2018 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
VEHICLE ELECTRONICS AND ARCHITECTURE & GROUND SYSTEMS CYBER ENGINEERING (VEA/CYBER) TECHNICAL SESSION
AUGUST 7-9, 2018 - NOVI, MICHIGAN

DETERMINISTIC ETHERNET CHALLENGES AND SOLUTIONS

David A. Verbree, PhD
DornerWorks Ltd.
Grand Rapids, MI

Brian C. Douglas
DornerWorks Ltd.
Grand Rapids, MI

Anthony J. Torre
RDECOM TARDEC
Warren, MI

ABSTRACT

DornerWorks has developed a high-assurance communication solution for military use that leverages open standards and COTS technologies to lower cost, reduce SWaP, and provide a roadmap that protects against obsolescence. Throughout our R&D efforts, we discovered shortcomings with existing COTS solutions that we sought to address in developing a custom, yet standards-based, solution. This paper describes the unique approaches we used to overcome these shortcomings, to optimize performance, and to improve ease-of-use to develop a time-synchronous AVB-capable MAC (Media Access Controller) FPGA IP core that can interface with most COTS 1-Gbps PHYs.

INTRODUCTION

AVB (Audio Video Bridging; IEEE 802.1BA) is a collection of standards that describes mechanisms for time-synchronization and bandwidth reservation over a COTS network. It consists of several protocols including generalized precision time protocol (gPTP; IEEE 802.1AS), AVB Tunneling Protocol (AVBTP; IEEE 1722), Multiple Registration Protocol (MRP; IEEE 802.1Q-2014, Clause 10), Multiple Stream Registration Protocol (MSRP; IEEE 802.1Q-2014, Clause 35), and Multiple VLAN Registration Protocol (MVRP; IEEE 802.1Q-2014, Clause 11.2).

The AVB protocols are complex and usually implemented in software that leverages several operating system (OS) services such as multi-process scheduling, run-time memory allocation, several timers, DMA and network interface drivers, and a network stack. It usually requires a hard-processor and DDR memory. This is an unacceptable burden and cost to levy on simple devices that only require time synchronization and the ability to send and receive data in deterministic fashion.

Our goal is to eliminate this burden by implementing these protocols in custom logic, thus providing an easy-to-use and affordable high-assurance communication solution with broad industry support. Since AVB has a road map to the emerging Time-Sensitive Networking (TSN) standards, it also mitigates the risk of obsolescence.

In this paper, we describe approaches we used to overcome shortcomings with existing solutions in developing an AVB-capable MAC FPGA end-point IP core. It also describes our innovations to improve time-synchronization and communication performance, and ease-of-use while adhering to the standards.

TIME-SYNCHRONIZATION

Making Sense of the PTP Standards

State-of-the-art distributed electronics systems for military and industrial use often require high-accuracy time-synchronization. Several commercially available solutions exist that provide various mechanisms and degrees of time-synchronization. Most solutions require dedicated hardware and infrastructure (e.g. wiring) and can increase a system's SWaP-C considerably. Many solutions can only adjust for drift (relative time) and have no inherent mechanism to synchronize to absolute time.

A mechanism of time-synchronization over COTS Ethernet was first standardized in IEEE 1588-2002 (PTP). It describes a mechanism to synchronize each device's absolute time to that of a "grand-master" (GM). Each device periodically sends a delay request packet to the GM and the GM sends delay response back to the device. The packets contain ingress and egress timestamps added by the hardware that each time-aware device uses to calculate propagation delays. Each time-ware device uses propagation delays to adjust GM timestamps periodically sent in separate packets in order to calculate the time difference between the GM and local device clocks. Typically, a device uses this difference to adjust a clock increment value that causes the time of the local clock to converge with the GM time. PTP has the benefit of being able to use the same physical medium (e.g. Ethernet) used for general communications, thereby reducing size and weight.

