IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received November 24, 2020, accepted November 30, 2020, date of publication December 10, 2020,
date of current version December 21, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3043887

Mapping TSN Traffic Scheduling and Shaping to FPGA-Based Architecture

ZIFAN ZHOU, MICHAEL STÜBERT BERGER, (Member, IEEE), AND YING YAN
Department of Photonics Engineering, Technical University of Denmark, 2800 Kongens Lyngby, Denmark
Corresponding author: Michael Stübert Berger (msbe@fotonik.dtu.dk)

This work was supported by the HI2oT–NordForsk's Nordic University Hubs Programme under Grant 86220.

ABSTRACT Time-Sensitive Networking (TSN), which evolves from the Ethernet standards, has been developed to ensure deterministic transmission in data networks. Asynchronous Traffic Shaping (ATS) extends the conventional synchronized TSN with an asynchronous scheduler to guarantee a bounded transmitting delay. In this work, we present a Field Programmable Gate Arrays (FPGA) implementation of a TSN scheduling entity, which leverages ATS for the frame forwarding process. We explore the ATS design by function blocks and compare it with a benchmark design utilizing strict-priority scheduling. In terms of operating frequency, our results indicate that strict-priority scheduling performs 1.05% to 9.56% higher maximum frequency than ATS with the same configurations. Regarding resource utilization, ATS consumes 51% to 119% more logic blocks and 51% to 101% more registers than strict-priority scheduling. Based on the synthesis and fitting results from Register-Transfer Level (RTL) simulations, we provide a general vision of designing and implementing considerations of the ATS function. Specifically, we show the influences of the buffer and bus width configuration on the FPGA implementation scale and data rate.

INDEX TERMS Ethernet networks, real-time systems, hardware, scheduling algorithm, FPGA.

I. INTRODUCTION

The amount of data to be shared/exchanged has increased exponentially in recent embedded systems, such as automotive, industrial control, and avionic networks [1]–[3]. Since real-time applications sensitive to latency are common in most systems, it raises a requirement for data transmission in the network. In order to build real-time communication capability over the data link layer, it is crucial to guarantee a deterministic low End-to-End (E2E) latency and jitter in technology such as Ethernet. TSN standards provide an improvement to the IEEE 802.3 Ethernet network regarding the deterministic services. One of the main contributions of TSN is to guarantee bounded low latency for frame transmission. Relying on the time synchronization mechanism in TSN, IEEE 802.1 Qbv standard [4] defines the Enhancements for Scheduled Traffic, i.e., Time-Aware Shaping (TAS). With the precise time alignment of all nodes in the network, the TAS-enabled domain assigns dedicated time slots for the time-critical flows to avoid interference from other non-critical traffics.

The associate editor coordinating the review of this manuscript and approving it for publication was Liang-Bi Chen.

However, since the gate operations have to comply with constant time intervals in the scheduler, the Scheduled Traffic (ST) classes in TAS are limited to the flows with a periodic transmission feature. A sporadic flow is not supported as an ST class. From the perspective of hardware implementation, the device needs to allocate hardware resources (i.e., computation and memory) for synchronization maintenance and the execution of the time-triggered gate actions, accordingly.

In order to achieve a real-time guarantee without global time synchronization, the ATS is introduced to the TSN standards [5]. ATS utilizes a token-bucket-based scheduling approach to achieve bounded low latency, and jitter [6]. The ongoing ATS standardization is led by the IEEE 802.1 Qcr working group [7]. It has been proved in [8] that ATS has better performance than TAS for sporadic flows. Nevertheless, the transmissions of periodic critical flows are less predictable in ATS than in TAS. The reason is that ATS offers certain fairness to all traffic classes. Overall, ATS can still provide critical flows with bounded low latency and requires no prerequisite on the traffic pattern (i.e., periodic or sporadic). It makes ATS a promising solution to be part of a deterministic transmission network used for distributed real-time systems.

To ensure the Quality-of-Service (QoS) in networks, ATS implements the flow classification based on the incoming frames' priority level. It supports a flexible classifying mechanism with multiple classes and per-stream scheduling. Hence, the assignment of queuing and priority levels becomes a critical synthesis issue at design time as it determines the latency of each stream. Depending on the network size and QoS requirements, the maximum number of traffic classes supported in the entity also needs to be decided at design time. Nevertheless, the influence of classification on the scale of needed Input or Output (I/O) pins and other hardware resources, such as logic blocks and memory blocks, are still unknown. For hardware platforms, the available resources and performance vary significantly among products. To tailor the devices used for specific cases at justifiable costs, the capability and scalability of the specific type of device are some of the necessary issues that the network equipment providers have to address based on the implementing consideration.

Various hardware is available in the market, such as Application Specific Integrated Circuit (ASIC) and FPGA. FPGA is highly customized and can be quickly reconfigured to support different configurations, so we implement ATS on an FPGA platform to test various configurations. Since the main objective is to investigate the hardware's resource consumption and throughput, we leverage the Intel Quartus and Modelsim software to implement ATS in RTL simulation and validate the implementation with multiple testbenches.

A. CONTRIBUTIONS

ATS has been verified to meet tight real-time constraints with an effective synthesis solution in [9]. The novelty and objective of this paper are to explore the influence of ATS classification policy on the hardware resource consumption and compare the scale and throughput performance of ATS to strict-priority scheduling with the same configurations. Specifically, the contributions of this paper are:

- We present a generic FPGA implementation of ATS that can be configured with different bus widths and maximum numbers of traffic classes, and we give a sophisticated demonstration by function blocks.
- We implement the RTL simulations of a strict-priority component as well as the ATS component on an Intel Stratix V series FPGA device, respectively. Then we compare the hardware frequency, resource utilization, and throughput results between ATS and strict-priority regarding the entire design and individual blocks.
- We evaluate the ATS and strict-priority designs by setting up 32 different configurations for each scheme. We investigate the influences of the number of traffic classes and bus width on the design.

Strict-priority is a basic method for traffic scheduling in a switched network, which uses simple scheme to prioritize the high-priority class. ATS also combines prioritization to execute transmission selection. Furthermore, ATS needs function blocks to record timing information and calculate eligibility time on a per-stream basis to regulate the outbound flows. Therefore, it is interesting to compare ATS with the traditional strict-priority policy in a hardware form.

