we restrict our attention to architectures that can fit within a single FPGA chip for the following reasons. First, multi-chip implementations introduce a significant latency overhead due to the long round-trip time (RTT). According to Shen, the measured average latency between FPGAs using the Interlaken IP was approximately 170 ns [18] which is roughly the same as the port to port latency of our entire single-chip implementation. Furthermore, multi-FPGA implementations are far more complicated than a simple, single-chip design.

1) Fabric: A crossbar based switch consists primarily of buffers, a scheduler, and the crossbar fabric itself. Crossbar-based switches are non-blocking since any input port can be connected to any output port. In addition, the behaviour of the crossbar can be easily emulated with a group of multiplexers making them an attractive fabric choice for FPGA based switches.

2) Buffering: Since low memory access latency is crucial to any switch design, fast on-chip memory resources must be used for buffering instead of the off-chip memory. Naively using BRAM FIFOs on FPGA for VOQs implementation, however, would limit the design's scalability and would make use of the memory resource available on-chip inefficiently.

Example: The Xilinx UltraScale+ XCVU9P consists of 2160 on-chip BRAMs. Assume that a line-rate of 10Gb/s can be supported with a data width of 256 bits and a queue depth of 64 operating at a frequency of 40MHz. Since the 36 Kb BRAMs on the FPGA can be configured to have a maximum width of 72 bits, a total of four 36 Kb BRAMs would be required to write 256 bits in parallel. Since the number of VOQs required for a NxN switch is N^2, the following constraint must be satisfied: $4 \times N^2 \leq 2160$. This in turn limits the maximum number of ports that can be supported to 23. Additionally, most of the BRAM resources are also wasted as shown in Fig. 3. Utilizing the distributed RAMs instead of the BRAMs relaxes the constraint to $256 \times 64 \times N^2 \leq 36.1$ Mb [19]. Nonetheless, the maximum port number is still limited to

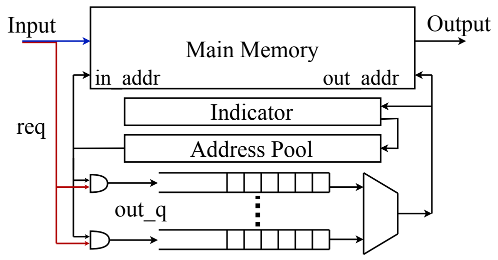


Fig. 4: Shared Memory Architecture

46. While UltraRAMs present larger storage capacities, they are as limited as BRAMs in flexibility and utilizing them for implementing VOQs would similarly be inefficient.

To efficiently use the on-chip FPGA memory resources, the shared memory architecture used by the IBM Prizma switch [20] is adopted to replace large VOQs. This architecture enables flexibly sharing a single BRAM/UltraRAM based memory across VOQs belonging to the same input port. Another advantage of this architecture is its high tolerance to bursty traffic, as the busy queues will be allocated with more buffer space in the shared memory space. The shared memory structure has similarly been used in the work of Dai and Zhu [12] when implementing a memory-based switch, and is shown in Fig. 4.

The proposed shared memory architecture is implemented using a dual-port RAM that supports concurrent read and write operations. The address pool is implemented as a simple FIFO populated with available memory addresses. When a packet arrives at a shared memory block, the destination field is fed to the output pointer queues whereas the data get stored in the shared memory with the address provided indicated by the address pool. The indicator block shown in Fig. 4 is designed to handle multi-cast packets that are destined to multiple output ports. Instead of storing the same data multiple times in the shared memory, only one copy of the multi-cast data is kept. This minimizes storage overhead since the number of bits required to represent the request is much smaller than the data itself. The indicator stores a bit-map indicating the output ports requested by a packet when it is first written into memory. The bit-map entry representing a certain output is cleared when the header is popped out of the corresponding output port, and the memory address is freed and returned to the address pool only after the bit-map value is 0, which indicates that all the multi-cast requests have been handled.

Although utilizing an asymmetric dual port memory for implementing the indicator might appear optimal as it allows the modification of individual bits within the bit-map with a single clock cycle delay, such an approach would limit the scalability of the shared memory. This is because the read and write width ratio of the dual port memory provided by Xilinx [21] memory is limited to 32. This in turn would restrict the maximum number of queues that can be shared. To enhance design scalability, we choose to implement the indicator block using a symmetric dual port memory at the expense of a cycle overhead required for updating the entire bit-map on each access. In other words, the symmetric memory now requires the entire old bit-map content to be read before writing back the new value thereby introducing an additional read delay.

