A FPGA-based Implementation of an Intravenous Infusion Controller System

Cristiano C. de Araújo , Marcus V. D. dos Santos[1] and Edna Barros
Depto. de Informática - UFPE[2]
Caixa Postal 7851 - Cidade Universitária
CEP. 50740-940 Recife - PE - Brazil
{cca2, mvds, ensb}@di.ufpe.br

Abstract

In this paper we present the development and implementation of an intravenous infusion controller system based on fpga's. The system receives information of an infusion drop sensor and controls the drop flow by giving the direction and number of steps of a stepper motor, which compress the drip-feed hose. The system consists of a mixed implementation of software and hardware. The software was implemented in C++ and the hardware was implemented by using FPGA's.

1 - Introduction

The aim of this work is the development and implementation of a system to control the flow of an intravenous infusion gravity system. Such kind of infusion system is the simplest and cheapest, but the infusion rate can vary with the volume to be infused. In order to keep the infusion rate constant, the drip-feed must be adjusted. In the most cases this adjust is done manually, which implies that someone must take care all the time in order to keep constant the infusion flow and to prevent some situations like: the end of infusion is reached, the sensor does not function, the drip chamber is obstructed. In the case where the patient receives medicine or nutrition through the infusion (e.g. parenteral feed), this kind of monitoring can become critical, since the absorption, in this case, depends on the accuracy of the infusion rate. Additionally, the manually monitoring of a large numbers of patients can lead to a stressful work environment. All these facts state the need of an automatic monitoring system, which should keep the infusion flow rate constant and should be able to signal when something is wrong. An automatic monitoring would be very helpful in order to guarantee a safe and accurate infusion process.

The main aim of this work is the development of a control system for a gravity infusion system. The system consists of a board, which can be connected to some personal computer (486 or Pentium). This feature makes the system very useful for home IV infusion therapy.

An infusion controller has been implemented by Ferraz [4] and another one has been implemented by Brandão [7]. The first one was implemented in software and has used a DC motor for pressing

[1] Master student of Department Electronics and Systems at the UFPE.

[2] This work was partially supported by the brazilian research council, CNPq, research project number 680074/94-5 and by the Esprit program KIT (Keep In Touch), project number 128.

the drip-feed hose. The second one was implemented by using discrete hardware and is a stepper motor. The proposed work has been implemented in hardware and software and has used also a stepper motor for pressing the drip-feed hose. As it will be seen later, the proposed system is faster than the first one and more flexible and has more scalability than the second one. Another feature of the proposed work is the hardware implementation by using the fpga technology, which allows the implementation of a first prototype in a very short design time and at low development cost. This paper is organized as follow: in the next section an overview of the intravenous gravity infusion controller system is presented. After that, some background in the FPGA technology is given. The implementation of the intravenous infusion controller system, proposed in this work, is detailed in section 4. First, the software implementation is described. After that, the developed hardware is described followed by the description of the implemented hardware/software interface. This section is finished with the presentation of some results based on experimental measurements obtained by using a first prototype of the proposed controller. The paper concludes with some comments and some ideas for future works.

2 - The Intravenous Infusion Controller System - IVICS: An Overview

The Intravenous Infusion Controller System - IVICS- is a single patient control system , which controls the intravenous infusion flow (in drops/min.) and the infused volume (ml), and sends messages when something wrong is happening. The IVICS system aims to keep constant the serum flow between values going from 4 up to 80 drop/minute, with a dynamic maximum error of ±1 drop/minute. Using a personal computer, the operator gives the proper dosing rate , the volume to be infused and the used kind of drip-feed unit. After the input data has been given , the monitoring can be started and the desired flow is kept constant while no error event happens.

