MÄLARDALEN UNIVERSITY SWEDEN

BACHELLOR THESIS REPORT

Servo Motors Control With FPGA : Introdution of control system with torque change

Director: Dr. Mikael Ekström
School of Innovation, Design and Engineering
Mälardalen University
Email: Mikael.Ekstrom@mdh.se
Phone: +46 (0)21 10 16
&
Fredrik Ekstrand (PhD Student)
School of Innovation, Design and Engineering
Mälardalen University
Email: fredrik.ekstrand@mdh.se
Phone: +46 (0)21 1015731

Alvaro Polo Brihuega
Estudiante de Ingenieria Técnica Industrial: Electrónica
Escuela Politécnica Superior
Univesidad de Alcalá
Email: alvaropolo84@hotmail.com
Phone: 0700160509

ABSTRACT

This document represents the final thesis of Mr. Alvaro polo Brihuega, which consists of a study of the servo motors dynamisel and design a solution for controlling them.This solution will use a VHDL programmed FPGA connected to the servo motors and the subsequent study of the changes that occur in the servo to get some resources to control their movement and the torque which plays such movent with respect to external forces. All this will be pointing to an ultimate goal as the control of changes instrength and dampen the same as falls and bumps in systems with installed servo

INDEX

INTRODUCTION

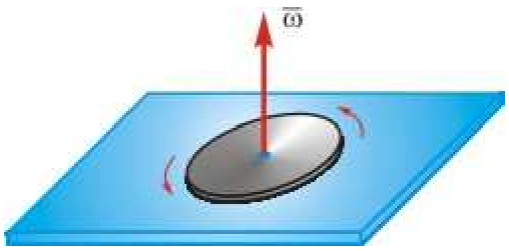
In this introduction we will see some concepts of both physical and electronic engineering that will help us understand the whole of the project developed and that in turn will serve as guidelines to introduce in the analysis of project performance in an easy and simple way. These concepts are presented in a linear manner from a simple explanation of a physical force so common in nature as is the torque till the programming languages not so common for those not familiar with this branch of engineering

What is the torque?

We say that a body is in mechanical equilibrium when its state of motion as a whole does not change over time. This concept is relative because the state of motion of a body depends on the chosen reference system.

Distinguishes two kinds of balance: translational and rotational.
We say that a body is in translational equilibrium, for a reference system, when its center of mass is at rest or moves with constant velocity (motion) about him.

We say that a body is in rotational equilibrium with respect to some reference system, when this does not rotate or is rotating with constant angular velocity (uniform rotational motion), about him.



If a body is at rest with respect to a reference system, it is said that the body is in static equilibrium, which is the most common form of mechanical equilibrium.

When you exert a force on a rigid body pivots around an axis, the body tends to rotate around that axis. The tendency of a force has to rotate an object around any axis is measured by a vector quantity called torqu or torque.

The key concept of balance of angular accelerations of bodies and large bodies, is the torqu (τ) or torque. For the system is in balance the total or net torque must be equal to zero. The balance when playing on a swing. If two people of equal weight suspended from the swing at the same distance from the axis of rotation, this is balanced. However, if a person hangs closer to the axis, the balance is broken and it is further raised to that center. The closer to the transcendent forces involved in this example are the weights of each individual. Also upset the balance if a person is heavier than the other, but it is possible to maintain balance if you hang something heavier closer to the axis of rotation.

Torque = force x arm x b τ = F

Arm means a force to the perpendicular distance separating force, the fulcrum axis of rotation. At this point it is called torque, τ.
The forces applied at the ends of the swing produces torque with opposite signs in the absence of one, the other turns the one way system

How to apply knowledge of torque to our project ?

The main objective of this project is to achieve stabilization, sudden changes in torque of jointed appendages mounted on robots or machines with servo motors in the joints the main body, ie control of the legs or arms of robots to blows or falls. To avoid these sudden changes in order to reduce the touch to the movement of appendages, ie, if get appendages move quickly and in a small space with little movements or get it on every move the torque is minimum.

Reduce the torque by the movements are very quick and consistent programming we will get our system in a special way and inputting orders right business. These orders will be executed quickly on the engine and this will give a quick response as we shall see which will have to rescan the system and return to give an order. We can also to program the servomotor to support greater flexibility to changes in torque cushionwhich will make much more rapid change of position

The analysis of torque to the impacts shows an increase of the same immediately and if we tried to rectify its position quickly get the only thing that will break the servomotor connected to the appendix, so the rapid changes of position have to be made in very small and an increase in the range allowed by the motor torque will get an optimal damping in these systems

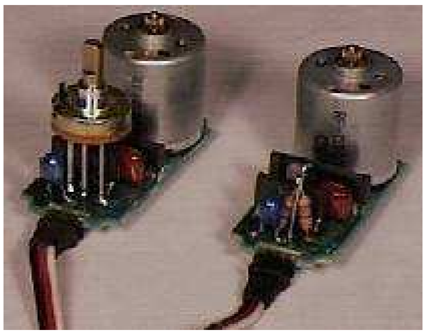
What is a servo motor?

A Servo is a small device that has a line of controlled performance. This can be taken to specific angular positions by sending a coded signal. As long as a coded signal exists on the input line, the servo will maintain the angular position of gear. When the coded signal changes, the angular position of the gears change. In practice, servos are used to position control surfaces like the movement of levers, small elevators and rudders. They are also used in radio control, puppets, and of course, robots.