TABLE I: Utilized custom packet format

Field	Source	Data	Destination
Width	$\log_2 N$	$W - \log_2 N - N$	N

3) Schedulers: Schedulers are among the core components of crossbar-based switches as they are needed to resolve potential port conflicts that could arise in a switch. We consider two schedulers in our work: **1)** a centralized iSLIP scheduler that implements the iterative maximal size matching scheduling algorithm proposed in [22] and guarantees 100% throughput for uniform Bernoulli traffic. We chose the iSLIP algorithm after simulating a group of input buffered algorithms including [23] [24] [25]. Our simulation results show that under uniform traffic, iSLIP has the highest throughput and the lowest average packet latency while allowing for simple hardware implementation. **2)** a round-robin (RR) scheduler that can provide additional implementation simplicity over iSLIP in architectures where complex centralized scheduling is not required.

IV. Feasibility Study

To investigate the scalability of FPGA based switches, two architectures are considered. The first switch architecture, ***SMiSLIP***, uses shared memory input-buffering with a centralized iSLIP scheduler and is chosen to represent the broad class of input-buffered switches with centralized schedulers. The second design, ***GCQ***, is the switch architecture proposed by Dai and Zhu in [12], and is chosen to be representative of CICB and HC+CIOB architectures since the CICB architecture is an extreme case of the HC switch with S=1 as explained in Section II-A. We also implemented a simple crossbar switch architecture with an iSLIP scheduler and with no buffer sharing to serve as a baseline. We specifically focus on simple scheduling algorithms since they are more likely to scale to higher line rates and number of ports while achieving timing closure on FPGAs.

In the rest of this section, we first introduce our design setup and then discuss the results of our study. Specifically, we present the trade-offs of the various architectures in terms of their maximum achievable line rate and number of ports (aggregate bandwidth), port-port latency, and FPGA resource utilization based on the simulation results obtained.

A. Design setup

1) Design Flow & Tools: Vivado Design Suite 2017.2 is used for hardware implementation, and our results are obtained targeting Virtex Ultrascale+ XCVU9P FPGA with `out_of_context` flag to prevent IO insertion. Our results do not include any transceiver logic and are limited to the switching architectures themselves.

2) Packet Format: All implementations assume uniform traffic and a fixed packet size representing a single transfer unit, namely the cell. Though this approach increases the

TABLE II: Design specification

Platform	Operating frequency	Line rate	Data width
Virtex UltraScale+ xcvu9p	40MHz	10Gb/s	256
	40MHz	25Gb/s	640
	40MHZ	50Gb/s	1280

TABLE III: Buffer depth configurations

	iSLIP	SMiSLIP	GCQ
VOQ depth	64	n/a	16
Shared memory depth	n/a	64N	64N
Memory pool depth	n/a	64N	64N
Output address pointer queue depth	n/a	64N	64N

control resource overhead, it allows for a simpler flow control logic. We simply use buffer occupancy flags (full or empty) as a notifier to the sender. A customized packet format summarized in Table I is chosen since our primary goal is investigating the basic switching functionality. The radix of the switch is represented by N, and W is the total packet width. Each bit in the destination field represents the request to the corresponding output port. $Log_2(N)$ bits are used to represent the input port, and are only included to provide a more realistic packet format even though they are not necessarily required for correct switch operation.

3) Design specification: We consider switches with the three line rates shown in Table II. The reason why the operating frequency is fixed at 40MHz is twofold: First, 40MHz is the boundary operating frequency for the iSLIP scheduler to support 128 input based on our experimental results. Second, the GCQ switch requires four times speedup in the design, using 40MHz not simultaneously eases achieving timing and provides a fair comparison with the previous work.

4) Buffer Depth: Through simulation with Omnet++ [26] we found that for the iSLIP based switch, a VOQ buffering depth of 40 is required for no packet loss. To ease the implementation process, we use a buffer depth of 64. The GCQ switch requires a smaller VOQ depth for no packet loss and a VOQ depth of 16 is sufficient as presented in [12]. The utilized buffer depths for the various switching architectures considered are further summarized in Table III.

B. Performance analysis

1) Maximum Achievable Line rate: Table IV summarizes the obtained results targeting line rates of 10 Gb/s, 25 Gb/s, and 50 Gb/s respectively. It also demonstrates the extent to which each design can be scaled in terms of both line rate and number of ports. This is indicative of the maximum achievable aggregate bandwidth of each switch architecture which is simply the product of line rate and port count.

