Using the CAN Protocol and Reconfigurable Computing Technology for Web-Based Smart House Automation

Fernando Moraes, Alexandre Amory, Ney Calazans, Eduardo Bezerra, Juracy Petrini

Pontifícia Universidade Católica do Rio Grande do Sul (FACIN-PUCRS)
Av. Ipiranga, 6681 - Prédio 30 / BLOCO 4
90619-900 - Porto Alegre – RS – BRASIL
{moraes, amory, calazans, eduardob}@inf.pucrs.br

Abstract

This paper presents the hardware implementation of a multiplatform control system for house automation using FPGAs. Such a system belongs to a domain usually named domotics or smart house systems. The approach combines hardware and software technologies. The system is controlled through the Internet and the home devices being connected use the CAN control protocol. Users can remotely control their houses using a web browser (Client). Locally, instructions received from the Client are translated by the Server, which distributes the commands to the domestic appliances. The implemented system has the following characteristics, which distinguish it from existing approaches: (i) the client interface is automatically updated; (ii) a standard communication protocol (CAN) is used in the hardware implementation, providing reliability and error control; (iii) new appliances are easily inserted in the system; (iv) system security is provided by user authentication; (v) user rights can be set up by an administration interface.

1. Introduction

Smart house is the integration of technologies and services applied to home, office and small buildings environments, with the purpose of acquiring enhanced safety, comfort, communication and power saving with less human interaction [1] [3]. Teleaction is the ability to control devices remotely [3].

Combining smart house and teleaction it is possible to associate the internal network of a house with the external world network (Internet). This association allows controlling and administrating houses remotely.

The benefits of a *smart house system* (SHS) can be classified in four groups: safety, comfort, power saving and communication.

To make possible the implementation of a SHS, there are some technical requirements that must be respected: low cost (easy to install and cheap devices), plug and play, flexibility (modular and extensible system), easiness of use and reliability. The system presented in this paper contemplates these requirements (some of them partially).

To illustrate the application scope of this work, we present, in Figure 1, some possible situations and adequate actions in a house with a SHS connected to the Internet. For example, in situation 1, Mary is leaving her workplace. She could use her wireless PDA (personal digital assistant) with access to the Internet to connect to her house homepage and adjust the temperature of the bathtub and illuminate it before she arrives home. In situation 2, the oil level sensor of the heating system indicates that refueling is needed. Immediately, the system sends a message to the oil distribution service requiring refueling. In situation 3, John is working in his office when suddenly he receives an urgent message communicating an invasion at his home. Immediately, he connects to his house homepage and, through the security cameras, he may confirm the presence of a thief. In parallel, the system may already have sent a message to the police. In situation 4, John and Mary are going to the movies. When they are getting out, the system sends a message to Mary's PDA, indicating that they may have forgot a window opened.

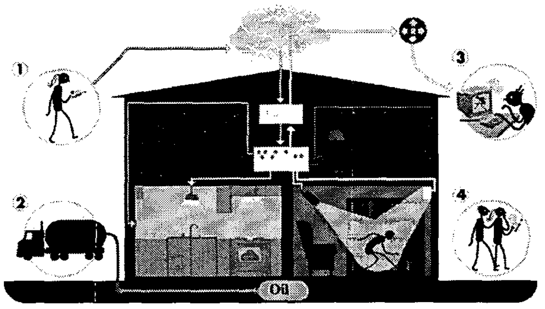

Figure 1. Possible application to this project.

These situations may seem far from reality, but they are not. Nowadays, there are PDAs and cellular phones with access to Internet and, during this work, we found some systems (see references [4] to [7]) with objectives similar to ours. This shows that there are technologies converging to the same objective of this project.

The main motivation for this work is the fact that SHS may become a mass-product, once the associated technologies evolve to provide low cost implementations. Additional motivations are the challenge of developing an application with hardware/software/Internet integration and creating new applications to personal computers and Internet.

The main goal of the paper is to present the hardware part implementation of a SHS. A previous publication [2] exploits the system implementation with emphasis in the software part. Section 2 describes the general architecture of the system. Section 3 describes the hardware implementation, presenting the local controller, the master controller and the application protocol and also describes techniques and tools used to verify the hardware of the system. Finally, Section 4 presents some conclusions and directions for future work.

