IMPLEMENTATION OF A CONFIGURATION SERVER FOR A HARDWARE SNTP SYNCHRONIZATION PLATFORM BASED ON FPGA

J. Quiros, J. Viejo, A. Millan, A. Muñoz, J. I. Villar and D. Guerrero

Grupo ID2 (Investigación y Desarrollo Digital)
Departamento de Tecnología Electrónica - Universidad de Sevilla
E. T. S. Ing. Informática, Campus Universitario Reina Mercedes
41012 Sevilla (SPAIN)
Email: jquiros@dte.us.es, julian@dte.us.es, amillan@us.es, amrivera@dte.us.es, jose@dte.us.es, guerre@dte.us.es

ABSTRACT

This paper presents the implementation of a configuration server for a SNTP synchronization platform which implements accurate synchronization solutions for Remote Terminal Units commonly used in industrial control processes. The configuration server provides settings to others platform devices using the BOOTP protocol and an interface that allow to administer the system. This environment requires a compact (specific dimensions) and low power and low cost device. Thus, a general purpose device (e.g. a PC) is discarded and an embedded one with these features has been developed. However, in addition to these requirements it offers a flexibility similar to the PC. Thereby it is able to update and carry out tasks beyond synchronization platform easily.

1. INTRODUCTION

The time synchronization is critical in a great wide industrial processes. Depending on the application, a precision of up to few microseconds can be required. This problem has been tackled by the industry, which has developed systems to resolve it. However, these solutions are based on software and need complex and expensive hardware to achieve an acceptable accuracy.

In this sense, a synchronization platform based on the industry norm IEC 61850 [1], which defines the Simple Network Time Protocol (SNTP) [2] over Ethernet as a standard way to synchronize a set of substations with a time server, has been fully developed in hardware, achieving a low cost and power, compact and accurate platform which provides synchronization in the range of the microseconds [3, 4].

The SNTP synchronization platform consists of three types of devices: SNTP server, SNTP client and configuration server (Fig. 1). It can be used in industrial environments where there are several Remote Terminal Units (RTUs) which require time synchronization. The SNTP server receives accuracy time information from a Global Positioning System (GPS) device. The SNTP clients in turn synchronize with the server through Local Area Network (LAN) using the SNTP protocol, and transmit the time information to RTUs emulating a GPS device: SNTP clients generate the GPS signals required, Pulse per second (PPS) signal and Recommended Minimum Navigation Information (RMC) frame [5, 6].

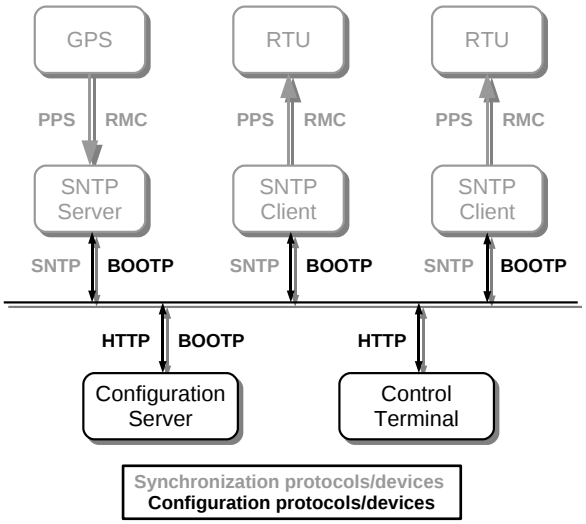

Fig. 1. Typical scenario for deploying hardware SNTP client and servers (synchronization devices and protocols are marked in gray, and configuration devices and protocols are marked in black).

SNTP is a protocol of the Application Layer of the TCP/IP model. Thereby a correct configuration of the lower layers (Network Interface, Internet and Transport) is necessary for a successful communication. Media Access Control (MAC)

address is assigned and stored at bitstream generation time and the NTP standard port (123 UDP) is used. Thus, Internet Layer configuration is only needed to be configured at operation time. The configuration required by the communication between GPS and SNTP server is the default speed of the serial port used to transmit RMC frame and when the PPS signal indicates the start of a second: rising or falling edge. When the SNTP client emulates the GPS, the start of a second is always indicated by the rising edge of the PPS signal, so the default speed of the serial port is just needed.