The maximum frequency and link capacity (maximum running frequency $\times$ data width) that could be supported for each data width and port count is determined and provided in Table V. Note that GCQ consists of two clock domains, with one being 4 times faster than the other. The slower clock frequency is displayed in Table V because this clock determines the link speed of the switch. As can be seen from

TABLE IV: Implementation results summary

	iSLIP			SMiSLIP			GCQ		
N	10G	25G	50G	10G	25G	50G	10G	25G	50G
8	✓	✓	✓	✓	✓	✓	✓	✓	✓
16	✓	✓	T	✓	✓	✓	✓	✓	✓
32	✓	X	X	✓	✓	✓	✓	✓	✓
64	X	X	X	✓	C	X	✓	✓✓	X
96	X	X	X	C	X	X	X	X	X
128	X	X	X	X	X	X	X	X	X

✓: Successful implementation
✓✓: Best aggregate throughput
C: Failed to resolve global congestion
T: Failed to resolve timing closure
X: Implementation failed due to resource shortage

Fig. 6a, for switches with 32 or fewer ports, SMiSLIP can support a higher link capacity than GCQ. This is because no speed up is required for SMiSLIP so that the entire switch can run at the highest frequency. However, due to the centralized architecture of SMiSLIP, the maximum supported frequency decreases more rapidly than that of the GCQ switch. This is because the iSLIP scheduler contains three stages in one scheduling iteration and the input size of the scheduler is proportional to the port number squared. Larger input size leads to higher combinational logic delay and eventually reduces the maximum running frequency. While inserting registers could help increase the operating frequency, this would also introduce cycle overhead to the scheduler and as an iterative scheduler, this overhead would be multiplied by the number of iterations, which makes its overall efficiency less clear. We leave a more thorough exploration of this possible optimization to future work. GCQ on the other hand has a stable performance due to its hierarchical structure which makes it more suitable for higher radix switch designs.

2) Average port-port latency: Table VI reports the port-port latency obtained for each implementable configuration. The latency of all the designs increases with both the port number and the data width. Fig. 6b contrasts the average port-port latency of the SMiSLIP and GCQ switch designs for different port speeds. To provide a fair comparison against commodity switches, the latency of the transceivers are further accounted for in the result. According to [27], the latency of the transceivers ranges from 13.65ns to 96.75ns for 10 Gb/s line rate resulting in an average latency of 55.2ns. The shaded area represents the latency boundary of commodity switches [28]–[30]. Overall, both design performs well in latency compared with the commodity products, and SMiSLIP has much lower latency compared with GCQ. This is because GCQ switches require clock domain crossing at both input and output interfaces as well as the use of pipeline registers to break the long wires that inter-connect the memory based switches.

C. FPGA Resource Utilization

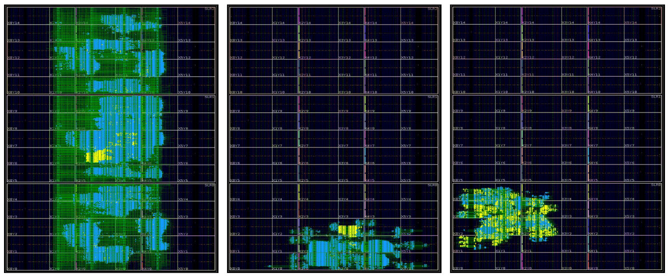
Three floorplans of a 16 port switch are shown in Fig. 5 to demonstrate the scaling of each architecture in terms of

TABLE V: Frequency and line rate scaling with switch size and data width

	Maximum Frequency (MHz)						Link Capacity (Gbps)	
Data Width	256 bit		640 bit		1280 bit			
# Ports	SMiSLIP	GCQ	SMiSLIP	GCQ	SMiSLIP	GCQ	SMiSLIP	GCQ
8	228.68	55.49	206.31	54.14	206.65	48.31	264.52	61.83
16	201.49	54.29	192.53	52.69	183.79	43.67	235.25	55.88
32	135.65	51.28	109.89	50.64	56.727	46.42	70.33	59.42
64	52.46	49.29	-	48.56	-	-	13.43	31.08