B. RELATED WORK

In order to accelerate system speed, FPGA hardware is widely used to distribute workloads from the software. A number of works existing in the literature have investigated various FPGA-based Ethernet applications. An FPGA implementation of real-time Ethernet functions is introduced in [10], the FPGA is embedded in a Commercial Off The Shelf (COTS) switch and is used to accelerate the forwarding of real-time frames in the switch. The work in [11] presents an Ethernet switch structure that uses an FPGA for cut-through forwarding, which also supports the IEEE 1588 Precision Time Protocol (PTP). Moreover, specialized FPGA components are often used for dedicated functionalities. In [12], [13], FPGA implementations are used to generate highly accurate timestamp on the frames. The works in [14], [15] propose an FPGA-based implementation performing TSN frame preemption. A hardware/Software co-design of an Ethernet controller is proposed in [16], where the FPGA is used in the Ethernet MAC to achieve time-triggered transmission. The paper includes a hardware consumption of the internal modules in the FPGA device, indicating the task partitioning of hardware and software stack. Memory space is an essential resource for the TSN hardware switch, especially when the on-chip memory is preferable for a faster processing duration. In [17], the author proposed a switch architecture for time-triggered TSN switches with a memory usage optimization. The paper investigated the hardware resource utilization of the proposed TSN switch architecture on an FPGA device. A TSN switch is required to deal with a high data throughput in the network. The switch architecture needs to be faster enough for line-rate operations. In [18], a TSN switch architecture supporting TAS, frame preemption, and Credit-Based Shaper at high throughput is proposed. The architecture utilizes a Virtual Output Queuing (VOQ) framework to save hardware memory utilization. An experimental setup of TSN TAS scheduling is introduced in [19], which was implemented in a hardware platform with prototypical FPGA-based ports that support synchronization and TAS scheduling. The platform achieved guaranteed worst-case latency for time-triggered frames.

ATS has drawn much attention since it offers an alternative to TSN synchronous scheduling methods. The asynchronous feature enables all types of flows with potential deterministic transmissions. Several works have documented the performance analysis of ATS [8], [20]–[23]. However, to the best of the author's knowledge, no one has investigated the hardware implementation of ATS as we introduce in this work.

C. PAPER OUTLINE

The rest of the paper is organized as follows. In Section II, we introduce the basic function blocks of the ATS and the mappings of these function blocks to FPGA. In Section III,

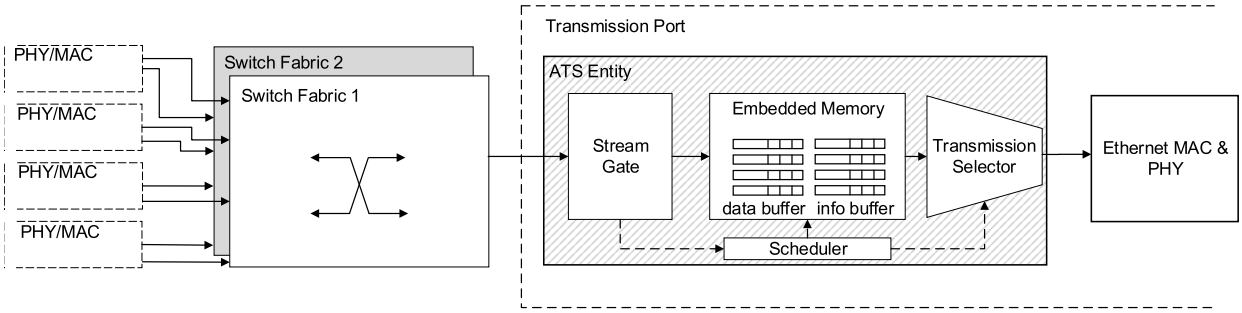


FIGURE 1. An abstraction of the ATS entity (highlighted) in a switching port, the figure demonstrates the location of the ATS entity inside an example switch system.

the primary hardware implementation metrics relevant to the scheduling entity are described. The experimental setup and evaluation of our design are given in Section IV. Additionally, the comparison between the ATS and the strict-priority components are shown in Section IV. The paper is concluded in Section V.

II. MAPPING ATS FUNCTION BLOCKS TO HARDWARE

Switched Ethernet provides effective data transmission in terms of bandwidth utilization. Strict priority is widely adopted in the switched network to provide classified QoS services. The time-critical frames are offered with a high-priority traffic class, eliminating part of the latency caused by low-priority traffic classes. However, a low-priority frame may start ahead of high-priority frame transmission. This condition leads to excessive latency for the high-priority frame.

By creating dedicated transmitting time slots for scheduled flows, TAS scheduling prevents the contention of transmission within the same port. In contrast, ATS utilizes a local system clock for scheduling operations, such as assigning per-stream eligibility times for transmission selection. According to their priority values and the classification rules in ATS, frames are classified into different classes. Later, the frames are buffered in a per-class queuing manner. This section briefly describes the mapping of ATS functional modules to FPGA and demonstrates the hardware implementation.

Figure 1 demonstrates the ATS entity and several relevant parts in a switching device. The ATS entity is deployed at the egress path in each port. Frames from the reception ports are forwarded through the switch fabrics and arrive at the corresponding transmission port's scheduling entity. We implement the ATS entity (highlighted in the figure) and test it in a simulated surrounding. The entity uses the Avalon interface [24] to communicate with other parts.

A. STREAM GATE

In ATS, flows are classified and filtered once arriving at the egress port. At the stream gate, the original priority of the frame can be mapped to an Internal Priority Value (IPV) so that a per-hop implementation of QoS can be achieved. If Per-Stream Filtering and Policing (PSFP) is supported, the stream gate also carries out functions such as gate control operations and flow metering. It is beyond the scope of the paper to explore PSFP in more detail, an overview can be found in [25]. During the frames are parsed in the stream gate, the frame length and arrival time will be transferred to the ATS scheduler module to calculate eligibility time (described in Section II-C).

In this work, all frames follow the Ethernet 802.3 tagged frame format. The format includes a four-byte field for the VLAN tag, as well as a two-byte field indicating the length of the Ethernet frame. To reduce the complexity of the classifying process, we use the VLAN tag for the proprietary classification scheme in our design. Depending on the bus width of the FPGA design, the VLAN tag field comes at a different round of input. Therefore, before the tag field is read, the data must be stored temporally until the traffic class decides the output direction.

B. DATA BUFFER

In the case of ATS, frames are queued separately based on the assigned IPV at every egress port. Therefore, buffering memory needs to be partitioned based on the number of queues. Our designs use the embedded memory blocks provided by the FPGA. The available space for *data buffer* is set to 16384 word in total, where each word equals the bus width. It is important for a designer to consider delineating and sharing the available memory among multiple queues. However, since the buffer depth can only be base-two values, the buffer depth is set to 4096 word in the 3-class design, consuming $3 \times 4096 = 12288$ word memory in total. The 5-, 6-, and 7-class designs use buffers with 2048-word depth, consuming $5 \times 2048 = 10240$, $6 \times 2048 = 12288$, and $7 \times 2048 = 14336$-word total memory space, respectively. While the 1-, 2-, 4-, and 8-classes designs use 1×16384-word, 2×8192-word, 4×4096-word, and 8×2048-word buffers, utilizing all the 16384 word memory space.