Table 1. Major revisions of the PTP/gPTP protocols

Features	IEEE 1588-2002	IEEE 1588-2008 (v2)	IEEE 802.1AS-2011	IEEE 802.1AS-REV (draft only)
Residence time correction	No transparent clocks – e.g. switches are not time-aware	Transparent clocks – switch adjusts packet timestamps with residence time		
Hardware timestamping	one step only	one and two-step	two-step only	one and two-step
Delay calculation mechanism	path delay	peer delay or path delay	peer delay	
Bridge compatibility	time-aware and non-time-aware		time-aware only	
Protocols supported	layer 2-4, IPv4 multicast only	layer 2-4, IPv4 or IPv6, multicast or unicast	layer 2 only	
GM time transmit interval	up to 1/s	up to 10/s		
Grandmaster redundancy	Multiple domains supported simultaneously (different subnets)		Single domain (single active grandmaster)	Redundant GMs in "hot-Standby"
Syntonization	optional	logically required		
Asymmetry correction	none	optional		
Conformance	not specified	specified		
Min End to End Sync Accuracy	< 1 us	< 1 us for <= 7 hops	<1 us for <= 7 hops FFO +/- 100 ppm; RR: +/-0.1 ppm Jitter <2ns/60s; Granularity: <40ns	

Although the basic concept of PTP has remained the same over the years, the protocol itself has gone through several substantial revisions, which are not all compatible with each other (Table 1). In general, there are two dominant variants, the IEEE 1588 series, commonly referred to as PTP, and IEEE 802.1AS series, commonly referred to as gPTP. However, IEEE 1588-2008 (known as v2) introduced the most substantial differences in functionality. Unfortunately, this version introduced so many new variations that it became difficult to ensure interoperability among implementations. IEEE 802.1AS-2011 reduced the number of supported options in order to provide better performance guarantees and consistent interoperability.

The most notable features introduced by IEEE 1588-2008 that persists in the IEEE 802.1AS standard are the peer delay and transparent clock mechanisms. In the previous version of the standard (2002), end-to-end (path) latency delay is determined from the end-point all the way to the GM. This is inefficient on a multi-hop network especially if either topology or routing is dynamic. IEEE 1588-2008 introduced a peer delay mechanism whereby each device in a path only calculates its delay to its peer or neighbor(s). When the GM transmits its timestamp in a packet, each time-aware device in the path accounts for its latency to its neighbor. All time-aware devices also have a transparent clock feature, which allows them to update the timestamp in packets on the fly to account for residence time.

IEEE 1588-2008 introduced one other important aspect that persists in the IEEE 802.1AS standards. It specified what is required for a device to be conformant with the standard, which is essential given the number of possible different variations introduced by the standard.

IEEE 802.1AS-2011 eliminates several variants to improve interoperability. It does not support layer 3 and 4 PTP protocols. All gPTP protocols are layer 2, which greatly reduces complexity for hardware or logic-based solutions. It eliminates the path delay mechanism in favor of the peer delay mechanism and it mandates that all bridges in the path between a time-aware device and the GM must be time-aware (i.e. provides transparent clock functionality). It also adds performance requirements on the stability and variance associated with the accuracy of time-synchronization.

IEEE 802.1AS-REV is the latest draft standard. Its goal is tighten-up the performance requirements further to better accommodate the emerging TSN standard. Our implementation is based on this latest draft standard to be forward-looking to TSN.

Improving Time-Synchronization Accuracy

According to the IEEE 802.1AS standards, gPTP implementations must obtain < 1 us end-to-end synchronization accuracy for 7 or fewer hops during steady state conditions. Currently, the IEEE 802.1AS-REV draft is considering 1-us end-to-end synchronization accuracy for 64 or fewer hops for industrial applications which is approximately 15.6 ns per hop. This has been our goal as well.

Typically, the gPTP protocol is implemented in complicated software that requires a hard processor, DDR memory, and operating system services. The software running in an operating system may experience substantial and non-deterministic delays in retrieving the time-counter register value used for hardware timestamping of PTP packets. Depending on the exact mechanism used to synchronize the local clock to the GM clock, sources of error may include the following:

1) The mean latency and variance of reading the local clock. This may be strongly influenced by the latency of context switches, task preemption, interrupt preemption, system call overhead, etc.
2) The time between reading the local clock and applying an adjustment to the local clock or the local clock increment. By the time the correction is applied, the local clock may have drifted further away from the GM or may

already be converging due to a previous adjustment.

