CLOCK DUPLICITY FOR HIGH-PRECISION TIMESTAMPING IN GIGABIT ETHERNET

Carles Nicolau, Dolors Sala*
Dept. of Information and Communications Technologies
Universitat Pompeu Fabra
carles.nicolau, dolors.sala@upf.edu

Enrique Cantó
Dept. of Electronics, Electrical and Automation
Universitat Rovira i Virgili
ecantó@etse.urv.es

ABSTRACT
Hardware-timestamping is essential for achieving tight synchronization in networking applications. This mechanism is selectively used on few high-cost tailored systems. Actual μP-based implementations fail on deterministic timestamp retrieval and insertion inside the message. This problem degrades significantly the synchronization between network nodes. This paper describes the analysis, implementation, and evaluation of a HW-timestamping technique for minimal-latency timestamping at Gigabit Ethernet using a low-cost off-the-shelf FPGA board. The effectiveness of the method is validated through a point-to-point synchronization scheme achieving a best-case synchronization accuracy of 150 ns.

1. INTRODUCTION
Since the first introduction at early 80s by Kopetz [1] and later by Schmid [2], HW-timestamping technique has become a requirement for achieving high-precision synchronization in many different areas such as industrial environments [3, 4], precision network data analysis and logging [5, 6], and network measurement [7].

In the context of synchronization of networks, HW-timestamping refers to the generation and insertion of a time field in a message by means of dedicated hardware. SW-timestamping alludes to the same concept but the timestamp is accomplished by software instructions running at the upper layers (fig. 1), on top of the jittery scheduling tasks of the operating system. Both dimensions of the mechanism establish the performance division line between the SW-based protocols, such as the Network Time Protocol (NTP)[8], and the HW-based synchronization protocols like the ubiquitous IEEE 1588, the Precision Time Synchronization protocol (PTP) [3].

The HW time of 80x86 μPs is based on a Time Stamp Counter register (known as the TSC register) which is generally implemented with a 64-bit counter running at the μP frequency (fig. 1). Actual HW-timestamping techniques obtain timestamps directly from the TSC register and suffer from inserting it in the current outbound frame. In Gigabit Ethernet the delay between the timestamp reading to the insertion in the message must be below hundred of nanoseconds. The source of this delay is mainly due to the bus arbitration and, in less extend, the jitter of the internal synchronization logic. This limitation degrades the precision of the synchronization between nodes as the time conveyed in the current message is only an estimation.

Fig. 1. Timestamping points in Ethernet node.

In this paper, we duplicate the μP time register and allocate the copy in our custom Timestamp Unit (TSU), located at the bottom part of the MAC just before sending the frame to the GMII interface, the region that delimits the FPGA chip from the on-board Ethernet transceiver (PHY) (fig. 1). This duplicity of clocks benefits from the elimination of the bus delays for outbound synchronization messages as the timestamp is just in time inserted by near dedicated hardware. With the synchronization code running on the μP, we achieve tight synchronization between two peer boards at

*The authors acknowledge the financial support of Generalitat de Catalunya, under grant SGR2005-00767, and Ministerio de Educación y Ciencia of Spain, under grant TSI2005-07293-C02-02.
the TSU level. The synchronization is also extended to the μP level by dumping the TSU time into the μP’s localtime register. Such transfer accounts for the internal bus delays, temporary memory accesses and the high-speed frequency of the destination time register. In the next section we introduce relevant HW-timestamping techniques. Section 3 shows the internals of the TSU and its implementation targeting a Virtex-4 FPGA. Section 4 summarizes the used FPGA logic resources to implement the TSU. In Section 5, we provide metrics to account for the delay of the internal time transfers and verify the low offset error, and hence the high synchronization precision, between the μP and the TSU localtime with a sequence of synchronization messages between two boards.

2. EXISTING SYNCHRONIZATION TECHNIQUES

HW-based synchronization techniques started promisingly at the later 90s within the SynUTC project [9] to support external clock synchronization at Fast Ethernet speeds (100 Mbps). The legacy of SynUTC was the Network Time Interface (NTI)[10] which was an ASIC-based network interface card, consisting of a CPU and an adder-based clock allocated at the Media Dependent Interface (MII). The adder-based clock was an unconventional 56-bit counter that used a high-resolution adder instead of a common counter for summing oscillator ticks. It could be paced at an arbitrary frequency in the range from 1 to 25 MHz, and thus timestamping with a resolution spanning from 1 μs to 40 ns, which was enough for the 100 Mbps Ethernet. All internal blocks of the NTI including the μP where directly wired to the adder-based clock, and thus becoming a single clock architecture around the ASIC TSC clock.

