

# Clock Synchronization for Networked Control Systems Using Low-Cost Microcontrollers

Pau Martí, Manel Velasco, Camilo Lozoya and Josep M. Fuertes  
Automatic Control Department, Technical University of Catalonia

Research Report: ESAII-RR-08-02

April 2008

## Abstract

This report details the implementation of the IEEE 1158 clock synchronization protocol for Networked Control Systems (NCS). NCS can be implemented using control algorithms that operate on absolute time measurements taken in the distributed nodes. Therefore, clock synchronization becomes an issue.

The IEEE 1588 precision time protocol (PTP) provides a standard method to synchronize devices on a network with sub-microsecond precision. The protocol synchronizes slave clocks to a master clock ensuring that events and time-stamps in all devices use the same time base.

The implementation we present targets nodes build using low-cost microcontrollers. These type of microcontrollers usually work with low-frequency clocks that have a poor time resolution, typically running at the millisecond range, with a periodic tick of 1 ms or greater when they are enabled with a real-time kernel. Considering this limitation in nodes clocks, the targeted synchronization precision for our PTP implementation is of 1 ms. This precision, although being far away from the sub-microsecond precision achievable by this protocol, is in many cases enough for the control purposes.

Rather than using specific hardware support for the PTP implementation (such as message detectors and time stamps units), the solution here described is based on software interrupts.

In general messages generated or received to implement the protocol can be delayed in an unpredictable way by different factors (different software execution times, resources consumed by other communication processes, scheduling and bus arbitration, etc). We show that the application of the Controller Area Network (CAN) as a networking infrastructure can remove some of this unpredictability (which usually is done by hardware assistance).

**Keywords:** Clock synchronization, networked control systems, low-cost microcontrollers, controller area network

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Overview of IEEE 1158</b>	<b>5</b>
<b>3</b>	<b>Implementation</b>	<b>8</b>
<b>4</b>	<b>CAN overview</b>	<b>10</b>
<b>5</b>	<b>NCS timing requirements</b>	<b>11</b>
<b>6</b>	<b>Results</b>	<b>11</b>
6.1	Setup . . . . .	12
6.2	Implementation results for pure IEEE 1588 . . . . .	13
6.3	Alternative implementation . . . . .	14
<b>7</b>	<b>Network scheduling issues</b>	<b>16</b>
<b>8</b>	<b>Conclusions</b>	<b>17</b>

# 1 Introduction

Control loops that are closed over communication networks are known as networked control systems (NCS) [2]. In these systems, control nodes such as sensors, controllers and actuators exchange the physically distributed control data through networks. The network is a fundamental component supporting all the interactions among the nodes. The purpose of the network is to deliver control messages in a reliable, secure, efficient and timeliness fashion. For NCS, timeliness becomes the most critical aspect because (varying) time delays prevent the successful operation of the networked control loops [12].

Some approaches to control system design that can be applied to NCS demand accurate clock synchronization among the networked nodes because the resulting control algorithms base their operation on absolute time measurements. For example, [10] proposed, for CPU-based systems, an approach to embedded control systems design where the underlying control method is synchronized by the actuation instants, and control signals are computed using absolute time measures. For CPU systems, a simple clock can provide the correct timing required. However, for NCS with distributed nodes, the application of those results require precise clock synchronization between nodes. Other approaches like [9] or [5] also require at some degree clock synchronization due to the time-stamping of the control messages.

Motivated by these observations, in this work we report our experiences in implementing the IEEE Standard Precision Time Protocol (PTP) IEEE1588 [6] as a means for clock synchronization. IEEE 1588 addresses the clock synchronization requirements of measurement and control systems. These systems have a number of requirements that must be met by a clock synchronization technology. In particular:

- Timing accuracies are often in the sub-microsecond range,
- These technologies must be available on a range of networking technologies including Ethernet but also other technologies found in industrial automation and similar industries,
- A minimum of administration is highly desirable,
- The technology must be capable of implementation on low cost and low-end devices,
- The required network and computing resources should be minimal.

These requirements could also be met by other protocols such as the SynUTC protocol [3] (see [4] for an comparative study of both protocols). However, we have focused on IEEE 1588 because it is the most accepted synchronization protocol in the market.