Therefore, it is necessary a method to configure all platform devices. Because all platform devices are connected throught Ethernet, Bootstrap Protocol (BOOTP) will be the protocol used for this purpose. This functionality is implemented by the configuration server, which must have the same features than the platform devices. In this way, the design can be based on a general purpose device (e.g. a PC) or on a dedicated one. The first is more expensive and higher power consumer than a dedicated device, furthermore adapting it to specific dimensions is more difficult than the last one. Thereby, an embedded device is the best option, but a fully hardware implementation is inflexible: it often presents long development and support time and cost. So that, the configuration server implementation is based on a System on Chip (SoC) design which share the best features of the last two designs mentioned. On the one hand, it presents a flexibility similar to a PC: reduced development time and cost, it is easily updated and even of being able to carry out functionalities beyond the synchronization platform. On the other hand, it is an embedded device, so it fulfills power consumption and size requeriments. Moreover, the cost can be reduced if the developed design shares most hardware elements with other platform devices.

This paper describes the configuration server implementation and is organized as follows: in section 2 some concepts of BOOTP protocol are presented, section 3 enumerates the device specifications, section 4 gives some design and implementation details, section 5 includes some results, and section 6 discusses some conclusions.

2. BASIC CONCEPTS OF BOOTP PROTOCOL

The BOOTP is a network protocol based on the client-server model, and it is used by a device (client) to obtain its Internet Protocol (IP) settings [7]. Its detailed operation is described below (Fig. 2).

When a client needs IP configuration it sends a BOOTP request with the broadcast address as destination address. This packet is received and processed by the BOOTP server, which keeps a pool of settings. If it has an entry for this client, which is identified with its MAC address, the server will send a BOOT reply with the client IP settings. Otherwise, the server will not answer (Fig. 2). This protocol provides not only IP settings, the packet format includes other configuration fields [7, 8] as the boot file name used to boot through network.

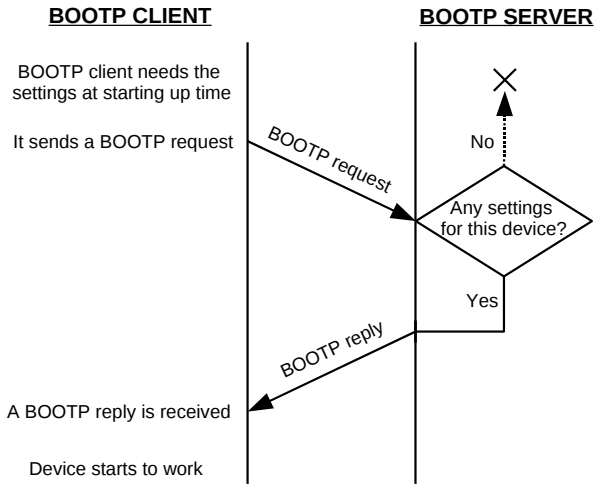

Fig. 2. Operation of the BOOTP Protocol.

Historically, this protocol has been used by devices at starting up time to obtain the needed settings to connect a server and download the Operative System (OS) image to execute it. Moreover, the BOOTP protocol can be used to obtain only an IP configuration and the requests can be sent by a client anytime. In this way, a dynamic IP address assignment can be implemented sending periodic BOOTP requests, although the protocol was designed to provide a static IP address assignment. Nowadays, another protocol has limited the use of BOOTP: Dynamic Host Configuration Protocol (DHCP) [9]. It offers more functionality than BOOTP (e.g. dynamic IP address assignment), but its implementation is more complex. However, DHCP and BOOTP can be used in the same network because most DHCP servers are compatible with BOOTP. In our platform, the SNTP clients and server are fully implemented in hardware, so the BOOTP is the configuration protocol chosen because it reduces the complexity and implementation time. Furthermore, it is possible to modify some aspects of the protocol to transmit more information with minimal changes in the clients.

