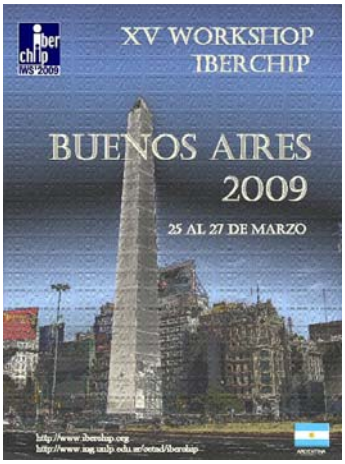
XV Workshop IBERCHIP



Palacio de Correos y Telecomunicaciones

Ciudad Autónoma de Buenos Aires

Marzo 25-27, 2009

Fernando G. Tinetti (Ed.)

Accurate and compact implementation of a hardware SNTP Client

J. Viejo, J. Juan, E. Ostua, M. J. Bellido, A. Millan, A. Muñoz, and J. I. Villar
Grupo ID2 (Investigacion y Desarrollo Digital)
Departamento de Tecnologia Electronica-Universidad de Sevilla
E. T. S. Ing. Informatica, Campus Universitario Reina Mercedes
41012 Sevilla (SPAIN)
Email: julian@dte.us.es, jjchico@dte.us.es, ostua@dte.us.es, bellido@dte.us.es,
amillan@dte.us.es, amrivera@dte.us.es, jose@dte.us.es

***Abstract*—This contribution presents a compact, accurate and cost-effective SNTP client implementation. Once synchronized to a (S)NTP server, the client functions as a GPS unit generating a PPS (Pulse Per Second) signal and NMEA information through a serial port. This device is part of a technological platform implementing accurate synchronization solutions for Remote Terminal Units (RTUs) commonly used in industrial control systems.**

I. Introduction

Time stamping is a critical task in many industrial control systems. Data acquisition by Remote Terminal Units (RTUs) being a typical example. In this sense, the industry norm IEC 61850 [1] defines the Simple Network Time Protocol (SNTP) [2] over Ethernet as a standard way to synchronize a set of substations with a time server. SNTP is a simplified version of the more general Network Time Protocol (NTP) [3] that is commonly used in Internet servers and routers. Both SNTP and NTP share the same communication protocol and data format, the main difference being that NTP uses sophisticated algorithms that ensures a correct synchronization with multiple servers under highly variable latency data links, which is common in a world wide network like the Internet. On the contrary, SNTP covers the synchronization with a single server and uses a simplified stateless algorithm, which makes it suitable for embedded systems in a controlled industrial environment. Nevertheless, a SNTP client may communicate with either a SNTP server or a full NTP server.

In this scenario, a time server will typically gather accurate time information from an absolute reference like an accurate clock or a GPS receiver. SNTP clients located at the substations will synchronize with the server through the Local Area Network (LAN) making it unnecessary to install absolute references at the substations. SNTP clients will then provide the nearby electronic equipment with the necessary time information.

This contribution is part of a project supported by the Ministry of Industry of Spain and leaded by the Telvent company [4] which finality is to develop a Common Technological Platform (PTC) to easy the implementation of the functionality typically found in Remote Terminal Units (RTU) used to control the public power grid. A key point in the project is to assure the synchronization of electronic equipment within the range of a few microseconds. This synchronization is achieved by the use of SNTP clients and servers fully implemented in hardware. SNTP client and server design is divided in three main phases:

1) Basic protocol stack implementation and integration with Ethernet controller. At least the following protocols are needed: IP, ARP, UDP and NTP.
2) NTP client implementation: issuing of requests, answer processing and local clock synchronization.
3) NTP server implementation: synchronization with an external reference and request processing.

This paper describes the current status of the client implementation that includes phase 1 and phase 2.

The rest of this contribution is organized as follows: a brief introduction to NTP and SNTP is included in section 2, section 3 enumerates the requirements of the system and general specifications, section 4 gives the details of the design and implementation of the hardware SNTP client, section 5 includes some implementation results and section 6 discusses some conclusions.