TABLE VI: Port-port latency scaling with switch size and data width

	Port-port Latency (ns)						Average port-port latency (ns)	
Data Width	256 bit		640 bit		1280 bit			
# Ports	SMiSLIP	GCQ	SMiSLIP	GCQ	SMiSLIP	GCQ	SMiSLIP	GCQ
8	26.238	172.52	32.082	173.09	32.034	175.875	31.118	173.83
16	33.778	182.23	35.164	183.213	36.646	186.22	35.196	183.89
32	49.232	203.627	59.6	204.307	141.02	209.23	54.416	203.97
64	171.53	246.36	-	247.81	-	-		247.09



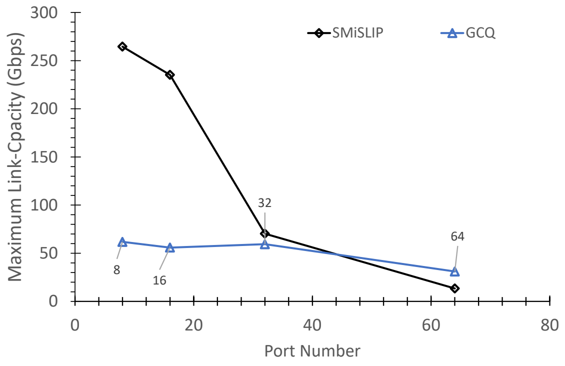
(a) iSLIP 10G 16-port **(b)** SMiSLIP 10G 16-port **(c)** GCQ 10G 16-port

Fig. 5: Implemented device view of three architectures. The leaf cells highlighted in blue represent the memory resources used by buffers. The yellow cells represent the scheduler in iSLIP design and memory based switch in GCQ design.

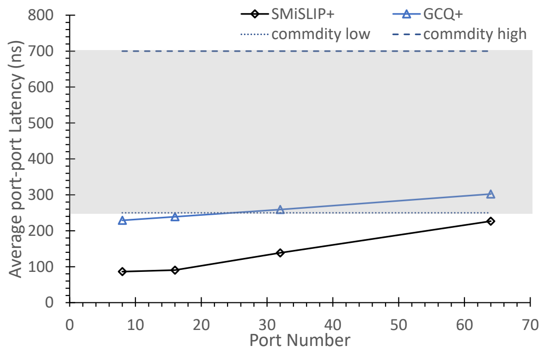
resource utilization. As expected, the basic crossbar switch with no buffer sharing has a high memory utilization and is limited in scalability. The resource usage versus port number for various port line rates are presented in Fig. 7a and Fig. 7b for the SMiSLIP switch and the GCQ switch respectively. As can be seen, GCQ requires a higher number of logic resources compared with the SMiSLIP switch (FFs and LUTs). However, comparing the utilization of the other resources is not as straightforward. Since the compiler views the rest of available resources as a collective pool of memory resources, it attempts to make optimal use of the pool of resources while compensating the lack of one resource with another based on the total resources available on the target device. This, in turn, obfuscates the relationship between the scale and the resource requirements of the design, as it is now subject to target dependent optimization constraints. We therefore propose a target independent model in Section V.

V. Analytical Model

We present a target independent model that can be used to estimate the resource requirement of our proposed architectures at various scales for future adoption on next-generation FPGAs. This enables us to estimate the resource requirements of each architecture and predict whether they can be implemented on a target FPGA while relying on



(a) Maximum link capacity vs. port number



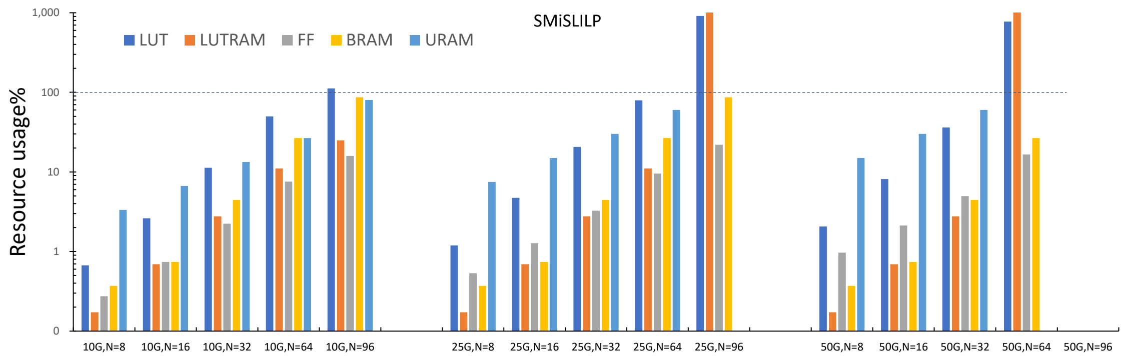
(b) Average port-port latency vs. port number, "+" means with transceiver latency added