2. System Architecture Overview

Figure 2 shows the block diagram of the SHS system architecture, where there is a *server computer*, at least one *client computer*, and a network of home appliances. Client computers are connected to the house server by means of the HTTP protocol. The Controller Area Network (CAN) protocol ([11] and [12]) is used to connect the server to the *home appliances network* (HAN). The communication takes place when the *master controller* of the HAN exchanges data with the server through a serial connection.

Users (clients) send control signals to the server computer using a web browser in a client computer [1][2]. These signals are called *control packets*. The client keeps its interface updated according to the status of the home appliances. The server receives control packets from clients (Figure 2-1), interprets the received packets (Figure 2-2), updates the data base (Figure 2-4) and sends the control packets (Figure 2-3) to the master controller, (Figure 2-5) controlling the home appliances (Figure 2-6).

The web server Apache, running on the server computer, provides the HTML interface to the clients. A program written using the PHP language [8] (Figure 2-2) receives control packets from clients using the web browser, updates the database (Figure 2-4) on the server and improves system security by using data authentication and encryption with the MD5 algorithm. Another function performed by the PHP program is to keep updated the system interface (the client homepage), using information stored in the database. This action is transparent to the user. The database is automatically updated when the status of any home appliance changes. An advantage of this strategy is that there is no need for the use of the browser refresh button. Another one is that the SHS can automatically send alarms to the inhabitants.

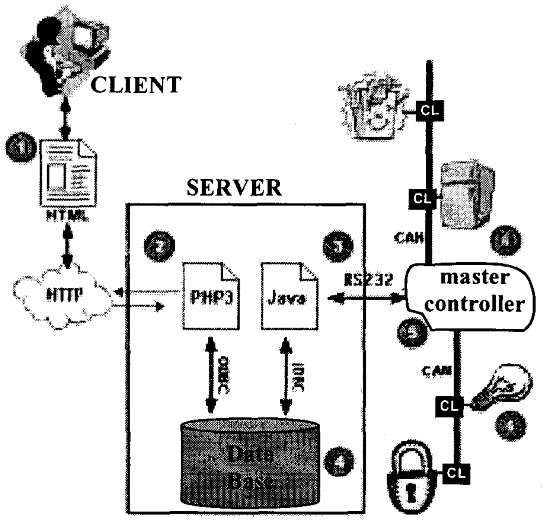

Figure 2. General SHS structure.

There is a Java application [9] running on the server (Figure 2-3) which is responsible for the serial communication [10] between the master controller and the server itself. Another function of the Java application is to guarantee the consistency between the database and the status of the home appliances.

The master controller is the interface between the server and the CAN bus, translating serial and CAN frames.

A serial bus installed in the house connects all the home appliances. The server (PC) together with the CAN protocol, is the basis for the home automation. Each appliance has its own *local controller* (CL in Figure 2-6), used to provide an interface with the bus. The CAN protocol is implemented using an IP soft Core described in VHDL [13].

3. Hardware Implementation

The SHS is written in VHDL [14] and implemented using FPGAs. An important motivation for using FPGAs is the availability of ready to use soft IP cores. The use of IP cores contributes to decrease the complexity of the implementation. The hardware implementation was divided into six steps: control protocol choice, control protocol characteristics, and the implementations of the user application node, the master controller and the application protocol. The steps are detailed in the next sections.

3.1. Control Protocol Choice

The choice of the features of a protocol depends on its application field. A set of features and requirements should be evaluated to select an adequate protocol. Examples of such features are [1][12]: the cost/benefit tradeoff of the cabling system; reliability, availability, flexibility and standardization; communication bandwidth, real time support, topology, maximum physical length of the nets, maximal number of bytes in the data frame.

Several control protocols have been studied in order to select the most adequate for this project. The studied control protocols were EHS, BDLC, LON, EIB, X-10, CeBus and CAN. Details of this research and a comparative table for these protocols can be found in [1]. From the comparison of the protocols arose the choice of the CAN protocol, to be discussed next.

3.2. Controller Area Network (CAN) Characteristics

CAN ([11] and [12]) is a serial control protocol that supports distributed real time applications and has an adequate security level for the objectives of this work. This protocol was originally developed for automobile applications, but nowadays there are several other applications where it is used. Examples are industrial control systems and embedded applications such as medical systems. One of the originality of this work resides in the use of CAN in SHS.