3. SYSTEM SPECIFICATION

The objective is to develop a configuration server which have to establish two processes of communication. On the one hand, it will configure the other devices of the platform using the BOOTP protocol through the LAN. On the other hand, it will allow to administer the system through an user interface.

Below a more detailed system specification is listed:

- The configuration server hardware must be as similar as possible to SNTP client/server hardware. In this sense, the same hardware design, which is based on a SPARTAN-3E XC3S500E FPGA, has been used. Therefore, it was just necessary to add a FLASH memory (non-volatile memory) and a SDRAM memory module in order to achieve a hardware which be able to run an OS. Thereby a more flexible, cheaper and easier to assembly hardware is obtained, due to all hardware design is reused, adding the needed memories is considered as an assembly option, and any platform device can be implemented on the server hardware, it is only necessary to change the design synthesized on FPGA.

- The SoC will be implemented on FPGA. An OS will be executed by it to achieve a reduction in the development and support time.

- The BOOTP protocol will be used in the communication with SNTP clients and server. So, the server has to execute a BOOTP server.

- The device will administer the synchronization platform through a web application. A HTTP server is required for this purpose.

- The implemented web application must have the following features:

  - It will be based on configuration profiles.
  - It will check and verify new settings.
  - It will store all settings in a non-volatile memory.

- The devices will be identified by its MAC. Therefore, the server has to remember the MAC of all devices that send a BOOTP request and has not been configured previously.

- The interconnection between administration interface (web application), BOOTP server and HTTP server must be implemented. In that way, the system will be autonomous.

The developed device conforms with these requeriments and, in addition, it is possible to add new funcionalities to it. In the next sections, the development and the results of the configuration server are showed.

4. SYSTEM DESIGN AND IMPLEMENTATION

The hardware is based on a SPARTAN-3E XC3S500E FPGA. The SoC synthesized on this device has been designed with XPS software and it is based on XILINX IP Cores which are optimized to be implemented on XILINX devices, so MicroBlaze is the system processor. Petalinux has been the chosen embedded Linux distribution and uClinux [10] the Linux kernel.

The server functionality is implemented in software where is divided into three modules which are detailed below (Fig. 3).

Administration interface makes up the first module. This communication is based on a web application using the HTTP protocol. Thus, the configuration information is administered through a web browser. Two Common Gateway Interfaces (CGIs) have been developed in C: one of them (*Load CGI*) starts the process sending the web application and the current system configuration, and the second one (*Apply CGI*) receives, checks and applies the modifications. All the settings are checked in the application (client side) as well. This last feature is implemented in JavaScript using the *jQuery* library to achieve a cross-browser code. When the new configuration is completed, this application sends and sets it up reducing the amount of FLASH writings (intermediate settings are not saved). Finally, the *thttpd* web server has been the HTTP web server used.

The second module consists of the interface between server and others platform devices. The settings provided by the server depends on the type of device: SNTP server or client. Internet Layer configuration and default speed serial of the port are required by them both. In addition to these settings, SNTP server needs the active edge of the PPS signal, and the SNTP client needs the SNTP server IP address and the interval time between SNTP requests. The BOOTP protocol is used for this purpose. The packet format defined by this protocol has fields to transmit Internet Layer configuration (IP address, netmask, and default gateway), but it do not have fields to platform specific information. However, this settings are send in the boot file name field. The settings are transmitted as a string of hexadecimal digits codified as ASCII characters which will be different depending on the type of device. This method only modifies the semantic of a field producing valid BOOTP packets. So the server can operate with standard BOOTP clients. The *Internet Systems Consortium (ISC) DHCP software*, which is a DHCP and BOOTP server, is the BOOTP server used.