Fig. 6: Switch Performance Scaling Results

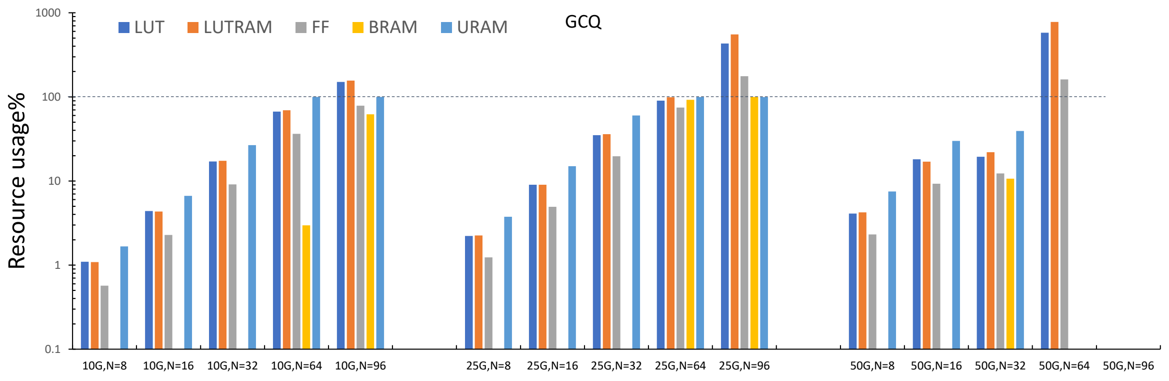
the FPGA datasheet. This approach reduces design time by eliminating the need to repeatedly implementing the designs while targeting different boards and scales. We also estimate the expected latency of each architecture to show their trade-offs in terms of both latency and resource utilization.

A. Memory Resource Model

We rely on our existing implementations of the SMiSLIP switch and the GCQ switch to obtain a memory model that can be used to predict the memory resource requirements without being subject to target device constraints. When we consider the SMiSLIP switch, the shared-memory VOQs contribute to all the memory resource utilization resulting in the equality



(a) SMiSLIP resource usage vs. port number and line rate



(b) GCQ resources usage vs. port number and line rate

Fig. 7: A base-10 log scale is used for the Y axis with the dotted horizontal line indicating the 100% resource usage of the target platform; For designs with usage over 100%, synthesis results are provided as they cannot be implemented on the FPGA; URAM is similar to BRAM but with larger unit block size and less flexibility, therefore shared memory module are implemented to first exhaust URAMs prior to using BRAMs to avoid wasting memory resources.

shown in (1), where U_{sm} is the memory resources used by the shared memory VOQs.

$$U_{SMiSLIP} = U_{sm} \tag{1}$$

We further break down the shared memory VOQ architecture into the utilization of its four main sub-components: **(a)** a shared memory buffer (U_{shared_mem}), **(b)** the memory pool (U_{mem_pool}), **(c)** the output queues (U_{out_q}), and **(d)** the indicator ($U_{indicator}$). Consider an N port switch with data path width of W and queues with a depth of D. The shared memory buffer is expected to hold a total of N VOQs. This results in $U_{shared\_mem} = DNW$ as shown in Fig. 4. The memory pool should contain all the addresses for the shared memory. Given the depth of $D \times N$, the input width of the memory pool equals to $log_2(DN)$; hence the usage for memory pool $U_{mem\_pool} = DNlog_2(DN)$. The output queues store the addresses issued by the memory pool with a depth of D, so the usage for output queues $U_{out\_q} = DNlog_2(DN)$ The indicator module should have the same depth with the memory pool and the data width of N to indicate the request status for N outputs. This results in $U_{indicator} = DN^2$. Finally, a total of N shared memory units are needed. In summary the total memory usage U_{sm} for the shared memory can be expressed as:

$$U_{sm} = N \times (\underbrace{DNW}_{shared\_mem} + \underbrace{2DNlog_2(DN)}_{mem\_pool\&out\_q} + \underbrace{DN^2}_{indicator}) \tag{2}$$