II. NTP/SNTP protocol basic operation

The operation of the NTP/SNTP protocol is very simple (Fig. 1). The client sends a request to the server by issuing an UDP data packet where the time of its local clock ($_1$) is included. When the request is received at the server a new time stamp $_2$ is generated with the reception time as given by the server's local clock. After processing the request, the server issues a reply including the time at which the reply leaves the server ($_3$). When the client receives the reply the arrival time ($_4$) is also annotated. With this set of timestamps the client can calculate the round trip time ($_{rd}$) and the time offset between the server's and client's clocks ($_{offset}$). Assuming a symmetric connection it gives:

$$\begin{aligned} {}_{rd} &= (\ {}_4 - \ {}_1) - (\ {}_3 - \ {}_2) \\ {}_{offset} &= \frac{(\ {}_2 - \ {}_1) + (\ {}_3 - \ {}_4)}{2} \end{aligned} \tag{1}$$

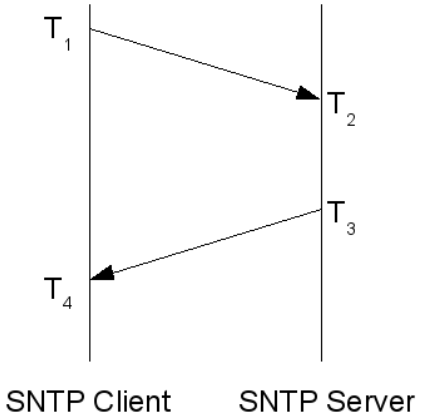


Fig. 1. Operation of the NTP/SNTP protocol.

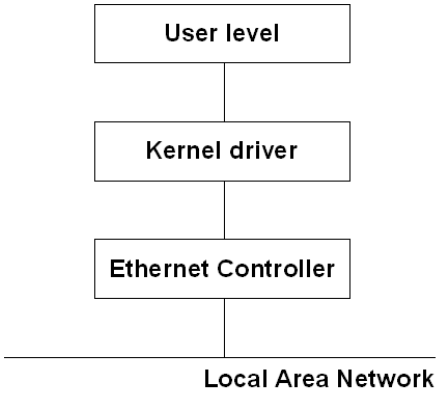


Fig. 2. Layers where NTP can be implemented.

Using the calculated offset the client can correct its local clock to match the server time. Software implementations of NTP tipically achieve time synchronization within a millisecond with respect to the server [3]. There are two main sources of error. The first one is the asymmetry in the network communication when the time spent by the client's request to reach the server is different to the time spent by the answer to reach the client. This is due to unpredictable latency in network equipment, specially when collisions take place and the number of the devices involved increases. The second main source of error is due to the variable time gap between the instant the time stamp is registered in the datagram and the real instant the datagram leaves or reaches the host. In typical software implementation, these time stamps are registered by client/server software running as a user level application (Fig. 2) so the time stamp error will depend on the time spent processing the datagram as it goes through the protocol stack and software layers. This error will largely depend on system load, detailed software implementation, etc. The precision of the NTP synchronization can be largely improved by doing the time-stamping operation in lower layers [5], therefore some operating system kernels like Linux of FreeBSD support NTP processing in the kernel [6]. This way, precision may reach some tens of microseconds.

The highest precision in the time-stamping operation is only achievable if done by the Ethernet device hardware as soon as the packets arrive or leave the interface.

III. System Specification

The main objective of this contribution is to build cost-effective, autonomous, compact and highly accurate SNTP client and server modules suitable for, but not limited to, IEC 61850 environments. The SNTP server will use a standard GPS receiver as a time reference. It will use the PPS (Pulse Per Second) signal and NMEA data from the GPS to synchronize its internal clock. The SNTP client will synchronize with the server through the local area network using the NTP protocol, and will provide a PPS signal and NMEA information through a serial interface thus emulating a GPS receiver. A typical scenario is depicted in Fig. 3 where the server (H-SNTPD) gathers the time from the GPS receiver and clients (H-SNTPC) provide time and synchronization information to remote terminal units (RTU).

