Network Time Synchronization: A Full Hardware Approach

Jorge Juan, Julian Viejo, and Manuel J. Bellido

Departamento de Tecnología Electrónica,
ETSI Informática, Universidad de Sevilla, Spain
jjchico@dte.us.es
http://www.dte.us.es/id2

Abstract. Complex digital systems are typically built on top of several abstraction levels: digital, RTL, computer, operating system and software application. Each abstraction level greatly facilitates the design task at the cost of paying in performance and hardware resources usage. Network time synchronization is a good example of a complex system using several abstraction levels since the traditional solutions are a software application running on top of several software and hardware layers. In this contribution we study the case where a standards-compliant network time synchronization solution is fully implemented in hardware on a FPGA chip doing without any software layer. This solution makes it possible to implement very compact, inexpensive and accurate synchronization systems to be used either stand-alone or as embedded cores. Some general aspects of the design experience are commented together with some figures of merit. As a conclusion, full hardware implementations of complex digital systems should be seen as a feasible design option, from which great performance advantages can be expected, provided that we can find a suitable set of tools and control the design development costs.

Keywords: digital systems, hardware, network time synchronization, FPGA.

1 Introduction

Complex digital systems are typically built on top of several abstraction levels: digital, RTL, computer, operating system and software application. Each abstraction level, together with design automation tools, greatly facilitates the design task at the cost of the overhead introduced by every abstraction layer. This is payed in the form of reduced performance (both timing and power) and a much higher hardware resources usage. However, some critical parts in complex digital systems still require a low level implementation in order to improve performance or reduce power consumption. This is the case of the numerous hardware accelerators used today for audio and video processing that can be found in high performance or resource-limited devices like graphic adapters or

smart phones. At the same time, there are an increasing number of devices implementing in hardware high level functions typically done in software, like the QuickTCP IP core from PLDA [1]. However, there is still plenty of room for performance improvement in some high-level traditional software applications that can get a big performance boost if fully implemented in hardware. These applications can be improved by several orders of magnitude in speed, lower power consumption and smaller hardware footprint (area, logic blocks, etc.). These gains are a consequence of doing without some of the abstraction layers like the software and computer abstractions.

A full hardware implementation has to face a number of challenges like handling the complexity at the low level, design tool availability, development cost and time-to-market. Design teams also need to believe that it can be done. Network time synchronization is a very good example of this kind of traditional software applications where we can find clients, servers, network protocols and elaborated processing algorithms. This contribution summarizes the implementation of an prototype embedded network time synchronization system that is being developed completely in hardware. Some preliminary results show that the system can be fully implemented in hardware and perform with an accuracy comparable to commercial industrial equipment at a fraction of the cost and resources usage. The system can be implemented stand-alone or as a soft-core in a low grade FPGA chip.

In the following section, a brief introduction to network time synchronization is given. Section 3 is an overview of the synchronization system while Sect. 4 describes the design platform and tools. Section 5 summarizes some figures of merit and some conclusions are derived in Sect. 6.

2 Network Synchronization

The two main network synchronization protocols in use today are the Network Time Protocol (NTP) [2], and the Precision Time Protocol (PTP) [3]. NTP is more used to synchronize Internet equipment and is used by almost any Internet router and server. NTP servers typically serve thousands of clients over world wide network connections so the poll intervals are in the range of several minutes to a few hours and the network latencies are difficult to predict. To cope with that, NTP includes sophisticated mitigation and clock disciplining algorithms achieving a clock accuracy in the range of the millisecond. On the other hand, PTP was designed to synchronize equipment in industrial environments connected to a local area network and aims to sub-microsecond clock accuracy, making it suitable for measurement and control systems. PTP uses poll intervals in the range of a second or lower and most implementations uses some kind of dedicated hardware to implement the most critical parts and gain accuracy. The NTP standard also defines the Simple Network Time Protocol (SNTP) a simplified version of NTP, that do not impose the use of the mitigation and clock disciplining algorithms found in NTP. SNTP servers are typically primary servers connected to a single reference clock and SNTP clients typically connect

to a single servers and do not have dependent clients and have been used in scenarios similar to PTP. Otherwise, NTP and SNTP clients and servers can inter-operate seamlessly.

