Design and implementation of a Robotic Arm Able to Play "TIC-TAC-TOE" controlled by a FPGA

Norma Elva Chávez Rodríguez[1], Fernando Arámbula Cosío[2], Joel Esquivel Villar[1], Miguel Ángel Valdes[1]

[1] Departamento de Ingeniería en computación, Facultad de Ingeniería UNAM, MÉXICO
[2] Laboratorio de Análisis de Imágenes y Visualización, Centro de Ciencias Aplicadas y Desarrollo Tecnológico UNAM, MÉXICO

e-mail :norma@fi-b.unam.mx

Abstract. This paper show a field programmable gate array (FPGA) based control of a robotic arm able to play TIC-TAC-TOE. The architecture of the control scheme is simple, and thus facilitates realization of the proposed digital controller. The designed controller has been implemented using the SPARTAN-3 STARTER KIT from Xilinx, Inc. Our "IC" can be used as a microprocessor in applications of robotic arm control. Due to the high-speed nature of FPGAs, the sampling frequency of the IC can be raised to values that cannot be reached using a conventional digital controller based on a microcontroller. The strength of this paper is the implementation of a 9-choice algorithm that makes the robotic arm win the game. "VHDL" language and Synthesis tool (view logic) was used to provide the FPGA with the information of the moves and the algorithm to select the most appropriate move.

Keywords: VHDL, digital design, Robotic Arm, FPGA

1 Introduction

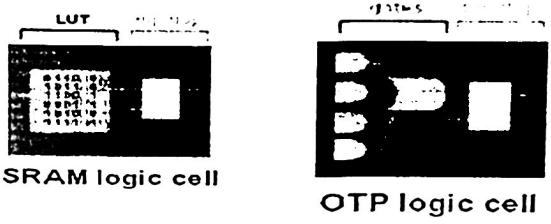
Robots have become important due a wide range of applications from manufacturing and surgery to the handling of hazardous materials. One of our robotic arm's functions is to move to a specific location or along a determined path. Moves can be performed in nine different ways thanks to the algorithm specially design to play wisely (intelligently?). A FPGA is used as control of the robotic arm with five motors and five joints to allow flexibility, five Axes of motion: Right / Left 350 degrees; Shoulder 120 degrees; Elbow 135 degrees; Wrist rotate CW & CCW 340 degrees; Gripper Open & Close 50 mm. Its dimensions are: Max Length Outwards = 360 mm Max Height Upwards = 510 mm Max Lifting Capability = 130g.

Field-Programmable Gate Arrays FPGAs are a special type of Application Specific Integrated Circuits ASICs which can be configured or reconfigured by the user instead of the manufacturer. One of the most important characteristics of FPGAs is fast circuit prototyping. A digital device of up to thousands of logic gates can be implemented and/or revised in days or even hours. In addition, FPGAs have a low cost of manufacturing, and have the property of being fully testable. FPGAs are gate-array-like devices, and they are typically used to implement multi-level logic functions. This paper shows that a FPGA can control directly a chopper driven brush DC motor and in addition generate a sequence of robotic arm commands.

The paper is organized as follows: Section 2 presents an overview of how and where programmable logic devices are used, including both "CPLD" & "FPGA" devices, and discusses the synthesis and implementation process for FPGAs. The design targets a Spartan-3 "FPGA". Section 3 takes the "VHDL" language through to a working physical device, Section 4 shows the sensor used in the board. Section 5 presents the implementation of our control "IC", finally, section 6 is dedicated to conclusions.