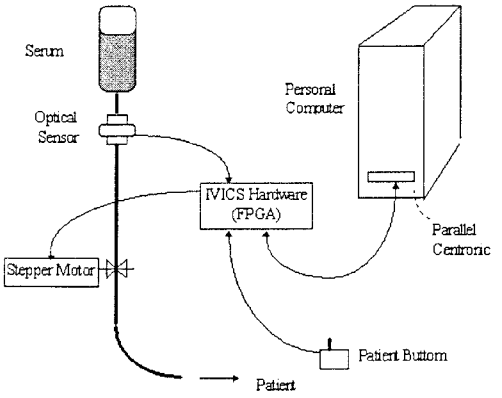

Figure 2.1 - The IV Infusion Controller System - IVICS

As it can be seen in the figure 2.1, the IVICS system is a closed-loop control system, which measures the time interval between successive drops falling through the drip-feed unit and correct the flow by actuating on a stepper motor, which compress or decompress the drip-feed hose. The flow rate, given by the operator, is converted into time value (desired time between drops) in order to be compared with and the measured time value. Depending on the comparison results, the system activates a stepper motor in order to correct the error, if necessary. The optical sensor has a infra-

red LED (light emitter diode) and a photo-transistor which sends 1 (drop) or 0 (no drop) to the hardware. The computer receives the drop data, calculates the error and the number of steps for correct it by applying distinct algorithms and sends to the hardware a 8-bits instruction informing the direction and number of steps. The IVICS hardware receives this information and converts it in pulses and signals for the stepper motor. The IVICS hardware communicates with the computer (software) by using a proper protocol. The hardware receives a 8-bits instruction and a start signal to convert the 8-bits instruction in 0 up to 127 pulses sequence and sent it to stepper motor. The patient button is a push button that sends 1 (patient ready to be infused) or 0 (patient not ready) to the FPGA Hardware. For the sake of simplicity, the communication between the hardware and the personal computer has been implemented by using the "Centronics" parallel port (I/O addresses 378h, 379h and 37ah). The IVICS control logic uses four distinct algorithms for error correction depending on the error value and on the flow range. The flow range (from 4 up to 80 drops/min.) has been divided in five rate regions due to non-linear pulses response (the system behavior for high rates is different than the behavior for small rate values) [4]. When the error is greater than ± 1 drop/min and smaller than some tolerance value[3], the used algorithm is the called "Bang-bang" algorithm, which keeps the error in ± 1 drop/min. The pre-control algorithm is called for negative error values and the post-control algorithm corrects errors greater than the tolerance value. When the measured flow value is in a distinct region than the desired flow, the range control algorithm is executed. More details about these algorithms are given in section 4.1.3. The IVICS system has also fault tolerance capability for the case when the patient makes abrupt movements causing a pseudo changing of the flow rate. The system waits for three subsequent drops and returns to the normal operation mode.

3 - Implementing Digital Circuits by using FPGA's

The field-programmable gate array (FPGA) is a relatively new type of component for the implementation of electronics systems, particularly for digital systems. It consists of an array of functional blocks along with an interconnection network, and as the name implies, its configuration can be determined in the field, that is , at the point of application. A FPGA circuit provides, for many applications, an adequate number of transistors in a single chip package, which includes the functional blocks, switches for the routing network, and the memory to control both. In this implementation we are using a SRAM-based re-programmable FPGA from Xilinx. , which address applications in new ways [5]:

- **Instantaneous Implementation:** as memory-wiring times are usually short, designs can effectively be implemented "instantaneously".
- **Dynamic reconfiguration**: parts of the FPGA can be reprogrammed at run-time, that is, in the application situation.
- **Design security**: an FPGA configuration disappears when the chip is powered-down.
- **Field programmability:** systems built by using FPGAs can be upgraded in the field in order to correct bugs, to recover from damage, or to add new functionality.

The IVICS hardware was described in VHDL [6] at the RT-level and synthesis in XNF description was done by using the Alliance synthesis system [1].

[3] The tolerance value depends on the region of the desire flow rate.

