Time Synchronization for an Asynchronous Embedded CAN Network on a Multi-Processor System on Chip

Gabriela Breaban*, Sander Stuijk*, Kees Goossens*†

*Eindhoven University of Technology, The Netherlands
{g.breaban,s.stuijk,k.g.w.goossens}@tue.nl

†Topic Embedded Products, The Netherlands

***Abstract*—Distributed cyber-physical systems cover a wide range of applications such as automotive, avionic or industrial automation. These applications require a global notion of time to fullfill their timing requirements. Multi-processor system on chips (MPSOCs) are an attractive implementation option since they offer several benefits such as parallelism and power efficiency. However, MPSOCs have a Globally Asynchronous Locally Synchronous (GALS) architecture in which different processors are driven by independent clocks. Hence, to obtain a global notion of time on a MPSOC, a clock synchronization solution is required.**

In this paper we propose a time synchronization technique for a Controller Area Network (CAN) network implemented on an asynchronous MPSOC that offers a trade-off between HW/SW cost and time synchronization performance. We evaluate our method on a FPGA platform and show that it can achieve a minimum accuracy of 860 ns and a precision of minimum 2 μs.

I. Introduction

Distributed cyber-physical systems (DCPS) are a set of interconnected processing devices that closely interact with the physical environment in a distributed manner. They cover a wide range of applications such as automotive, avionic, industrial automation etc. These applications are time-aware, meaning that they are required to execute their functionality in a timely fashion, and often, also require a common notion of time [13]. Both the time notion and timeliness are needed to provide the necessary quality to the user. While a timely execution ensures that tasks are completed within a given deadline and that the communication latencies are deterministic, a common notion of time allows for collaborative actions (e.g. synchronous actuation/sampling, sensor data fusion) and leads to a consistent view on the occurance order of global system events. A common notion of time brings the need for time synchronization.

While DCPS can be implemented by interconnecting multiple uniprocessor chips, more recently, Multi-Processor Systems on Chip (MPSoCs) proved to be a good alternative as they offer various benefits such as power efficiency, reliability, resource management. Nowadays MPSoCs are GALS with multiple clock domains that inherently suffer from drift [14]. Hence for implementing time-aware applications on such a system, a time synchronization technique has to be in place.

The Precision Time Protocol (PTP) is a state of the art time synchronization technology that has been applied and standardized for a multitude of communication networks. While a significant amount of the related research work focuses on Ethernet and wireless [12], recent extensions of automotive standards such as AUTOSAR release 4.2.2 spotlight the need to implement time synchronization on domain-specific networks [1]. CAN is a mature automotive network. It has a standard defined bit synchronization mechanism that can be exploited to obtain clock synchronization [9]. Nonetheless, the arbitration mechanism of CAN causes latency jitter. Time Triggered CAN (TTCAN) was introduced to address this problem [6]. It offers exclusive time windows for time triggered and event triggered messages and it achieves clock synchronization via the HW correction of clock drift done in the dedicated TTCAN controllers.

A time synchronization solution can be assessed by quantifying a set of relevant properties such as tightness, cost-effectiveness or reliability [11]. In this paper we propose a time synchronization technique for a CAN network implemented on an GALS Multi-Processor System on Chip (MPSoC) that, next to a NOC, contains an on-chip CAN bus between processors. Our technique offers a trade-off between HW/SW cost and time synchronization performance and gives the possibility of integrating existing CAN-based applications that require a global notion of time without changing their code. The time synchronization protocol involves a minimum communication overhead of one message per synchronization round.

The remainder of the paper is organized as follows: Section II gives an overview of the related work, Section III presents the CAN network design method for a MPSoC, Section IV explains the time synchronization protocol for CAN, Section V discusses the experimental evaluation of the proposed solution and finally Section VI concludes.

II. Related Work

Watwe et al. offer a time synchronization technique for wireless sensor networks [15]. It improves energy efficiency and uses a TDM-based (Time Division Multiplexing) channel access to avoid message collision. Our method also makes use of TDM, but the purpose is to allow multiple applications to access the processor during one CAN bit period in a time isolated manner.

Lim et al. evaluate in [7] the IEEE 802.1AS time synchronization standard for switched Ethernet in in-car networks and show that the accuracy required by the standard (1 μs) can be obtained with a synchronization interval of maximum 125 ms. Our work uses a different network, namely the CAN, on top of which a simplified version of the AUTOSAR time synchronization protocol is implemented and the main advantages are the reduced required bandwidth (4 orders of magnitude lower) and the larger synchronization interval (1 s).