SynUTC’s adder-based clock is actually adopted in some ASIC implementations of the PTP. The Precision Time Protocol is a standard for a precision clock synchronization protocol for local area networked (LAN) measurement and control systems [3]. It is devoted to 100 Mbps Ethernet subnets interconnected via special switches (boundary clocks (BC)). Each PTP node and BC have a timestamp logic at the MII interface card, consisting of a CPU and an adder-based clock allocated at the Media Dependent Interface (MII). The adder-based clock was an unconventional 56-bit counter that used a high-resolution adder instead of a common counter for summing oscillator ticks. It could be paced at an arbitrary frequency in the range from 1 to 25 MHz, and thus timestamping with a resolution spanning from 1 μs to 40 ns, which was enough for the 100 Mbps Ethernet. All internal blocks of the NTI including the μP where directly wired to the adder-based clock, and thus becoming a single clock architecture around the ASIC TSC clock.

In figure 2 is shown our timestamp unit (TSU). It is divided into five major regions. The centerpiece of the TSU is the control adjustment unit (CAU) implemented as a 32-bit wide time register. The time is maintained as a fixed point representation of a 0 integer part and 32 bit fractional part (i.e. the lower part). The integer part is not kept at the TSU, but at the PPC upper part as it is not sent in the message. The upper part increments after the lower part runs over (i.e. every ~14.3 secs). The lower part can be read with a nominal resolution of 3.33 ns, as it is driven by a frequency clock of 300 MHz, the same as the μP.

The other four TSU regions are the transmit engine (tx_engine), the receive engine (rx_engine), the register stack and the bus interface logic. The tx_engine generates synchronization messages (SYNC) by modifying on-the-fly standard PAUSE control frames. The message formats are shown in Fig. 2. The controller of the transmit engine is a Moore machine (FSM_tx) that enables the internal blocks as the outbound PAUSE frame is passing through. First, the pattern recognizer waits for matching the fixed content of the Destination Address (Pause DA) and type fields and asserts hit. At this point it starts replacing the PAUSE opcode (0x0001)
for a SYNC opcode (0x0002). The contents of SYNC num and SYNC oper fields are user-selectable and automatically retrieved from one of the registers of the register stack to be byte-sliced with a shift register (not shown) into the frame. At the time of the insertion of the SYNC num field (i.e. 25th byte), the transmission timestamp is captured from the CAU block, byte-sliced by the shift register (shr) and inserted from the 29th field to the 32th. The frame is zero-padded until a new CRC code is inserted at the tail.

The rx_engine performs the reverse process by transforming the inbound SYNC frames to PAUSE frames. The pattern recognizer asserts hit when detecting the DA field and type field codes. After two clock cycles, two processes start concurrently on the incoming SYNC. On one hand, the retrieval of the message information (i.e. timestamp, time-of-arrival, SYNC num, SYNC oper) and also, its restructuring to a PAUSE. The rx_engine block contains a CRC check block to account for possible line errors of the incoming frame and a CRC generator (CRC gen) block to insert in the modified frame. Once zero padding starts, two time values are retrieved: at the 25th byte, the time of arrival (ToA) of the SYNC frame and, from the 29th field to the 32th, the timestamp of the sender. Inbound and outbound freeze time points must be the same in order to cancel the inbound/outbound frame latencies (which is the case). The register stack is a pool of 16 32-bit data registers that stores the relevant information of the last sent/received SYNC frame (i.e. timestamps, ToA, no. frame, user operand, external user time, actual time), and 1 control register for setting/reading flags. The register stack also contains capture registers used for internal timing test purposes.

The OPB if logic allows single user 32-bit read/write accesses from/to the data register stack, and one-bit read/write from/to the control register.

One of the major challenges of the TSU design is the multiple clock domain scenario. TSU deals with 3 non-related clock frequencies (100, 125, 300 MHz) that can potentially introduce timestamping/capture errors due to metastability errors. Here we adopted a solution for the higher speed domains (e.g. the CAU register) that must cross to a lower clock domain (e.g. the tx_engine). It consists of a sanity log-ic that stretches the small periods of the CAU up to the duration of the longer periods of the data transmission (tx_clk). This solution swallowing in the worst case 3 clocks (∼10 ns) of the CAU reading but prevents totally from unacceptable timestamping read errors.

3.2. Embedded platform

Our TSU has been targeted on a ML403 board which mounts a Virtex-4 (XC4VFX12FF668)[12]. It embeds a PowerPC 405 (PPC) μP and a hardcore media access controller (HARD _TEMAC) that interfaces with a Marvell Alaska 10/100/1000 Mbps Ethernet transceiver (PHY) (fig. 2). We adapted the output synchronization logic of the GMII interface of the
MAC to allocate TSU just before the GMII, between the HARD_TEMAC and the PHY transceiver.