Finally, the third module defines the interconnection between the two interfaces previously described (Fig. 3). In this sense, web application must be able to read and modify the current settings, which are used by the BOOTP server to configure the platform devices. These settings are saved in the BOOTP server configuration file, which is located on the non-volatile memory. At starting up time, the server checks and verifies this file to insure the system operation. The functionality of showing the current configuration is implemented by the Load CGI, which reads and parses this file. When the web application sends new settings to be saved

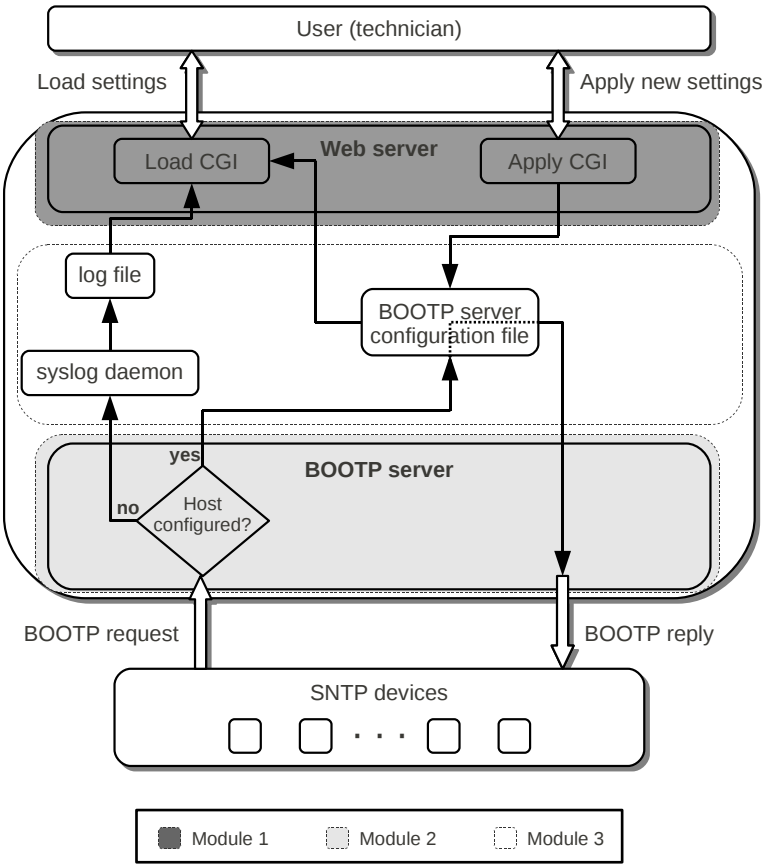

Fig. 3. Overall diagram of the configuration server operation.

and applied, *Apply CGI* checks the validity and coherence of the new configuration and then generates the new BOOTP server configuration file. Moreover, the server must save all addresses of platform devices, because the devices are identified with its MAC address. In this way, the MAC of the known devices are located in the BOOTP server configuration file. If an unknown device sends a BOOTP request its address has to be saved. The BOOTP server has been configured to send log information to *syslog daemon*, which in turn writes this information to a log file. *Load CGI* reads and parses this file to show unknown devices addresses. Furthermore, the system shows the relative and absolute date and time of the first BOOTP request of each device.

5. RESULTS

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

Resource	Usage (%)
Slices	4238 (91%)
Slices Flip Flops	4958 (53%)
4 Input LUTs	7444 (79%)
Bonded IOBs	75 (47%)
Block RAMs	19 (95%)
GCLKs	6 (25%)
Maximum operation frequency	75.857 MHz

Table 1. Hardware implementation results on SPARTAN-3E XC3S500E.

5.1. Hardware results

The SoC design has been implemented on a SPARTAN-3E XC3S500E FPGA. The Table 1 shows the design results. The maximum operation frequency (75 MHz) is enough, due to it is over PCB clock frequency (50 MHz). It is remarkable that 95% of Block RAMs are used by the design. Therefore, the Memory Management Unit (MMU) support in MicroBlaze Processor can not be used, due to the fact

Usage of RAM memory		
Total usage of RAM memory	**Item**	**Maximum memory usage in KB (%)**
	Total Memory	28900 (100%)
	Wating a BOOTP/HTTP request	11344 (39.25%)
	Maximum load (Saving settings)	11628.28 (40.24%)
Usage of RAM memory by processes	**Process**	**Maximum memory usage in KB (%)**
	BOOTP server	578.336 (2.00%)
	HTTP server	292.916 (1.01%)
	Load CGI	141.492 (0.49%)
	Save GCI	271.248 (0.94%)
	Total	1283.992 (4.44%)

Usage of FLASH memory		
Usage of FLASH memory	**Item**	**Memory usage in KB (%)**
	Total Memory	15625 (100%)
	System loader	800 (5.12%)
	Operative System	4000 (25.6%)
	User Flash partition	8000 (51.2%)
	Total	12800 (81.92%)

Table 2. Usage of RAM and FLASH memory.