More specifically, the SNTP client should meet the following criteria:

- The client will operate in a standard 10/100/1000MHz Ethernet LAN.
- The client will configure automatically using the BOOTP [7] protocol so that the configuration for all the clients can be centralized in a single BOOTP server.
- The precision of the local clock at the client should be within 10 with respect to the server's clock in optimal conditions: hardware time-stamping in the server and direct LAN connection without switches. In typical conditions (software server and standard switch connection) precision should be always within 1 .
- The whole client design should fit in a single, low density FPGA chip and should need no additional hardware, so that cost of system parts will be under $20 before mass production.
- Low power. Implemented in a low density, low frequency FPGA, the client will consume under 1W of average power which is much lower than a computer-based implementation that would consume about 100W.

IV. Design and implementation

In this section, the most important aspects of design and implementation are commented. A diagram of the modules that form the SNTP client is shown in Fig. 4. We can distinguish the following parts: control unit, Ethernet MAC controller, SNTP client module and PPS generation and RMC frame transmission module. Next, we will briefly explain the functionality of each one of these subsystems.

The control unit is in charge of arbitrating the operation of the rest of the modules in order to perform the adequate task in each moment. The module has been modeled as a Finite State Machine using Verilog coding according to the structure

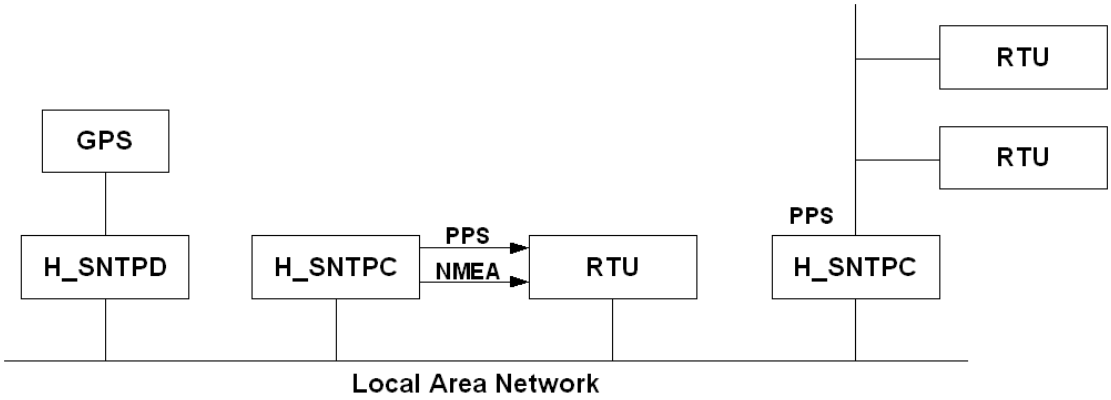


Fig. 3. Typical scenario for deploying hardware SNTP client and servers.