3) Accuracy of the hardware timestamping and clock phase alignment of each device in the chain. If every link in the chain only provides hardware timestamps with the minimum granularity of 50 ns, then the maximum theoretical error would be 50 ns/hop if phase and frequency aligned or up to 100 ns/hop if unaligned. This could be extensive in a multi-drop / multi-hop configuration

We overcame these challenges using several unique approaches. The entire gPTP protocol was implemented in custom logic (FPGA) making all packet parsing, packet transmission, calculations, and clock adjustments highly deterministic (Figure 1). All calculations are made using 16-B math representing EPOCH time with a resolution down to the $2^{(-16)}$ nanoseconds. This is to prevent any loss of precision.

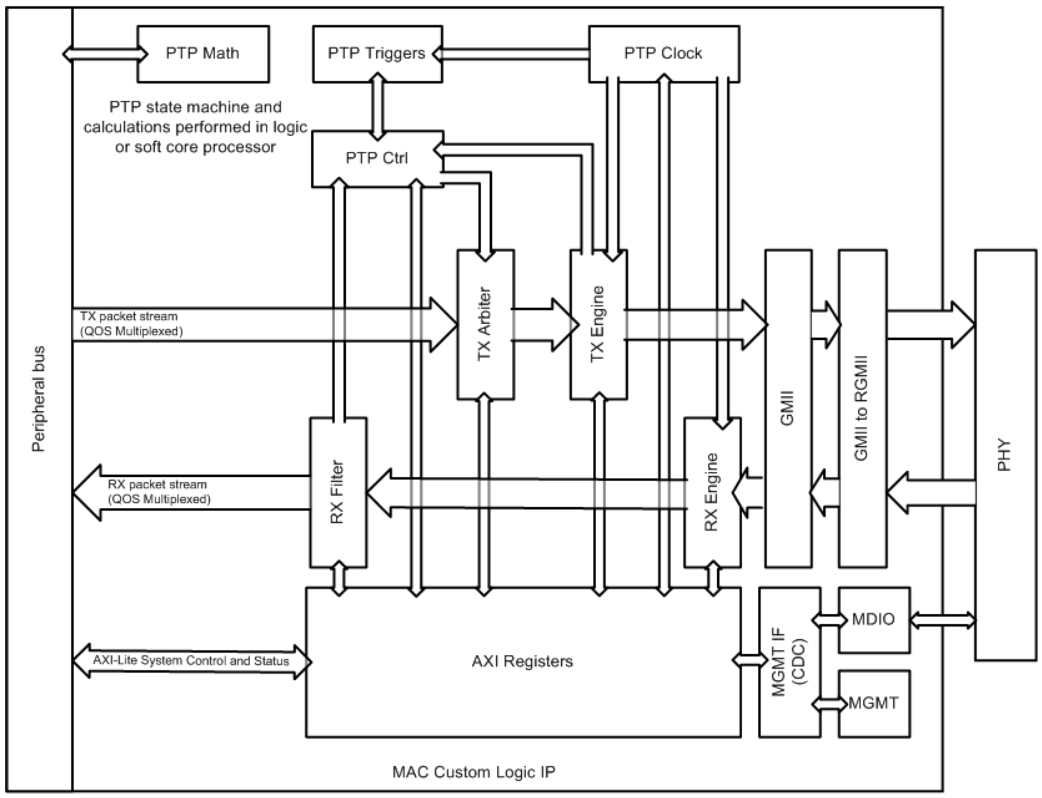
PTP Math
PTP Triggers
PTP Clock
PTP state machine and calculations performed in logic or soft core processor
PTP Ctrl
TX packet stream (QOS Multiplexed)
TX Arbiter
TX Engine
GMII
GMII to RGMII
PHY
Peripheral bus
RX packet stream (QOS Multiplexed)
RX Filter
RX Engine
AXI-Lite System Control and Status
AXI Registers
MGMT IF (CDC)
MDIO
MGMT
MAC Custom Logic IP

Figure 1. DornerWorks AVB-capable MAC IP core architectural overview

The logic updates the clock increment using a P-I loop (Equation 1) to minimize the master-local offset, which is the error term. This compensates for most frequency differences or phase offsets between the device and the grandmaster as well as any indeterminism introduced via the SGMII/RGMII interfaces or the PHY itself. The clock increment base value (e.g. 8 ns) is scaled up 2^{24} to allow very fine adjustments.