Similarly, the GCQ switch memory utilization can be broken down into 3 sub-components: **(a)** the memory based switches (U_{MBS}), **(b)** the VOQs (U_{VOQ}), and **(c)** the output buffers (U_{outbuf}). The core of the GCQ switch is the memory based switch (MBS) which extends the shared memory architecture by adding multiplexers and de-multiplexer at the beginning and end respectively. As a result, the memory usage model for a MBS is very similar to that of U_{sm}, with the eception that the size of MBS used in GCQ is fixed. With a speed up of S, the total number of MBS needed in the GCQ design is $(\frac{N}{S})^2$, hence U_{MBS} can be expressed as:

$$U_{MBS} = (\frac{N}{S})^2 \times (\underbrace{DSW}_{shared\_mem} + \underbrace{2DSlog_2(DS)}_{mem\_pool\&out\_q} + \underbrace{DS^2}_{indicator}) \tag{3}$$

The GCQ switch reduces the number of queues at the input to $(\frac{N}{S})$. The U_{VOQ} usage can then be calculated as follows with D_{VOQ} representing the depth of the VOQ at the input:

$$U_{VOQ} = N(N/S) \times W \times D_{VOQ} \tag{4}$$

GCQ requires additional asynchronous output buffers at the output whose usage can be expressed as the product of the number of buffers and a factor F to compensate the resource variation when using the IP generator. The factor F is calculated by the ratio of actual memory resources used by the IP to the theoretical memory usage without the IP. The memory utilization of a GCQ switch can then be given by:

$$U_{GCQ} = \underbrace{(\frac{N}{S})^2 \times (DSW + 2DSlog_2(DS) + DS^2)}_{U_{MBS}} \\ + \underbrace{F \times N(N/S) \times W \times D_{VOQ})}_{U_{VOQ}} + \underbrace{F \times (N/S)^2 \times SWD}_{U_{outbuf}} \quad (5)$$

B. Port to port Latency model

1) SMiSLIP: The latency of iSLIP based switches are dominated by the minimum clock period T of the scheduler. According to [8], the iSLIP scheduler needs $log_2(N)$ iterations to converge, which means each cell waits for at least $log_2(N)$ cycles in the buffer before it is scheduled to the output port. The latency of iSLIP scheduler can then be calculated as $log_2(N)T$. Therefore, the latency of SMiSLIP in clock cycles can be given by:

$$D_{SM_iSLIP} = TC_{MBS} + log_2(N)T \quad (6)$$

where C_{MBS} is the number of clock cycles that are required for the cell to pass through the shared memory architecture.

2) GCQ latency model: The cross clock domain logic and the pipeline stages in the GCQ design introduce latency overhead to the design. The latency model for the GCQ switch with N port in clock cycles is shown below:

$$D_{GCQ} = T_sC_{wr} + T_sC_{rd} + T_fC_{MBS} + T_sC_p \quad (7)$$

where T_s and T_f are the clock period of slow and fast clock region respectively. C_{wr} and C_{rd} are the write and read delay in terms of clock cycles. Additionally, C_p is the number of pipeline stages inserted to break the long wire connections within the switch and it is equal to the number of memory based switches in a row.

C. Case study: Resource model use case

As an example of how our model could be used in practice, we sketch a possible workflow in Fig. 8. While we do not expect our model to give us the exact number of memory resources that end up being used on the FPGA by each architecture, we can use that to answer the following questiones.

1. Is an SMiSLIP or GCQ switch implementation likely to fail due to insufficient memory resources on a target board?

We answer this question by computing the expected memory resource requirements for the points at the boundary conditions where the first design fails implementation due to insufficient memory resources. This corresponds to a 64-port design with a line rate of 50 Gb/s as can be seen in Table IV. Based on our model, we estimate that 356.515 Mb and 368.738 Mb are required for the SMiSLIP and GCQ architectures respectively. Considering that the target board has a total of 382 Mb of cumulative memory resources (LUTRAM, BRAMs, and UltraRAM), our estimates are within a threshold (6%) for SM-SILP and (3.4%) for GCQ from the overall memory