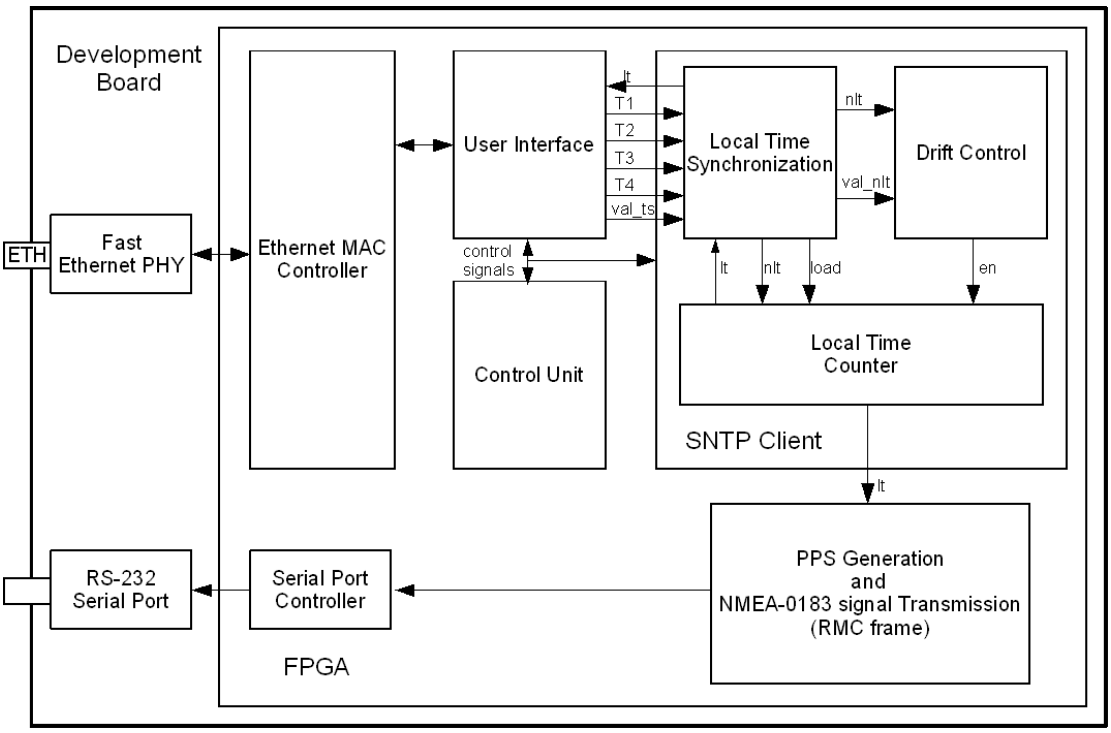


Fig. 4. Block diagram of the SNTP client.

described in [8]. The control unit defines two main operating modes:

1) Configuration. When the SNTP client starts to operate or after a system reset, an automatic configuration process is performed according to the Bootstrap Protocol (BOOTP) [7]. This process consists of finding the SNTP client IP address and a series of configuration parameters like the RS-232 serial port baudrate. The use of BOOTP is due to the its simplicity compared to DHCP [9] which makes it more suitable to be implemented in hardware, while the extended capabilities of DHCP are not useful for the intended application and would only introduce extra development and resource costs.
2) Normal operation. Once the configuration process has finished, the SNTP client begins to work into the normal operation mode. In this mode, the device carries out different tasks which we are going to summarize next. Firstly, the SNTP client needs to know the SNTP server IP address. In this way, the client sends a request to a broadcast address (manycast mode) and expects a reply from one or more servers. The client uses the first reply received to establish the particular server for

subsequent unicast request. Once the client knows the SNTP server IP address, it is necessary to find out the SNTP server MAC address; so the client includes a simple implementation of the Address Resolution Protocol (ARP) [10]; as well, the client must be able to send a ARP reply packet whenever another device requests its MAC address. Secondly, the client must transmit a time request packet (SNTP message) at secondly intervals. Finally, when the SNTP client receives the time reply packet, the timestamps obtained are registered so that the SNTP client module can synchronize the local clock.

The Ethernet MAC controller is in charge of controlling a standard Fast Ethernet PHY device, allowing us to transmit and receive Ethernet frames conforming to IEEE 802.3 specification. The implementation of this module has been carried out using the Tri-mode Ethernet MAC IP-core available from the OpenCores project in its web portal *opencores.org*. Moreover, this IP-core has a FIFO interface to user applications which facilitates the SNTP client design.