The implementation we present targets nodes build on low-cost microcontrollers such as Microchip<sup>1</sup> PIC16F or PIC18F families. These type of nodes usually work with low-frequency clocks (such as 20MHz frequency clocks) that have a poor time resolution, typically running at the millisecond range to not overload their operation with time management. For example, in [8] a real-time kernel targeting these microcontrollers has been presented, where the time management is based on a periodic tick that can be configured from  $2ms$  to  $65534ms$ , and periods, offsets and deadlines are expressed as integer multiples of the tick duration. Considering these type of configurations, the targeted synchronization precision for our PTP implementation is of 1 ms. This precision, although being far away from the sub-microsecond precision achievable by this protocol, is in many cases enough for the control purposes.

IEEE 1588 has been conceived for multi-cast networks, and specifically for Ethernet. However, since the Controller Area Network (CAN [1]) has been shown [11] to have interesting properties for NCS, our implementation has been done in CAN. In addition, the characteristics of CAN permit to solve some of the timing unpredictability that the software implementation of the protocol could introduce.

## 2 Overview of IEEE 1158

The objective of IEEE 1588 is defined in the 'Scope' section of the Project Authorization Request approved by the Standard Board of the IEEE as follows.

*"This standard defines a protocol enabling precise synchronization of clocks in measurement and control systems implemented with technologies such as network communication, local computing and distributed objects. The protocol will be applicable to systems communicating by local area networks supporting multi-cast messaging including but not limited to Ethernet. The protocol will enable heterogeneous systems that include clocks of various inherent precision, resolution and stability to synchronize. The protocol will support system-wide synchronization accuracy in the sub-microsecond range with minimal network and local clock computing resources. The default be-*

---

<sup>1</sup>[www.microchip.com](http://www.microchip.com).

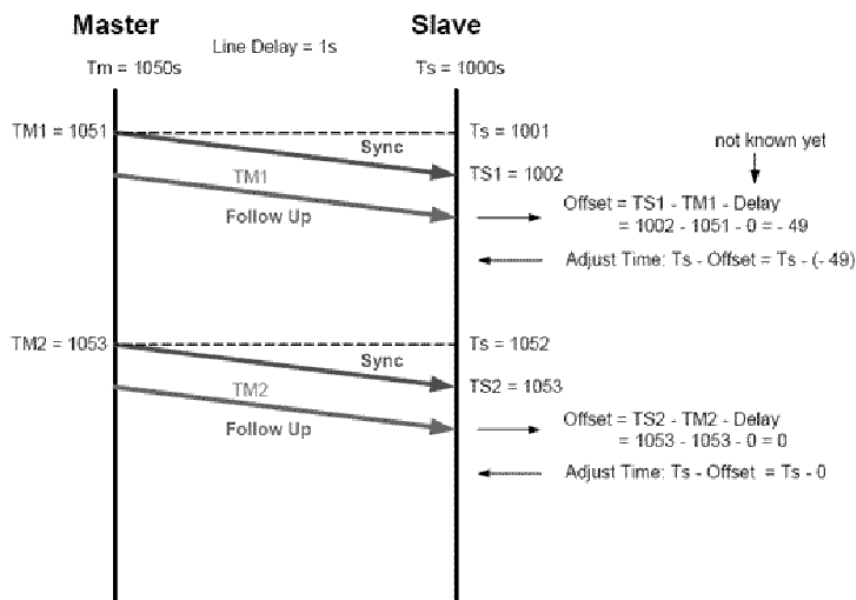


Figure 1: Offset correction

*havior of the protocol will allow simple systems to be installed and operated without requiring the administrative attention of users.”*

The synchronization is as follows<sup>2</sup>. Every slave synchronizes to its master’s clock by exchanging synchronization messages with the master clock.

The synchronization process is divided into two phases. First (see figure 1) the time difference between master and slave is corrected; this is the offset measurement.

During this offset correction, the master cyclically transmits a unique synchronization (SYNC) message to the related slave clocks at defined intervals (by default every 2 seconds). This sync message contains an estimated value for the exact time the message was transmitted.

For highly accurate synchronization a mechanism is now provided that determines the time of transmission and reception of PTP messages as precisely and as closely as possible to the hardware, best of all directly on the medium.