Equation 1. Proportional integral loop clock increment adjustment.

$$newInc(t) = baseInc + K_p e(t) + K_i \int_{t0}^{t} e(\tau)d\tau$$

Where:

- $newInc$ is the new clock increment value
- $baseInc$ is the base clock increment value (e.g. 80000000_{16})
- K_p is the proportional gain constant (e.g. 3.0 x 2^{16})
- $e(t)$ is the most recently calculated MLO value which represents the error
- K_i is the integral gain constant of (e.g. 0.3 x 2^{16})
- t is the present time.
- $t0$ is the initialization time of the calculation.

The proportional and integral gain constants may be modified either statically or dynamically to minimize convergence time and maximize clock stability and accuracy. The clock increment value is updated with every sync message received from the GM (up to 10/s). The clock increment is applied to the local clock at the link rate frequency (125 MHz for a 1-Gb link).

We instantiated two of our end-points on Zed Boards with quad-port FMC Ethernet cards connected to an AVB-capable switch and configured each to provide a 1-PPS output at the top of the second of the synchronized time. We connected both 1-PPS outputs to a 1-GHz oscilloscope to verify synchronization accuracy. This difference between the rising-edge of both signals was less than 8-ns and stable. This is the theoretical maximum accuracy achievable with a 1-Gbps clock frequency of 125 MHz.

The most notable feature of this approach is that time-synchronization is done automatically behind the scenes. System integrators and software developers do not need to allocate any additional resources (CPU processing time, DDR memory, etc.) to support time synchronization.

Improving Synchronized Clock Distribution

An accurately synchronized local clock is of little use if it cannot be distributed accurately especially to software processes running on the same device. It is relatively easy to synchronize software applications to system time derived from an RTC or CPU clock. However, it is not as easy to synchronize software to a synchronized network time that is only accessible through a physical network device (e.g. NIC, PHY, or MAC) without incurring high and non-deterministic latency as previously described.

We developed a general-purpose solution that is ideal for distribution to other custom logic as well as for routing to a hard processor via an interrupt controller. Included within our AVB-capable MAC IP core are eight user-configurable triggers that are externally available. Each trigger is configurable with the start time in EPOCH time down with 1-ns resolution, the period in nanoseconds up to 1s for recurring triggers, and the number of iterations to trigger on. Each trigger is asserted when the synchronized network time is within 8 ns of user-configured trigger time.

When routed to an interrupt controller, the user triggers can synchronize software in multiple ways. They can directly drive the system tick and/or OS scheduler to ensure a process awakes at a synchronized time interval. They can also be configured as general-purpose periodic timers that an application can register interrupt handlers for. Whether routed to an interrupt controller or to down-stream logic, they provide an effective way to clock distributed hardware or processes nearly in lock-step.

TRAFFIC SCHEDULING

IEEE 802.1Q-2014 describes how AVB traffic and best-effort traffic is scheduled and how multiple traffic classes or QoS levels are supported. We implemented our AVB-capable

MAC IP according to the standard. In some cases we went beyond the standard to bring in new innovations that reduce latency and maximize throughout, particularly when using multiple traffic classes and QoS levels. We designed our IP core to support 8 QoS levels and any or all QoS levels can be runtime configured to use either the credit-based scheduler (802.1Q-2014, section 8.6.8.2, Forwarding and Queuing for Time-Sensitive Streams; FQTSS) or the strict priority scheduler (802.1Q-2014, section 8.6.8.1). The bandwidth of each QoS using the credit-shaper can also be runtime configured.

Reducing latency

Nearly all implementations of network stacks include packet queues or buffers for transmission and reception of packets. The queues or buffers are implemented in either software, hardware, or most often in both. Every queue introduces bounded but non-deterministic latency that depends on the number of packets enqueued and the fill capacity of the queue. For example, every 1500-B packet enqueued adds 12.3 us of latency.