The user interface has been developed according to the specification document [11]. This interface has been implemented as a finite state machine coded in Verilog, and is formed by a transmitter module and a receiver module. So, on one hand, the transmitter module is able to transmit three different Ethernet frames: BOOTP Request, ARP Request/Reply and time request packets, which are stored in a RAM. This module also includes a memory updating component that is in charge of updating the different packet fields before transmitting them. On the other hand, the receiver module is able to identify the following frames: BOOTP Reply, ARP Request/Reply and time reply packets, ruling out the rest of Ethernet frames.

The SNTP client module is in charge of calculating the clock offset using the timestamps and synchronizing the local time. Additionally, a drift control is carried out in order to improve the local clock accuracy. This component has been developed using the System Level tool System Generator for DSP according to the methodology presented in [12].

Finally, the PPS generation and RMC frame transmission module is in charge of generating a synchronization signal (PPS+NMEA) which will be sent through the serial port to a Remote Terminal Unit. To implement this module the same methodology used to build the SNTP client module has been employed. A block diagram of the module is shown on Fig. 5. The main tasks developed with this subsystem are: NTP format conversion, basic UART interface and a controller module, mainly build with the Picoblaze microprocessor, with some added components designed in sinthesizable VHDL and implemented in the FPGA. A brief description of each of these blocks follows.

Firstly, the time format conversion module, takes the date in NTP format (64 bits, 32 for seconds and 32 for fractions) and outputs the date format DD/MM/YY and HH:MM:SS, as will be required for the NMEA line later. This task is completely build with a sinthesizable VHDL block attached, designed as a Finite State Machine implementation and some extra logic like adders and dividers cores. The output is interfaced to Picoblaze through the input ports multiplexer, which will read the data as it's needed.

Secondly, the initial specifications for the external serial interface, were:

- Fixed data output format, "8N1" (8 data bits, 1 stop bit and no parity).
- Communication through TX and RX signals, no other signaling needed.
- No flow control.

The design of this simple UART controller was acomplished and later a few other features were included to make it more flexible. For example:

- Baudrate online reconfiguration: for this concerns, a baudrate generator is included in the design in order to be able to program the desired baudrate of the serial trasmission. The default baudrate was fixed to 4800 bauds.
- PPS signal generation: this output was connected to the DSR output pin of the RS232 port.

Also, the center block integrates the Picoblaze soft-core microprocessor, with its ROM memory where the application program is loaded. This is the main controller of the system and interfaces to the others described subsystems with its input and output ports. It builds and sends the NMEA RMC sentence based on the data received and also calculates the global checksum (that's basically the XOR operation on the characters sent) for its inclusion at the end of the line transmission.

The serial interface has a 16 bytes FIFO for the tx-queue. In order to interface with the Picoblaze micro, two of the status signals are used, buffer_full and buffer_half_full, so the processor checks its status periodically to control the flow of characters.

Finally, the program running on the Picoblaze is designed in assembler code, and it includes two main subroutines: first, to control the transmission FIFO of the UART in order to achieve a fluent communication; second, to send each one of the ASCII characters previously computed and keep the checksum updated at the same time. On the other hand, main program first waits an active event of the PPS signal to start the sending process, where it should check if the synchronization was accomplished, as the RMC sentence differs in each case. Later, it converts the decimal digits (coming from the input ports) to ASCII characters. The checksum value at the end of the sentence also needs a conversion to hexadecimal characters.

V. Results

In this section, simulation and hardware implementation results are described in some detail.

A. Simulation results

In order to check that the designs works correctly, the following simulation process has been carried out. At the first stage, the design has been verified using Simulink and ModelSim. For the generation of the input stimuli, the Source Blockset of Simulink has been employed. At the second stage,