Rodrigues et al. propose in [9] a new fault tolerant software algorithm that exploits the unique properties of the CAN protocol related to reliability and tightness. It can achieve a synchronization precision of maximum 50μs, but it has the side effect of causing an additional accuracy loss of several ms/hour. Our hybrid HW/SW solution improves the synchronization precision while minimizing the HW/SW cost, but it does not address faults.

Rodriguez-Navas et al. offer in [10] a HW solution that extends a CAN node with a dedicated module called clock unit. The clock unit consists of an enhanced CAN controller and a synchronization submodule. The synchronization submodule generates a synchronized clock. Our solution also uses a HW time stamp mechanism and a simillar synchronization protocol consisting of a single CAN message but it has a significantly lower HW cost. The HW cost of our technique per CAN node is given by the PHY layer implementation of the CAN controller, while the one proposed by Rodriguez-Navas et al. adds the CAN MAC layer and the clock unit described above.

III. Designing an Embedded CAN Network on a GALS MPSoC

Our solution for implementing a CAN network on a MPSoC consists of a hybrid HW/SW design for the CAN controllers: the physical layer as specified by the standard [3] is realized in hardware, while the Media Access Control (MAC) layer version 2.0A is implemented in software as specified by the ISO11989 standard [2]. The hardware PHY module can be instantiated for each processor so that the software running on the processor can access the CAN PHY layer. The HW/SW architecture concept was initially proposed in [5] for a MPSoC platform using a single clock oscillator, in which a simplified CAN PHY design was used. The following subsections present the updated architecture that includes a standard compliant CAN PHY design and accounts for clock drift in the software.

A. MPSoC Hardware/Software Architecture

Figure 1 shows the hardware architecture of the multi-processor GALS platform. It consists of three tiles interconnected by the CAN network. There are two independent clock sources provided by separate crystals, CK 1 and CK 2. CK 1 runs at 100 MHz and provides the clock signal for tiles 1 and 2, while CK 2 runs at 64 MHz and generates the clock for tile 3. Each tile comprises a Microblaze processor, its instruction and data memory (IMEM, DMEM), a Timer Interrupt and Frequency Unit (TIFU) and the CAN PHY. The CAN PHY module implements the physical layer of the CAN protocol. Its interface consists of a TX output and a RX input. The RX is the result of a wired-AND of all the TX outputs coming from the CAN PHY instances on the platform. The processor communicates with the CAN PHY via an AXI4LITE interface to access the TX/RX registers.

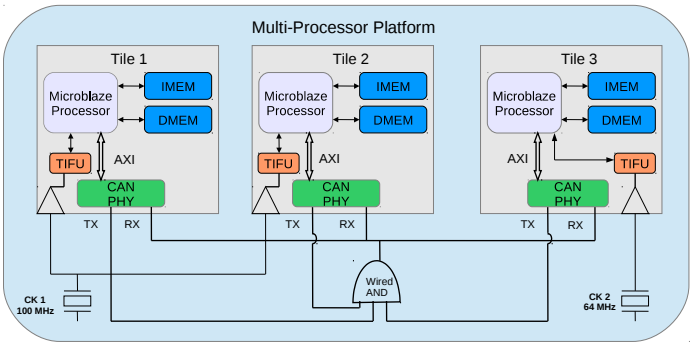

Fig. 1. Multi-Processor Tile-based Architecture

To accommodate a set of individual applications that communicate via CAN on the MPSoC platform, we use fine-grained precise Time Division Multiplexing (TDM) in software to offer independent processing resources to all the applications running on each processor together with the CAN MAC layer. This is realized by the CoMik microkernel [8]. When a certain processor requires access to the CAN bus, at least one TDM slot has to be allocated for the CAN MAC. The MAC layer is responsible for creating or receiving the bit stream of each CAN frame and it monitors the bus to detect its state. For this, it needs to access the bus at least once per bit period. Thus the attainable CAN bit period depends on the frequency of the TDM slots allocated to the MAC layer.

When integrating a set of individual applications that communicate via CAN on the MPSoC platform, we need to consider several design parameters, such as the number of applications sharing a processor and the minimum required CAN bit rate. For an extensive explanation of the design space exploration for CAN, we refer the reader to [5].