The main features of the CAN protocol, are: priority levels to access the bus; bandwidth up to 1 Mbps; support to real time; configuration flexibility; support to message multicasting; techniques to keep data consistency (to detect and signaling errors); support for multi-master communication; support for remote data requesting; messages with up to 8 data bytes. Figure 3 illustrates the CAN frame.

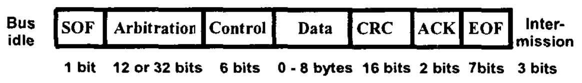

Figure 3. CAN frame.

The IP Core HurriCANe has been adapted to be used to implement the CAN protocol in this project. HurriCANe is written in VHDL and developed by the European Space Agency (ESA) [13]. Figure 4 shows the hierarchical structure of HurriCANe. The numbers in each module indicate the typical amount of slices (an equivalent area measure) for a Virtex XCV300 Xilinx FPGA device [17].

The CAN interface module is a standard interface to a microcontroller or microprocessor. As all of our system is implemented in FPGA, the CAN controller and the CAN interface modules were not used in this work. The module used is the CAN Core and its hierarchy.

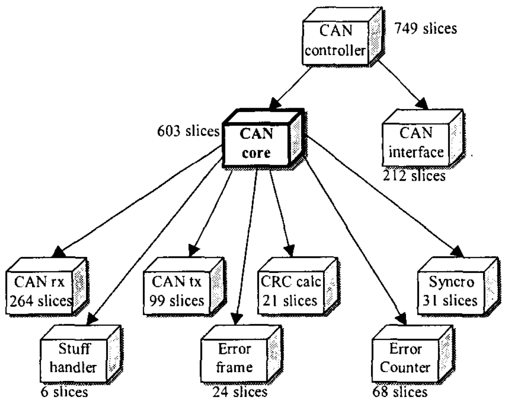

Figure 4. HurriCANe structure.

3.3. User Application Nodes Implementation

The *local controllers*, also called *user application nodes*, are hardware modules responsible for controlling the home appliances. They are distributed through the house and may be embedded within each appliance. Examples would be lamp dimmers, temperature sensors, video cameras, and air conditioning control. The functionality of a local controller varies according to its application.

For each home appliance there must be a hardware device and a software procedure responsible for integrating the appliance into the SHS. For example, in order to control lamps, it is necessary a hardware device that interfaces the lamp to the CAN bus and software in the server to manage lamps. A lamp control application has been implemented as a case study.

In the case study, the local controller was implemented in the Xess XS40 prototyping platform [15], which has a XC4010 Xilinx FPGA on it. This node supports up to 64 lamps, and each bit of a CAN frame data field represents the status of one lamp (see Figure 3). The structure of a lamp local controller is depicted in Figure 5.

The *CAN Core* module is the interface between the CAN bus and the lamp control logic. When a CAN frame is received, the *Data Verification* module is activated. It verifies the frame destination comparing the identification of the node to the identification received. If they are the same, the *Lamp Register* will be written, activating the lamps according to the information in the data field. The *Alteration Detection* module detects changes in the switches (a local user action). When a change occurs, it sends a new message to the master to update the status of the lamps in the server database. So, there are two ways to change a lamp state: (*i*) manually, as a result of a local user action and (*ii*) remotely, executed by the system as a result, for example, from a remote user action.

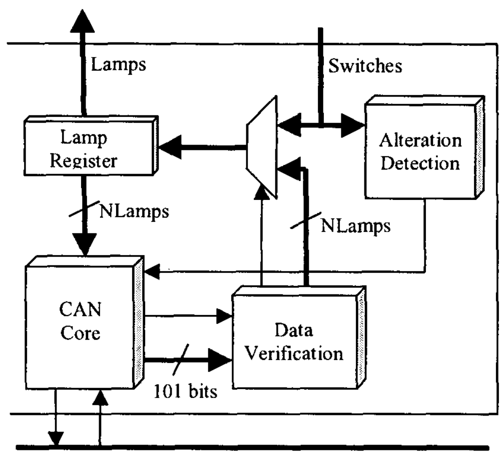

Figure 5. Architecture of the lamp local controller.