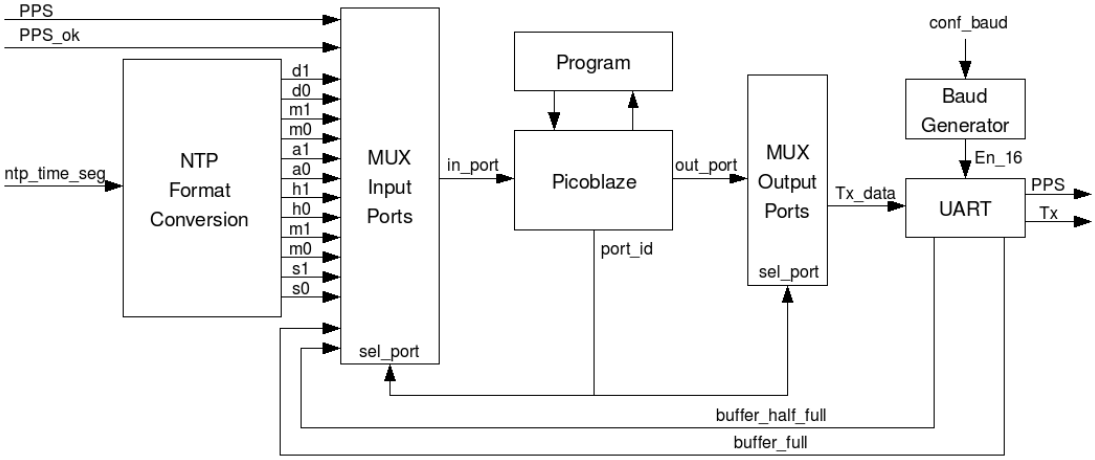


Fig. 5. Block diagram of the PPS generation and RMC frame transmission module.

TABLE I
HARDWARE IMPLEMENTATION RESULTS ON SPARTAN-3E XC3S500E.

Figure of merit	Usage (%)
Slices	3,411 (73%)
Slice Flip Flops	3,651 (39%)
4 input LUTs	4,493 (48%)
Bonded IOBs	37 (15%)
Block RAMs	9 (45%)
GCLKs	8 (33%)
Maximum operation frequency	43 MHz

we have used the Xilinx tool ChipScope Pro to perform the on-chip verification of the client. In this way, we have verified the correct transmission and reception of the different packet types: BOOTP, ARP, and SNTP messages. In relation with the scenario depicted in Fig. 3, the client has been tested against a software NTP server since the hardware SNTP server has not been developed yet. Fig. 6 compares a PPS signal generated by a GPS unit (top curve) to the PPS generated by the client. Fig. 7 shows a sequence of NMEA traces generated by the client. Synchronization accuracy is within 20 microseconds, limited by the use of a software NTP server.

B. Hardware implementation results

In this subsection, hardware implementation results will be presented. Specifically, two figures of merit will be analyzed: hardware resources and maximum operation frequency.

The design has been implemented on a Spartan-3E XC3S500E FPGA. The Table I shows the design results in the current development stage. It is remarkable that although the implementation is finished, an optimization process must be carried out. Thus, the final resource requirements will be reduced.

VI. CONCLUSION

The design of a SNTP client completely done in hardware has been presented. By using a high level methodology and standard FPGA technology it is possible to produce a high accurate, cost-effective and flexible solution for accurate time distribution and time stamping in industrial environments that agrees with international standards. First prototypes are expected to provide synchronization in the range of the microsecond in a compact and cheap device that would substitute expensive computer-based solutions or dedicated GPS receivers.

ACKNOWLEDGMENT

This work has been partially supported by the Ministry of Education and Culture of the Spanish Government through the TEC2007-61802/MIC (HIPER) project and the PROFIT-MITC SEPIC TSI-020100-2008-258 project.