The Servos are extremely useful in robotics. The motors are small, have an internal circuitry and internal control is extremely powerful for their size. A normal or standard servo like the Hitec HS-300 is 42 ounces per inch or better 3 kg per cm. Of torque is pretty strong for his size. Power also proportional to mechanical stress. A servo therefore does not consume much energy. Shows the internal composition of a servo motor in the box below. You will see the control circuitry, motor, a set of gears, and the box. You can also see the 3 wires for external connection. One is to Vcc (+5 volts), GND and the white wire is the control wire.

GENERAL OPERATION:

The servo motor has some control circuits and a potentiometer (variable resistor) that is connected to the central axis of the servo motor. In the picture you can see on the right side of the circuit. This potentiometer allows the control circuitry, monitor the current angle of the servo motor. If the shaft is at the correct angle, then the engine is off. If the circuit checks that the angle is not correct, the motor rotates in the right direction until the correct angle. The axis of the servo is capable of reaching around 180 degrees. Usually, some reach 210 degrees, but varies by manufacturer. A normal servo is used to control angular movement between 0 and 180.



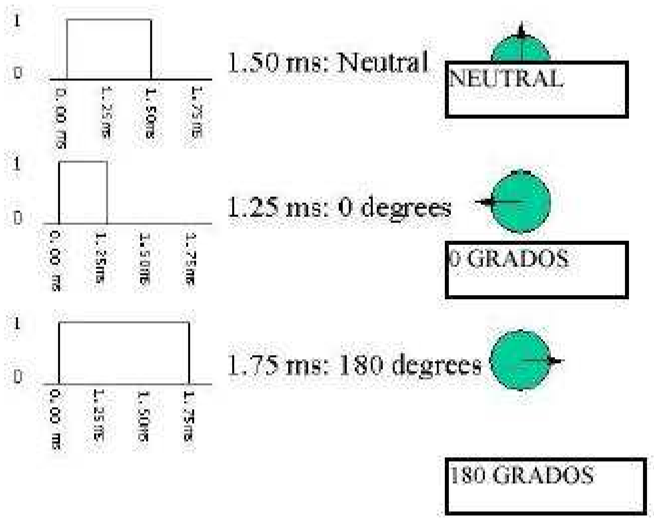
The amount of voltage applied to the motor is proportional to the distance it needs to go. Thus, if the shaft needs to return a large distance, the engine will return at full speed. If this needs to return only a small amount, the motor will run at a slower rate. This is called proportional control.

FEATURES AND OPERATION:

These servos have an amplifier, servo motor, reduction gears and a feedback potentiometer, all built on the same set. This is a position servo (which means that one tells what position should go), with a range of approximately 180 degrees. They have three electric cables, Vcc, GND, and control input.

To control a servo, you order a certain angle, measured from 0 degrees. You send a series of pulses. In a pulse ON time indicates the angle to be positioned; 1ms = 0 degrees, max = 2.0ms. degree (about 120) and a value between them gives a proportional output angle. Generally considered that 1.5ms is the "center." Between 1 ~ 2ms limits are the manufacturers' recommendations, you can usually use a wider range of 1.5ms to obtain a wider angle and even 2ms performance for an angle of 180 degrees or more. The limiting factor is the ceiling of the potentiometer and the mechanical limits built into the servo. A buzzing sound usually indicates that you're forcing over the servo, then you should slow down a little.

OFF time the servo is not critical, can be around 20ms. We used between 10 ms and 30 ms. This does not have to be this way, can vary from one pulse to another. Pulses that occur frequently in the OFF time may interfere with the internal synchronous servo and could hear a buzzing sound or a vibration in the shaft. If space pulse is greater than 50ms (depending on manufacturer), then the servo could be in sleep mode between pulses. Come to work in small steps and performance would not be optimal.



As shown in the figure, the pulse duration indicates or dictates the angle of the shaft (shown as a green circle with arrow). Note that the illustrations and the actual times depend on the motor manufacturer. The principle, however, is the same.

The control cable is used to communicate the angle. The angle is determined by the duration of a pulse is applied to control wire. This is called PCM Pulse Coded Modulation. The servo expects to see a pulse every 20 milliseconds (.02 seconds). The pulse length determines the motor turns. A pulse of 1.5 ms., For example, will cause the motor becomes the position of 90 degrees (called the neutral position). If the pulse is less than 1.5 ms., Then the engine will be close to 0 degrees. If the pulse is greater than 1.5ms, the shaft is closer to 180 degrees.

why did we choose dynamisel Motors?

This motor is much more than a digital servo, is a sophisticated robotic component. Each servo has the ability to control your speed, temperature, position, tension andsupported load. The algorithm used to maintain the position of each servo can be adjusted individually, allowing the feedback control of speed and load on each actuator. It really is more expensive than any servo motor for smaller applications but for our project we chose this complex because we allow servomotor full control of the necessary parameters and the robustness of its structure which will allow us to use in the practice of our Project.

The special feature of AX-12^{a}, made a high performance robotic actuator to be unique in the market, is clearly that each actuator Dynamixel AX-12A has a microcontroller means 50 commands, most of which set or read parameters that define its behavior. The typical radio control hobby servo only understands the order "target angle (given by a PWM signal), but Dynamixel actuators allow you to use as an actuator-sensor professional: the software that runs on the CM-5 or CM510 can react to the environment using information from the sensors read the AX-12 +.
This information can be read current position, the current drawn or the variation in temperature under load servo in it, allowing sophisticated feedback control by controlling the couple supporting each robot joint. This has applications eg in biped robots, because without inclinometers or accelerometers, are available equilibrium effects.

What is DLP-HS-FPGA controller card?

The controller card, or simply "controller," is a piece of hardware that acts as the interface between the motherboard and the other components of the computer. For example, hard drives, optical drives, printers, keyboards, and mice all require controllers to work. Most computers have all the necessary controllers built in the motherboard as chips, not full-sized cards. However, if you add additional components such as a SCSI hard drive, you may need to add a controller card as well. Controller cards are typically installed in one of the computer's PCI slots.