The master clock measures the exact time of transmission  $TM1$  and the slave clocks measure the exact times of reception  $TS1$ . The master

<sup>2</sup>The following text and images 1 and 2 have been extracted from [www.ieee1588.com](http://www.ieee1588.com).

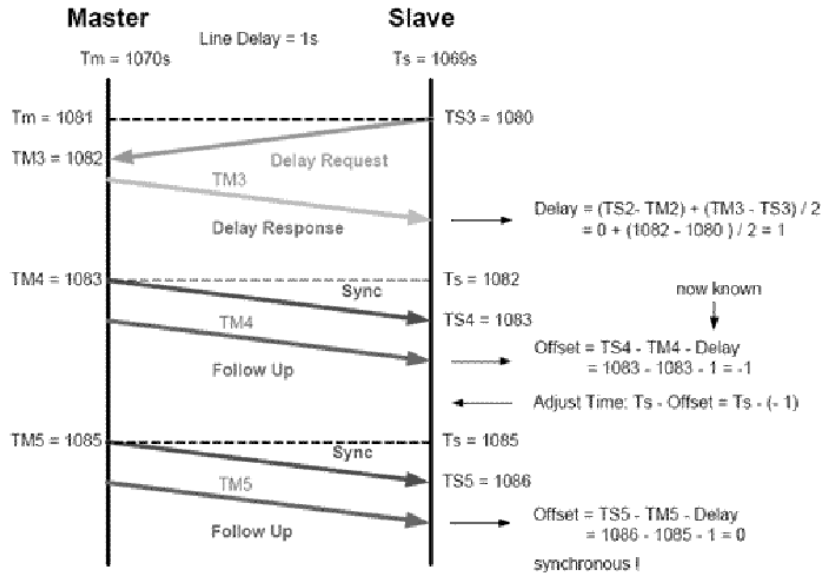


Figure 2: Delay measurement

then sends in a second message, the follow-up message, the exact time of transmission  $TM1$  of the corresponding sync message to the slave clocks.

On reception of the sync message and, for increased accuracy, on reception of the corresponding follow-up message, the slave clock calculates the correction (offset) in relation to the master clock taking into account the reception time stamp of the sync message. The slave clock  $T_s$  must then be corrected by this offset. If there were to be no delay over the transmission path, both clocks would now be synchronous.

The second phase of the synchronization process (see figure 2), the delay measurement, determines the delay or latency between slave and master. For this purpose the slave clock sends a so-called "delay request" packet to the master and during this process determines the exact time of transmission of the message  $TS3$ . The master generates a time stamp on reception of the packet and sends the time of reception  $TM3$  back to the slave in a "delay response" packet.

From the local time stamp for transmission  $TS3$  and the time stamp for reception provided by the master  $TM3$ , the slave calculates the delay time between slave and master.

The delay measurement is performed irregularly and at larger time in-

tervals (random value between 4 and 60 seconds by default) than the offset measurement. In this way, the network, and particularly the terminal devices, are not too heavily loaded. However, a symmetrical delay between master and slave is crucial for the delay measurement and its precision, i.e. same value for both directions.

Using this synchronization process, timing fluctuations in the PTP elements especially the protocol stack and the latency time between the master and slave are eliminated.

### 3 Implementation

The implementation we present performs the synchronization by software, and specifically, using software interrupts. This permits to avoid using extra hardware in the PTP implementation such as message detectors and time stamps units.

It does the offset correction by using only one SYNC message transmitted by the master every 2 seconds (as suggested by the protocol). And the delay measurement is performed right after reception of the SYNC message. As we will discuss later, the delay measurement could be avoided in our implementation, and the SYNC message could be transmitted at a lower frequency.

In the following, we provide the pseudo-code for the master and slaves nodes. Both codes make use of a structure named *time\_struct* that is used to store the time. The protocol mandates that the time format should be a 32-integer for storing the accounting of nanoseconds up to seconds, and a 32-integer for storing the accounting of seconds up to years. Note this second integer goes up to 150 years. A third 32-integer is suggested to account for epoques. Note that the three integers occupy 12 bytes. An taking into account that a CAN message can only carry 8 data bytes, sending these three integers would require two messages.