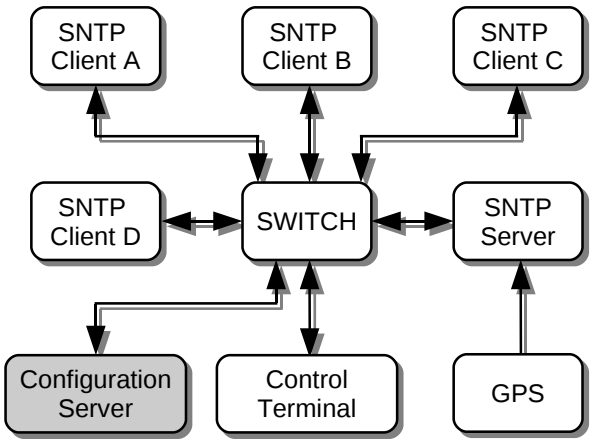

Fig. 4. Environment used to test the configuration server.

that four additional Block RAMs are required for this purpose. For this reason a non-MMU support Linux Kernel (uClinux) has been used. This is because the server hardware should be as similar as possible to hardware of other platform devices, and a low cost FPGA is used by them. If the immediately above device in the Spartan-3E family is chosen, the MicroBlaze Processor with MMU support may be synthesized and a full Linux kernel (with MMU support) may be used. Moreover, the usage of Slices will be below 50% and it would be possible to synthesize other hardware functionalities on the same device.

5.2. Software results

The environment where the tests took place is showed by Fig. 4, where all devices were turned on and the settings were modified along the time to check stability, possible errors and memory usage of the system. Fig. 5 shows the web interface of the configuration server. The CPU results are not relevant because the configuration process is not a critical task, so they are omitted. Regarding to RAM memory usage (Table 2), it is remarkable that more than 50% of the available memory is free. This fact allows to add new software implemented functionalities to the system. The 82% of FLASH memory is used by the system (Table 2). However, the user FLASH partition size can be reduced if more FLASH memory is required by the OS image. Settings for a new device require $\approx$ 99 bytes and for a new configuration profile $\approx$ 64 bytes. Thereby, if partition size is reduced by half, the system can continue working, but a fit size can reduce the FLASH life.

6. CONCLUSION

The design and implementation of a configuration server for the SNTP synchronization platform has been presented. It has been based on a SoC synthesized on a FPGA, which executes an OS. The functionality is implemented in software to achieve a flexible, low power and low cost and compact device that it is not limited to the platform environment being able to carry out tasks beyond it.