(from web page: http://www.iwebtool.com/what_is_controller_card.html)

In this case more or less is the same, we have an element of control is the servo motor and a controlling element is the FPGA and we need something in between that we act as liaison between the two systems.

The DLP-HS-FPGA module is a low-cost, compact prototyping tool that can be used for rapid proof of concept or within educational environments. The module is based on the Xilinx Spartan 3A and Future Technology Devices International’s FT2232H Dual-Channel High-Speed USB IC. The DLP-HS-FPGA provides both the beginner as well as the experienced engineer with a rapid path to developing FPGA-based designs. When combined with the free ISE™ WebPACK™ tools from Xilinx, this module is more than sufficient for creating anything from basic logical functions to a highly complex system controller.

(from web page: http://www.electronicspecifier.com/Tech-News/DLP-HS-FPGA-High-Speed-FPGA-Module.asp)

What is a FPGA?

The FPGA's (Field Programmable Gate Array) are general-purpose logic devices programmable by users, consisting of logic blocks connected by programmable connections. The size, structure, number of blocks and the amount and connectivity of connections vary in different architectures.

Is an integrated circuit containing identical logic cells (64 to 8'000 .000) which can be viewed as standard components. The logic cells are interconnected by a matrix of wires and programmable switches

-Structure: two-dimensional array of logic blocks surrounded by configurable connections. A family contains identical logic blocks and connections, but differ in the size of the array.
-Programming technology, is programmed by loading configuration memory cells that control the logic and interconnections.
-Features: volatility, non-volatile, external storage, reschedule, standard manufacturing process and low consumption.

The FPGA is one of the latest developments in programmable logic device technology, it is important to note that a really FPGA configured with a program, unlike what is commonly known as programmed system (microcontroller, microprocessor, etc) where a hardware fixed to interpret and execute a program specified as a set of instructions by the programmer, in the FPGA what you have is a hardware that is set by physical connections that are specified by a program or setup string.It is important to note that when making a FPGA design with the same drawbacks that make a system with discrete components, ie become relevant phenomena and propagation delay associated with clock signals. (Jitter etc).

The first programmable logic devices were PAL or PLD, they had gates (AND / OR) fixed that could be programmed to respond to certain transfer functions.
The FPGA as opposed to PLD and PAL, is that its structure is not composed of gates AND / OR, instead containing logic blocks to implement the required functions.

What is a programming language (VHDL)?

A programming language is an artificial language that can be used to control the behavior of a machine, especially a computer. These consist of a set of syntactic and semantic rules that allow to express instructions that are then interpreted. Be distinguished from "computer language", which is a broader definition, since they include other languages such as HTML or PDF that format to a text itself is not the same. The programming programmer is in charge of using a language for creating a set of instructions that ultimately constitute a computer program.

In use, a programming language can approach the human form of expression and, therefore, this type of language is called high level. This means that they use words and forms in their structures that resemble natural language (especially English). In contrast, languages that are closer to the way in which the computer is operated, are called low-level languages. This means that the programmer should write is closer to machine language, which is ultimately what computers can interpret.

The source code is the set of instructions in a program (or subprogram or module).Source code must be compiled to be interpreted and executed by the computer. The compilation translates the source code (depending on the programming language) to machine language (depending on the host machine.)

Examples of programming languages: php, prolog, asp, actionscript, ada, python, pascal, c, basic, java, VHDL, etc.

VHDL means very-high-speed Integrated Circuits or hardware description language hardware description language integrated circuits high speed.As evident from the name is a language similar to Verilog ADHL or used to describe the internal circuitry and programming of FPGAs. These languages have the same objective and differs from the classic C (or one similar to this one) for being a non-sequential parallel language.

Xilinx FPGA (that used in the project.) It has 400,000 gates and runs at 66 MHz. Like this that the difference between an FPGA and a micro is that we program each cell to function as a block of logic. This means that, because each block is so independent, they are all operating at the same time. Unlike a bus where each line of code is processed in turn. Therefore FPGA programmers can achieve much higher speeds processing (in theory).

This previous point is very important to keep in mind. A program in VHDL may seem like a classic computer program (sequential) and can process data at the edge of the clock changes, but remember that everything is running at once (parallel).

VHDL language is very simple but very powerful. It is not as flexible as a bus in the eyes of some people, but a good programmer can generate things you can not buy or replicate with a microphone, designing the hardware according to our requirements.

To program with this language there are many powerful software as the case of ISE XILINX we will use in this project. These products allow the user to design a code, compile, and test of advanced simulation systems for subsequent programming of the FPGA.

To finish this introduction ...

With knowledge of these concepts can reach a total project understanding easy analysis of what is intended to do the same. I would also like to aware that this project is part of a project goblal take charge of building a machine with appendices and analyze angle sensors to help with stabilization. therefore this project will be embraced by other projects that will:

> -A possible scheduling of a sensory part will analyze the body with respect to space and may add information, for example the angel of fall, to the FPGA through a serial port and help stabilize the body against impacts.
> -And building such a system with damping mechanisms that help improve the control of the impact and the best response of the actuators.

All these projects may be used in future applications especially in motion applications for robots or other navigation equipment. The use of such systems have to support ongoing change in position is very common in both everyday use as in a more specific use such as in space in which robots use similar systems to the study.

MAIN OPERATION

It will start with a few ready-made products such as motors and controller card DYNAMISEL so we must adapt the project to such products. Vhdl code will be created based on the model of division into smaller blocks with small blocks with different functions that allow us to make future changes if needed and a quick understanding of what you want to get with the program.