However, in our implementation, the time structure has two fields: nanos and seconds. The first one is an 16 integer, enough to store the accounting of 0.0 ms to 1000.0 ms. Note that this simplification can be done because our implementation targets a millisecond precision. The second one is a 32-integer for storing the accounting of seconds up to years. The epoque integer is not implemented. Therefore, our time structure occupies 6 bytes, which can be sent in a single CAN message. In both codes, a *timer* is in charge of updating the time in the timer interrupt service routine (ISR). The timer overflows/interrupt every 0.1 ms.

```

time_struct time;

/*----- Time update and TM1 transmission -----*/

void isr_timer(void){
    update_time(time);
    if (mod(time,2)==0) send_message(TM1,time);
}

/*----- TM3 transmission on reception of TS3 -----*/

void isr_RX(void){
    send_message(TM3,time);
}

/*----- Main -----*/

void master(void) {
    setup_timer();
    enable_interrupts(timer, RX);
    ...
    ----- main code -----
    ....
    disable_interrupts(timer, RX);
}

```

Figure 3: Master node pseudo-code

Figure 3 shows the pseudo-code for the master node. The clock synchronization is done at the interrupts level, so, no code is required in the *main*. In the master, the timer isr also sends the SYNC message (or TM1) of the offset correction stage. Finally, the isr associated to the reception of messages sends message TM3 upon reception of the message TS3 sent by the slave in the delay measurement stage.

Figure 4 shows the pseudo-code for the slaves nodes. It is very similar to the master code. The time update isr only updates the time. The message reception isr performs the offset correction and triggers the delay measurement phase upon reception of the SYNC message, and it also performs the delay measurement correction upon reception of the TM3 message.

Note that depending on the specific underlying network and hardware, these pseudo-codes will have to be adjusted. Since the implementation of the IEEE 1588 that we report was done on CAN, in the following, we briefly describe the main features of CAN that at some extend have some influence of the implementation.

Note also that the delay measurement stage is performed at each SYNC message, much more frequent that the protocol suggests. This modification

```

time_struct time;

/*----- Time update -----*/

void isr_timer(void){
    update_time(time);
}

/*----- offset and delay correction -----*/

void isr_RX(void){
    if message_id(TM1){
        time=correct_offset();
        send_message(time);
    }
    else{
        /* message is TM3 */
        time=correct_delay();
    }
}

/*----- Main -----*/

void slave(void) {
    setup_timer();
    enable_interrupts(timer, RX);
    ...
    ----- main code -----
    ....
    disable_interrupts(timer, RX);
}

```

Figure 4: Slave node pseudo-code

has been performed in order to simplify code and in order to use as fewer resources as possible. However, as previously announced, an alternative implementation that we present later on shows that eliminating the delay measurement phase does not have any influence on achieving the targeted accuracy.

## 4 CAN overview

In general messages generated or received to implement the protocol can be delayed in an unpredictable way by different factors (different software execution times, resources consumed by other communication processes, scheduling and bus arbitration, etc). We show that the application of the Controller Area Network (CAN) as a networking infrastructure can remove some of this unpredictability (which usually is done by hardware assistance).

The CAN protocol regulates bus access by bit-wise arbitration. The priority, at which a message is transmitted compared to another less urgent message, is specified by the identifier of each message. The identifier with the lowest binary number has the highest priority. Bus access conflicts are resolved by arbitration of the identifiers involved by each station observing the bus level bit for bit. This happens in accordance with the wired-and-mechanism, by which the highest priority message wins the bus access.

Tacking advantage of this arbitration mechanism, any message transmitted by a node is received in all the nodes during the same bit time, that is, within  $1\mu s$  (if the CAN bus speed is 1Mbps) and  $100\mu s$  (if the speed is set to 10Kbps).

The maximum length for a data frame in Standard CAN is  $66 + 8n$  and for Extended CAN is  $90 + 8n$ , where  $n$  is the number of data bytes, which can be up to  $8^3$ . Therefore, in both cases, having a speed of 125Kbps, the transmission of a standard frame with 8 data bytes is around  $1ms$ . For higher speeds, the transmission time decreases to the order on  $\mu s$ . Remind that the messages used in our PTP implementation use 6 data bytes.