Each traffic class is assigned to a separate buffer to enable better traffic flow isolation, where the buffering memory is implemented through a First-In-First-Out (FIFO) approach. Each buffer is constituted by one or a set of FIFOs, with the corresponding memory space. According to the ATS algorithm, the frames' eligibility time are assigned in ascending order so that the frames arrive earlier will be

forwarded before the later ones. The FIFO feature satisfies the mechanism since the frame stored at the head of the FIFO gets processed first.

C. ATS SCHEDULER MODULE

The scheduler module realizes the computational functions and maintains the scheduling parameters on a per-stream basis. The ATS buffering approach provides the possibility of fine-grained traffic management. The scheduler separately makes scheduling decisions and applies to frames that belong to different traffic classes. The following piece of pseudo-code demonstrates the calculation of the frame's eligibility time in the ATS scheduler module. The $D_{lengthRecover}$ and $D_{emptyToFull}$ denote the required duration to recover a full bucket, equaling to the frame length and the committed burst size, respectively. Then the *shaperEligibility* time and *bucketFull* time can be calculated based on the *bucketEmpty* time. Before the frames get queued in the buffer, the module calculates and assigns the *eligibility* time to the frames. The *eligibility* time is the maximum among *arrival* time, *groupEligibility* time, and *shaperEligibility* time. Afterwards, the *bucketEmpty* time is updated based on the values of *eligibility* time, *bucketFull* time, and *shaperEligibility* time.

/* ATS eligibility time calculation */
$D_{lengthRecover} = framelength/rate;$
$D_{emptyToFull} = burstsize/rate;$
$T_{shaperEligibility} = T_{bucketEmpty} + D_{lengthRecover};$
$T_{bucketFull} = T_{bucketEmpty} + D_{emptyToFull};$
$T_{eligibility} = max(T_{arrival}, T_{groupEligibility}, T_{shaperEligibility});$
if $T_{eligibility} \leq (T_{arrival} + MaxResidencetime/1.0e^{9})$
  $T_{groupEligibility} = T_{shaperEligibility};$
  $T_{bucketEmpty} = (T_{eligibility} < T_{bucketFull})\ ?$
    $T_{shaperEligibility}$:
    $T_{shaperEligibility} + T_{eligibility} - T_{bucketFull}$

In our design, the Hardware Description Language (HDL) tool assigns general circuit for the arithmetic functions in the scheduler module instead of Digital Signal Processor (DSP) blocks. DSP is a dedicated block in FPGA to accelerate massive floating-point calculation. The speed improvement in our design is trivial since the ATS algorithm is a relatively simple process. The calculation mostly uses integer addition/subtraction and comparison. The arithmetic function is programmed with less than 20 lines of HDL codes. It is translated to the adder and general FPGA circuit in the scheduler module. The benefits of using a general circuit are retaining the design's simplicity and keeping the design portable among any FPGA platforms. However, this means the design will have higher logic block consumption because of the calculation [26].

D. INFO BUFFER

After calculating eligibility time, a group of *Info buffers* is employed after the scheduler module to store all the eligibility time information. Same as the *data buffer*, the *info buffer* is also separated by the traffic classes. A group of shift registers is used in the scheduler module to delay the eligibility info until their corresponding traffic class is determined. Then the scheduler module forwards the calculated eligibility times to the corresponding *info buffer*. A *valid* signal will be sent to the *Write request* port of the *info buffer* from the scheduler module at the same time so that the positions where the frames and their eligibility info are stored are identical in the *Data* and *info buffer*.

Regardless of bus width, the total space for the *info buffer* is fixed at 4096 word, where each word equals to 32 bit. The 3-class design uses 1024-word buffer for *info buffer*, taking $3 \times 1024 = 3072$-word memory space. The 5-, 6-, and 7-class design use 512-word buffers and occupy $5 \times 512 = 2560$, $6 \times 512 = 3072$, and $7 \times 512 = 3584$-word space. The 1-, 2-, 4-, and 8-class designs have 1×4096-word, 2×2048-word, 4×1024-word, and 8×512-word buffers, utilizing all 4096-word memory space for *info buffer*.

E. TRANSMISSION SELECTOR

After the scheduler module executes the arithmetic function and writes the eligibility times in the *Info buffers*, the stored data are used for transmitting operations by the transmission selector.

When a frame comes to the head of the *data buffer*, the selector sends a read request to the corresponding *info buffer* and matches the eligibility time with the frame. This process recurs every time a frame is transmitted. Accordingly, the selector reads out the frames with eligibility time earlier or equal to the current time. Finally, frames are transmitted when the egress port is idle. Similar to a multiplexer in functions, the selector receives frames from several buffers and transmits through one output. When multiple frames are eligible for transmission, the one with the earliest eligibility time will be selected first. The selector assures one frame of getting transmitted every round to avoid port contention.

The eligibility-time-driven feature decides the transmission selector works in a non-work-conserving manner. It means the selector is not always busy, despite the presence of stored frames in the buffers. The selector only reads out the frames when one or more eligibility times are met with the current time, otherwise, the frames will remain in the FIFO until the time. In this way, the transmission channel may become idle during consecutive transmissions, making lower bandwidth utilization. On the other hand, since the non-work-conserving selector regulates frame transmission according to time instant, it is one of the primary features to achieve deterministic transmission and mitigate jitter in the network.

Figure 1 describes the schematic relationship between the scheduler module and the selector. However, the two function blocks are implemented within the same entity. Because of that, some of the signals and variables can be shared easily between the two blocks.

F. STRICT-PRIORITY BENCHMARK

To benchmark the hardware resource utilization and throughput of traffic scheduling hardware, we also develop a strict-priority scheduling entity as a reference. It is the default transmission selection algorithm supported by all bridges in the IEEE 802.1Q networks [30]. The strict-priority demands no complex scheduling algorithms. It uses a work-conserving selector that always performs transmission when frames exist in the buffers. It is common to integrate strict-priority scheduling with another scheduler to offer a deterministic guarantee.

As depicted in Figure 2, the strict-priority design utilizes a similar structure as the ATS, including a stream gate instance, a group of buffers for per-class queuing, and one transmission selector. The strict-priority stream gate is responsible for classifying and filtering the incoming data frames. The gate supports a maximum of eight different classes. Compared with ATS, the time stamping process is removed from the stream gate, as the transmissions are not triggered by time instant. Frames with the same priority level are stored in one buffer. The memory space applies the same division scheme as ATS. On the other hand, since the scheduling policy requires no frame metadata, the strict-priority design requires no memory space to *info buffer* to store frame information. When frames present in the *data buffer* and the transmission selector is idle, it polls buffers starting from the highest-priority class. As a reference group, the strict-priority design is also evaluated with the configurations of 64-, 128-, 256-bit, and 512-bit bus width, combining with one to eight maximum supported traffic classes.