We designed our primary AVB-capable MAC IP core to eliminate all packet queues or buffers. All data is streamed via AXI interfaces. For transmitters, the only back-pressure applied to the sender external to the IP core is via the AXI TREADY signal. This approach provides considerable flexibility to the user. The user can choose to add a queue of any size to any egress QoS. For highly time-critical applications, the user can omit it completely and send data as soon as the TREADY signal is asserted (for strict-priority traffic). More importantly, they can vary the FIFO sizes for each QoS depending on the specific application requirements, keeping in mind that smaller FIFOs are likely to result in lower latency and greater determinism. Figure 2 is an example of how the traffic scheduler selects packets from multiple sources and outputs a multiplexed stream based on the source QoS priorities. It also demonstrates how the traffic scheduler controls the TREADY (data ready) AXI signal. In this example, the traffic scheduler is holding off traffic (policing bandwidth) until data ready is asserted high for source 1.

Our IP core does not have built-in buffers for incoming packets either, other than a small FIFO for filter matching. The logic computes the CRC on the fly. If the computed CRC does not match the one at the end of the received packet, it asserts a CRC failed signal, but the logic does not automatically discard failed packets like most implementations. This has the advantage of allowing a receiver to start processing an incoming packet before it is completely received and validated. For receivers capable of handling this situation, it can reduce latency by up to 12.3 us for a 1500 B packet. The user can choose to add receive queues of any size to an ingress QoS and flush the queue if the CRC failed signal is asserted.

Maximizing throughout

The credit shaper (FQTSS) does an excellent job of rate-limiting bandwidth for a single QoS. AVB-centric implementations are likely to run the scheduler at a period of ≤125 us (8 kHz) which is the scheduling frequency required by AVB SR Class A. This is major source of latency and non-determinism when transmitting packets. It is also a major source of lost bandwidth caused by gaps between scheduled packets. Through simulations, we discovered that the effective throughput is highly influenced by three important factors when multiple QoS are in use:

1) frequency that traffic is scheduled
2) size of packets being sent
3) number of different QoS being scheduled

Since we are not able to control the size of packets transmitted or the QoS on which they are transmitted, we can only optimize the traffic scheduler frequency. We found that 125 MHz is the only scheduler frequency that maximized effective throughput for all packet sizes regardless of how many QoS were in use. This was not surprising since it is the frequency of the 1-Gbps link. Implementing the traffic scheduler at this frequency eliminates all gaps between successive packets sent on different QoS levels (e.g. AVB SR Class A and AVB SR Class B or best-effort).