The SYNC messages (SYNCs) are triggered by the synchronization software part running on the PPC. The out-bound synchronization messages are initiated by a PAUSE control frame triggered by the PLB_TEMAC block (asserting pause req), built into the HARD_TEMAC and converted to a SYNC by the transmission logic of the TSU. Before sending a SYNC, the SYNC num. and SYNC oper. fields (fig. 2) are updated by filling the corresponding register of the TSU data register stack.

The incoming SYNCs are first checked against CRC errors by the rx logic block of the TSU. If there is an error, it is signalled by the TSU and the frame is aborted. In absence of line errors and at the end of the SYNC reception, rx logic asserts good crc which is connected to the capture input of the OPB_TMR1 block. The interruption raised by OPB_TMR1, triggers the OPB_INT which in turn asserts a non-critical interruption (intc irq) to the PPC for starting the interruption routine handler. The interruption routine picks the last received information kept at the TSU data register stack and proceeds depending on the value of the received SYNC oper. We defined a set of OPER codes to perform different operations.

The glue logic between the PLB_TEMAC and the HARD_TEMAC allows triggering PAUSEs from the PPC or from a HW loopback. The HW loopback can be enabled by setting the TSU in ‘precise rtt mode’ and allows for automatic reply of a SYNC message upon a non-errored SYNC indication (good crc). This mode will be used for the precise calculation of the link delay (Section 5).

4. IMPLEMENTATION

At the design level, the TSU has been developed with ISE 9.1 SP3 and integrated within the embedded platform with EDK 9.1 SP2. Both phases have been verified through simulations with Modelsim 6.2c. Table 1 summarizes the hardware resource utilization of the TSU after the map phase. Utilized BRAM resources are 0% as the TSU does not contain memory blocks, but individual and independent registers.

<table>
<thead>
<tr>
<th>Module</th>
<th>Regs.( %)</th>
<th>LUT( %)</th>
<th>Slice( %)</th>
<th>BRAM(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSU</td>
<td>382(8 %)</td>
<td>1289(11%)</td>
<td>758(13%)</td>
<td>0(0%)</td>
</tr>
</tbody>
</table>

5. EVALUATION

5.1. Clock duplicity

In many situations it is necessary to internally transfer the time from one register to another. This transfer will suffer from the delay to read the source register, storing the value temporarily, accessing the bus and setting the destination register. These sources of delay prevent from having a perfect coupling of two copies of the same time stored in different registers. In our implementation, each overall delay depends on the intermediate operations performed from the time of the reading to the time of writing to the destination. Eq. 1a identifies the delay components of the transfers for TSU to PPC, and reverse direction in eq. 1b. \( \delta_{LOAD_CAU} \) and \( \delta_{READ_CAU} \) denote the delay to write to the CAU register of the TSU and to read from any register inside the TSU, respectively. Both are drawn from a user-defined functions consisting of few XIO_In32 and XIO_Out32 low-level C macros, already in the PPC driver library. \( \delta_{DDR_{wr}} \) and \( \delta_{DDR_{rd}} \) correspond to the write and read access delays to/from the on-board DDR SDRAM memory (Fig. 2).

\[
\delta_{TSU\rightarrow PPC} = \delta_{READ_CAU} + \delta_{DDR_{wr}} + \delta_{PPC_{rd}} + \delta_{DDR_{rd}} + \delta_{LOAD_CAU} \tag{1a}
\]

\[
\delta_{PPC\rightarrow TSU} = \delta_{PPC_{rd}} + \delta_{DDR_{wr}} + \delta_{DDR_{rd}} + \delta_{LOAD_CAU} \tag{1b}
\]

We have obtained and annotated the statistics of the timing of the time transfers over 100k samples. In all tests the cache memory of the processor has been disabled in order to force external DDR SRAM memory accesses. The obtained metrics are summarized in Table 2. The left column corresponds to the average of the transfers using an exerciser loop in a synthetic program. The right column contains the timing of the transfers measured in the SW synchronization application code, as described in the next section. The distribution functions of both transfers delays are shown in Figure 3.

| Table 2. Number of clock cycles to perform the transfer from PPC to TSU and viceversa. |
|-----------------------------------------------|---------------|
| DDR SDRAM | loop | int. handler |
| Ave. | Std. Dev. | Max. |
| \( \delta_{PPC\rightarrow TSU} \) | 1688 | 1797 | 20 | 1822 |
| \( \delta_{TSU\rightarrow PPC} \) | 941 | 1005 | 19 | 1047 |