Both NTP and PTP synchronization is based in the interchange of packets between clients and servers. This mechanism is called the on-wire protocol, which objective is to determine the offset of the local clock at the client with respect to the server and the latency of the network connection. Both NTP and PTP on-wire protocols are very similar. Here we describe the operation of the NTP/SNTP on-wire protocol (Fig. 1). The client sends a request to the server by issuing an UDP data packet where the time of its local clock (T_1) is included. When the request is received at the server a new time stamp T_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 (T_3). When the client receives the reply the arrival time (T_4) is also annotated. With this set of timestamps the client can calculate the round trip time (t_{rd}) and the time offset between the server's and client's clocks (t_{offset}). Assuming a symmetric connection these times can be calculated according to eq. (1).

$$\begin{aligned} t_{rd} &= (T_4 - T_1) - (T_3 - T_2) \\ t_{offset} &= \frac{(T_2 - T_1) + (T_3 - T_4)}{2}. \end{aligned} \tag{1}$$

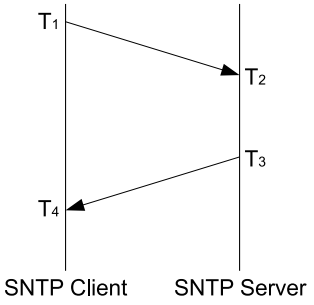

Fig. 1. On-wire (S)NTP protocol operation

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 [2,4]. 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 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.

As in PTP, the precision of the NTP synchronization can be largely improved by doing the time-stamping operation in lower layers [5,6,7]. 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. Thus, precision at the time client can be better than one microsecond [7].

3 Synchronization System Overview

The objective of the project leading to the design experience summarized in this contribution was to design a very compact and low cost SNTP client and server suitable for the scenario depicted in Fig. 2, where the SNTP server gathers the time from the GPS receiver and SNTP clients provide time and synchronization information to remote terminal units (RTU's). The SNTP server will use a standard GPS receiver as a time reference. It will use the PPS (Pulse Per Second) signal and NMEA [8] data from the GPS to synchronize its internal clock. The SNTP clients 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, emulating a GPS receiver. With this solution, there is no need to provide a GPS receiver with every RTU to avoid extra costs, complexity and solve the cases where the installation of a GPS antenna is not feasible. A summary of the system specifications follows:

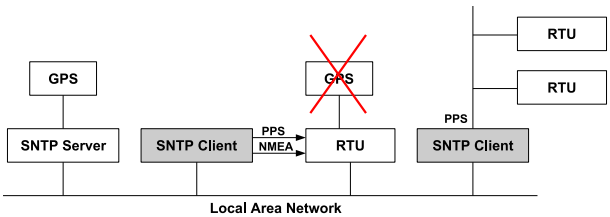

Fig. 2. Synchronization system overview

- Clients and servers should operate in a standard 10/100/1000 MHz Ethernet LAN.
- Clients and servers should gather their configuration parameters automatically using the BOOTP protocol [9] so that the configuration for all the clients and servers can be centralized in a single BOOTP server.
- The precision of the local clock at the clients and the server should be within 10 μs of a GPS reference in optimal conditions: low network load and direct LAN connection without switches. In typical conditions precision should be always within 1 ms.

– The whole client and server designs should fit in a single, low density FPGA chip and should need no additional hardware, so that the system can work stand-alone or embedded in a bigger system.
– Low power: implemented in a low density, low frequency FPGA, the client or server should consume under 1 W of average power.
– System clock frequency is 50 MHz.

After thoughtful consideration, it was decided to implement both the client and server completely in hardware without a software abstraction level. The most important reasons leading to this decision were that in order to achieve a high accuracy, some of the key parts of the system, like timestamping, need be done in hardware anyway; and that the lack of a processor and associated subsystems (RAM, file system, etc.) should give a good footprint in terms of resources and power consumption. In the rest of the section, the most important aspects of the design and implementation are commented. Diagrams of the modules that form the SNTP server and client are shown in Fig. 3. Server and client share the Ethernet MAC controller and the UART with no modifications. The protocol and configuration interface and the synchronization module are the most complex blocks and most of its functionality is shared between the client and the server. The transmission and receptions modules are specific to the server and client respectively. Next, we will briefly describe the functionality of these block.