The set represented as a square block in the FPGA were implemented on the card as a single program and to conduct the tests used an externally created force that will run on any type of engines installed on the same arm or the simple observation of these parameters to changes produced in their tables

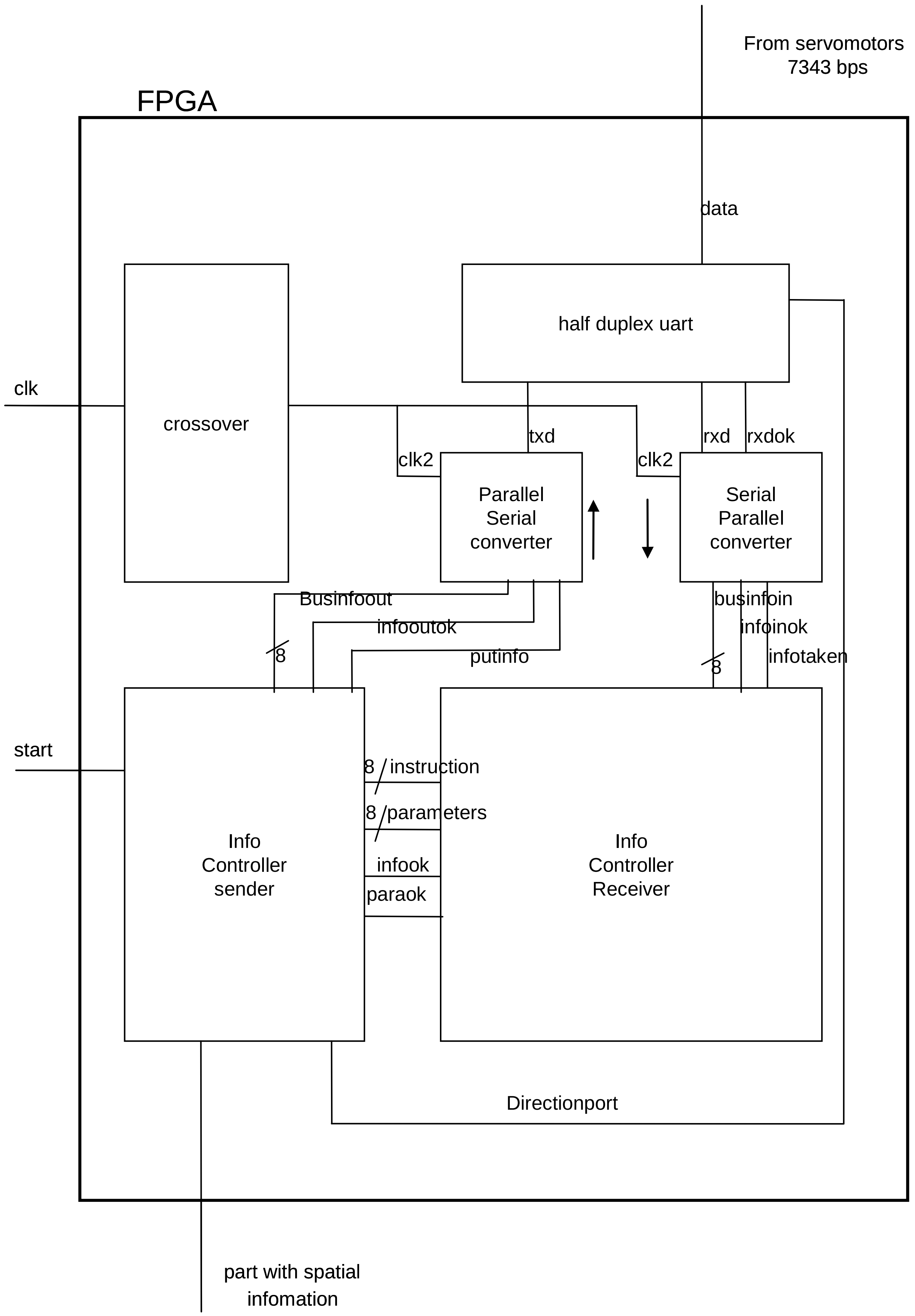
Before proceeding we need to know how communication works between fpga and servomotors.

Dynamisel servo motors operate with a special protocol "question-answer", ie, we will send a query or order and the motor is going to answer with an immediate response. Once we know that the programming process is simple first order will be activated with the pin and immediately start the system that sends information to send a first order analysis of state of the servo or servos. This order will be the analysis of the current engine torque and the response is the current state of the engine on the parameters that are needed at the time. However you could ask for more information relating for example to temperature, angle, etc. and create answers for them by creating a total control on the engine but this project will focus on the torque. This information is passed through the nine-bit bus to the party that will make this series and will be sent to the actuator via the UART you will need to be at that moment in "issuer." Once all information is sent to the UART mode will change the mode to "receiver" and expecting the response of the servo. This is given by e serial port and will be converted to parallel data in serial-parallel converter and then scanned by the block that receives the information and sent back to analyze the information block. This contrasted the information block that has both the servo and the outside (part of another project) and create a response sent back to the servo starting the cycle again.

The data analysis will be through the more complex block the system and produce a fuzzy control because the answer will be created from the servo information and will be a computation of all the information you receive, ie if several actuators, the answer must be given in terms of offering the same information and the expansion of the project we need to add external sensor information as well.

The control system is in continuous communication with the servos and modify the parameters of these and go depending on varying the torque at all times trying to get the angle of these at all times does not vary.