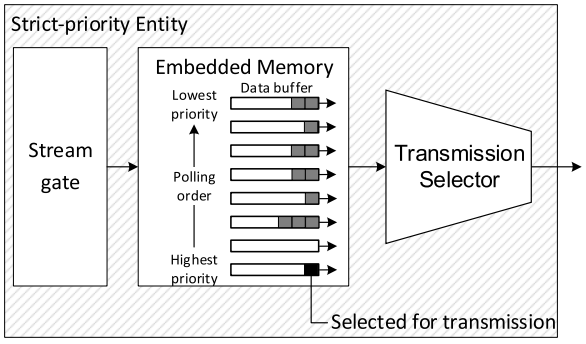


FIGURE 2. An illustration of the modules in the strict-priority entity. The frame stored in the buffer with the highest priority is selected for transmission by the entity.

III. HARDWARE IMPLEMENTATION METRICS

A. LOGIC BLOCK

FPGA comprises arrays of basic logic blocks combined to implement logic-gate, arithmetic, and register-related operations. Each logic block has multiple inputs and outputs pins, and it has access to programmable connections to adjacent blocks. One fracturable Look-Up Table (LUT) receives the inputs to the logic block. It stores the values that correspond to all input variables and drives the outputs according to the input values. In this way, the LUT is able to implement any user-defined Boolean functions. Each logic block also contains several registers before the output pins of the block. In some cases, logic blocks also include adders for simple arithmetic capability [27]. With a fixed designing structure, the number of logic blocks needed in the ATS design is driven by multiple elements. In this work, we investigate the effect of the traffic classification (Section III-E) and bus width (Section III-D) on resources utilization.

B. BRAM

In the FPGAs, large FIFOs are composed of Block Random Access Memorys (BRAMs) [28]. The amount of used BRAM blocks is used to reflect the memory usage of hardware design in this work. The specific Intel Stratix V device used here contains 2640 M20K BRAM blocks with 20480-bit space each. The M20K BRAM supports a flexible memory configuration depending on the bus width of the design. The space of each BRAM (denoted by θ, and $\theta = 20480$ in this paper) can be calculated using Equation 1, where τ and ω denote the depth and data width:

$$\theta_{BRAM} = \tau_{BRAM} \times \omega_{BRAM} \tag{1}$$

Depending on the depth of one FIFO, the number of M20K BRAMs in one FIFO can be calculated by:

$$\delta_{BRAM} = \frac{\tau_{FIFO} \times \omega_{BRAM}}{\theta_{BRAM}} \tag{2}$$

δ_{BRAM} denoted the amount of BRAM per FIFO. Figure 3 shows the number of BRAMs needed per FIFO as a function of FIFO's depth. Four different bus-width cases used in our designs are given. As can be seen from the figure, the amount of required BRAMs per FIFO increases with the bus width and FIFO's depth.

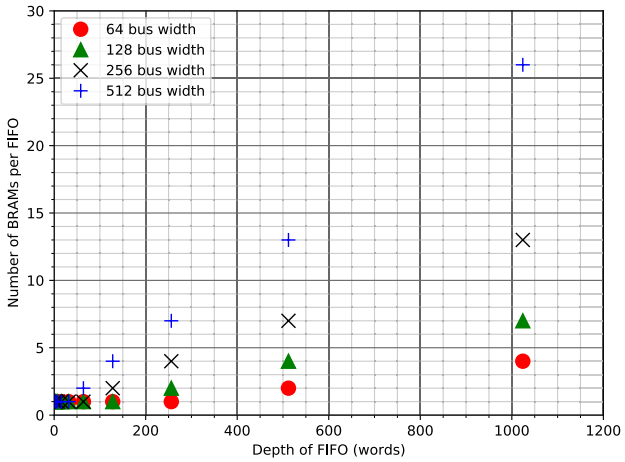


FIGURE 3. Number of memory BRAMs per FIFO as a function of FIFO depth, with 64-, 128-, 256-, 512-bit bus width.

C. I/O PINS

The I/O pins are used to connect the FPGA with external, as well as some reserved pins to specialized peripherals, e.g., external clock signals, Double-Data-Rate Synchronous

dynamic random-access memory (DDR SDRAM). In this work, we use the customized Avalon-Streaming interfaces to connect the ATS entity.

Table 1 shows the I/Os of the ATS entity. The entity uses one *data_in* interface to receive frames from the upstream entity, such as a switch fabric, one *startoffframe* interface to mark the beginning of the frames. The ATS entity transmits all frames via one *data_out* interface to the downstream entity, such as an Ethernet MAC and PHY. The total number of I/O pins used in the design can be calculated by:

$$\Delta_{pin} = \omega_{input} + \omega_{output} + \omega_{clock} + \omega_{start\_in} + \omega_{reset} + \omega_{start\_out} \quad (3)$$

where Δ_{pin} denote the total number of I/O pins. ω_{start_in} and ω_{start_out} are the number of I/Os for the *startoffframe* interfaces. The design also has one *clock* interface for a single timing reference, one *reset* interface, which are represented by ω_{clock} and ω_{reset}, accordingly. ω_{input} and ω_{output} are the data widths of the input and output interfaces.

TABLE 1. I/O pins used in the ATS entity.

I/O	Width (bit)	Direction	Description
data_in	64/128/256/512	Input	Data input from upstream block, e.g. switch fabric
startoffframe_in	1	Input	Mark the beginning of input data
reset	1	Input	Reset bit to initialize all signals
clk	1	Input	Clock signal
data_out	64/128/256/512	Output	Data output from the entity
startoffframe_out	1	Output	Mark the beginning of output data

D. DATA RATE

Traffic scheduling plays an important role in the forwarding process, therefore, the achieved maximum data rate is a crucial parameter for the ATS design. The maximum rate (R) supported by one interface is calculated by:

$$R_{max} = \omega_{bus} \times F_{max} \quad (4)$$

F_{max} is the maximum frequency of the design, indicating how many input data cycles the design can handle in one second. The maximum rate is calculated under the situation when $(64 \times n)$ Byte frames are transmitted, which leaves no spare bit on the data bus. In reality, the data rate declines when there are spare bits in the data bus while processing a frame. We use the maximum results of data rate for comparison in this work.

Modern FPGAs are compatible with high-speed data communications. The Stratix V GT devices come with 28.05-Gbps and 12.5-Gbps transceivers that are suitable for fast switching applications [29]. With a targeted clock frequency of 200MHz, 64-bit, 128-bit, 256-bit, and 512-bit designs correspond to 10G, 25G, 40G, and 100G interfaces, respectively. Bus width plays a critical role in affecting the FPGA utilization. For instance, specific operations may be executed on a per-bit base, especially the registers created for interfaces that increase proportionally with the width.