Relevant results obtained from the synthesis for a XC4010E-3-PC84 Xilinx FPGA [15] are the CLB occupation (294 CLBs, corresponding to 73% of FPGA utilization) and the estimated maximum frequency (9.018 MHz). This shows that a typical local controller can be implemented in small, low cost FPGAs. This is in accordance to the motivation stated in Section 1.

3.4. Master Controller Implementation

The *master controller* interfaces the CAN network and the server. It was implemented in VHDL in the VW300 [16] prototyping platform with a XCV300-5-BG352 [17] FPGA which supports digital systems with up to 300.000 logic gates.

The architecture of the master controller is shown in the Figure 6. Due to the distinct transmission data rates of the serial interface (up to 115 Kbps) and the CAN bus (up to 1 Mbps), the master controller is built around a FIFO-based hardware structure. When the CAN interface receives a frame, it is divided in bytes (up to 13, comprising the arbitration, data and control fields of the CAN frame, according to Figure 3). This information is stored in the CAN FIFO. While the CAN FIFO is not empty, the serial interface transmits data from the CAN FIFO through the serial port. In the opposite direction, the serial interface receives data from the server and stores it in the Serial FIFO. The main difference in the FIFOs management comes from the distinct size of serial and CAN frames. A CAN frame has variable length (5-13 bytes) and a serial frame has fixed length (1 byte). The transmission of a CAN frame by the CAN Interface occurs only after the whole frame is stored in the Serial FIFO.

The FIFOs are implemented using internal FPGA RAM blocks, called *Block Select Ram*. In this implementation, such blocks are configured as a dual port RAM (one for reading and other for writing) with 512 words of 8 bits, allowing to simultaneously store up to 32 CAN frames. For practical reasons frames are stored aligned in 16-byte frontiers. The *Serial Interface* sends and receives data to/from the server.

The most relevant signals of a CAN to serial conversion are presented in the simulation shown in Figure 7. The conversion occurs when the master node receives a valid CAN frame, which must be sent to the server. In Figure 7, the simulation shows two 5-byte CAN frames being received. The meaning of each signal is listed below:

- *CANBitStream* represents the CAN data being received;
- *SerialDataIn* is the Hexadecimal encoding and the *SerialTX* is the binary encoding of the serial data being transmitted to the server;
- *CANBusy* is in high logic level when the CAN interface is busy;
- *SerialTXBusy* is in high logic level when the transmission serial interface is busy;
- *FIFOCANEmpty* indicates when the CAN FIFO has no more data to be transmitted through the serial interface.

To indicate the end of a CAN frame reception, the signal *CANBusy* goes low. When the Serial interface is idle and the CAN FIFO is not empty (*FIFOCANEmpty* in low logic level), the serial interface starts a transmission through pin *SerialTX*. When a Serial transmission starts, the *SerialTXBusy* goes high until the transmission of a whole byte. When the serial interface finishes the transmission of the data in CAN FIFO, the *FIFOCANEmpty* goes high.

Table 1 presents an abstract of the meaningful information generated by the synthesis tool. This information is related to the XCV300-5-BG352 Xilinx FPGA.

Again, this shows that a typical master node, although not very small, can also be implemented in low cost FPGAs, e.g. in devices of the Spartan II family of Xilinx.

Table 1. Abstract of the master node synthesis results report.

Modules	Usage (Slices)	Usage (%)	Max. Freq. (MHz)
Master (Total)	731	23	26.134
CAN Interface	512	16	25.028
Serial Interface	65	2	85.434
FIFOs	116	3	37.448

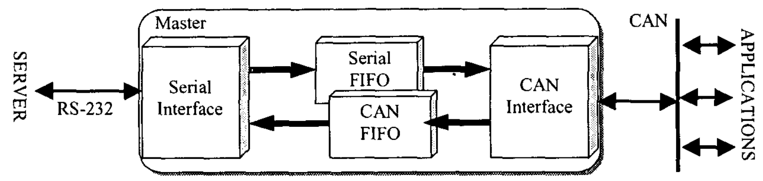

Figure 6. Architecture of master node.