4 - The FPGA-based Implementation of the IVICS System

As mentioned previously, the IVICS controller system was implemented in software and in hardware. An overview of the IVICS implementation can be seen in Figure 4.1. The drop sensor captures the occurrence of a drop. In the case a drop is falling (a signal is activated - 0 to 1) the time is captured and the current flow rate is calculated. When the signal goes from 1 to 0, it means the drop is leaving out and the volume is calculated. If the error flow is greater than 1 drop/min. and smaller than a tolerance value, the "bang-bang" algorithm is executed in order to correct this error. For error values greater than the tolerance value, it must be verified if the limit of region is reached in order to call the range-control algorithm or post-control algorithm. The range-control algorithm corrects whether the measured value and the desired value are in distinct regions.

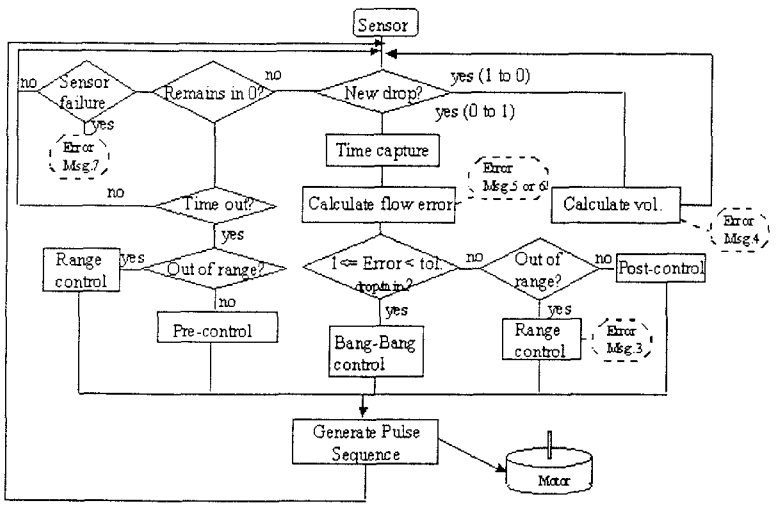

Figure 4.1 - An overview of the IVICS implementation

When a drop is not captured, that is the sensor signal does not change and remains in 0, the time is captured and a new drop is waited. If a time-out value is reached, either the range-control algorithm or the pre-control algorithm is called. Although it is not shown in the picture, if the range-control is called more than hundred times it means that something wrong is happening, probably the infusion liquid is finished. All the control algorithms generate an 8-bit instruction, which is the input for the block Generate Pulse Sequence. This block was implemented in hardware by using FPGA's. In the following, an overview of the software implementation is given. More details of the hardware implementation can be found in the section 4.2.

4.1 - The software of the IVICS system

The software of the IVICS has been implemented by using the C++ language, which was chosen due its versatility and the easy access to the I/O ports of the microcomputer. The IVICS program is composed of three main procedures: user interface, hardware/software interface and the control flow routines. These parts were implemented by using object orientation concepts of classes in the

C++ language. Three classes were created: the graphics class implements the user interface, the motor class implements the hardware/software interface and the algorithms implements the control flow routines.

4.1.1 - User interface: the role of the user interface is to make easier the communication with the system by receiving the input data and by giving out the output data as well as error messages. The data entered by the user are the desired flow, the desired infused volume and the time interval of infusion. The data displayed by the computer are the measured flow, the infused volume and the time of infusion already gone. The error messages are displayed each time some abnormal event happens. These messages indicate the kind of error such as: the sensor or stepper motor is failing, the value of the infused volume has reached the zero and the measured flow is smaller than the minimum limit value.

The user interface performs also some auxiliary tasks, which are routines performing activities not directly related to the control flow such as the manual adjustment of stepper motor position or the archive generation (for patient data and calibration purposes).