2 Overview of PLDs

By the late 70's, standard logic device were the rage and printed circuit, boards were loaded with them. Then someone asked the question: "What if we gave the designer the ability to implement different interconnections in a bigger device?" This would allow the designer to integrate many standard logic devices into one part. The two programmable planes provided any combination of AND and OR gates and sharing of AND terms across multiple OR's. This architecture was very flexible, but at the time due to wafer geometry's of 10um the input to output delay or propagation delay (Tpd) was high which made the devices relatively slow. Complex Programmable Logic Devices CPLD is the way to extend the density of the simple PLDs. The concept is to have a few PLD blocks or macrocells on a single device with general purpose interconnect in between. Simple logic paths can be implemented within a single block. More sophisticated logic will require multiple blocks and the use of the general purpose interconnect in between to make these connections. Field Programmable Gate Array FPGA is a regular structure of logic cells or modules and interconnect, which can be controlled completely. This means that changes to the circuit, its design and program can be done freely as required. There are two basic types of FPGAs: SRAM-based reprogrammable and OTP. These two types of FPGAs differ in the implementation of the logic cell and the mechanism used to make connections in the device. The dominant type of FPGA is SRAM-based and can be reprogrammed as often as you choose. In fact, an SRAM-FPGA is reprogrammed every time it's powered up, because the FPGA is really a fancy memory chip. That's why you need a serial PROM or system memory with every SRAM-FPGA.



a) b)

Fig. 1 a) SRAM based FPGA Logic Cell; b) OTP based FPGA Logic Cell.

In the SRAM logic cell, instead of conventional gates, an LUT determines the output based on the values of the inputs. (In the SRAM logic cell, Figure 1a, six different combinations of the four inputs determine the values of the output.) SRAM bits are also used to make connections. OTP FPGAs use anti-fuses (contrary to fuses, connections are made, not "blown," during programming) to make permanent connections in the chip. Thus, OTP-FPGAs do not require SPROM, or other means to download the program to FPGA. However, every time you make a change on design, you must throw away the chip. The OTP logic cell is very similar to PLDs, with dedicated gates and "flip-flops". The availability of CAD-software such as WebPACK ISE from Xilinx Inc. and Max+Plus II from Altera Inc. has made it much easier designing with programmable logic. Designs can be described easily and quickly using a description language such as ABEL, VHDL, Verilog-HDL, and AHDL or with a schematic capture tools.

3 Introduction to VHDL

VHDL is a language for describing digital electronic systems. It arose out of the United States Government's Very High Speed Integrated Circuits (VHSIC) program, initiated in 1980. In the course of this program, it became clear that there was a need for a standard language for describing the structure and function of integrated circuits (ICs). Hence the VHSIC Hardware Description Language (VHDL) was developed, and subsequently adopted as a standard by the Institute of Electrical and Electronic Engineers (IEEE) in the US. VHDL is designed to fill a number of needs in the design process. Firstly, it allows description of the structure of a design and its decomposition into sub-designs, and how those sub-designs are interconnected. Secondly, it allows the specification of the function of designs using familiar programming language forms. Thirdly, as a result, it allows a design to be simulated before being manufactured, so that designers can quickly compare alternatives and test for correctness without the delay and expense of hardware prototyping.

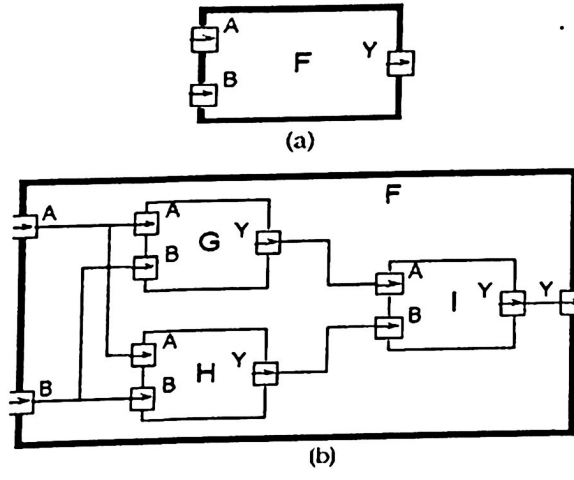


Fig.2 Example of a structural description.