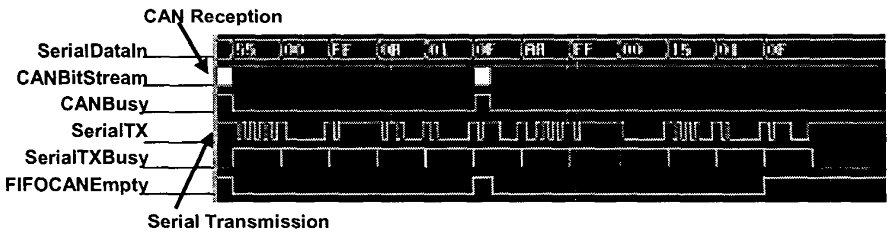

Figure 7. CAN to serial conversion simulation results.

3.5. Application Protocol Implementation

The CAN protocol has a limitation to be applied to SHSs, namely it can transmit at most 8 data bytes per frame. If a SHS is expected to provide multimedia services like image transmission, higher level protocols must be implemented. The authors suggest a mechanism to support transmission of complex information, such as video, images and sound using CAN. The basic idea is to divide CAN frames in two classes: data and control frames. These classes are distinguished by the node identifier (node_id), a sub-field of the arbitration field contents of the frame [1]. An odd node_id identifies control messages, while an even node_id identifies data messages.

The use of this technique gives enhanced flexibility to the SHS. In a control frame there are 8 data bytes, 1 is used to identify the service class and 7 are used to store a parameter, which is dependent on the specific service class and the application node. For example, a local controller controlling a camera can send a control frame to the server informing that a transmission of an image with 100 Kbytes follows. Another example is a transmission of a control message from the server computer to a lamp local controller asking for a self-test of the node. In this way, the system supports up to 256 service classes to each application. For example, if a node requires a transmission of more than 8 bytes of information, the length in bytes of this information may be specified in the 7-byte parameter sub-field. Thus, the hardware supports a theoretical transmission of data bundle with up to 2^{56} bits of data, for this hypothetical service.

It is important to note that the use of this technique preserves compatibility with the CAN protocol. These additions are implemented in a higher abstraction level on top of the CAN protocol.

3.6. System Verification

The validation tools used to debug the SHS prototype are the QHSIM VHDL simulator, the Tektronix TDS 220 oscilloscope and the HP 16663E logic analyzer. The following resources were employed: (*i*) a personal computer, used as server computer; (*ii*) an oscilloscope, for monitoring the CAN bus; (*iii*) a logic analyzer, to check some master node signals; (*iv*) the VW300 [16] prototyping platform, containing the master controller; (*v*) the Xess XS40 [15] prototyping platform, containing the lamp local controller application.

Two decisions were taken to facilitate hardware verification. The first one is to adopt the *use of a pre-tested IP Core to implement the CAN protocol* instead of implementing our own CAN description. This decision decreased the complexity and the development time of the system. The second decision is the use of structured techniques to implement the testbench [18][19].

The testbench contains a set of *test patterns* or, in other words, a set of predefined data representing the input of the system and the expected response. If the output coincides with the expected response, the system presumes that the operation was successfully executed. The use of a *self-testable testbench* [18][19] accelerates the development process of the master node because it is not necessary to verify complex waveforms to evaluate the most recent changes. The testbench compares the output of the system against the expected one, generating an error occurrence report.

To test the master node, a testbench to simulate sending and receiving CAN and Serial frames to the master was implemented. To do so, an auxiliary CAN core module (see Figure 5) is used to simulate the lamp application. Also, a serial interface module of a server computer is used to simulate the server (in fact, a simulation of the RS232 protocol). The structure of the testbench is presented in Figure 8.

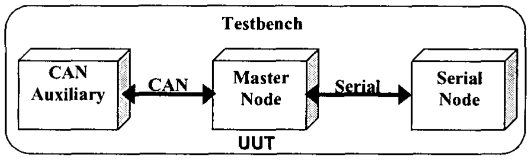

Figure 8. Structure of the master node testbench.

4. Conclusions and Future Work

The implemented system supports remote control of homes through a web browser. The work provides the integration of technologies at several abstraction levels (hardware, software and Internet). The hardware was implemented using VHDL, employing a soft IP Core for the CAN protocol. The software was developed in Java using methods to access the serial port and the database. The system interface homepage is implemented in HTML and PHP.