To illustrate the concepts presented above, we show the CAN controller design in Figure 2. At the bottom, the CAN PHY hardware implements the bit timing logic according to the standard. The CAN bit period is made of 3 time segments: SYNC, SEG 1 and SEG 2. The bus is driven with the new bit value at the end of SEG 2 (TX point) and the value of the current bit is sampled at the end of SEG 1 (sampling point). We will explain further the role of these segments in the following subsection III-B. In software, the CoMik microkernel depicted as MK in each of the three TDM slots

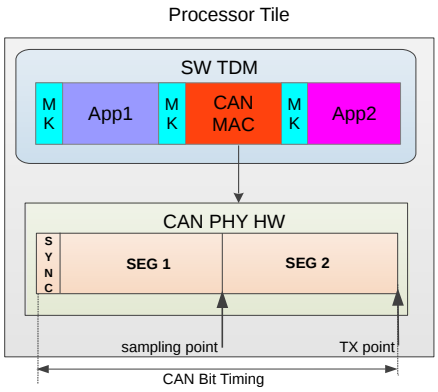

Fig. 2. HW-SW Bit Timing for the CAN Controller

schedules each application in its corresponding TDM slot and performs the context switch. Each TDM slot has a fixed-duration and it is divided into a CoMik sub-slot and an application sub-slot. The TDM table has three slots, two of them allocated to applications 1 and 2, respectively, and one to the CAN MAC layer. Both applications use the CAN controller running on tile 1, shown in Figure 1. In the CAN slot, the value of the last sampled bit is read and the value of the next bit to be transmitted is provided to the CAN PHY HW. In this case, the TDM table width is equal to the CAN bit period, which results in a relatively low bit rate in the range of several kbits/s. To increase the bit rate, the TDM slot assignment has to be changed and more slots must be allocated to the CAN such that it can keep up with a higher bit rate [5].

B. Accounting for Clock Drift in the HW-SW Design

The clock drift is compensated first in hardware (PHY) and then propagated in software (MAC). We will explain how this is achieved in each layer.

In HW, at the PHY layer, the clock drift is tolerated through the quasi-static timing of the bit segments. The duration of the bit segments can be configured at design time depending on the number of applications running on the processor and their communication requirements. The duration of each segment is defined as a multiple of the basic time unit called *time quantum* (TQ). The duration of TQ has to be equal for the CAN controllers in the network. The SYNC segment has a fixed duration of one time quantum and it is the time during which a new bit value is expected (see Figure 3(a) circle 1). SEG 1 and SEG 2 are used to compensate for signal propagation time and phase errors caused by oscillator drifts, and their duration can be adjusted at run time to correct the phase error. Every time a negative edge (a transition from a '1' bit to a '0' bit) is transmitted at run time, SEG 1 or SEG 2 is shortened or lenghtened such that all the controllers align their bit sampling point. The maximum phase error correction is called *synchronization jump width* which is a design-time parameter. Figure 3(a) illustrates the effects of the clock drift in the platform shown in Figure 1. The CAN bit segments on tiles 1 and 3 are initially misaligned. Then tile 3 starts driving the bus and as a result the bit timing restarts on tile 1 and in consequnce the sampling point is aligned on the two tiles.

In software, the CAN MAC layer has to follow the PHY layer. For this, the software alingns the sampling point in the CAN PHY with a fixed point inside the CAN TDM slot, as seen in Figure 3(b) for the second sampling point. The software alingnment is done by decreasing or increasing the duration of the CoMik sub-slots. The application sub-slots keep a constant duration but they will shift to the left or right as a result of the CoMik slot adjustment. The CoMik adjustment is equal to the phase compensation done in the CAN PHY and it is computed based on the hardware timestamp taken at the TX point. This can be seen in Figure 3(b) that illustrates 3 successive TDM periods corresponding to 3 CAN bit periods. In the first bit period, we can see that the CAN bit sampling points of the two tiles have a relative offset equal to R (circle 2), which is also reflected in the offset between the TDM tables (shown above). In the second bit period, tile 2 sends a negative edge causing the CAN PHY modules to synchronize and as a result the bit time for the second bit starts at the same moment on both tiles. Tile 1 takes a hardware timestamp TS of the negative edge which is then read in the CoMik slot (MK) preceding the application sub-slot App1 (circle 3). The offset R is computed in the CoMik slot as follows:

$$\begin{aligned} Mk_{start_{i+1}} &= TS + SEG_1 - OFS - C \\ R &= Mk_{start_{i+1}} - Mk_{start_i} - (C + P) \end{aligned} \tag{1}$$

where Mk_{start_i} represents the time when CoMik slot i starts, SEG_1 is the value of the design parameter SEG 1 of the CAN bit, C is the CoMik sub-slot duration, P is the application sub-slot duration and OFS is the CAN bit sampling point offset with respect to the application sub-slot start time. All these parameters are measured in clock cycles.