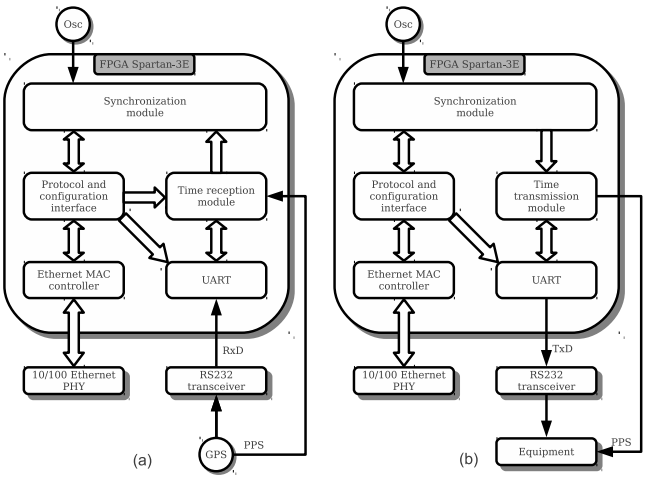

Fig. 3. SNTP client and server block diagram. a) server, b) client

The protocol and configuration interface is probably the most complex block in the system. It is in charge of handling the configuration process and the various communication protocols involved during the configuration and normal operation phases: BOOTP, IP, ARP, UDP and NTP. It also acts as the control unit of the system since many system tasks are triggered by the protocol

interface. At startup time, the configuration phase starts by requesting a configuration packet through the BOOTP protocol. A BOOTP response provides the client with an IP network address and mask client configuration parameters like the NTP server address, the poll interval, UART baud rate and some internal clock tunning parameters. After configuration, the interface enters normal operation and issues NTP requests at the configured poll intervals, and collect the responses. It is also in charge of registering and reading the timestamps of the NTP packets as they are processed. This information is then transferred to the synchronization module to make the necessary clock adjustments.

The task of the synchronization module is to maintain the local clock time as accurate as possible. In the client, when a NTP response arrives, the timestamps are transferred to the synchronization module that calculates the local clock offset according to eq. (1). In the server, the offset is calculated by using the reference provided by the GPS trhough the reception module. Then, the local time is corrected by introducing slight frequency variations to the local clock counter, so that a good frequency stability is achieved. The frequency control is done with the clock discipline algorithm published in [11] which provides both a high accurate control and a low time to synchronization. By design, the local clock resolution for the fractional part of the second is 22 bits (238 ns time resolution) which is intended to provide local clock accuracy in the range of 1 μs.

The transmission (server) and reception (client) modules handles the conversion between the local clock time format (that follows NTP standards) and NMEA-0183 Recommended Minimum sentence C (RMC) [8] frame format used to communicate with the GPS unit in the server, and with the external equipment in the client. The Ethernet MAC controller provides the standard functionality and is in charge of controlling a standard Fast Ethernet PHY device, allowing the transmission and reception of Ethernet frames conforming to IEEE 802.3 specification.

4 Platform and Tools

The systems are implemented on FPGA chips from Xilinx using Xilinx tools. Development has been done in a Digilent's Spartan 3E Starter Board [12] that includes a mid-range Xilinx Spartan-3E XC3S500E FPGA chip. Most of the design is coded in hardware description languages (mostly Verilog but also some VHDL) that has been synthesized using Xilinx's XST synthesizer. Xilinx's System Generator for DSP (SGDSP) [13] has also been extensively used to implement complex arithmetic operations.

The Ethernet MAC controller is the Tri-mode Ethernet MAC IP-core available from the OpenCores web portal [14]. The block is available as Verilog code and has been slightly customized to make a more efficient memory usage of the FPGA resources. The top-level design in the protocol and configuration interface and the synchronization module have been designed using SGDSP. This has facilitated a fast design and interconnection of processing blocks, registers and

memories. Controlling units in these blocks have been designed as state machines coded in Verilog as black boxes inside the SGDSP project. The reception and transmission modules have been implemented using custom time format converters coded in VHDL and a simple processing unit implemented using Picoblaze [15], a very simple soft microprocessor from Xilinx.