In addition, the bitwise arbitration on message identifier permits to code the PTP messages in such a way that scheduling and bus arbitration delays are eliminated or, if existing, perfectly characterized. In fact, SYNC messages could use the highest priority identifier, and the delay measurement messages the next ones.

## 5 NCS timing requirements

For control loops closed over networks that operate on absolute time measures, in many cases, nodes are required to be synchronized at the millisecond level [7]. It is important to point out this timing requirement depends on the specific controlled plant. For example, a Ball and Beam process can be considered slow in front of a voltage stabilizer. However, both can be controlled if the accuracy of the distributed nodes is at the millisecond.

## 6 Results

In this section we report the experimental results. We describe the hardware, and then we explain the results for the implementation of the IEEE 1588 protocol, as well as for an alternative implementation that also meets the NCS requirements and demands less hardware and software.

---

<sup>3</sup>The maximum length is accounting for the maximum number of stuffing bits.

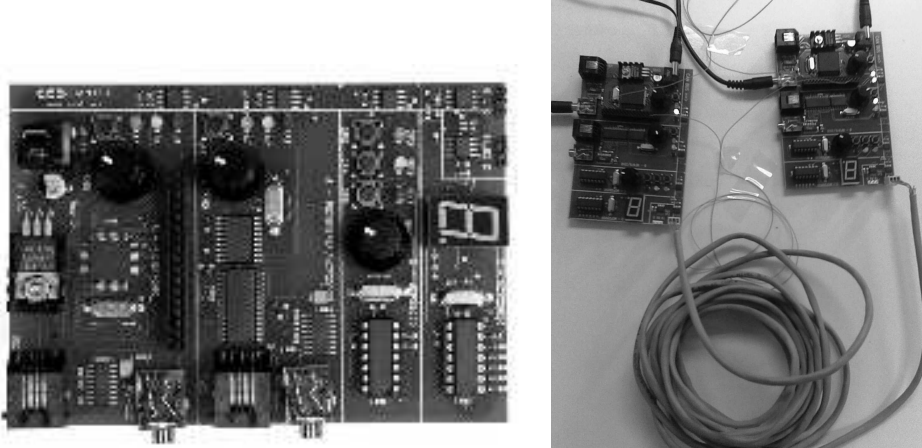


Figure 5: Board with four CAN nodes (left) and setup (right)

## 6.1 Setup

In our experimental setup we have use two CCS CAN prototype boards<sup>4</sup>. Each board, shown in figure 5 (left), has four nodes connected via CAN. From left to right in the figure, the first node is a PIC18F4580 which includes an integrated CAN peripheral. The second node is a PIC16F876A connected to an MCP2510, an external CAN peripheral which is connected to a micro-controller over SPI. Both nodes are connected to a 20MHz oscillator. The last two nodes, which have not been used in our experiments, are MCP25050s, stand-alone CAN expanders. The board also has an CAN transceiver that permits to connect the CAN bus prototype board with other CAN systems, such as other CAN bus prototype boards. Both PIC-based nodes have been programmed using the PCWH IDE Compiler for 12-bit, 14-bit, and PIC18 MCU parts, also from CCS.

In all the experiments, the time management in all the PICs was done every 0.1ms. That is, the real-time clock maintained by the each timer in each node had a precision of 0.1ms.

In some experiments, we have used MiniMon, a CAN software sniffer from IXXAT<sup>5</sup>, that can be used together with the USB-to-CAN compact, a low-cost, active CAN module for connection to the USB bus.

---

<sup>4</sup><https://www.ccsinfo.com/>.

<sup>5</sup><http://www.ixxat.com/>.