We will see the scheme:


From servomotors
7343 bps
FPGA
data
half duplex uart
crossover
clk
txd
rxd
rxdok
clk2
clk2
Parallel Serial converter
Serial Parallel converter
Businfoout
infooutok
putinfo
8
businfoin
infoinok
infotaken
8
start
8 instruction
8 parameters
infook
paraok
Info Controller sender
Info Controller Receiver
Directionport
part with spatial infomation

Viewing scheme we can see that the entire program will consist of a main block that will send information to the servo motors and analyze the information received from both servo motors and the source of external information provided by the sensory part of the project. It will create a continuous information exchange between the servo motors and the FPGA, a second block to codify the information obtained from the servo when they give their response to the information sent, two blocks to take charge of transforming the information of the servo motors serial to parallel and parallel to serial, another main block is responsible for administering the half duplex UART type, and create and rdx tdx signals required for further analysis and control of the servos, and a last block to be responsible for managing all necessary clock signals for the proper functioning of the project.

The whole block is controlled by the signal CLK2, which is a signal obtained from clk fpga own speed according to the need for communication with the servo motor, synchronous to the necessary exchange of information with the servomotor and activation signals such as “infoinok” or “infooutok” that will allow asynchronous communication between the blocks. This blocks will have an information analysis of data with all the data transmission.

which means that a part is synchronous and one asynchronous?

As we can see the two modules that are controlled by CLK2 are part to be responsible for transmitting data, ie the part that transmit or move information through the fpga and they need a activation or synchronization by a clock, that is the reason why these two sides always operate using the changes of “timer” that will be totally in syncronism with the “timer” to the actuator has a perfect communication.

The other part of the program, the part that deals with the analysis of information does not need any “timer” to run because it is an entirely combinational part. This part took the information from the blocks or put it on the blocks and communicate with them by signals that are generated internally when finished working with the information.

PROGRAMMING: DESCRIPTION OF EACH OF THE BLOCKS

Block 1: crossover

DESCRIPTION:

This is the part that will take care of getting a timer suitable for communication between two devices. This is given by the appropriate conversion clock signal of the FPGA is 66 MHz with appropriate calculations and taking into account that the motor operates at a speed of 7343 bps.

This conversion is achieved through the implementation of a meter that will put a one in the output when it reaches the end of his account which will be the number obtained from the load obtained.

CODE:

```
entity frecuencies is
   Port ( clk : in  STD_LOGIC;
          clk2 : out  STD_LOGIC);
end frecuencies;

architecture Behavioral of frecuencies is
signal valor : integer range 0 to 10000;

begin
   process (clk)
                   begin
                            if (clk'event and clk = '1') then
            if valor = 8988 then
                                                                      clk2<= '1';
               valor <=0;
            else
               valor <= valor+1;
               clk2 <= '0';
            end if;
       end if;
    end process;

end Behavioral;
```

Block 2: half duplex uart

DESCRIPTION:

This part is one of the most important part because that is what is responsible for controlling the direction of the information. It is a very simple because only depends on the incoming pin pin "port direction. " If this is in high then the incoming data by the servomotor provinientes pin "data ", in this case entry will be forwarded to the RXD output and if the controlled low level txd outgoing data to switch to "data "act as output in this case. If nothing happens on both outputs are high impedance would be
It has a very simple but no less important, because if this part fails it will ruin the comumicacion between the two peripheral because the pin "data " is the only pin of communication between them.

```
entity halfduplexuart is
   Port ( data : inout  STD_LOGIC;
         directionport : in  STD_LOGIC;
         txd : in  STD_LOGIC;
         rxd : out  STD_LOGIC);
end halfduplexuart;

architecture Behavioral of halfduplexuart is
begin
        process (data,txd)
                begin
                if(directionport='1')then
                        data<=txd;
                elsif(directionport='0')then
                        rxd<=data;
                else
                        data<='Z';
                        rxd<='Z';
                end if;
        end process;

end Behavioral;
```

Block 3: parallel serie converter

DESCRIPTION:

the conversion of parallel to serial will be responsible for transmitting all the information block and transform information in series so that the servomotor can understand it. The operation is as follows:

Has two processes with different sensitivity lists. The first is activated when the pin "infooutok"comes from block of information comes to one, that would mean that there is new information on the input bus and this one will be copied to an auxiliary signal

The other process runs synchronously with the clock is set at the same speed and activate all the rising edges of the clock and provided that "infooutok" is activated. The function of this part is to transmit the bit bus with more weight, moving the rest, and increase the counter that we use to calculate the information blocks of 9 in 9 bits with the "stop " bit. When the count reaches 9 to send the "stop" bit and a signal called "putinfo" to block of information that indicates that it has finished transmitting and that is again ready to send information.

CODE:

```
entity paralellserie is
   Port ( txd : out  STD_LOGIC;
          clk2 : in  STD_LOGIC;
                                infoout:in  STD_LOGIC;
                                putinfo: out STD_LOGIC;
          businfoout : in  STD_LOGIC_VECTOR (7 downto 0));
end paralellserie;

architecture Behavioral of paralellserie is

signal aux : std_logic_vector (7 downto 0);
signal count : integer range 0 to 8;

begin

        process (clk2)
        begin
                if(clk2'event and clk2='1') then
                        if (infoout ='1') then
                                if count=8 then
                                        txd <='1';                              --stop bit
                                        count <=0;
                                        putinfo <='1';
                                else
                                        txd <= aux(7);
                                        aux(7 downto 1) <= aux(6 downto 0);
                                        count<=count+1;
                                end if;
                        end if;
                end if;
        end process;
```

```
	process (infoout)
	begin
		if(infoout'event and infoout='1')then
			aux <= businfoout;
		end if;
	end process;

end Behavioral;
```

Block 4: serial-parallel converter

DESCRIPTION:

In the parallel serial conversion we must achieve the opposite of the previous block, ie the serial communication need transform the information gives us the motor and a stop bit comunacion in a 8-bit port queue to pass the to receiver information block and filtered before forwarding it to the last block.