4.1.2 - Hardware/Software interface: the hardware/software interface is responsible for the signals exchanging between the microcomputer and the hardware. The functions performed by the HW/SW are the following one:

- the reading of the number of steps and the direction of turn, and the conversion of these two data in an eight bit instruction;
- the writing of the generated instruction to specific output port of the centronics parallel interface, port number 378H;
- the writing to the specific output port (37AH) of the control signals: EN, CLR_DROP and RST_CONT;
- the reading from the specific input port (379H), the signals BUTTOM_OUT, END, DROP_OUT and A_RD_CONT.

4.1.3 - Control flow routines: these routines calculate the number of steps and the direction of turn of the stepper motor, generate the error messages and control the hardware. The implemented routines are described in the following.

- **Bang-Bang routine**: is called when the measured error is greater than one drop per minute and smaller than a tolerance value.
- **Pre-control routine**: is executed when the time interval since the last drop fall is greater than a tolerance value plus the desired time between drops in order to correct the measured flow to a value in the Bang-Bang range of actuation.
- **Post-control routine**: is called when the measured time between drops is smaller than the desired time minus the tolerance value in the case when the measured flow and the desired flow are in the same region. Its goal is similar than the pre-control routine.
- **Range-control routine**: is called when the desired flow and the measured flow are in different regions, and its function is to force the measured flow value to a value in the desired region.

4.2 - The Hardware of the IVICS system

The IVICS hardware has been implemented by using the Xilinx 3020pc68-7 FPGA, which contains 64 CLB's (configurable logic blocks). Each CLB has two function generators, a combinatorial logic section, and an internal control section. The CLB also includes five logic inputs, a common clock input, an asynchronous direct reset input, a clock enable, and two outputs. A data-in input is also provided for direct input to the flip-flops within the CLB. This FPGA circuit is a re-configurable

one and uses static RAM's to store the hardware interconnections and the functions performed by the function generators [8]. As mentioned previously, two main tools were used to implement the hardware in the 3020pc68-7 FPGA: the ALLIGATOR tool and the XDM(Xilinx Design Manager). By having as input a VHDL description, the ALLIGATOR tool generates a XNF(Xilinx Netlist Format) file, that is a structural description of the circuit and is the input file for the XDM software. The XDM software has been used to map the XNF file to the FPGA and to make changes in the circuit manually after the FPGA mapping. This manual correction was necessary due to the fact that the circuit description generated by the ALLIANCE doesn't allow to use some of the FPGA characteristics such as the direct reset input.

The IVICS hardware is basically composed of three main blocks: the communication block, the execution unit and the motor controller, as it can be seen in Figure 4.2.a.

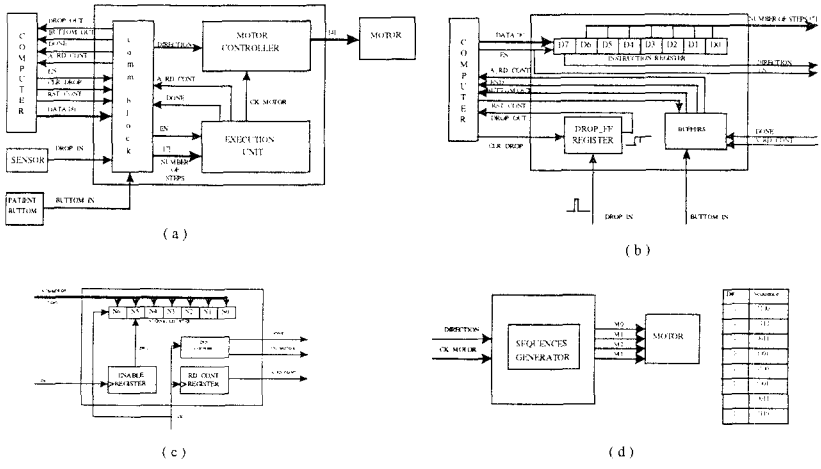

Figure 4.2 – (a) The IVICS hardware implementation, (b) The communication block, (c) The execution unit, (d) The motor controller