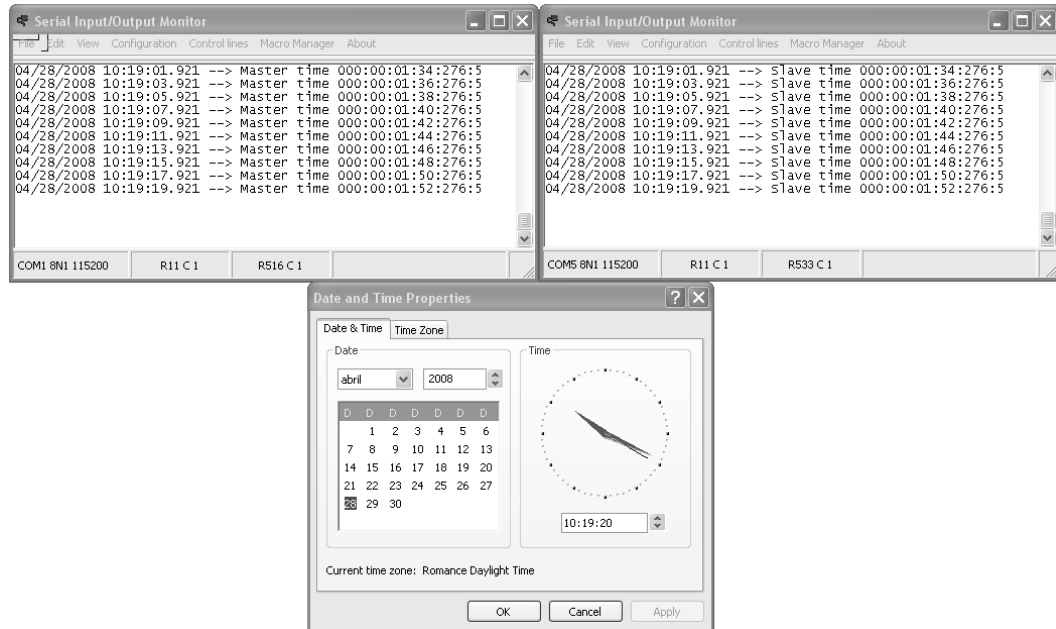


Figure 6: Measurements results for the IEEE 1588 clock synchronization

## 6.2 Implementation results for pure IEEE 1588

To experiment with IEEE 1588, we have used two boards connected via CAN, as shown in Figure 5 (right). In particular, on PIC18F plays the master, and the other PIC18F plays the slave. The bus speed was set to 125Kbps.

The most challenging task was, once clocks were synchronized, to read both clocks. To do so, we used the IO ports associated to the PIC18F. In order to read in both PIC's the *time* at the same time, one PIC18F acting as a master generated in one of the ports a flank every two seconds. In the master, after generating the flank, the time was read. In the slave, after detecting the flank, the time was read. Both read times are then sent through the RS232 ports of each node to a PC equipped with Windows XP, which making use of two serial port monitors, displays both times.

Figure 6 shows the tow monitors. The left monitor corresponds to the master node and the right monitor corresponds to the slave monitor. Also, the PC clock is shown, in order to give a time reference.

Both monitor show the same type of information. First of all, they show the PC time, which is a time stamp feature provided by the monitor. Then,

```

time_struct time;

/*----- Time update and TM1 transmission -----*/

void isr_timer(void){
    update_time(time);
    if (mod(time,2)==0) send_message(TM1,time);
}

```

Figure 7: Master node alternative pseudo-code

```

time_struct time;

/*----- Time update -----*/

void isr_timer(void){
    update_time(time);
}

/*----- offset correction -----*/

void isr_RX(void){
    time=correct_offset();
}

```

Figure 8: Slave node alternative pseudo-code

after the arrow, and starting by *Master time* or *Slave time*, it is displayed read time in the master and in the slave, respectively. The time is displayed in

*days : hours : minutes : seconds : milliseconds : microseconds*

In fact, they do not show microseconds, but their hundreds. As it can be seen the time synchronization achieved is of the order of 0.1ms, meeting our requirements.

### 6.3 Alternative implementation

As outlined in the CAN review section, the protocol establishes that all nodes receive a message withing the bit time, which is of a magnitude less than one millisecond. This permits to implement the clock synchronization by only performing the offset correction phase, because propagation times and transmission times introduce negligible clock drift if the accuracy has to be of the order of 1ms.