For this part we have used a design with a 10-state machine triggered by the rising edge of clock in which the first stage is to wait and be constantly waiting for the activation of the RXD signal to start the transfer. Once the transfer starts copying each state RXD anger in different bits of data bus until you reach the last stage that will match the state of the stop bit, and activate the signal infook to inform the rceiver block that you have complete information on the bus and that can take a turn back to standby.

CODE:

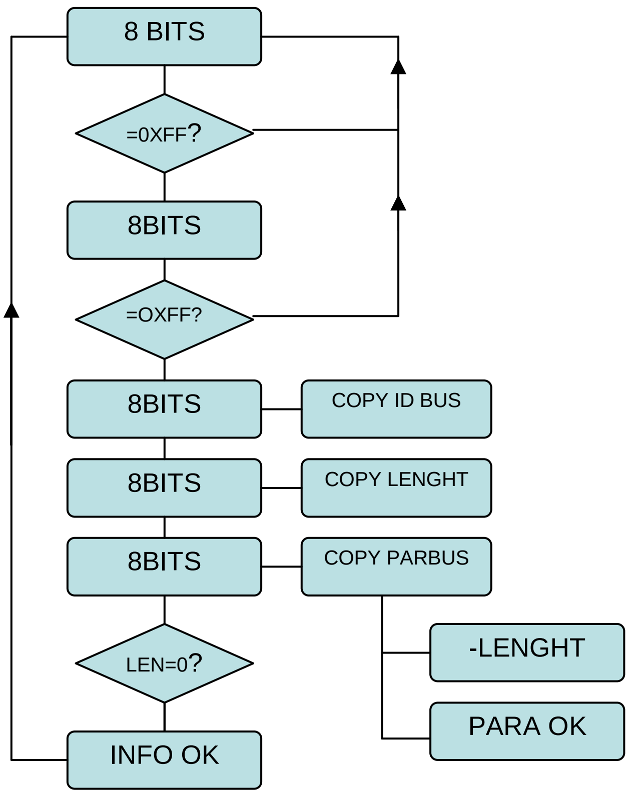
```
entity serialparallel is
   Port ( rxd : in  STD_LOGIC;
          infoinok : out  STD_LOGIC;
                                 rxdok:in  STD_LOGIC;
          businfoin : out  STD_LOGIC_VECTOR (8 downto 0);
          clk2 : in  STD_LOGIC);
end serialparallel;

architecture Behavioral of serialparallel is

type fsm is (st0,st1,st2,st3,st4,st5,st6,st7,st8,st9,st10);
signal act_st, next_st:fsm;
signal aux :std_logic_vector(8 downto 0);

begin

process(rxdok)
begin
            if rxdok'event and rxdok='1' then
                        act_st<=st1;
            end if;
            if rxdok'event and rxdok='0' then
                        act_st<=st0;
            end if;
end process;

process(clk2)
begin
            if clk2'event and clk2='1' then
                        act_st<=next_st;
            end if;
end process;

process(act_st)
begin
            case act_st is
```

```
                when st0 =>                                             --wait state
                        infoinok<='0';
                        next_st<=st0;

                when st1 =>
                        aux(8)<=rxd;
                        next_st<=st2;

                when st2 =>
                        aux(7)<=rxd;
                        next_st<=st3;

                when st3 =>
                        aux(6)<=rxd;
                        next_st<=st4;

                when st4 =>
                        aux(5)<=rxd;
                        next_st<=st5;

                when st5 =>
                        aux(4)<=rxd;
                        next_st<=st6;

                when st6 =>
                        aux(3)<=rxd;
                        next_st<=st7;

                when st7 =>
                        aux(2)<=rxd;
                        next_st<=st8;

                when st8 =>
                        aux(1)<=rxd;
                        next_st<=st9;

                when st9 =>
                        aux(0)<=rxd;
                        next_st<=st0;

                when st10 =>                                    --this happend when is stop bit
                        businfoin<=aux;
                        infoinok<='1';
                        next_st<=st0;

        end case;

end process;

end Behavioral;
```

Block 5: info controller receiver

DESCRIPTION:

Received information block is responsible for separating the information and forward it to analyze the same block. It comprises a state machine to compare incoming data with data expected from the information exchange protocol and depending on what a hard copy will go in different output buses by the end of each instruction is activated bit "infook "and analyze the information block to take it.
to see better what we see with a chart:



CODE:

```
entity infocontrollerreceiver is
   Port ( infoinok : in  STD_LOGIC;
         businfoin : in  STD_LOGIC_VECTOR (7 downto 0);
         instruction : out  STD_LOGIC_VECTOR (7 downto 0);
         parameters : out  STD_LOGIC_VECTOR (7 downto 0);
         infook : out STD_LOGIC;
                              paraok : out STD_LOGIC);
end infocontrollerreceiver;

architecture Behavioral of infocontrollerreceiver is

type fsm is (st0,st1,st2,st3,st4,st5,st6,st7,st8,st9);
signal act_st, next_st:fsm;
signal lenght :unsigned(7 downto 0);
signal activat :STD_LOGIC;
begin

process(infoinok)
```

```
begin
        if activat'event and activat='1' then
                act_st<=next_st;
        end if;
end process;
process(act_st)
begin
        case act_st is
                when st0 =>
                        if businfoin= X"FF" then
                                next_st<=st1;
                                activat<='0';
                        else
                                next_st<=st0;
                                activat<='0';
                        end if;

                when st1 =>
                        if businfoin= X"FF" then
                                next_st<=st2;
                                activat<='0';
                        else
                                next_st<=st0;
                                activat<='0';
                        end if;

                when st2 =>
                                next_st<=st3;
                                activat<='0';

                when st3 =>
                                lenght<=unsigned(businfoin);
                                next_st<=st4;
                                activat<='0';

                when st4 =>
                                instruction<=businfoin;
                                next_st<=st5;
                                activat<='0';

                when st5 =>
                                parameters<=businfoin;
                                paraok<='1';
                                lenght<=lenght-1;
                                activat<='0';
                        if lenght= X"02" then
                                next_st<=st6;
                                infook<='1';
                                activat<='0';
                        else
                                next_st<=st5;
                                activat<='0';
                        end if;

                when st6 =>
                                next_st<=st0;
                                activat<='0';
        end case;
end process;
end Behavioral;
```