4.2.1 - Communication block: The communication block is responsible for the signals exchange between the microcomputer and the circuit itself. As it can be seen in Figure 4.2.b, there are two kinds of signals: data and control. The input signal is a eight bit instruction to be executed by the hardware. This instruction is composed of one bit indicating the direction and seven bits giving the number of steps and is stored in the instruction register when the control signal EN has been activated by the microcomputer. The output signals are the DROP_OUT and the BUTTOM_OUT signals. The DROP_OUT signal is stored in the DROP_FF register, this one bit register is set to one when the drop is detected by the drop sensor and is cleared by the control signal CLR_DROP. The implementation of this register must be done manually by using the XDM tool due to the reasons mentioned previously. The signal BUTTOM_OUT is setup when the user press the button in the hardware, indicating to the software that the unit is activated. By this way, the computer starts the reading of signals coming from the hardware and sends data and control signals to the circuit. The control signals are used to synchronize software and hardware and are described below.

- **EN** (Input): enables to store the instruction into the Instruction Register.
- **RST_CONT** (Input): resets the internal flip-flop, avoiding the re-writing of the internal counter with the value stored in the instruction register.

- **CLR_DROP** (Input): resets the internal flip-flop in order to store the information about a drop fall.
- **BUTTOM_OUT** (Output): indicates to the microcomputer that the hardware is activated, enabling the software for reading the rate of drop falls.
- **DROP_OUT** (Output): it is turned to high when the drop is falling through the sensor's beam.
- **DONE** (Output): indicates that the stepper motor has completed the number of steps of the instruction, and enables the microcomputer to send another instruction.
- **A_RD_CONT** (Output): indicates to the microcomputer that it must send the RST_CONT signal.

4.2.2 - Execution Unit: The execution unit executes the instruction sent by the microcomputer and informs when the instruction is completely executed enabling, thus, the storing of another one in the internal instruction register. As it can be seen in the figure 4.2.c, this unit is composed of a internal counter, the RD_CONT register and the EN_I register. The Internal Enable (EN_I) is a one bit register that stores the external signal EN. This strategy of two enable signals, one external and one internal enable, has been used to permit the hardware itself to load the internal count, releasing, thus, the microcomputer of this task. The data stored in the internal register is loaded, asynchronously, in the internal counter at each clock falling edge when the EN_I signal is high. So whether the microcomputer wants the execution of one instruction by the hardware, it sends the EN signal. This action sets the EN_I signal to high and at the clock falling edge the data is loaded in the internal counter. After that, the signal A_RD_CONT is generated in the RD_CONT register, which causes the microcomputer to activate the signal RST_CONT in order to reset the EN_I register and the RD_CONT register. By this way, the data in the internal register is loaded once during the execution of one instruction. After the internal counter is loaded, its value is decremented until zero for each clock (CK) pulse. When the internal counter is zero, the execution unit sets the signal DONE to high, indicating the end of the instruction execution. The two one-bit registers RD_CONT and EN_I were implemented manually like the DROP_FF register, because they also need to use the internal reset signal, which is not supported by the ALLIGATOR tool.

4.2.3 - Motor Controller: The motor controller unit generates a sequence of four bits, which is necessary to turn the stepper motor in one of the two directions. As it can be seen in the figure 4.2.d, this unit receives the external clock signal (CK) and the turn direction from the instruction register while the internal counter is different of zero. The two sequences are also shown in the figure 4.2.d. The VHDL description for this block can also be seen in figure 4.3.