REFERENCES

[1] I. E. C. Technical Committee 57, "IEC 61850 Communication Networks and Systems In Substations," IEC 61850 Edition 2 and other extensions, Jun. 2008.
[2] D. L. Mills, "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI," RFC 4330 (Informational), Jan. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4330.txt
[3] D. L. Mills, "Network Time Protocol (Version 3) Specification, Implementation and Analysis," RFC 1305 (Draft Standard), Mar. 1992. [Online]. Available: http://www.ietf.org/rfc/rfc1305.txt
[4] *Telvent Company Web Portal*. http://www.telvent.com.
[5] T. Skeie, S. Johannessen, and O. Holmeide, "Highly accurate time synchronization over switched Ethernet," in *Proc. 8th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, Antibes-Juan les Pins (France), Oct. 2001, pp. 195–204.
[6] D. L. Mills and P. H. Kamp, "The nanokernel," in *Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting*, Reston VA (USA), Nov. 2000, pp. 423–430.
[7] W. J. Croft and J. Gilmore, "Bootstrap Protocol," RFC 951 (Draft Standard), Sep. 1985, updated by RFCs 1395, 1497, 1532, 1542. [Online]. Available: http://www.ietf.org/rfc/rfc951.txt
[8] C. E. Cummings, "The Fundamentals of Efficient Synthesizable Finite State Machine Design using NC-Verilog and BuildGates," in *International Cadence Usergroup conference (ICU)*, San Jose, California (USA), Sep. 2002.
[9] R. Droms, "Dynamic Host Configuration Protocol," RFC 2131 (Draft Standard), Mar. 1997, updated by RFCs 3396, 4361. [Online]. Available: http://www.ietf.org/rfc/rfc2131.txt

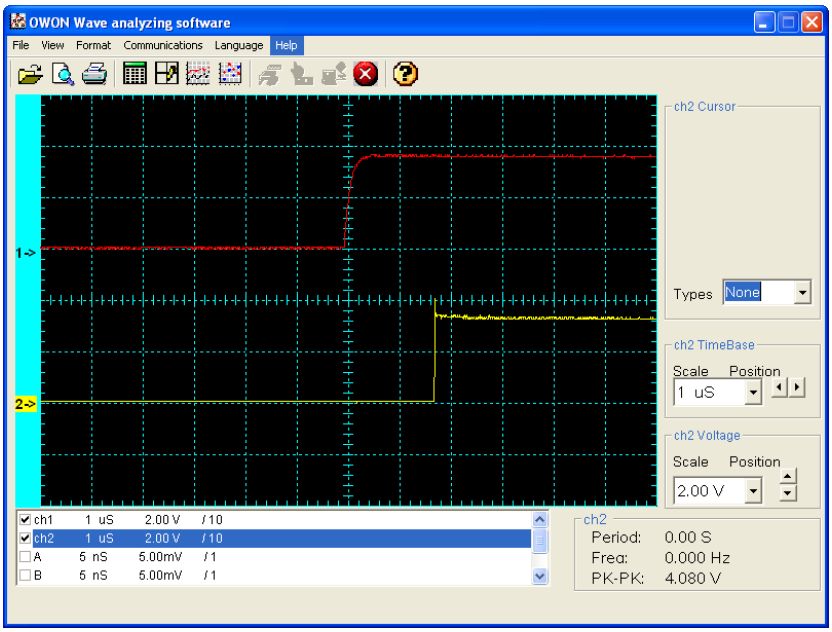


Fig. 6. PPS signal generates by the SNTP client designed.

Fig. 7. NMEA RMC frame generated.

[10] D. Plummer, "Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware," RFC 826 (Standard), Nov. 1982, updated by RFC 5227. [Online]. Available: http://www.ietf.org/rfc/rfc826.txt

[11] J. Gao, *10_100_1000 Mbps Tri-mode Ethernet MAC Specification*, OPENCORES.ORG, Jan. 2006.

[12] J. Viejo, M. J. Bellido, A. Millan, E. Ostua, J. Juan, P. Ruiz-de Clavijo, and D. Guerrero, "Efficient Design and Implementation on FPGA of a MicroBlaze Peripheral for Processing Direct Electrical Networks Measurements," in *Proc. 1st IEEE Symposium on Industrial Embedded Systems (IES)*, Antibes-Juan les Pins (France), Oct. 2006, pp. 1–7.