Block 6: info controller sender

DESCRIPTION:

This is the most important part of the project and is responsible to send all information to servomor receive, analyze and respond again creating a cyclic communication question-answering. This consists of several processes which are responsible for different functions within the program and in fact could have created another piece of code only to the analysis of information and the subsequent response.

The first process is sensitive to signal the start and take charge of putting the whole system runs may put a first-order and running to take charge of starting the communication. in the program this function is referenced as "?" to leave the progam completely open to change.

In a second process, sensitive to "infook" and "starttransfer" is manejera movement tates machine that is very similar to the previous block in 9 states where the last block will be given the order to stop transmitting resetting signal "starttransfer. this machine may have been putting less conditioning to the parameters (in the code it is assumed that has 3 parameters but this can always change these states conditional)

The main process can be developed in multiple ways and thus left open to include any of them at any time. Consists of data collection, modification of these depending on what parameters and instructions received and put in a dummy variable for later transmission to be triggered by "infook2" that will again run the state machine.

CODE:
```
entity infocontrollersender is
   Port ( directionport : out  STD_LOGIC;
          instruction : in  STD_LOGIC_VECTOR (8 downto 0);
          parameters : in  STD_LOGIC_VECTOR (8 downto 0);
          infook : in  STD_LOGIC;
                              paraok : in STD_LOGIC;
                              putinfo : in STD_LOGIC;
                              start : in STD_LOGIC;
          infooutok : out STD_LOGIC;
          businfoout : out  STD_LOGIC_VECTOR (8 downto 0));
end infocontrollersender;

architecture Behavioral of infocontrollersender is
type fsm is (st0,st1,st2,st3,st4,st5,st6,st7,st8,st9);
signal act_st, next_st : fsm;
signal instructionaux : STD_LOGIC_VECTOR (7 downto 0);
signal idaux : STD_LOGIC_VECTOR (7 downto 0);
signal lenghtaux :STD_LOGIC_VECTOR (7 downto 0);
signal parameter1aux :STD_LOGIC_VECTOR (7 downto 0);
signal parameter2aux :STD_LOGIC_VECTOR (7 downto 0);
signal parameter3aux :STD_LOGIC_VECTOR (7 downto 0);
signal checksumaux :STD_LOGIC_VECTOR (7 downto 0);
signal starttrans :STD_LOGIC;
signal infook2 :STD_LOGIC;
signal numberpara :integer range 0 to 10000;

begin
        process(start)
```

```
begin
        if start'event and start='1' then
                starttrans<='1';
                instructionaux<= X"??";
                idaux<= X"??";
                lenghtaux<= X"??";
                numberpara<=2;
                parameter1aux<= X"??";
                parameter2aux<= X"??";
                --parameter3aux<='??';
                checksumaux<= X"??";
        end if;
end process;

process(putinfo,starttrans,infook2)
begin
        if putinfo'event and putinfo='1' then
                act_st<=next_st;
                infooutok<='0';
        end if;
        if starttrans'event and starttrans='1' then
                act_st<=next_st;
                infooutok<='0';
        end if;
        if infook2'event and infook2='1' then
                act_st<=next_st;
                infooutok<='0';
        end if;
end process;
process(act_st)
begin
        case act_st is

                when st0 =>
                        businfoout<= X"FF";
                        infooutok<='1';
                        next_st<=st1;

                when st1 =>
                        businfoout<= X"FF";
                        infooutok<='1';
                        next_st<=st2;

                when st2 =>
                        businfoout<=idaux;
                        infooutok<='1';
                        next_st<=st3;

                when st3 =>
                        businfoout<=lenghtaux;
                        infooutok<='1';
                        next_st<=st4;

                when st4 =>
                        businfoout<=instructionaux;
                        infooutok<='1';
                                if numberpara=0 then
                                next_st<=st8;
                                else
                                next_st<=st5;
                        end if;

                when st5 =>
                    businfoout<=parameter1aux;
                    infooutok<='1';
                        if numberpara=1then
                        next_st<=st8;
                        else
                        next_st<=st6;
                        end if;

                when st6 =>
                    businfoout<=parameter2aux;
                    infooutok<='1';
                        if numberpara=2then
                        next_st<=st8;
                        else
                        next_st<=st7;
                        end if;

                when st7 =>
                    businfoout<=parameter3aux;
                    infooutok<='1';
                    next_st<=st8;

                when st8 =>
                    businfoout<=checksumaux;
                    infooutok<='1';
                    next_st<=st9;

                when st9 =>
                    starttrans<='0';
            end case;
        end process;

        process (paraok)
            begin
                if numberpara= 0 then
                    parameter1aux<=parameters;
                    numberpara<=numberpara+1;
                end if;
                if      numberpara= 1 then
                    parameter2aux<=parameters;
                    numberpara<=numberpara+1;
                end if;
                if      numberpara= 2 then
                    parameter3aux<=parameters;
                end if;
        end process;
        process (infook)
            begin
                if infook'event and infook='1' then
                --This section will look at the entrance of the bus and
                --given a response depending on what you want to achieve
                end if;
        end process;
    end Behavioral;
```