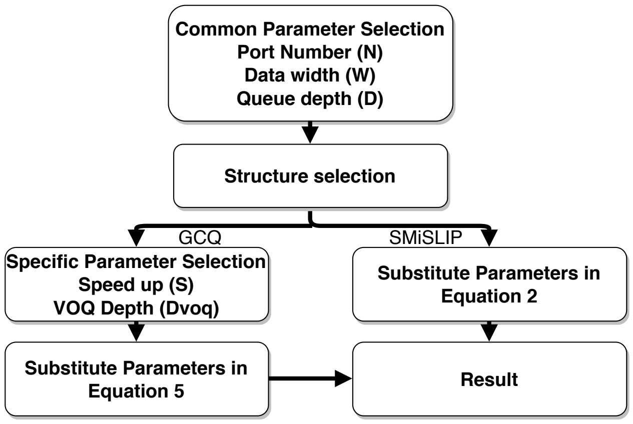

Common Parameter Selection
Port Number (N)
Data width (W)
Queue depth (D)
Structure selection
GCQ
SMiSLIP
Specific Parameter Selection
Speed up (S)
VOQ Depth (Dvoq)
Substitute Parameters in Equation 2
Substitute Parameters in Equation 5
Result

Fig. 8: Memory model usage flow diagram. Note common parameters are parameters shared by both architectures and specific parameters are exclusive to GCQ only

capacity. Based on those numbers, we hypothesize that designs are likely to fail due to insufficient memory resources when the model-based memory resource estimations are within close proximity of the determined thresholds. To verify our hypothesis, we ran a number of tests for designs sets that start to fail implementation, indicated by the first red cross under the green check in table Table IV. The results indicate that on average, the design fails with memory variation of 7.98% over for SM-iSLIP and 2.75% over for GCQ of the overall memory capacity. This implies that the synthesis tools might end up using more memory than actually needed. Therefore, while the model does not provide an exact matching number of resources, it can be used to predict the likelihood of failing implementation due to memory constraints, although more work is needed for rigorous statistical validation of the determined thresholds.

Next, we use our model to predict the lower bound of FPGA memory resource requirements for extrapolating the memory requirements of FPGA switches at various scales. An example is as follows.

2. What is the minimum memory requirement for implementing a 128-port SMiSLIP or GCQ switch operating at a line rate of 50 Gb/s ?

Based on our model, we determine that SMiSLIP and GCQ need 1503.65 Mb and 1474.9 Mb memory respectively for a switch of such a scale. Considering this, and the amount of on-chip memory resources available on FPGAs, we observe that the memory requirements of both designs cannot be supported by the current generation of UltraScale+ FPGAs.

VI. Conclusion

In this paper we study the feasibility for FPGA-based switches to answer our primary question of whether such architectures can be used to implement FPGA based switches that can match the performance of commodity switches used today. We conclude that with today's FPGA chip, there is still an evident gap between them and commodity switches. The best achieved aggregate bandwidth on our target FPGA is 1.6 Tb/s using the GCQ switch architecture for a 64-port switch with a port line rate of 25 Gb/s. This makes FPGAs today more suitable for implementing medium scale switches with lower port count requirements. We further determine the trade-offs of each proposed architecture in terms of performance and scalability. Though the SMiSLIP architecture has lower port-port latency than the GCQ switch, it exhibits limited scalability. This is due to its higher resource utilization as shown by the target independent model, as well as to the difficulty in achieving higher operating frequencies due to its centralized architecture. The GCQ switch, instead, demonstrates better scalability in terms of memory resource requirements and consistency in supported link capacity, although scaling it further is currently limited by routing resource congestion. Further, we observe that achieving timing closure became more challenging as the size of the switch increases since it is possible for a switch fabric to cross two super logic regions (in Xilinx FPGAs) to connect with the transceiver.

Nonetheless, the rapid growth in the size of FPGAs and their performance can help circumvent these challenges and enable larger and faster FPGA based switches that do not rely on a centralized scheduling architecture. Also, with hardened Networks-on-Chips (NoCs) finding their way into FPGAs, the challenges of meeting timing closure can be alleviated, thus enabling even lower latency and resource area usage, which in turn could help bring the performance of FPGA-based switches closer to commercial switches.

Acknowledgment

This work was partly supported by Microsoft Research through its PhD Scholarship Program. The support of the United Kingdom EPSRC (grant numbers EP/L016796/1, EP/N031768/1, EP/P010040/1 and EP/L00058X/1) is gratefully acknowledged.

References

[1] A. M. Caulfield *et al.*, "A Cloud-scale Acceleration Architecture," *MICRO*, 2016.

[2] J. Fowers *et al.*, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," *Proc. of 45th Intl. Symp. on Computer Architecture*, 2018.