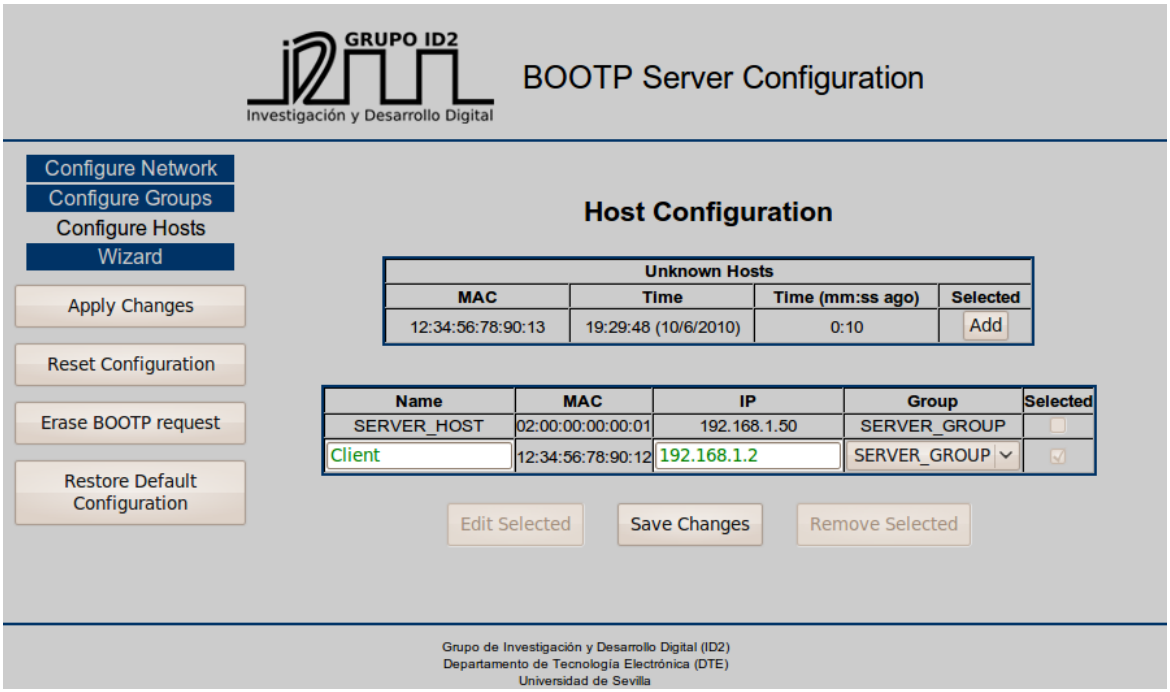

Fig. 5. Web interface of the configuration server.

7. 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.

8. REFERENCES

[1] H. Dawidczak, "IEC 61850 Communication Networks and Systems In Substations," International Electrotechnical Commission and Technical Committee 57, 1995.

[2] D. Mills, "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI," Internet Engineering Task Force (IETF), RFC 4330 (Informational), Jan. 2006.

[3] J. Viejo, J. Juan, M. J. Bellido, E. Ostua, A. Millan, P. R. de Clavijo, A. Muñoz, and D. Guerrero, "Design and implementation of a SNTP client on FPGA," in *Proc. 2008 IEEE International Symposium on Industrial Electronics (ISIE)*, Cambridge (United Kingdom), July 2008.

[4] J. Viejo, J. Juan, E. Ostua, M. J. Bellido, A. Millan, A. Muñoz, and J. I. Villar, "Accurate and compact implementation of a hardware SNTP Client," in *Proc. 15th Iberchip Workshop (IWS)*, Buenos Aires (Argentina), Mar. 2009.

[5] J. Quiros, J. Viejo, A. Muñoz, A. Millan, E. Ostua, and J. I. Villar, "Implementación sobre FPGA de un cliente SNTP usando MicroBlaze," in *Proc. 16th Iberchip Workshop (IWS)*, Iguazu Falls (Brazil), Feb. 2010.

[6] E. Ostua, M. J. Bellido, J. Viejo, A. Millan, A. Muñoz, and D. Guerrero, "Aplicación de Picoblaze como Emulador/Receptor de un GPS en el diseño hardware de un cliente/servidor SNTP," in *9th Jornadas de Computación Reconfigurable y Aplicaciones (JCRA)*, Madrid (Spain), Sept. 2009.

[7] W. J. Croft and J. Gilmore, "Bootstrap Protocol," Internet Engineering Task Force (IETF), RFC 951 (Draft Standard), Sept. 1985, updated by RFCs 1395, 1497, 1532, 1542.

[8] R. Droms and S. Alexander, "DHCP Options and BOOTP Vendor Extensions," Internet Engineering Task Force (IETF), RFC 2132 (Standards Track), Mar. 1997.

[9] ——, "Dynamic Host Configuration Protocol," Internet Engineering Task Force (IETF), RFC 2131 (DRAFT STANDARD), Mar. 1997.

[10] R. Klenke, "Experiences using the xilinx microblaze softcore processor and uclinux in computer engineering capstone senior design projects," in *Microelectronic Systems Education, 2007. MSE '07. IEEE International Conference on*, June 2007, pp. 123 – 124.