E. NUMBER OF SUPPORTED CLASSES

The forwarding process supports up to eight traffic classes with separate queues for each class [7]. The process in the one-class configuration is always straightforward. Since there is only one data path in the design, all frames are mapped to the same priority level. The scheduler module only needs a sole function block to calculate the eligibility info. The selector starts to transmit frames from the buffer whenever an eligibility time is reached, without checking if another traffic class occupies the port.

In more complex cases, where multiple classes are supported, the received frames are classified and forwarded to the corresponding buffers by the stream gate. Assuming a single input interface to the stream gate, it can support different numbers of classes with minor changes since one sequential case statement is used for the stream gating process in the design. In contrast to the stream gate, the scheduler module needs separate and concurrent code blocks to calculate per-stream status. The scheduler module scale is expected to vary significantly with the number of streams and traffic classes.

Depending on the network configuration, the required number of traffic classes differs among cases. Designs with more supported classes come with a cost of hardware resources and prices. As described above, the resource utilization changes with the number of traffic classes used in the design. They are caused by the difference of needed logic blocks in the scheduler module. Besides, the configuration and size of FIFOs vary with the number of supported classes, and these factors also significantly impact resource utilization.

IV. EXPERIMENTAL EVALUATION

A. METHODOLOGY

To evaluate the ATS design, we carried out synthesis, fitting, and timing analyzing within RTL simulations. The work is implemented on an Intel Stratix V FPGA (5SGXMABN3F45I4) in the Intel Quartus 16.1 and Modelsim 10.5b software. Firstly, the logic functions are validated with testbench in the RTL simulations. The conducted tests include: (a) stress tests that generate consecutive bursty Ethernet 802.3 tagged frames with the minimal length, (b) randomization tests that feed frames with variable length and interval.

After the functional validations, we run a series of simulated synthesis and fitting for the design in Quartus, these steps translate the code into device-specific primitives and map the primitives to the device and specify the usage of routing resources. The required data on the logic block utilization, registers, pins, and memory blocks can be acquired from the Analysis and Synthesis Summary in Quartus software.

Finally, the static timing analysis is performed using the timing analyzer. The tool measures the timing performance of all paths, as well as determines clock signals for all register-to-register transfers translated from the code. The tool generates a timing report that contains the maximum frequency of the clock in the design.

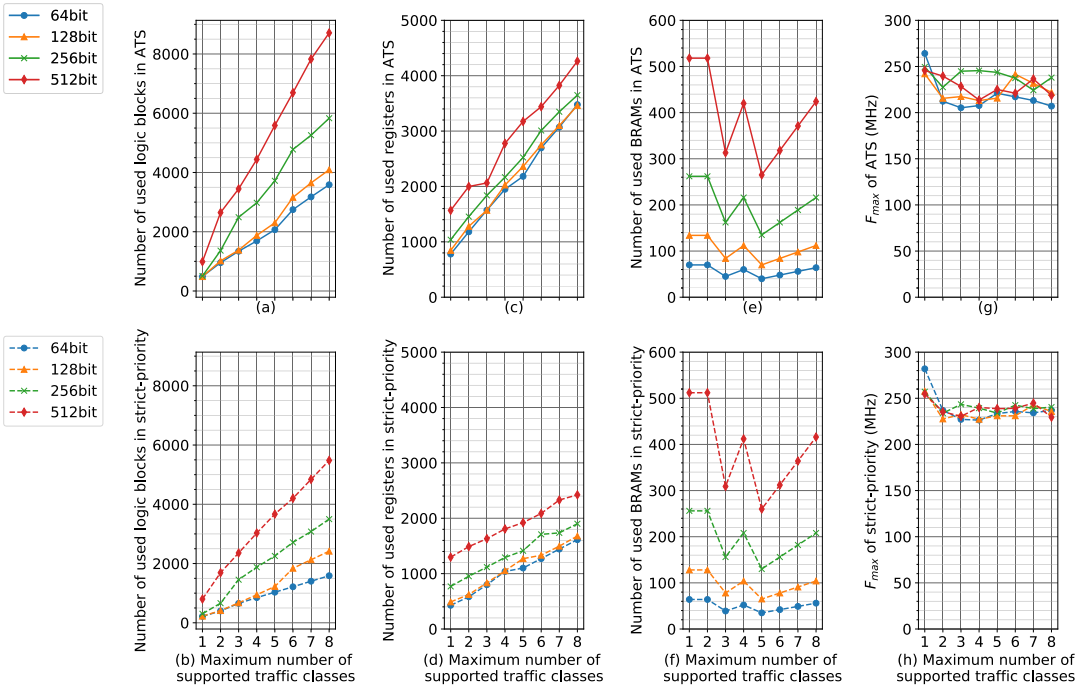


FIGURE 4. All experimental results of two scheduling methods. Four upper charts are the results of ATS scheduling: (a) number of used logic blocks, (c) number of used registers, (e) number of used BRAMs, (g) the maximum frequency (Fmax) as a function of the Maximum number of supported traffic classes. Each chart consists of four cases of different bus width used in the design, differentiated by color. The results of the strict-priority scheduling are given in the bottom row by (b) number of used logic blocks, (d) number of used registers, (f) number of used BRAMs, (h) Fmax as a function of the Maximum number of supported traffic classes.

B. EXPERIMENTS

The platform in this work uses an x64-based processor. Overall, the ATS and strict-priority entities each are tested with 32 different setups from the combination of the following variables:

Bus width. The ATS design presented in this work can be configured to 64-bit, 128-bit, 256-bit, and 512-bit use cases. These four options are picked with the consideration of the minimum Ethernet frame so that the input of one cycle contains only one Ethernet frame or pieces from the same frame.

Maximum supported traffic classes. To evaluate the effect of traffic classification on the design, the maximum supported classes in the design varies from one to eight classes.

C. EXPERIMENTAL RESULTS

All the experimental results are discussed in this section. In Figure 4, the first row consists of (a) number of used logic blocks, (c) number of used registers, (e) number of used BRAMs, and (g) F_{max} as a function of the maximum number of supported traffic classes in the ATS design. Charts in the second row: (b), (d), (f), (h) give the corresponding results of the strict-priority scheduling. Table 2 lists the maximum data rate achieved in each design. Finally, Figure 5 shows the logic block utilization as a function of the traffic class by entities: (a) stream gate, (b) transmission selector.