[3] D. Firestone *et al.*, "Azure Accelerated Networking: SmartNICs in the Public Cloud," in *USENIX NSDI*, 2018.

[4] "P4 Language." [Online]. Available: https://p4.org/

[5] Intel, "Intel Stratix10 TX Advance Information Brief." [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/s10_tx_fpga_aib.pdf

[6] M. Karol *et al.*, "Input Versus Output Queueing on a Space Division Switch," *IEEE Trans. Commun*, vol. 35, no. 12, pp. 1347–1356, 1987.

[7] N. Gebara *et al.*, "Scheduling Algorithms for High Performance Network Switching on FPGAs: A Survey," *Intl. Conf. on Field Programmable Technologies*, 2018.

[8] N. McKeown, "The iSLIP Scheduling Algorithm for Input-queued Switches," *IEEE/ACM transactions on networking*, no. 2, pp. 188–201, 1999.

[9] W. E. Denzel *et al.*, "A Flexible Shared-buffer Switch for ATM at Gbs Rates," *Computer Networks and ISDN systems*, vol. 27, no. 4, pp. 611–624, 1995.

[10] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queueing with a combined input/output-queued switch," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1030–1039, 1999.

[11] J. Kim *et al.*, "Microarchitecture of a High-radix Router," in *ACM SIGARCH Computer Architecture News*, vol. 33, no. 2. IEEE Computer Society, 2005, pp. 420–431.

[12] Z. Dai and J. Zhu, "Saturating the Transceiver Bandwidth: Switch Fabric Design on FPGAs," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. ACM, 2012, pp. 67–76.

[13] K. Yoshigoe *et al.*, "The RR/RR CICQ Switch: Hardware Design for 10-Gbps Link Speed," in *Conference Proceedings of the IEEE International. Performance, Computing, and Communications Conference.* IEEE, 2003, pp. 481–485.

[14] NetFPGA, https://netfpga.org/site/.

[15] A. Bitar *et al.*, "Efficient and Programmable Ethernet Switching with a NoC-enhanced FPGA," in *Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems*. ACM, 2014, pp. 89–100.

[16] M. S. Abdelfattah *et al.*, "Take the Highway: Design for Embedded NoCs on FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 98–107.

[17] I. Swarbrick *et al.*, "Network-on-Chip Programmable Platform in Versal TM ACAP Architecture," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2019, pp. 212–221.

[18] J. Sheng *et al.*, "Towards Low-latency Communication on FPGA Clusters with 3D FFT Case Study," *Proc. Highly Efficient and Reconfigurable Technologies*, 2015.

[19] Xilinx, "UltraScale+ FPGAs Product Tables and Selection Guide." [Online]. Available: https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf

[20] A. P. Engbersen, "Prizma Switch Technology," *IBM Journal of Research and Development*, vol. 47, no. 2.3, pp. 195–209, 2003.

[21] Xilinx, "Block Memory Generator v8.3." [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v8_3/pg058-blk-mem-gen.pdf

[22] P. Gupta and N. McKeown, "Designing and Implementing a Fast Crossbar Scheduler," *IEEE Micro*, no. 1, pp. 20–28, 1999.

[23] Y. Li, S. Panwar, and H. J. Chao, "On the performance of a dual round-robin switch," in *Proceedings IEEE INFOCOM. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, vol. 3. IEEE, 2001, pp. 1688–1697.

[24] A. Mekkittikul, "Scheduling non-uniform traffic in high speed packet switches and routers," Ph.D. dissertation, Stanford University, 1998.

[25] F. J. González-Castaño, C. López-Bravo, M. Rodelgo-Lacruz, and R. Asorey-Cacheda, "Decoupled parallel hierarchical matching schedulers," *International Journal of Communication Systems*, vol. 20, no. 3, pp. 365–384, 2007.

[26] "Omnet++." [Online]. Available: https://omnetpp.org/

[27] Xilinx, "7 Series GTX Transceivers - TX and RX Latency Values." [Online]. Available: https://www.xilinx.com/support/answers/42662.html

[28] "Juniper." [Online]. Available: https://www.juniper.net/assets/us/en/local/pdf/datasheets/1000480-en.pdf

[29] "Cisco." [Online]. Available: https://www.cisco.com/c/en/us/products/switches/nexus-3000-series-switches/models-comparison.html

[30] "FS." [Online]. Available: https://www.fs.com/uk/c/network-switches-3079