Verilog and VHDL code has been simulated using Xilinx's ISIM logic simulator included with the ISE design suit [16]. Inside SGDSP, ISIM is also used to simulate the user's black boxes. Otherwise, the high level simulation capabilities of SGDSP, that uses Matlab/Simulink have been extensively exploited for the associated modules, using various data sources and sinks (scopes) provided by SGDSP and custom Matlab scripts, which has greatly facilitated the simulation process. Xilinx's ChipScope [17] integrated logic analyzer has also been used for testing the FPGA chip during the first part of the development, specially to test and validate the clock discipline algorithms. Later, a custom logic analyzer named LEA (Logic Event Analyzer) [18] was developed for the long-term testing of the system to overcome the storage capabilities of ChipScope.

5 Results

In this section some preliminary results are presented in order to evaluate the system performance and fulfillment of specifications. First, Table 1 summarizes the hardware resources consumed by the implementation on a Xilinx's XC3S500E chip. Both client and server designs occupies less than 50% of the chip's resources except for the number of slices. Considering this is a low grade family we can say that hardware footprint is reasonably low and would be under the 10% of slice occupation in any mid-range Virtex-6 family of FPGA's of the same vendor. The resources are slightly higher in the client because of the transmission module, which needs a divider and other arithmetic blocks with an extra complexity when compared to the server's reception module.

Table 1. Hardware implementation results (FPGA Spartan-3E XC3S500E)

Resource	SNTP Client - Use (%)	SNTP Server - Use (%)
Slices	3615 (77 %)	2927 (62 %)
Slice Flip Flops	3753 (40 %)	2941 (31 %)
4-input LUT	4475 (48 %)	3424 (36 %)
RAMB16	5 (25 %)	5 (25 %)

Table 2 shows the accuracy of three servers as seen by the standard NTP software client running in a personal computer. The Internet server is a public NTP server available and located in another country, the commercial server Lantime M600 Network Time Server from Meinberg [19] and the prototype is an implementation of the full hardware SNTP server described in this contribution.

Both the commercial server and the prototype are connected to the same LAN than the software client. Delay measures the round-trip-time from the client to the server, offset is the displacement of the local clock and jitter measures the stability of the synchronization. As expected, both local servers provide much better synchronization than the Internet server. Also, both local servers give similar quality figures as seen by the client with slightly better results for our prototype.

Table 2. Software client synchronization results: estimated delay, offset and jitter, all in milliseconds

	Delay	Offset	Jitter
Internet server	33.89	−2.410	0.550
Commercial server	0.251	−0.044	0.054
Prototype	0.101	−0.013	0.045

Table 3 shows the mean offset and offset error as seen by the hardware client when synchronized to the hardware server operating in the same LAN, for various combinations of the poll interval exponent (p) and the attenuation factor (q). p controls the poll interval which is 2^p and q controls how aggressively the offset is corrected with softer corrections for higher values of q [11]. Nominal values of the implementation are $p = 0$ and $q = 2$. This mainly shows that the discipline algorithms implemented in the client are able to maintain a local clock accuracy below 1 μs which surpasses the initial specification of 10 μs.

Table 3. Hardware client-server synchronization. Mean offset ± error in microseconds.

	$q = 0$	$q = 1$	$q = 2$	$q = 3$
$p = 0$	0.06 ± 1.21	0.06 ± 1.17	0.07 ± 0.95	0.05 ± 0.85
$p = 2$	0.07 ± 1.71	0.05 ± 1.51	0.07 ± 1.69	0.20 ± 1.81
$p = 4$	0.04 ± 3.88	0.23 ± 4.34	1.01 ± 6.53	0.55 ± 14.86
$p = 6$	0.24 ± 13.18	0.59 ± 17.97	1.16 ± 35.29	9.19 ± 64.45

In order to test the performance of the server prototype it is loaded with a varying number of requests per second (rps) by injecting NTP traffic in the LAN. At the same time, the mean offset and offset error is collected from a client prototype. It has been checked that the synchronization accuracy is not affected with a low number of rps and that the server can easily handle 10000 rps maintaining an accuracy of 3 μs with a maximum rps estimated above 40000 rps. Typically, Meinberg equipment specifies a maximum of 10000 rps without mentioning the expected accuracy, while Symmetricom's [20] announces a time stamp accuracy of 14 μs under 3200 rps.