1) ATS RESOURCE UTILIZATION AND FREQUENCY VERSUS CONFIGURATION

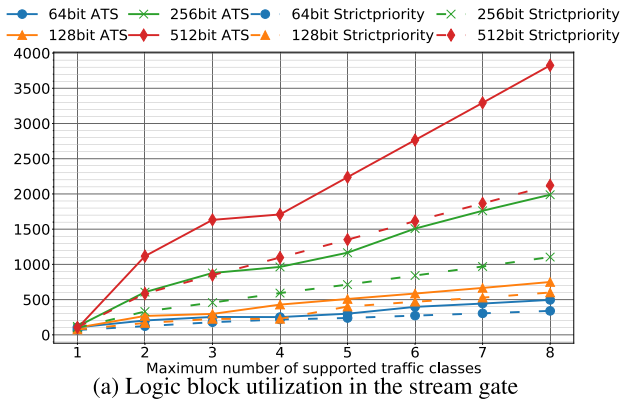
Figure 4a shows the number of logic blocks used in the ATS scheduling. Since the required logic blocks in the scheduling design only take a small portion of the total amount in the Stratix V devices, the utilization is compared through the actual number of logic blocks instead of the percentage, which is below 1% in most cases. As can be seen, the number of logic blocks increases with the maximum number of traffic classes by a factor of 6.31, 7.44, 10.57, 7.80 for 64-bit, 128-bit, 256-bit, 512-bit bus width, respectively.

The differences caused by the bus width is relatively slight for the single-class designs, where the 64-bit, 128-bit, and 256-bit ATS require a similar amount of logic blocks, and the 512-bit ATS utilizes around 500 more logic blocks. The difference becomes more evident with more traffic classes: when the design support eight classes, the 512-bit ATS has 2880, 4620, 5125 more logic blocks than the 256-, 128-, 64-bit ATS. The reasons are that logic blocks are used frequently and mostly on a per-bit basis. Each class requires more logic blocks with a broader bus width. Thus, with the increasing number of supported classes, the difference becomes more considerable.

The results for registers used in ATS are given in Figure 4c. The numbers of registers increase from the single-class designs to eight-class designs, and the growth rates get slower with a wider bus width. The respective increasing factors are

TABLE 2. The maximum data rate of all the strict-priority and ATS designs, values in Gbps.

	64-bit bus width		128-bit bus width		256-bit bus width		512-bit bus width	
Max. number of supported classes	Strict-priority	ATS	Strict-priority	ATS	Strict-priority	ATS	Strict-priority	ATS
1 class	18.04	16.90	32.99	30.98	65.81	63.86	130.40	125.77
2 classes	15.16	13.59	29.13	27.58	59.86	58.25	120.50	122.67
3 classes	14.54	13.13	29.74	27.82	62.32	62.68	117.98	116.92
4 classes	14.49	13.29	29.09	27.27	61.27	62.85	122.90	109.38
5 classes	14.94	14.14	29.56	27.60	59.88	62.36	122.24	115.24
6 classes	15.08	13.89	29.57	30.92	62.10	60.76	122.42	113.17
7 classes	14.99	13.64	31.05	29.66	61.12	57.41	125.35	121.10
8 classes	15.16	13.25	30.17	28.41	61.55	60.91	117.47	111.96



(a) Logic block utilization in the stream gate

(b) Logic block utilization in the transmission selector

FIGURE 5. Logic block utilization by entity in the ATS and strict-priority designs. The results of the stream gate module are given in (a) and the transmission selector's results are shown in (b). In both figures, solid lines represent the results for ATS scheduling, dashed lines represent strict-priority scheduling. Bus width used in the design are differentiated by color.

3.46, 3.10, 2.51, 1.72 for 64-bit, 128-bit, 256-bit, and 512-bit bus width ATS. Taking the vertical comparison, the differences among four bus width cases change slightly from a single class to eight classes. Because the registers' primary usage is to store data temporally, the number of registers scales lightly on a per-bit basis.

The number of required BRAMs in the ATS are plotted in Figure 4e. The results take both Data and *info buffer* into consideration. As described in Section II-B and II-D, the available memory space for *data buffer* is 16384 words of bus width, the available space for *info buffer* is 4096 words of 32 bit. While in the cases of 3-, 5-, 6-, and 7-classes designs, *data buffer* consumes 12288, 10240, 12288, and 14336 words of bus width, *info buffer* consumes 3072, 2560, 3072, and 3584 32-bit word. Since Quartus may employ extra BRAMs to improve timing, the single- and 2-class designs use the most and same amount of BRAMs with the same bus width. 5-class designs use the least BRAMs since they consume less memory space than other cases. 3- and 6-class designs have the same amount of BRAMs, as well as 4- and 8-class designs. More BRAMs are required with wider bus width due to the word width equals the design's bus width.

ATS design achieves higher than $200MHz$ frequency in all cases. Figure 4g indicates that the single-class designs have the highest maximum frequency than the others that support more classes. The results are due to the lack of classification and coordination functions for multiple traffic classes. So the designs can reach the shortest logic delay between registers. The F_{max} of the designs with multiple classes fluctuate within a certain range, the standard deviations of these ATS designs are: $\sigma_{64bit} = 5.7, \sigma_{128bit} = 10.5, \sigma_{256bit} = 8.5, \sigma_{512bit} = 9.4$. The fluctuation is caused by the differences in fitting and placing the modules in the circuit. Thus, the signal transitions among registers vary in every case, which results in F_{max} value changes. Moreover, the fitting results generated by the Quartus software are dependant on the values of the *Fitter Initial Placement Seeds*. In this work, each compilation is tested with multiple seeds, and the maximum results are picked.

Due to the fixed interfaces of the design, the number of I/O pins is independent of scheduling methods and traffic classes. According to the fitting results, the 64-bit, 128-bit, 256-bit, and 512-bit designs use 132, 260, 516, 1028 I/O pins, respectively. The amount of pins only scales with the bus width used in the design since the width of the *data_in* and *data_out* interfaces are the only variables of the I/O pins.

2) COMPARISON WITH BENCHMARK

As shown in Figure 4b, the same variation as ATS also applies to strict-priority scheduling. The number of logic blocks increases with the number of classes by a factor

of 6.39, 9.79, 10.55, 5.84 for 64-, 128-, 256- and 512-bit, respectively. In general, the ATS scheduling requires more logic blocks than strict-priority with the same configuration on traffic class and bus width. On average, the 64-bit, 128-bit, 256b-bit, and 512-bit ATS utilize 119%, 96%, 73%, 51% more logic blocks than the corresponding strict-priority, respectively.

Figure 4d indicates that the number of registers used in strict-priority also increase with the number of maximum traffic classes, but with a slighter change than ATS. The number increases by 2.77, 2.43, 1.47, 0.87 for 64-, 128-, 256- and 512-bit. ATS requires more registers than the corresponding strict-priority designs. The results indicate the 64-bit, 126-bit, 256b-bit, and 512-bit ATS require 101%, 96%, 70%, 51% more registers on average than the strict-priority groups.