3.1 Describing Structure

A digital electronic system can be described as a module with inputs and/or outputs. The electrical values on the outputs are some function of the values on the inputs. Figure2(a) shows an example of this view of a digital system. The module F has two inputs, A and B, and an output Y. Using VHDL terminology, we call the module F a design *entity*, and the inputs and outputs are called ports. One way of describing the function of a module is to describe how it is composed of sub-modules. Each of the sub-modules is an *instance* of some entity, and the ports of the instances are connected using *signals*. Figure 2(b) shows how the entity F might be composed of instances of entities G, H and I. This kind of description is called a structural description. Note that each of the entities G, H and I might also have a structural description.

3.2 Describing Behavior

In many cases, it is not appropriate to describe a module structurally. One such case is a module which is at the bottom of the hierarchy of some other structural description. For example, if you are designing a system using IC packages bought from an IC shop, you do not need to describe the internal structure of an IC. In such cases, a description of the

function performed by the module is required, without reference to its actual internal structure. Such a description is called a functional or behavioral description. To illustrate this, suppose that the function of the entity F in figure2-1(a) is the exclusive-or function. Then a behavioral description of F could be the Boolean function Y = A . B + A . B More complex behaviors cannot be described purely as a function of inputs. In systems with feedback, the outputs are also a function of time. VHDL solves this problem by allowing description of behavior in the form.

3.3 Discrete Event Time Model

Once the structure and behavior of a module have been specified, it is possible to simulate the module by executing its VHDL language description. This is done by simulating the passage of time in discrete steps. At some time, a module input may be stimulated by changing the value on an input port. The module reacts by running the code of its VHDL language description and scheduling new values to be placed on the signals connected to its output ports at some later time. This is called scheduling a transaction on that signal. If the new value is different from the previous value on the signal, an *event* occurs, and other modules with input ports connected to the signal may be activated. The simulation starts with an *initialization phase*, and then proceeds by repeating a two-stage *simulation cycle*. In the initialization phase, all signals are given initial values, the simulation time is set to zero, and each module's behaviors program is executed. This usually results in transactions being scheduled on output signals for some later time. In the first stage of a simulation cycle, the simulated time is advanced to the earliest time at which a transaction has been scheduled. All transactions scheduled for that time are executed, and this may cause events to occur on some signals. In the second stage, all modules which react to events occurring in the first stage have their behaviors program executed. These programs will usually schedule further transactions on their output signals. When all of the behaviors programs have finished executing, the simulation cycle is repeated. If there are no more scheduled transactions, the whole simulation is completed. The purpose of the simulation is to gather information about the changes in system state over time. This can be done by running the simulation under the control of a *simulation monitor*. The monitor allows signals and other state information to be viewed or stored in a trace file for later analysis. It may also allow interactive stepping of the simulation process, much like an interactive program debugger.

4 Control design and implementation

Due to limitation on the length of the paper the implementation of the VHDL code is not show, however we explain how the arm and interfaces work on the tic-tac-toe game.

4.1 The "Tic-Tac-Toe" Game

This game is played on a board with nine different holes, which are all of the same size and equipped with a pair of sensores, (The sensores are used to recognize if there is or there isn't a ball inside them, the sensores send a signal according to the presented condition to a "FPGA") 10 balls of the equivalent size are also used. The balls are divided in two group colors, 5 green balls, which are the balls that the robotic arm shall use, and 5 red balls, which are the balls that the Robotic's arm challenger shall employ.

The game is developed as a normal tic-tac-toe game, although the Robotic arm, will always have the first move. (This is so that the "FPGA" will be able to note a difference between the challenger's moves, and the Robotic's arm moves. What the "FPGA" does to recognize this difference, is that it separates and stores in a distinct place the moves that are an even number from the moves that are an odd number. The person will clearly always make the even moves, while the robotic arm will always make the odd ones.) This Robotic arm that plays "Tic-Tac-Toe", was programmed under an algorithm in "VHDL" language, which doesn't allow it to loose.

4.2 Power System

We use a FPGA as digital controller. The control of the robotic arm drives five DC motors, we used an L2930 driver, and each driver can control 2 motors. The figure 3 shows the control of two motors by driver.

