Development of application routines for a System on Chip dedicated to irrigation control

A. L. Silva, A.F. Rocha and J.C. da Costa

Departamento de Engenharia Elétrica, Universidade de Brasília, CP 4386, Brasília, DF 70904-970, Brazil

Abstract

A CMOS VLSI SoC (System on Chip), designed for Irrigation Control Applications was developed for a group of work. The SoC design allows wireless and serial communication capability, data processing, sensor signal acquisition and actuator control. All communication in this SoC is made bit-to-bit reading/writing through the interfaces, and the control time by software, because the system do not possess a time counter. In this work, three groups of the routines as well as the development of dedicated communication protocols are presented.

1. Introduction

Our University, together with seven other institutions, is developing a system to control irrigation on crops [1]. The system is composed by a base station, field stations and nodes [2]. The field stations gather information from the nodes through a wireless link, accurately identify areas of moisture deficiency and send instructions to the nodes, determining which ones should act on latch solenoid valves, in order to deliver the required amount of water to the plants. Each node is composed by a CMOS 0.35µm [3] SoC (System on Chip), a tensiometer with a solid state pressure sensor, an actuator which controls the water flow through a solenoid valve, a solar powered power supply, a RF antenna, and embedded software. The SoC consists of a RISC microprocessor, memories, a RF transceiver, digital interface and A/D interface.

An assembly tool was implemented [4], which is basically a two step assembler. It was made in such a way that it is possible to change, add or delete the instructions just by editing a text file.

This paper presents an overview of the three groups of routines: Boot Application (firmware), initialization of the system; Operational Application (RAM), control of the system; and Functional Application (RAM), test of the system. It also presents the protocols used for communication.

2. Programming structure

The microprocessor proposed is based on RISC architecture [5] and provides sixteen 16-bit registers, which are showed in **Table 1**. Besides that, seven memory positions are used as registers for communication with RF and serial interfaces and with the A/D converter.

The microprocessor, which follows a MIPS like structure, has sixteen fixed length 16-bit instructions, which are divided in four categories: arithmetic and logic, data transfer, conditional branch and jump. These instructions are showed in **table 1**.

Table 1: Microprocessor register and instructions

Registers	Instructions
$zero, $t0, $t1, $t2, $a0, $a1, $a2, $s0, $s1, $s2, $s3, $s4, $gp, $sp, $pc, $ra	**Add, Sub, Addi, Shift, And, Or, Not, Xor, Slt, Lw, Sw, Lui, Beq, Blt, J, Jal**

Table 2: Interface addressing for communications

	U. RF	U. Serial	Int. A/D
Setup	**$0BFD**	-	-
Status	-	-	-
transmission	**$0BFC**	**$0BFA**	-
reception	**$0BFB**	**$0BF9**	**$0BF8/7**

The register banks consist in 16 addressable registers organized in the following way:

$zero: the constant zero; t_0$, t_1$ and t_2$: hold temporary values; a_0$, a_1$ and a_2$: argument registers; s_0$, s_1$, s_2$, s_3$ and s_4$: hold program variables; $gp: global pointer register; $sp: stack pointer; $pc: program counter; $ra: return address register.

3. Protocols and software

Two protocols had been elaborated for the design of the communication in the system: one for Boot Application, PBOOT, that describes the load of instruction in RAM memory, and other, named MSNAP, for Operational Applications and Functional Applications, one level up in communication, **Figure 1**.

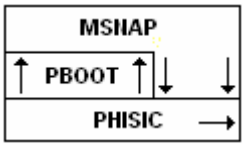

Figure 1: Protocol Layers.

The MSNAP protocol is the SNAP [7] protocol modified for sixteen bits.

When the frame of data arrives in the physics layer, the program Boot loads the packet in memory RAM, in accordance with the rules PBOOT, **Figure 2**. After, the Operational Software verifies the header MSNAP, it sees **Figure 3**, and calculates the checksum. Case some error be found, the packet is returned with one specific error

code. Otherwise, occur one jump for data field, and the application is executed.