During the development of this project some important decisions were taken. First to use the CAN protocol to implement SHSs. Second, the creation of a higher level abstraction protocol that makes possible supporting multimedia on top of the CAN protocol. Third, the use of structured and self-testable testbench.

The system is *multi-platform* (i.e. platform independent) because the client only needs a web browser running in any platform. The Server is also multi-platform, because the resources used (i.e. Apache web server, Java and PHP) are available to most platforms. Another important characteristic of the server computer is its low cost, since it capitalizes in the use of free software only.

As future works we intend to: (*i*) change the hardware/software interface to the use of the PCI bus, in place of the serial port; (*ii*) build a service that implements a plug and play protocol; (*iii*) build test techniques like keep alive and built-in self test (BIST), to verify the application nodes distributed through the house; (*iv*) implement a time to live mechanism to segmented frames (data bigger than 8 bytes); (*v*) web access with multiple permissions to disable the interaction of, for example, children, with the security system.

Acknowledgments

The authors are grateful to the institutions CNPq (Research grants 522939/96-1 and 520091/96-5) and FAPERGS (Research grants 94/01340-3 and 96/50369-5). The authors acknowledge the help of Luca Stagnaro from ESA, for providing us with the source code for the HurriCANe IP Core.

References

[1] A. M. Amory, J. P. Júnior, and F. G. Moraes. "Sistema Integrado e Multiplataforma para Controle Remoto de Residências". Trabalho de conclusão do curso de informática da PUCRS, December 2000, 167 p. [http://www.inf.pucrs.br/~amory/Trabalho_de_Conclusao/TCFinal.zip]

[2] F. G. Moraes, A. M. Amory, and J. P. Junior, "Sistema Integrado e Multiplataforma para Controle Remoto de Residências", VII Workshop Iberchip 2001, Montevideo – Uruguay, 21-23 March 2001.

[3] F. Baumann, B. Jean-Bart, and A. Kung, P. Robin. "Electronic Commerce Services for Home Automation", Trialog, [http://www.trialog.com/emmsec9-97.pdf].

[4] Sin-Min Tsai et al. "Integrated Home Service Network on Intelligent Intranet", IEEE Transactions on Consumer Electronics, Volume: 46 Issue: 3, Aug. 2000, pp. 499 -504.

[5] Corcoran, P.M., Desbonnet, J. "Browser-style interfaces to a home automation network", IEEE Transactions on Consumer Electronics, Volume: 43 Issue: 4 , Nov. 1997. Pp. 1063 -1069.

[6] Home Automation, Inc. HAI Web-Link Software Homepage, [http://homeauto.com/web-link/index.htm].

[7] Keware Tecnologies, HomeSeer Software, [http://www.homeseer.com].

[8] S. Hughes, A. Zmievski, "PHP Developer's Cookbook'. December 2000. 505 p.

[9] G. Cornell, C. S. HorstMann, "Core Java Second Edition", Califórnia, SunSoft Press, 1997.

[10] Java(TM) Communications API, [http://www.java.sun.com/products/javacomm/index.html].

[11] Robert Bosch GmbH, "CAN Specification Version 2.0", Stuttgart, 1991, [http://www.can-cia.de/CAN20B.pdf]. [http://www.bosch.de/k8/can/docu/can2spec.pdf].

[12] L. Wolfhard. "CAN System Engineering from Theory to Practical Applications", 1997, 468 p.

[13] HurriCANe[ftp://ftp.estec.esa.nl/pub/ws/wsd/CAN/can.htm]

[14] IEEE Standard VHDL Language Reference Manual. IEEE Standard Std 1076a-2000, Jan. 2000, pp. i -290

[15] XESS Inc. XS40 Board V1.4 User Manual, XESS Corp., [http://www.xess.com/manuals/xs40-manual-v1_4.pdf].

[16] VCC Inc, "The Virtual WorkBench Guide", Virtual Computer Corporation, [http://www.vcc.com].

[17] Xilinx Inc, "Virtex 2.5V Field Programmable Gate Arrays (DS003)", Virtex Series.

[18] M. Keating, P. Bricaud; "Reuse Methodology Manual for System-on-a-Chip Designs", Kluwer Academic, 1999, 286 p.

[19] J. Bergeron, "Writing Testbenches : Functional Verification of HDL Models", Kluwer Academic, 2000, 384 p.