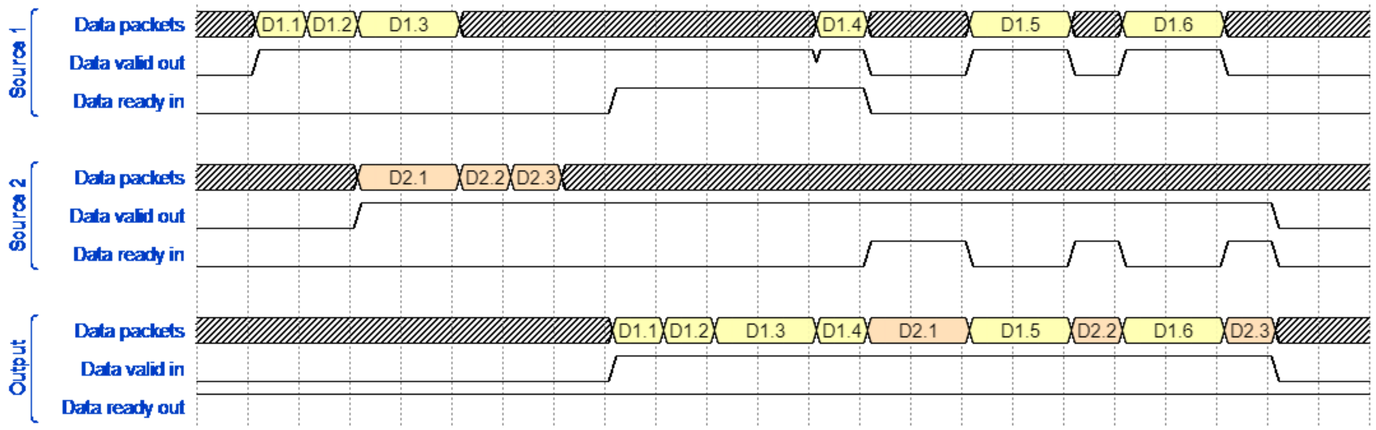

Figure 2. Example timing diagram of gapless traffic scheduler (overview) with multiple data sources. Source 1 is higher priority. Traffic scheduler is accepts data when Data ready is asserted high for Source 1.

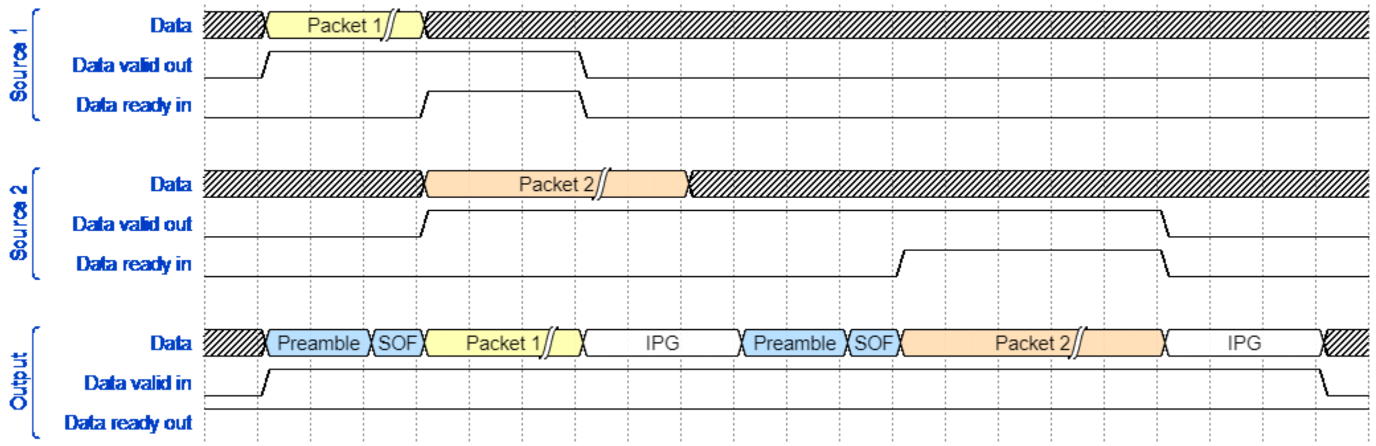

Figure 3. Example timing diagram of gapless traffic scheduler (detail) with multiple data sources. Traffic scheduler accepts data when Data ready is asserted high for Source 1. A subsequent packet is scheduled while the IPG of the previous packet is being transmitted, thereby reducing latency and eliminating the potential loss of throughput. This also prevents priority inversion (i.e.. higher priority packet waiting for a lower priority packet to be transmitted).

Implementing a traffic scheduler that runs every 8 ns would have been impossible in software, but we were able to implement the scheduler in logic. To achieve this, a subsequent packet needs to be scheduled for transmit while another packet is still being transmitted. The Ethernet standard (IEEE 802.3) specifies that there must be a 12-B gap between packets. We were able to take advantage of that interpacket gap by running the scheduler during that time. This allows a high priority packet enqueued during the current transmission to be transmitted immediately following it with no latency. Figure 3 is an example of how the traffic scheduler selects packets from different sources during the IPG and transmits the multiplexed output stream.