Finally, Table 4 estimates the power consumption and unit cost of various NTP server alternatives: a commercial server like the ones from Meinberg or Symmetricom, a software server implemented in an embedded computer like the Beagleboard [21] and our hardware SNTP server prototype. The stand-alone prototype power consumption is measured on a custom printed circuit board (PCB) that includes the FPGA chip and all the necessary peripherals. The reduced power consumption of the prototype is due to its much more simple hardware architecture compared to the other alternatives. The unit cost is also reduced since the prototype does not need additional devices like flash or RAM memory. Actually, if the prototype design is embedded in a bigger system as a soft IP-core, the power consumption and cost can be almost negligible.

Table 4. Power consumption and unit cost estimations for various NTP server implementation alternatives

	Commercial server	Embedded comp.	Prototype (stand-alone)	Prototype (embedded)
P (W)	20.4	$3,11$	0.624	≈ 0
Unit cost ($)	$6000 - 8000$	180	50	≈ 0

6 Conclusions

This contribution summarizes the implementation of a an embedded network time synchronization system completely implemented in FPGA hardware to illustrate the feasibility a full hardware implementation of high level functions typically found in the software layer. The accuracy of the system is in the same range of commercial equipment while the needed resources, power and cost is two orders of magnitude lower, at the cost of sacrificing some additional functionality and flexibility. The design is carried out using the Xilinx's tool set including DSP libraries, but only standard blocks and functions are used so it could be completely ported to a hardware description language to achieve vendor and technology independence.

This is a good study-case to supports the idea that high-level system functions tightly related to software like network protocols and network synchronization can be completely ported to hardware to obtain a very cheap yet much higher performance solution. This gain is mainly due to the simplification of the problem by removing some abstraction layers like the computer and the software abstractions. Newer design tools and the conception of a hardware operating system-like abstraction layer can make the full hardware approach widely available to a range of traditional software applications.

Acknowledgments. This work has been partially supported by the Ministerio de Ciencia e Innovacin of the Spanish Government under project TEC2011-27936 (HIPERSYS) and by the European Regional Development Found (ERDF).

References

1. PLDA: PLDA Home Page
2. Mills, D.L., Martin, J., Burbank, J., Kasch, W.: Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905 (Standards Track) (June 2010)
3. IEEE: IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. PTP Version 2 (1588-2008) (2008)
4. Mills, D.L.: Computer Network Time Synchronization: The Network Time Protocol. CRC Press, Inc., Boca Raton (2006)
5. Skeie, T., Johannessen, S., Løkstad, T., Holmeide, Ø.: Same time - Different place. ABB Review (2), 9–14 (2003)
6. Johannessen, S.: Time Synchronization in a Local Area Network. IEEE Control Systems Magazine 24(2), 61–69 (2004)
7. Holmeide, Ø., Skeie, T.: Synchronised: Switching. IET Computing and Control Engineering 17(2), 42–47 (2006)
8. NMEA: NMEA 0183 Standard. NMEA 0183 V 4.00 (January 2002)
9. Croft, W.J., Gilmore, J.: Bootstrap Protocol. RFC 951 (Draft Standard) (September 1985) Updated by RFCs 1395, 1497, 1532, 1542
10. Plummer, D.: Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826 (Standard) (November 1982) Updated by RFC 5227
11. Viejo, J., Juan, J., Bellido, M., Millan, A., Ruiz-de Clavijo, P.: Fast-convergence microsecond-accurate clock discipline algorithm for hardware implementation. IEEE Transactions on Instrumentation and Measurement 60(12), 3961–3963 (2011)
12. Digilent: Digilent Home Page
13. Xilinx: System Generator for DSP Getting Started Guide Release 10.1. Xilinx, Inc. (March 2008)
14. OpenCores: OpenCores Home Page
15. Chapman, K.: PicoBlaze 8-Bit Embedded Microcontroller User Guide for Spartan-3, Virtex-II and Virtex-II PRO FPGAs. Xilinx, Inc. (November 2005)
16. Xilinx: ISE In-Depth Tutorial. Xilinx, Inc. (March 2011)
17. Xilinx: ChipScope Pro 11.1 Software and Cores User Guide. Xilinx, Inc. (April 2009)
18. Viejo, J., Villar, J., Juan, J., Millan, A., Ostua, E., Quiros, J.: Long-term on-chip verification of systems with logical events scattered in time. Microprocessors and Microsystems 33(5), 402–408 (2012)
19. Meinberg: Meinberg Funkuhren GmbH & Co. KG Home Page
20. Symmetricom: Symmetricom Home Page
21. BeagleBoard.org: BeagleBoard Home Page