As a result of correcting the offset, the duration of the CoMik sub-slot preceding the CAN slot becomes C-R. The offset correction is applied once, after which the CoMik duration is restored to its design time value, C.

While a positive offset results in an increase of the CoMik slot, which has no upper bound, a negative offset signaling a decrease of the CoMik slot does have a lower bound. This lower bound is determined by the minimum duration that CoMik needs to perform the context switch and to update the TDM schedule. This is established by the implementation and it is equal to C_{min}. Thus the applied offset correction R has to satisfy the following inequality:

$$C - R \geq C_{min} \tag{2}$$

During normal operation, CoMik ensures temporal isolation between applications by starting each application sub-slot with a precise period of (C+P) clock cycles and then the application is swapped out precisely P cycles later. Thus no application can interfere with any of the other applications running on the same processor, a property that we call composability. The offset correction does not alter the amount of cycles received by each application, P, but it impacts the absolute moment

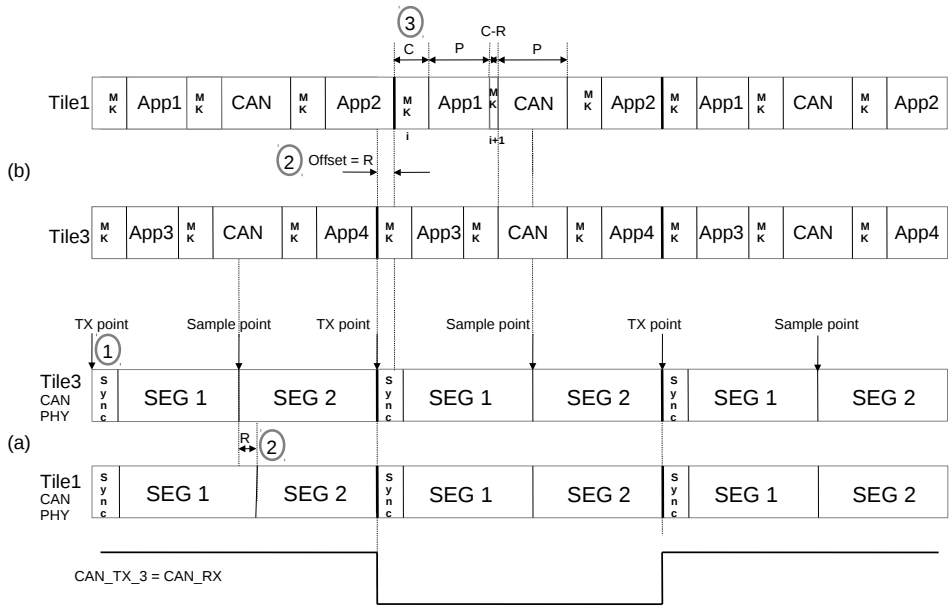

Fig. 3. Alignment of TDM slots based on CAN PHY Layer

in time when the application sub-slot starts, such that the application sub-slots become aligned among the processors.

IV. Time Synchronization on CAN

The time synchronization protocol for our CAN implementation requires a single message sent by the time master to the time slave(s). While the standardized PTP protocol [4] requires 4 messages for each synchronization round (Sync, Follow-Up, Delay Request and Delay Response), due to the favorable timing properties of the CAN protocol, they can be reduced to the first two messages. Furthermore, our specific design that implements the MAC layer in software makes it possible to use only a single Sync message.

The structure of the CAN bit timing can easily be exploited to obtain precise time synchronization [9]. First, a highly precise time synchronization requires a small variation of the network physical propagation delay. By properly configuring the CAN bit segments to account for signal propagation and phase errors, this variation can be bounded to a small percentage of the bit time. Second, the upper MAC layer features arbitration for collision resolution and includes an acknowledgement field driven by the slave which render the link delay detection redundant. With no link delay detection needed, the time synchronization protocol on CAN comes down to the Sync and the Follow-Up message.