The first software developed for the SoC was the Boot Application, which was recorded in the ROM. It was designed with the help of the previously mentioned assembler. It provides the routines for the initialization of the processor, the control of the serial and the radio frequency (RF) interfaces, and a program loader. Its design complies with some hardware constraints such as the fact that the serial and RF interfaces are bit-to-bit reading/writing and do not possess a hardware for the time counter. For that reason, the time is controlled by delay loops.

When the SoC is turned-on the Boot Program is executed. It sets interfaces, disables the interruptions and establishes communication with serial and RF interfaces. A simple protocol was developed for loading a complete program to RAM. It has four fields: 1- the beginning identification Code, 2- data field's size, 3- the first address of writing memory RAM, 4- data's field, **Figure 2.**

Code (16 bits)	N°Bytes (16 bits)	M.Position (16 bits)	Data (16 bits)

Figure 2: Protocol PBOOT.

Operational Applications are routines responsible for system's operation. They are the equivalent to a operating system for this rather simplified SoC architecture. They control the system from the moment that it is loaded in RAM memory. Besides of to manipulate the header MSNAP, it controls the checksum for transmission and reception of data, it identifies errors and transmits one specific code in the field DBn, makes the transmission by RF interface, among other things.

SYN	HB1/HB2	DAB	SAB	DBn	Chk

Figure 3: Protocol MSNAP, 16-bit-field.

The functional Applications are downloaded in the memory RAM indicates by "**Memory Position**" of the protocol PBOOT, and are executed for Operational Application. One application that makes tests in all functionalities of the system had been elaborated. It captures data in interface analog/digital (pressure sensor), makes eight samples and calculates the arithmetic media. After, the fields in protocol are modified for transmission, the data are stored in DBn field, the checksum is calculated, and the packet is returned as message reply.

4. Error detection

In the attempt of guarantee quality in the data for the system, three algorithms of error detection had been implemented: parity, checksum and *check_eot.* Parity and *check_eot* are implemented in the Boot Program and checksum in Operational Application.

Check_eot is the first algorithm of error detection in the system. It's related with the final of each byte load from interface. The last bit of one byte is the "stop bit", in high level; after a sequence of zeros are waited. Then, the routine does not stop reading at the stop bit, but, three bits later. That four bit set is then compared to the **1000** binary sequence, in order to detect eventual faults. The next step of error detection is even parity, a sequence of bit-to-bit "exclusive-or" verifications for each byte.

After that the packet is loaded in RAM memory, the algorithm of checksum is activated, verifying the entire packet.

5. Conclusion

The three groups of routine were successfully developed for the SoC. With the assembler tool, it was gotten 498-bytes of code for Boot Application, 472-bytes for Operational Application and 168-bytes for Functional Application. The protocols had given to stability in the communication, guaranteeing efficiency with simplicity. The next steps are development of new applications for de system and to make the best verification with the end of the simulator of the system [6].

6. References

[1] J. D. Costa et all, "Projeto de Estruturas de um Processador RISC para Aplicação em um SoC para Controle de Irrigação", Proceedings X Iberchip Workshop, Iberchip, Cartagena de Indias, Colombia, 2004.

[2] J. D. Costa et all., "Modulo I.P. de um processador para aplicações embarcadas sem fio", Proceedings IX Iberchip Workshop, Iberchip, Havana, Cuba, 2003.

[3] AMS (Austria Microsystems) 0.35 um CMOS, Model C35B4C3, 4 metal levels, 2 poly levels, 3.3V-5V power supply (http://www.austriamicrosystems.com/).

[4] L. Povoa, Master Thesis in Electrical Engineering, University of Brasília to be presented in this year, Brazil, University of Brasilia, Brazil.

[5] BENICIO, G. M., "Projeto de Microprocessador RISC 16-Bit para Sistema de Comunicação sem fio" (dissertação de mestrado em engenharia elétrica), Universidade de Brasília, 2002.

[6] A. J. O. Martins, Graduation's Project, University of Brasília, Brazil, 2005.

[7] SNAP Protocol, http://www.hth.com/snap/.