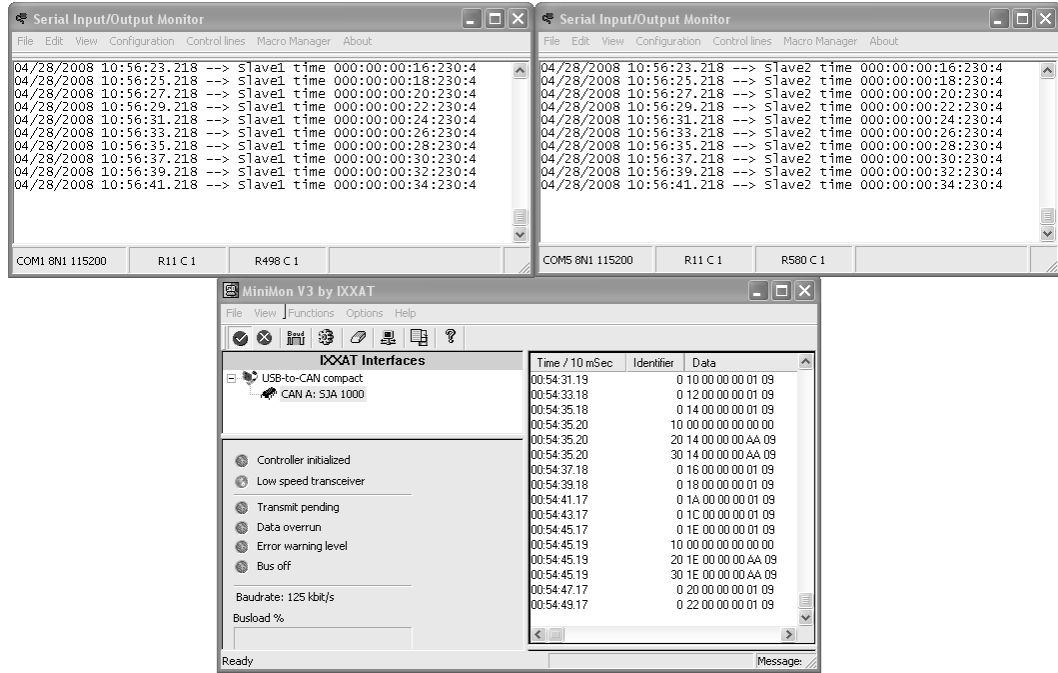


Figure 9: Synchronization results for the alternative implementation

From figures 3 and 4, the alternative synchronization mechanism changes the isr routines as shown in figures 7 and 8. As it can be seen, the only control code refers to the sending of the SYNC message in the master and offset correction in the slave. If required, the slave could update the synchronized time to take into account the transmission time of the SYNC message. However, for the required precision, this is not necessary in this case.

For validating this setup, we used PIC16F on one board as a master, and two PIC18Fs as slaves. After clocks were synchronized, the reading of the slaves times was made using interrupts and transmitting extra messages. To read the two slaves clocks at the same time, we sent periodically (every 10 seconds) a message from the master. The two slaves, upon reception of this message, stored their times, and sent them through CAN. In addition, the two slaves, at each SYNC message sent by the master every 2 seconds, printed their times. The slaves monitors, as well as the MiniMon screen are shown in figure 9. As it can be seen, both monitors show always the same time in same format as before, therefore corroborating that the synchronization was correct.

The MiniMon screen shows all the messages. Messages with ID 0 are SYNC messages (highest priority messages), sent every 2s by the master. Message with ID 10 is the message also sent by the master every 10 seconds to trigger the clock reading in the slaves. Slaves, upon reception of message ID 10, sent their clocks in messages with ID 20 (for Slave1) and ID 30 (for Slave2), respectively.

As it can be seen in the monitors, for example, the third slaves times starting from the bottom, displayed in the monitors is

000 : 00 : 00 : 30 : 230 : 4

that is, 30 seconds, etc.. At that time, the master sent the SYNC message (ID 0) and message ID10, to trigger the offset correction and the clock reading in the slaves, respectively. And the last two messages with ID 20 and 30 in the MiniMon carry the time read by the slaves. As it can be seen in the field Data, the time sent by both slaves is the same, but codified in hexadecimal. From right to left, the 8 first digits correspond to the nanos field, 0000AA09, and the last four digits correspond to the seconds field, 1E00. Therefore, the nanos field carries a 247.4 (0x09AA=0d2474) and the seconds field carries a 30 (0x001E=0d30). That is, both messages show the same time, meaning the the slaves are perfectly synchronized. The time they carry is a bit greater than the time displayed in the monitors, due to messages latencies and execution time delays.