The Sync message signals the start of the time synchronization and the Follow-Up includes the master time stamp, taken as close as possible to the physical layer. Both messages are required when the entire CAN protocol stack is implemented in hardware and the software interacts with it at the frame level. Our design however implements the MAC layer in software which offers the advantage of a bit level interaction with the hardware. This gives the possibility of modifying the content of a frame while it is being sent. The time master can then easily insert the time stamp in the data field after it wins arbitration. Hence the Follow-Up message is no longer needed.

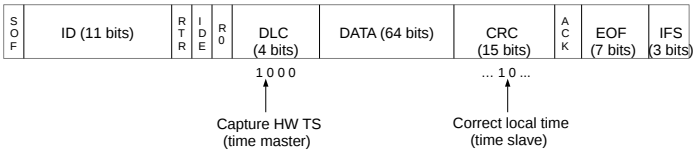

Fig. 4. Time Synchronization Frame Structure

Figure 4 shows the structure of the time synchronization CAN frame. We use a 64-bit hardware time stamp which is captured at the beginning of SEG 1 of every bit. This creates a payload of 8 bytes (DLC = ‘1000’). After winning the arbitration, the time master reads the time stamp after the first bit of the DLC field is sent, as seen in the figure and it inserts it in the following DATA field. At the slave side, after it detects the time synchronization message ID, it counts the total number of bits N transmitted between the first DLC bit, when the master took the time stamp, and the first negative edge (a ‘1’ to ‘0’ bit transition) following the DATA field and corrects its local time. This total number of bits can vary due to bit stuffing. Two important choices are made here: the one of reading the time stamp after the arbitration finished on the master side and that of applying the time correction at a negative edge at the slave side. Both have the role of maximizing the time synchronization accuracy. After the arbitration finishes, during the DLC and DATA fields, the CAN bus is solely driven by the time master, which is the only one dictating the bit timing in this interval. This makes it possible for the slave to precisely predict the master’s local time by adding the duration of the N counted bits to the

received timestamp. The obtained new time is given by the following equation and it is expressed in clock cycles:

$$t_{new} = TS + N \cdot BitTime_M \quad (3)$$

where $BitTime_M$ is the CAN bit length at the master expressed in clock cycles.

Our time synchronization mechanism aims at establishing a global notion of time in the multi-core system and can be evaluated in terms of accuracy and precision. The accuracy is defined as the difference between the global time value shown by the slave and the global time value shown by the master at the same point in absolute time. The precision is the maximum variation of the global time value shown by the slave at the same absolute or relative point in time. To maximize the accuracy when the time synchronization occurs, we chose to apply the correction on a negative edge, which enforces the synchronization in the CAN PHY layer between the master and the slave.

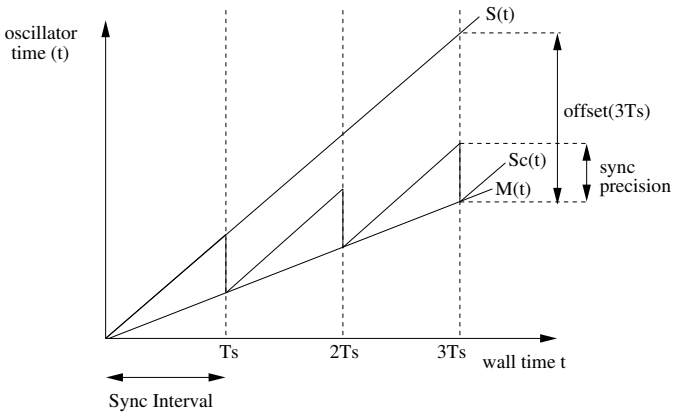

Fig. 5. Time Offset Correction

Figure 5 illustrates the time offset concept and the effects of time synchronization. The oscillator drift is a linear function of time. The figure shows two drifting clocks, a master clock M(t) and a slave clock S(t) and the corrected slave clock $S_c(t)$. As time advances, the offset between clock M and clock S, defined as the difference between the shown time values at a given moment t, M(t) - S(t), is linearly increasing. To correct this increasing offset, we apply the time synchronization with a period T_s. When the time correction is applied, the slave shows the same time as the master, after which it continues counting with its own rate. The accumulated offset in between two synchronization rounds depends on the relative drift rate between the master and the slave. In our method, the time correction is done in software, the hardware clock is not affected.