The FPGA implementation of the IVICS hardware can be seen in the figure 4.4. This implementation was carried out by using the family 3020 of Xilinx and has occupied 37 CLB's. The FPGA chip was placed in the Xilinx demonstration board.

```
-- The block that commands the signals sent to stepper motor

motor: block (ck_motor='0' and not ck_motor'stable)
begin
        a_motor <= guarded "1100" when (a_motor="0110" and dir='0') else
                "1001" when (a_motor="1100" and dir='0') else
                "0011" when (a_motor="1001" and dir='0') else
                "0110" when (a_motor="0011" and dir='0') else
                "0011" when (a_motor="0110" and dir='1') else
                "1001" when (a_motor="0011" and dir='1') else
                "1100" when (a_motor="1001" and dir='1') else
                "0110" when (a_motor="1100" and dir='1') else ini_motor;
end block;

--Bloco de Direction
dir: block (en_i='1' and not en_i'stable)
begin
       dir <= guarded reg(7);
end block;
```

Figure 4.3 - Motor controller VHDL description

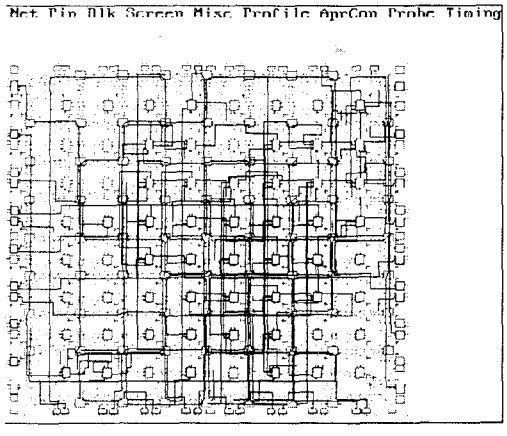

Figure 4.4 - The FPGA Implementation of IVICS hardware

4.3- Results

A first prototype of the IVICS system has been implemented by using a 486-IBM PC and all results were obtained for the system in closed-loop environment. As mentioned previously, the main goal of the proposed system is to keep constant the infusion flow for one patient, which could be a value in the typical range from 4 to 80 drops/min, allowing a maximum transitory error of ±1 drop/minute. This goal was reached as it can be observed in the figure 4.5.a, which shows the system behavior for a 40 drops/minute flow rate in steady state. The measured transitory error of the system is ±1 drop/minute.

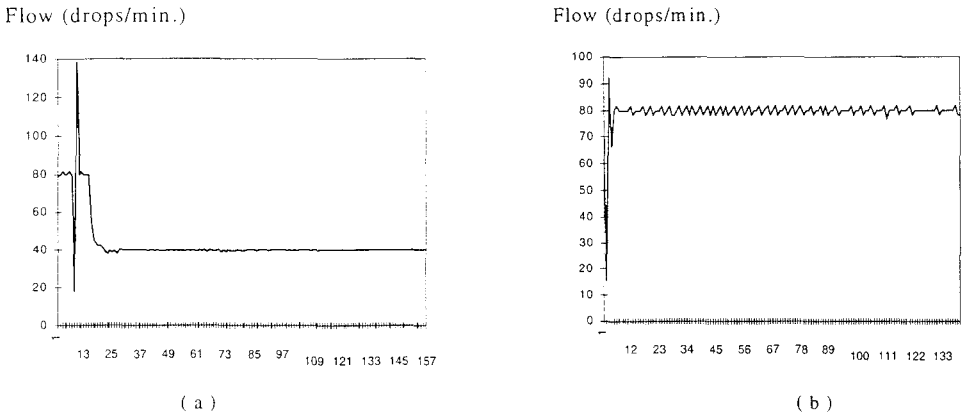

Fig. 4.5 – System behavior in steady state (a) flow rate of 40 drops/min. flow rate (b) flow rate of 80 drops/min.

The figure 4.5.b shows the IVICS behavior for a flow rate of 80 drops/minute. The reader can observe for high rates that the system try to correct the error but the step angle is higher than the necessary. This feature leads to an alternation of the error between positive and negative values. This problem occurs due to the small precision of the used stepper motor (it has a smaller number of step angles than the system needs) and can be correct by using a more precise stepper motor.