Due to the use of *info buffer*, ATS requires slightly more logic blocks than strict-priority, as shown in Figure 4f. At the same time, two scheduling methods require the same amount of BRAMs used for *data buffer* with the same configurations.

Compared with ATS, the strict-priority performs 9.56%, 4.91%, 1.05%, and 4.73% higher F_{max} on average across the 64-bit, 128-bit, 256-bit, 512-bit designs, as given in Figure 4h. Nevertheless, in some particular cases, ATS also achieves higher F_{max} than the corresponding strict-priority. The strict-priority is a basic scheduling method, which requires a less complicated circuit than the ATS to be implemented, therefore, it is more likely to achieve higher frequency in the strict-priority designs. Moreover, these two approaches differ significantly in the achievements, the frequency performance is affected by multiple factors besides the logic complexity, such as the fitting and routing among the hardware circuit. Thus, ATS can also achieve higher frequency than the strict-priority.

3) MAXIMUM DATA RATE

Table 2 lists the maximum data rate achieved by all the designs. The values are collected after running seed sweep for place-and-route in the Quartus software. The maximum values we got from our experiments are given. Within every column, the single-class designs achieve the highest rate compared with the others that support multiple classes. Depending on the higher F_{max} results, strict-priority achieves a higher rate than the corresponding ATS in most cases. Every time the bus width doubles, the rates also approximately become as twice as faster. ATS reaches $13.98Gbps$, $28.78Gbps$, $61.13Gbps$, $117.03Gbps$ data rate on average for 64-bit, 128-bit, 256-bit, and 512-bit designs. The designs are compatible with the 10G, 25G, 40G, and 100G Ethernet, accordingly.

4) LOGIC BLOCK UTILIZATION BY ENTITIES

The number of logic blocks used in the stream gate is given in Figure 5a, where it increases with both traffic classes and bus width. Besides, increasing from one class to eight classes, the number increases by a factor of 3.68 for 64-bit ATS, 6.35 for 128-bit ATS, 15.58 for 256-bit ATS, and 39.69 for 512-bit ATS. The same trend is also observed in the strict-priority scheduling: the number of logic blocks increases by a factor of 3.93, 6.90, 10.04, 19.02, accordingly. Figure 5b shows the number of logic blocks used in the transmission selector, including the scheduler module in ATS cases. As can be seen, the numbers of logic blocks increase linearly with the number of classes. However, the numbers do not scale with the bus width used in the designs. The number of logic blocks used in the transmission selector increases by a factor of 6.63, 6.61, 6.58, 6.72 for 64-bit, 126-bit, 256b-bit, and 512-bit ATS. In the corresponding cases, the factors are 10.21, 9.14, 14.98, 10.74 for the strict-priority. Compared with the entities in the strict-priority, ATS needs to send temporal information to the scheduler module, and the module does the arithmetic functions. Therefore, in order to achieve in circuit, ATS requires more logic blocks than strict-priority in both stream gate and selector module.

V. CONCLUSION AND FUTURE WORK

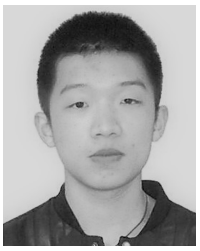
Traffic Scheduling of TSN is a critical feature in the Ethernet ecosystem to achieve real-time communications. In this paper, we map the ATS function blocks to hardware implementation and introduce the FPGA design of ATS. Based on that, we analyze the hardware performance and resource utilization of ATS scheduling. Furthermore, we investigate the impacts of traffic classification and bus width setup on the results. We also create a benchmark using a strict-priority scheduling entity in order to compare ATS with a widely-used scheduling method.

Firstly, our results show the overall utilization and performance of ATS. The frequency of ATS design achieves above $200MHz$ in all cases. The average data rates for 64-bit, 128-bit, 256-bit, and 512-bit designs are compatible with 10G, 25G, 40G, and 100G Ethernet interface. Both the number of logic blocks and registers used in ATS scales with the number of traffic classes. Regarding individual entities, the results indicate the number of logic block in the stream gate increase with both the number of traffic class and bus width. While the amount in the scheduler module only scales with the number of traffic classes. Finally, we compare the results in parallel with the implementation of strict-priority scheduling. Our results show that strict-priority achieves a slightly higher frequency performance than ATS. Meanwhile, ATS requires more hardware resources in the entire design and individual entities than strict-priority. It is worth mentioning that the performance and resource utilization results should be implementation-dependent. The results in this paper represent our selected designs and the experimental setup. All tests are created with consistency for comparing purposes.

As the objective of ATS is to achieve deterministic low-latency transmission. Latency performance is another important metric of the ATS design, therefore, future work can be carried out to investigate the switching delay generated by the ATS entity.