One important final remark regarding our implementation has to be made. The alignment of the TDM CAN slots with the CAN PHY is realized independently of the time synchronization between the time master and the time slave(s) and it is required to maintain a correctly functioning CAN bus on all tiles. In the same time, as stated in the previous section, the correction of the CoMik sub-slot has a lower bound, C_{min}. For this bound not to be violated, the traffic on CAN has to be frequent enough such that when a transmission starts, the relative offset between the transmitter and the receivers is lower than C_{min}. This can be realized by analyzing the CAN traffic generated by the applications and inserting, if needed, dummy messages to keep the offset in the required bound.

In conclusion, the time synchronization in our method is performed at three levels: at the CAN bit level, at the TDM slot level and at the time synchronization protocol level.

V. Experiments

We implemented our method on the Xilinx ML605 FPGA board. The HW platform comprises 3 processor tiles: tile 1 and tile 2 are connected to an onchip crystal oscillator running at 100 MHz, while tile 3 is connected to an offchip oscillator running at 64 MHz. The platform architecture is the same as the one shown in Figure 1. The relative drift rate of the two crystals is 15.6 μs/s. The CAN PHY HW was configured to have a bit period of 240 μs, a time quantum TQ of 250 ns and a synchronization jump width of 8 μs. The design of the PHY module follows the Bosch standard and has a single additional feature: a 64-bit HW timestamp in the tile clock domain, which is updated in every CAN bit period at the beginning of SEG 1.

For the time synchronization, we select tile 1 to be the master and tile 3 the slave. The synchronization interval is equal to 1 s.

In software, every processor executes a TDM table made of 3 slots, as shown in Figure 2. The TDM table duration is equal to the CAN bit duration, 240 μs, and it is the same for all the processors. The microkernel sub-slot was dimensioned based on the minimum required duration, C_{min} (see Section III-B) plus the amount required for the alignment with the CAN PHY, R (see Section III-B). The value of R depends on the relative drift rate and on the time synchronzation interval. Given that we perform the time synchronization every second and that the relative drift rate is 15.6 μs/s, it follows that the accumulated offset in 1 s is 15.6 μs. The time slave has to be able to adjust the TDM alignment according to this value and thus the microkernel sub-slot was overdimensioned with ~1000 cycles (15.6 μs).

To evaluate the synchronization tightness, we use two GPIO wires and toggle the signal value at predefined points in time at the slave and master side. The slave and master print their current global time value at the moment that the signal is toggled. The global time format is the one given by the master timer, which runs at 100 MHz and thus is incremented every 10 ns. The software ran for 20 minutes and it collected 1205 time values for the master and slave, respectively, corresponding to 1205 synchronization intervals.

A first measurement was done at the moment when the time slave corrects its global time according to the value received from the master. The difference between the global time values printed by the master and slave at this moment varies between 71 and 86 master clock cycles. The slave receives the timestamp taken by the master as explained in

Section IV, estimates the current time at the master based on the timestamp and on the time elapsed since the timestamp was taken and corrects its value of global time. The master estimates the computation delay at the slave side, waits for this amount of time and then prints the value of its global time as close as possible to the slave. The difference of 71 to 86 master clock cycles (710 to 860 ns) mentioned above comes thus from the estimation error and represents the accuracy. Figure 7 shows the empirical cummulative distribution for the values obtained during the 20 minutes run. It can be seen that the majority of the values (about 70%) are equal to 71 and the remaining ones cover the 71 to 86 range. Figure 6 shows the pulse edges triggered by the master and slave. The upper signal (in yellow) comes from the slave and the lower one (in blue) from the master. The measurement on the oscilloscope was triggered on the master pulse edge and the slave edges were displayed with infinite persistency. It can be noticed that there is a jitter of about 552 ns (displayed in the upper left corner as ΔT) that characterizes the precision of our approach. This is composed by the HW jitter at the slave and the SW polling jitter at the slave and master. The HW jitter is given by the time quanta (250 ns) that gives granularity of the HW timestamp. The SW polling jitter is the given by the instructions used to wait for a new CAN bit sampling point and it is around 10 cycles measured in the tile clock domain.