Another important feature to be measured is the setup time. The system maximum top-down setup time is 23.64 seconds and the maximum bottom-up setup time is 15.89 seconds. The figures 4.6.a and 4.6.b show some values for top-down and bottom-up setup times, respectively.

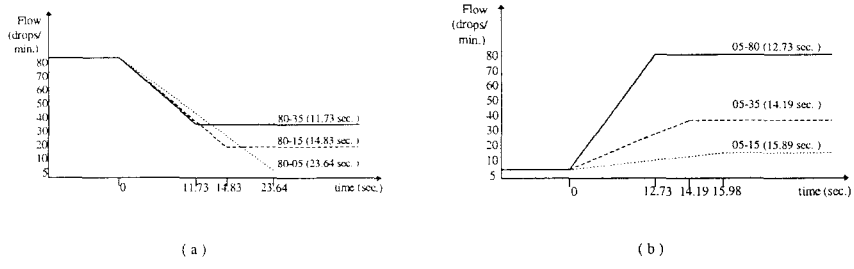

Fig. 4.6 – (a) Top-down setup times (b) Bottom-up setup times

Concerning the system precision and setup time, the results obtained with the IVICS system are better than the results obtained with others controller implementations. For example, the IVICS setup times are 15.89 sec.(bottom-to-up) and 23.64 sec.(top-down) while the CIIM system [Ferraz-93] setup time are 54 sec. (bottom-up) and 30 sec. (top-down). The transitory error to IVISC is ±1 drop/min and the CIIM system transitory error is ± 2 drops/min [Ferraz-93]. Another application to be compared is the CPIAH system [7]. The CPIAH system has a setup time between 1 and 2 minutes for the same application by using a similar stepper motor actuation mechanism. The transitory error of this system is also ±1 drop/min like the IVICS system.

5- Conclusions and future works

This work has presented a FPGA-based implementation for a intravenous infusion controller system. The whole system was implemented in software and hardware, where the hardware part was implemented by using FPGA´s. In comparison with other systems the proposed system has shown a better performance and accuracy. An advantage of a mixed implementation of hardware and software is a better scalability of the system for more than one patient. Additionally the use of FPGA´s has allowed a rapid prototyping of the system.

In order to improve the performance of the system and its capability of fault tolerance we have started the implementation of some software functions in hardware. Additionally, we are using this system as a case study in the development of a hardware/software codesign methodology [2][3] and, for this purpose the system functionality has been described in occam and is being used as input description for the partitioning tool developed in the context of the PISH project, ongoing in our department.

6 - References

[1] CAO-VLSI - Alliance's Reference Manual -Public Domain Software – 1995

[2] Barros, E. and Sampaio, A., Towards Provably Correct Hardware/Software Partitioning Using Occam, Proceedings of the third International Workshop on Hardware/Software Codesign Codes/CASHE94, 1994

[3] Barros, E., Xiong, X. and Rosenstiel, W., Hardware/Software Partitioning of UNITY Language, Proceedings of EURODAC 94, 1994

[4] Ferraz, Claudio H. C., Um Sistema Controlador de Infusao Intravenosa Multi-Paciente para Salas de Recuperacao Hospitalares, Master Thesis, UFPE, 1993 (in portuguese).

[5] Oldfield, John V., Dorf, Richard C., "Field-Programmable Gate Arrays- reconfigurable logic for rapid prototyping and implementation of digital systems", John Wiley &Sons, inc., 1995.

[6] Perry, D. "VHDL" , Mc Graw Hill, 1994.

[7] Brandão, Gláucio Bezerra, Controlador Programável de Infusão para Ambientes Hospitalares,Master Thesis, UFPE, 1996 (in portuguese).

[8] Xilinx XACT Manuals - 1994

Acknowledgments

We would like to thank Prof. Mauro R. dos Santos for the fruitful ideas and discussion about this implementation. Additionally, we thank the Xilinx University Program for the donation of the Xilinx tool we have used in the prototype implementation.