The logic-based traffic scheduler we designed has additional features to optimize throughput. We provide high-resolution configuration (0-

125000000 B/s for each credit-based QoS to allow very fine bandwidth allocations.

The IEEE 802.1Q-2014 standard only allows credit-based traffic to be reserved up to 75% of the total bandwidth. The standard is also somewhat inconsistent on whether the packet overhead (preamble, start delimiter, and interpacket gap) is included in the portion of reserved bandwidth or in the remaining unallocated bandwidth. Our implementation takes all packet overhead out of the reserved portion of bandwidth so it does not consume bandwidth assumed to be reserved for strict-priority traffic. This allows up to 100% of the total bandwidth to be reserved for credit-based streams.

USABILITY OF AVB

One of the most challenging aspects of AVB is its usability. It requires several different processes (e.g. gPTP, MSRP, and MVRP) running concurrently with the user's application. As previously described, we implemented gPTP entirely in logic, so the next challenge was to simplify MRP, MSRP and MVRP used for bandwidth reservations.

Simplifying stream reservations

MSRP and MVRP, variants of MRP, are protocols used to request bandwidth for streams and to dynamically join VLANs, respectively. The reference implementation of MSRP and MVRP comes from the MRP daemon included in the Open Avnu software distribution (https://github.com/AVnu/OpenAvnu) maintained by Avnu Alliance (the certification body for AVB and the emerging TSN standard). This software is written for Linux and requires many OS services.

As a first step, we substantially re-wrote the MRP daemon so it runs as a bare-metal embedded application. We removed all OS dependencies including runtime memory allocation and deallocation, direct driver accesses (IGB, PHC, and DMA), use of software timers, and all multi-threading. We created a simple API for the application to call directly rather than going through a MRP client API via a loopback interface to an MRP server daemon.

While streamlining the MRP daemon is a good step forward for a simple embedded application, it does not eliminate the dependency on a hard CPU or DDR. However, our current effort to implement MSRP and MVRP entirely in logic will remove that dependency.

Simplifying AVBTP (IEEE 1722)

Another area of usability that we improved upon is to provide automatic AVBTP packet encapsulation entirely in logic. We created two IP modules. One module encapsulates Ethernet frames with an appropriate L2 header based on values configured in registers. The other encapsulator is much more generic. It uses a flexible configuration stored in block RAM to encapsulate the packet with additional headers (L3-4), including an AVBTP header. With these encapsulators, the user only needs to configure an encapsulator and transmit the raw packet payload. The rest is handled in logic. This is particularly useful for simple sensors that send the same kind of data packet on a periodic basis.

FUTURE WORK

Presently, our AVB-capable MAC IP core is actively under development. We designed it from the beginning with the plan to add TSN once most of the relevant standards were ratified or at least mature. This has been our rationale for minimizing latency throughout the development of our AVB-capable IP core. Our current roadmap consists of a completely logic-based implementation of AVB and eventually TSN.

REFERENCES

[1] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," IEEE Std 1588-2002, pp. i–144, 2002.

[2] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked

Measurement and Control Systems," IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002), pp. 1–300, Jul. 2008.

[3] "IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks," IEEE Std 802.1AS-2011, pp. 1–292, Mar. 2011.

[4] "IEEE Standard for Layer 2 Transport Protocol for Time Sensitive Applications in a Bridged Local Area Network," IEEE Std 1722-2011, pp. 1–65, May 2011.

[5] "IEEE Standard for Local and metropolitan area networks--Bridges and Bridged Networks," IEEE Std 802.1Q-2014 (Revision of IEEE Std 802.1Q-2011), pp. 1–1832, Dec. 2014.

[6] "IEEE Standard for a Transport Protocol for Time-Sensitive Applications in Bridged Local Area Networks," IEEE Std 1722-2016 (Revision of IEEE Std 1722-2011), pp. 1–233, Dec. 2016.

[7] "IEEE Approved Draft Standard for Local and Metropolitan Area Networks--Bridges and Bridged Networks," IEEE P802.1Q-REV/D2.2, January 2018, pp. 1–2000, Jan. 2018.

[8] "IEEE Draft Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications," IEEE P802.1AS-Rev/D6.0 December 2017, pp. 1–496, Jan. 2018.

[9] "IEEE Standard for Local and metropolitan area networks--Audio Video Bridging (AVB) Systems," IEEE Std 802.1BA-2011, pp. 1–45, Sep. 2011.