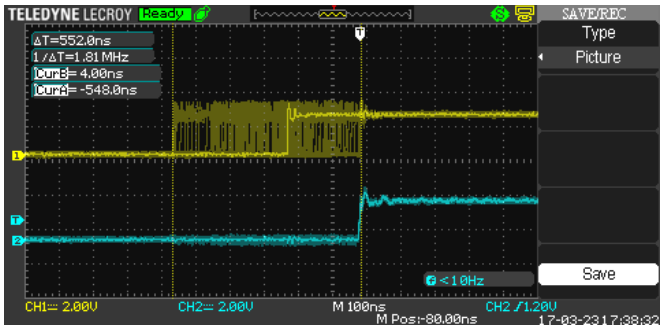

Fig. 6. Accuracy at synchronization time

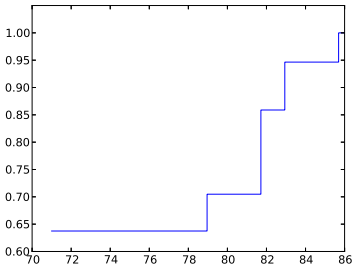

Fig. 7. Empirical CDF for the accuracy

A second measurement was done at an intermediate value within a synchronization interval, namely 405 ms after synchronization. For this, both the master and the slave wait for 405 ms as measured by their local timer after which the master prints its global time and the slave computes the global time and prints it. The difference between the global time values printed by the master and slave (the accuracy) varies between 62 and 68 master clock cycles (620 to 680 ns). It can be easily seen that it is simillar to the one obtained in the previous measurement, as it is influenced by the same factors. The precision in this case is about 1.96 μs. This value comes from the timer software polling and the computation of the global time at the slave side.

VI. Conclusions

In this paper we propose a time synchronization technique for an embedded CAN network implemented on an GALS MPSOC. The CAN controller has the physical layer implemented in HW and the MAC layer in SW, time sharing the processor with other applications using TDMA. First, the TDM slot allocated to the MAC layer is aligned to the PHY layer. Next, a global notion of time is obtained with a single message time synchronization protocol between the master and the slave. The experiments show that for 100 MHz and 64 MHz clocks our solution has a minimum accuracy of 860 ns and a precision of maximum 2 μs.

References

[1] "AUTOSAR release 4.2.2 - Specification of Time Synchronization over CAN," Tech. Rep.

[2] "ISO11989-1:2015 Road Vehicles – Controller Area Network (CAN) – Part 1: Data link layer and physical signalling," Tech. Rep.

[3] "R.Bosch CAN Specification – Version 2.0," Tech. Rep., 1991.

[4] "IEEE standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, 2008.

[5] G. Breaban *et al.*, "Virtualization and Eemulation of a CAN device on a Multi-Processor System on Chip," in *MECO*, 2016.

[6] W. Elmenreich, "Time-triggered fieldbus networks state of the art and future applications," in *ISORC*, May 2008, pp. 436–442.

[7] H. T. Lim *et al.*, "IEEE 802.1AS Time Synchronization in a Switched Ethernet based In-Car Network," in *VNC*, Nov 2011, pp. 147–154.

[8] A. Nelson *et al.*, "CoMik: A predictable and cycle-accurately composable real-time microkernel," in *DATE*, 2014.

[9] L. Rodrigues *et al.*, "Fault-Tolerant Clock Synchronization in CAN," in *RTSS*, 1998.

[10] G. Rodriguez-Navas *et al.*, "Orthogonal, Fault-Tolerant, and High-Precision Clock Synchronization for the Controller Area Network," *IEEE Transactions on Industrial Informatics*, vol. 4, no. 2, 2008.

[11] G. Rodriguez-Navas and J. Proenza, "Clock Synchronization in CAN Distributed Embedded Systems," in *RTN*, 2004.

[12] I. Shkliarevskyi *et al.*, "IEEE 1588 Protocol Profiles' Comparative Analysis According to Different Applications and Standards," in *ISPCS*, 2016.

[13] A. Shrivastava *et al.*, "Time in Cyber-Physical Systems," in *CODES+ISSS*, 2016.

[14] S. F. Smith, "A Multiple-clock-domain Bus Architecture Using Asynchronous FIFOs as Elastic Elements," Ph.D. dissertation, Moscow, ID, USA, 2003, AAI3110345.

[15] S. Watwe and R. C. Hansdah, "Improving the energy efficiency of a clock synchronization protocol for wsns using a tdma-based mac protocol," in *AINA*, March 2015, pp. 231–238.