In summary, using this alternative clock synchronization mechanism, clocks are synchronized with the required accuracy, and less overhead in introduced in the applications and in the network

Finally, we have to point out that in our alternative implementation, the SYNC message was sent every 2seconds, as suggested by the IEEE 1588 protocol. However, experiments using the described hardware have shown that this frequency could dramatically decreased without jeopardizing the achieved time accuracy.

## 7 Network scheduling issues

A common architecture for a single control loop in a NCS is shown in Fig. 10, where sensor, controller and actuator nodes exchange data via network messaging. In this architecture, any node could play the master clock role.

The networked control loop operation relies on control messages, which introduces delays within the loop, as shown in Figure. 10. These message have to be considered together with the clock synchronization messages.

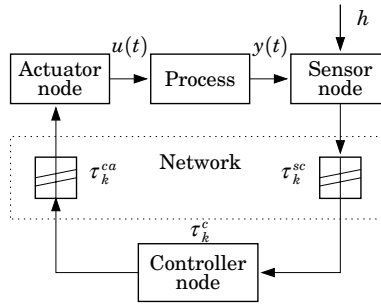


Figure 10: Common architecture for NCS

Therefore, careful design of messages as well as schedulability analysis are key aspects for implementation feasibility of networked control loops whose control algorithm relies on absolute time measurements.

## 8 Conclusions

This work has analyzed clock synchronization mechanisms for NCS. The required accuracy of 1ms has been achieved using IEEE 1588 protocol, but also using a light-weight protocol that takes advantages of the CAN properties.

## References

- [1] ISO 11898-1: Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling, 2003
- [2] Y. Tipsuwan and M. Y. Chow, *Control methodologies in networked control systems*, Control Engineering Practice, Vol. 11, pp. 1099-1111, 2003
- [3] R. Holler, M. Horauer, G. Gridling, N. Kero, U. Schmid, K. Schossmaier, “SynUTC - High Precision Time Synchronization over Ethernet Networks,” Proceedings of the 8th Workshop on Electronics for LHC Experiments, 2002.
- [4] R. Holler, T. Sauter, and N. Kero, “Embedded SynUTC and IEEE 1588 clock synchronization for industrial Ethernet,” IEEE Conference on Emerging Technologies and Factory Automation, 2003.

- [5] W. Hu, G. Liu, and D. Rees, “Event-driven networked predictive control,” *IEEE Transactions on Industrial Electronics*, vol. 54, n.3, 1603-1613, 2007.
- [6] IEEE 1588<sup>TM</sup>-2002. Standard for A Precision Clock Synchronization Protocol for Networked Measurement and Control Systems
- [7] C. Lozoya, P. Martí, M. Velasco, and J.M. Fuertes, “Analysis and design of networked control loops with synchronization at the actuation instants,” submitted to 34th Annual Conference of the IEEE Industrial Electronics Society, 2008.
- [8] R. Marau, P. Leite, M. Velasco, P. Martí, L. Almeida, P. Pedreiras, and J.M. Fuertes, “Performing Flexible Control on Low Cost Microcontrollers using a Minimal Real-Time Kernel”, accepted to *IEEE Transactions on Industrial Informatics*, 2008.
- [9] P. Martí, J.M. Fuertes and G. Fohler, “An Integrated Approach to Real-time Distributed Control Systems Over Fieldbuses.” *8th IEEE International Conference on Emerging Technologies and Factory Automation*, Antibes Juan-les-pins, France, October 2001.
- [10] P. Martí and M. Velasco, “Toward Flexible Scheduling of Real-Time Control Tasks: Reviewing Basic Control Models,” *10th International Conference on Hybrid Systems, Computation and Control*, LNCS, 2007.
- [11] M. Velasco, P. Martí, R. Castañé, J. Guardia and J.M. Fuertes, “A CAN Application Profile for Control Optimization in Networked Embedded Systems”, *32th Annual Conference of the IEEE Industrial Electronics Society*, Nov. 20
- [12] B. Wittenmark, J. Nilsson and M. Törngren, “Timing Problems in Real-Time Control Systems,” *1995 American Control Conference*, 1995.