RESOURCES:

for this project will be used with a separate power actuators, a controller and a computer support for progamacion controller. The card with corresponding servo driver are actuators for precision control of speed, torque and position. These replace hydraulic and pneumatic drives (except for high torque applications) and are the best alternative performance against drives using frequency converters, because they do not provide control of position and are ineffective at low speeds, as compared solutions with stepper motors, since the latter provide no position control of such precision and are limited to low power applications. The main disadvantage of servo systems is that they are generally more expensive than electric alternatives.

This project will use the type servomotors Dynamixel ax-12. Which are among the most competent servo market. AX-12 servo motors are driven by a serial link to 1Mbauds and can be attached to each other to create chains of servomotors (Daisy-Chain). Up to 254 operators can be connected in series with the Dual-POB in one cable. These motors have a resolution of 1024 steps and a variable speed also in 1024 steps. In reading, you may find your current position, its speed, torque, etc.. It is possible to detect an obstacle, the wheel slippage or the presence of a slope without additional sensor.

This is the way that we have to handle the engine. The actuator has an internal memory eprom that will allow us to read or modoficar data at will through the functions that we send at any time

The specifications of these actuators allow an amazing amount of intelligent actuator aplications.This gives information on his condition. Furthermore, the AX-12 + actuator enables all types of robotic assembly significantly reducing the number of sensors required. You can connect up to 254 types of AX-12 + servo in series in the Dual-POB. Dual-POB can know and control the position, torque and engine speed. Dual-POB can be used as a super card I / O module LEGO Mindstorms NXT, and thus can control the servomotor AX-12 + from the NXT.

General Characteristics:

Torque: 16.5 kg. Cm below 10V (12 under 7V).
Speed: 0.196 sec / 60 degrees below 10V (0.269 low 7V).
Resolution: 0.35 °.
Maximum current: 900mA.
Racing: for multipunzado TTL serial link.
Subjugation over 300 ° C maximum (download at will, programming).
Ability to pilot in continuous rotation without change of substance.
Size (L x H x D): 32 x 50 x 38mm.
Weight: 55g.

For this project we will also use a controller of the type HS-DLP-FPGA. Module DLP-FPGA-HS is a low cost tool, prototyped and compact that can be used for a quick test of concept or educational environments. The module is based on the Xilinx Spartan ™ devices 3A and Future Technology Devices International's FT2232H Dual-Channel High-Speed USB IC. The DLP-HS-FPGA provides both the beginner and experienced engineer, a quick way to access the development of FPGA based designs. When combined with free tools from Xilinx ISE ™ WebPack, this module is more than enough to create anything from basic logic functions operating at a large complex system controller.

Fortunately, a channel of the dual-channel USB interface is used to upload files directly to user bits of the SPI Flash and requires no external programmer. This represents a considerable savings because there is no need for additional programming cable for configuring the FPGA. All you need to upload the files on the PLD-HS-FPGA is a Windows software utility (free), a Windows PC and a USB cable. The module can also be programmed into the Xilinx ISE program using a Xilinx programming cable (sold separately).

The DLP-HS-FPGA is fully compatible with free software from Xilinx ISE ™ WebPack. This provides the ideal development environment for FPGA designs with HDL synthesis, simulation, implementation, and additional mounting devices and JTAG programming.

The DLP-HS-FPGA has embedded voltage regulators to generate all power supply voltages required from a single source of 5 volts. Power for the module can be taken from the USB port or an external source of 5 volts supplied by the user through a standard connector DC Barrel Jack

Connections to the outside of the plate is done through 50 pins arranged in the bottom of the plate, 0.9 inches wide, and another 26-pin, 0.05 inch wide 2x13 sited at the top . The bottom plate 50 pin provides access to 41 inputs / outputs of the FPGA and the top gives access to 22 input / outputs. The set also contains a 50pin socket, connector space DIP 0.9-inch so that the user can connect at the bottom of the plate and another 0.05 inches 2x13conector space, as FFSD-13-D-xx, xx-01, so that the user can place at the top.

General Characteristics:

Equivalent logic cells: 4032
CLB Array:
Rows: 32
Columns: 16
CLB Total: 448
Slices Total: 1792
Flip Flops Total: 3584
4-Input LUT Total: 3584
Bits distributed RAM: 28K
Bits block RAM: 288K
Dedicated Multipliers: 16
DCM: 4

BIBLIOGRAPHY

- **ALONSO M., FINN E.**, "FÍSICA" (vol 1, mecanica)
  Edit. Addison-wesley / Pearson. España 1999 , 530p

- **SERWAY R., JEWETT JR J**. "FÍSICA PARA CIENCIAS E INGENIERIAS"
  Edit. Thompson Mexico 2005, 695p

- **CRUSTCRAWLER: DYNAMISEL AX-12+ MANUAL (2011)**
  http://www.crustcrawler.com/motors/AX12/index.php 38p

- **DLP DESIGN: DLP-HS-FPGA & HS-FPGA2 USB - FPGA Module (2011)**
  http://www.dlpdesign.com/fpga/hsfpga.shtml 18p

- **PONG P. CHU** ,"FPGA PROTOTYPING BY VHDL EXAMPLES"
  Edit. Wiley-interscience , New Jersey 2008 , 440p