In the next section, we will assess the clock duplicity at each node by comparing the offset error between the PPC and the TSU time registers. We will use the metrics in Table 2 to accurately synchronize the two clocks internally at particular time instants.
5.2. Clock Synchronization

This section proposes a clock synchronization method to synchronize two nodes in a direct link configuration. Our goal is to measure and to adjust the clock offset between each pair of clocks of two nodes (master-slave) in order to synchronize them and measure the accuracy of the clock duplicity. The perfect synchronization occurs when the error between TSU’s localtime and between PPC’s localtime is null. But as discussed in previous sections, there are in-accuracies due to unavoidable constraints. We achieve our goal by using a synchronization mechanism based on precise measurements of the round trip time between the two nodes. First, the clock offset between TSU’s is compensated, and then, utilizing the correction factors of the table 2, the already slave’s TSU adjusted time is transferred up to the PPC to set the time at the μP level. As shown in figure 4, the mechanism is divided in three steps.

1) Internal synchronization, where each node transfers at boot time the localtime of the PPC to the TSU applying the corresponding correction factor (δ_{PPC→TSU}).

2) Calculation of the link propagation time, δ_l, where we first synchronize at the TSU level, namely from master’s TSU to slave’s TSU. The master node sends a SYNC message (t_{TSU_{ms}}) and the slave sets its localtime according to the timestamp inside the message (t_{TSU_{slv}} = t_{TSU_{ms}}). Then, the slave replies immediately through the wired loopback asserted with prec_rtt (Fig. 2). The delay from prec_rtt assertion to the generation of the SYNC (ε_0) is fixed (125ns). As explained in Section 3, the tx_engine logic freezes the timestamp after ε_1 (200ns). The slave’s reply conveys a timestamp of value t_{TSU_{slv}}. The master TSU snapshots the time-of-arrival (t_{TSU_{ms}}) and pass it to the PPC to start the calculation of δ_l (eq. 2).

\[ δ_l = \frac{t^2_{TSU_{ms}} - t^1_{TSU_{ms}}}{2} - (ε_0 + ε_1) \] (2)
Thereafter, $\delta_l$ is sent to the slave node inside a SYNC.

3) **Peer synchronization**, where the current slave’s local-time of the TSU and the PPC is modified by adding $\delta_l$. As the TSU does not perform arithmetic calculations, the correction of the local-time on the slave’s side is also executed by the SW synchronization part on the PPC. First, the TSU is adjusted by the PPC while applying the correction factor and the link propagation time, i.e., $t_{TSU_{slave}}^{TSU} = t_{PPC_{slave}}^{PPC} + \delta_l + \delta_{PPC\rightarrow TSU}$. Finally, TSU’s time is transferred up to the PPC with the other correction factor, $t_{PPC_{slave}}^{PPC} = t_{TSU_{slave}}^{TSU} + \delta_{TSU\rightarrow PPC}$.

The three steps above take less than 25 $\mu$s to accomplish and are restarted at intervals of 12.5 s. During the periods of no re-synchronization, we monitor the performance of the four clocks, i.e., $t_{TSU_{master}}^{TSU}$, $t_{TSU_{slave}}^{TSU}$, $t_{PPC_{master}}^{PPC}$, $t_{PPC_{slave}}^{PPC}$, at intervals of 500 ms. Figure 4 shows a 50 s. window of the time series of the offset between PPC’s and between TSU’s. Each sawtooth starts with a null offset and increments until the next re-synchronization event, namely every 12.5 s. The slope of the offset reveals the drift difference between the crystal oscillators of the two boards. They drift apart 12k $\mu$P clock, and thus bypassing the arbitration delay and jitter of the buses. We verified our concept in a point-to-point synchronization mechanism achieving a best synchronization accuracy of less than 50 clock cycles between timestamping units and $\mu$P’s. Embedding this clock solution in an Ethernet system it accomplishes a best-case 150 ns and 167 ns on-average link synchronization between contiguous nodes.

## 6. CONCLUSION

We presented a HW-timestamping unit for Gigabit Ethernet on an off-the-shelf Virtex-4 board based on the new approach of duplicating the $\mu$P Time Stamp Counter register. We allocate the mirrored clock/register at the GMII interface. To achieve an accurate copy of the clock, we measure the additional time offset to read/write to/from one clock to another. The low error between the two clocks allows to perform HW-timestamping without the necessity to convey the $\mu$P clock, and thus bypassing the arbitration delay and jitter of the buses. We verified our concept in a point-to-point synchronization mechanism achieving a best synchronization accuracy of less than 50 clock cycles between timestamping units and $\mu$P’s. Embedding this clock solution in an Ethernet system it accomplishes a best-case 150 ns and 167 ns on-average link synchronization between contiguous nodes.

## 7. REFERENCES