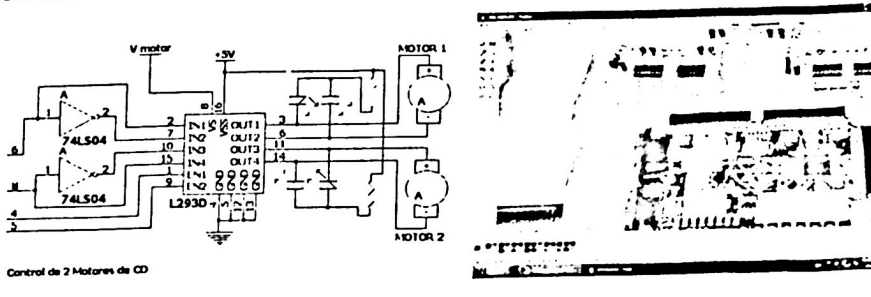


Fig. 3 The control of two DC motors

4.3 How the Controller Works

The robotic's arm control is in the FPGA. The FPGA has as an input the signals that come from the ultra red sensores. With these signals, the FPGA decides how the movement of the robotic's arm motors should be, according to the input, the FPGA sends signals to the motors that will make them move in the desired way. Between the FPGA and the motors, we have the interface of the drivers, which are used to boost the output power of the signals that the FPGA sends to the Robotic arm. See Figure 4.

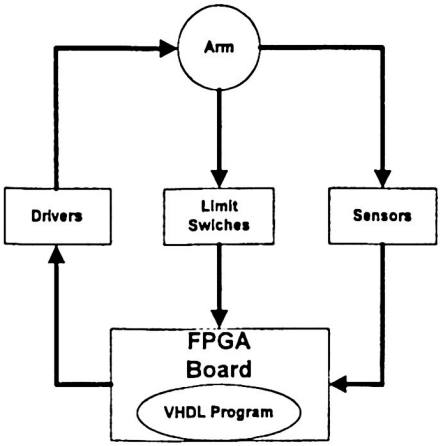


Fig. 4 Block Diagram of System Controller

5 RESULTS AND CONCLUSIONS

With all this, we created a robotic arm that is able to be an exciting tic tac toe challenger for any player who comes before it. The Robotic arm has good movements that allow it to present a high exactitude when placing the ball inside the hole. On the other hand, this Robotic arm never looses a game. It either ties or wins its matches. The longest time that a game with the Robotic arm can last is 15 minutes approximately. In the following Figures (5, 6), we can see the Robotic arm in action:

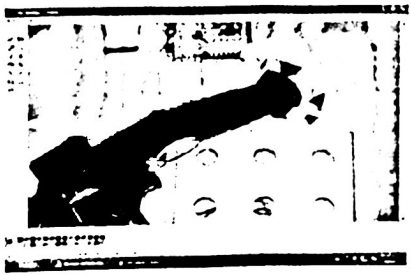


Fig. 5 The arm reaching for a ball to throw

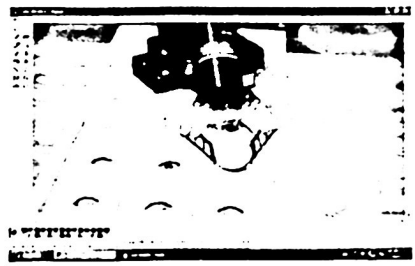


Fig. 6 The arm making the first move in the game.

References

[1] S. Brown, Z. Vranesic "Digital Logic with VHDL Design" McGraw Hill 2nd Edition.
[2] A. Lloris, A. Prieto, L. Parrilla "Sistemas digitales" Mc Graw Hill.
[3] J. Martin "Lenguajes formales y teoría de la computación" Mc Graw Hill , 3ª Edición
[4] D.Givone"Digital Principles and Design", Mc Graw Hill.. 2nd Edition
[5] A. Marcovitz "Introduction to Logic Design" ,Mc Graw Hill. 2nd Edition.
[6] "FPGA Starter Kit Manual" Book, 4ª Edition, Xilinxs .inc.
[7] "Getting Start" Manual, 5ª Edition Altera. inc.