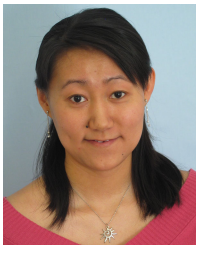
REFERENCES

[1] L. L. Bello, "The case for Ethernet in automotive communications," *ACM SIGBED Rev.*, vol. 8, no. 4, pp. 7–15, Dec. 2011.
[2] J.-D. Decotignie, "Ethernet-based real-time and industrial communications," *Proc. IEEE*, vol. 93, no. 6, pp. 1102–1117, Jun. 2005.
[3] H. Charara and C. Fraboul, "Modelling and simulation of an avionics full duplex switched Ethernet," in *Proc. Adv. Ind. Conf. Telecommun./Service Assurance Partial Intermittent Resour. Conf./E-Learn. Telecommun. Workshop (AICT/SAPIR/ELETE)*, 2005, pp. 207–212.
[4] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 25: Enhancements for Scheduled Traffic*, IEEE Standard 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q), Mar. 2016.
[5] J. Specht and S. Samii, "Urgency-based scheduler for time-sensitive switched Ethernet networks," in *Proc. 28th Euromicro Conf. Real-Time Syst. (ECRTS)*, Jul. 2016, pp. 75–85.
[6] P. Perry Tang and T.-Y.-C. Tai, "Network traffic characterization using token bucket model," in *Proc. IEEE Conf. Comput. Commun., 18th Annu. Joint Conf. IEEE Comput. Commun. Societies Future Now (INFOCOM)*, vol. 1, Mar. 1999, pp. 51–62.
[7] *IEEE Draft Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment: Asynchronous Traffic Shaping*, IEEE Standard P802.1Qcr, Draft 2.3, 2002.
[8] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. Elbakoury, "Performance comparison of IEEE 802.1 TSN time aware shaper (TAS) and asynchronous traffic shaper (ATS)," *IEEE Access*, vol. 7, pp. 44165–44181, 2019.
[9] J. Specht and S. Samii, "Synthesis of queue and priority assignment for asynchronous traffic shaping in switched Ethernet," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2017, pp. 178–187.
[10] H. Flatt, S. Schriegel, J. Jasperneite, and F. Schewe, "An FPGA based approach for the enhancement of COTS switch ASICs with real-time Ethernet functions," in *Proc. IEEE 17th Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2012, pp. 1–4.
[11] H. Flatt, J. Jasperneite, and F. Schewe, "An FPGA based cut-through switch optimized for one-step PTP and real-time Ethernet," in *Proc. IEEE Int. Symp. Precis. Clock Synchronization Meas., Control Commun. (ISPCS)*, Sep. 2013, pp. 7–12.
[12] H. Kirrmann, C. Honegger, D. Ilie, and I. Sotiropoulos, "Performance of a full-hardware PTP implementation for an IEC 62439-3 redundant IEC 61850 substation automation network," in *Proc. IEEE Int. Symp. Precis. Clock Synchronization Meas., Control Commun.*, Sep. 2012, pp. 1–6.
[13] P. Loschmidt, R. Exel, and G. Gaderer, "Highly accurate timestamping for Ethernet-based clock synchronization," *J. Comput. Netw. Commun.*, vol. 2012, pp. 1–11, Dec. 2012.
[14] Z. Zhou, Y. Yan, S. Ruepp, and M. Berger, "Analysis and implementation of packet preemption for time sensitive networks," in *Proc. IEEE 18th Int. Conf. High Perform. Switching Routing (HPSR)*, Jun. 2017, pp. 1–6.
[15] M. Kovacevic, V. Skobic, M. Knezic, and Z. Ivanovic, "Towards implementation of frame preemption mechanism on FPGA platform," in *Proc. 19th Int. Symp. INFOTEH-JAHORINA (INFOTEH)*, Mar. 2020, pp. 1–7.
[16] F. Gross, T. Steinbach, F. Korf, T. C. Schmidt, and B. Schwarz, "A hardware/software co-design approach for Ethernet controllers to support time-triggered traffic in the upcoming IEEE TSN standards," in *Proc. IEEE 4th Int. Conf. Consum. Electron. Berlin (ICCE-Berlin)*, Sep. 2014, pp. 9–13.
[17] Z. Li, H. Wan, Y. Deng, X. Zhao, Y. Gao, X. Song, and M. Gu, "Time-triggered switch-memory-switch architecture for time-sensitive networking switches," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 1, pp. 185–198, Jan. 2020.
[18] A. Pruski and M. Berger, "Design considerations for high-performance time sensitive networking switches," in *Proc. 10th Int. Conf. Netw. Future (NoF)*, Oct. 2019, pp. 114–117.
[19] M. H. Farzaneh and A. Knoll, "Time-sensitive networking (TSN): An experimental setup," in *Proc. IEEE Veh. Netw. Conf. (VNC)*, Nov. 2017, pp. 23–26.
[20] A. Grigorjew, F. Metzger, T. Hossfeld, and J. Specht, "A simulation of asynchronous traffic shapers in switched Ethernet networks," in *Proc. Int. Conf. Netw. Syst. (NetSys)*, Mar. 2019, pp. 1–6.
[21] Z. Zhou, M. S. Berger, S. R. Ruepp, and Y. Yan, "Insight into the IEEE 802.1 Qcr asynchronous traffic shaping in time sensitive network," *Adv. Sci., Technol. Eng. Syst. J.*, vol. 4, no. 1, pp. 292–301, 2019.
[22] E. Mohammadpour, E. Stai, M. Mohiuddin, and J.-Y. Le Boudec, "Latency and backlog bounds in time-sensitive networking with credit based shapers and asynchronous traffic shaping," in *Proc. 30th Int. Teletraffic Congr. (ITC)*, vol. 2, Sep. 2018, pp. 1–6.
[23] E. Mohammadpour, E. Stai, M. Mohiuddin, and J.-Y. Le Boudec, "Latency and backlog bounds in time-sensitive networking with credit based shapers and asynchronous traffic shaping," 2018, *arXiv:1804.10608*. [Online]. Available: http://arxiv.org/abs/1804.10608
[24] Intel. *Avalon Interface Specifications*. Accessed: May 26, 2020. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf?language=en_US
[25] J. L. Messenger, "Time-sensitive networking: An introduction," *IEEE Commun. Standards Mag.*, vol. 2, no. 2, pp. 29–33, Jun. 2018.
[26] E. Monmasson and M. N. Cirstea, "FPGA design methodology for industrial control systems—A review," *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1824–1842, Aug. 2007.
[27] Altera, Intel Corporation. *Stratix V Device Handbook, Volume 1:, Device Interfaces and Integration*. Accessed: Apr. 13, 2020. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-v/stx5_core.pdf
[28] I. Kuon, R. Tessier, and J. Rose. *FPGA Architecture: Survey and Challenges*. Boston, MA, USA: Now, 2008.
[29] Altera, Intel Corporation. *Stratix V Device Overview*. Accessed: Jun. 15, 2020. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/documentation/sam1403476018909.html#sam1403475986993
[30] *IEEE Draft Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks*, Standard P802.1Q-Rev/D0.2, Jul. 2020.



ZIFAN ZHOU received the M.Sc. degree in telecommunication engineering from the Technical University of Denmark (DTU), in 2018. He is currently pursuing the Ph.D. degree in latency-critical transmission with the Network Platform and Service Research Group. His research interest includes simulations and hardware design of ultra-low latency networks.

MICHAEL STÜBERT BERGER (Member, IEEE) was born in 1972. He received the M.Sc. degree in electrical engineering and the Ph.D. degree from the Technical University of Denmark, in 1998 and 2004, respectively. He is currently an Associate Professor with the University of Denmark, within the area of switching and network node design. He has been participating in several projects with relation to TSN. He was a Project Leader on the national project Ethernet for RAN (ERAN), where TSN was explored in the Front haul Network. Furthermore, he coordinated the participation from DTU in a Eurostars Project on TSN (Fronthaul for CRAN). He is also responsible for the Department participation in a Nordic University HUB project on industrial IoT, fog computing, and TSN. He is currently a Mentor of one Postdoctoral position and two Ph.D. students in the area of TSN and deterministic networks.



YING YAN received the B.Eng. degree in electrical engineering from the Beijing University of Technology, China, in 2002, and the M.S. degree in electronics engineering and the Ph.D. degree in telecommunication engineering from the Technical University of Denmark, in 2002 and 2010, respectively. From 2006 to 2007, she worked as a Research Scientist with the Department of Communication Platforms, Technical Research Centre of Finland (VTT), Finland. Her research interests include time sensitive network (TSN), the Internet of Things (IoT) networks, 5G Mobile networks, network simulation and emulation, traffic data mining and analysis, and network security.