Final Project

VERSATILE PULSE-WIDTH MODULATOR

Author: Miguel Padilla Pérez
Director: Gabriel José Capellá
Degree: Industrial engineering in electronics and industrial automation
Department: Power electronics
Year: 2013
I want to thank all the people, my family, my girlfriend, my friends and teachers, support during these years of university. And I want to congratulate me get myself for this.

I will never lose the anxiety of flying.
8. EXPANSION OF THE APP ........................................................................................................ 43
  8.1 Adaptation to three phase structure. .................................................................................. 43
    8.1.1 3 carriers .................................................................................................................... 43
    8.1.2 External device ........................................................................................................... 43
  8.2 Carrier modifications ....................................................................................................... 44
    8.1.2.1 Frequency carrier increase ..................................................................................... 44
    8.1.2.2 Increasing the number of carriers .......................................................................... 44
    8.1.2.3 Signal offset ............................................................................................................ 45
  8.3 Work with more SPARTAN peripherals ............................................................................ 45
  9. CONCLUSION .................................................................................................................... 46
ANNEX ....................................................................................................................................... 47
  PROGRAM CODE .................................................................................................................. 48
  INFORMATION ...................................................................................................................... 66
  BIBLIOGRAPHY .................................................................................................................. 66
  WEBGRAPHY ....................................................................................................................... 66
1. INTRODUCTION

When I try to explain what I'm doing in this project people say: "Why don't you do it with a microprocessor, it's faster, easier and you can program using C language?". At first, when I started the project, I wasn't able to explain because I hadn't changed my mentality. At this moment I know what I'm doing and know the answer because "I don't design software but I design hardware"

On this project I’ll try to do a design based on an FPGA that is inserted in SPARTAN 3AN board. This design will consist of modulating a sinusoidal waveform signal so that the FPGA output will provide a PWM signal. To do this application will use a carrier signal. This signal can be configured in several ways: face shift, level shift, etc. With this method we could use this output signal for applications like: controlling motors, modulating inverters or energy management.
2. WHAT IS AN FPGA?

The acronym of FPGA is Field Programmable Gate Array, in contrast to Application Specific Integrated Circuits (ASIC) that are design for only one specific application, this type of device could be programmable many times. Although One-Time Programmable (OTP) FPGAs are available, the dominant types are SRAM-based which can be reprogrammed as the design evolves.

This device is based around matrix of configurable logic blocks (CLB) interconnected (Fig. 1).

![Figure 1](image)

The CLB is a device that consists of a configurable switch matrix with 4 or 6 inputs, some selection circuitry (MUX, etc), and flip-flops. The switch matrix is highly flexible and can be configured to handle combinatorial logic, shift registers or RAM (Fig. 2).

While the CLB provides the logic capability, flexible interconnect routing routes the signals between CLBs and to and from I/O’s.
2.1 FPGA VS. MICROCONTROLLERS OR MICROPROCESSORS

As I explained more or less in the introduction for understanding this type of device we will need change our mind and follow the philosophy “Think hardware not software”.

When you work with microcontrollers or microprocessors what you really do is design software, actually you are describing the behavior of this type of device because a microprocessor is an already designed hardware design embedded and you must work with these hardware specifications.

Then when you are work on an FPGA project what you do is design a hardware that means that the device being adapted you, that’s true up the point when you design some app you can draw like a hardware scheme and compile.

An FPGA is not better than a microprocessor because an FPGA doesn’t include A/D converters or another output. Another aspect to consider is FPGA works with RAM memory, this means that if you will do reset on FPGA you will erase all that you design while on microcontroller no because works with flash memory. But for this reason we can’t fall on the comparisons of these types of devices.
2.2 APPS THAT WORKS WITH FPGA

The FPGAs are used in many industries. Below see a few:

AUTOMOTIVE

FPGAs are increasingly more within the automotive sector. Enter an FPGA in this sector favors the following:

- Increase system performance
- Reduce BOM cost
- Meet stringent requirements with fully-tested, automotive-grade devices

Nowadays we will see too many applications be used FPGA, on next figure can see the different parts to use FPGA in a car (figure3).

![Figure3](image)

PROTOTYPING

The reprogramming capability and easy adaptation to other peripherals make the FPGA a device very suitable for the construction of prototypes. Once the objective of the project the same settings can be applied to an ASIC or composed entirely designed.

AUDIO

The speed of FPGA is a very good factor for the use of this device in the audio industry. This speed can contribute to greater accuracy when digitizing sound, besides the multiple options signal transformations that can be done to generate sounds or effects.
Due to its high speed FPGAs are well suited for capturing images and videos. Also for modification, filtering, processing and issuance due to the high versatility with industry peripherals like video cameras, cameras, IP cameras, sending pictures, etc.
3. SPARTAN 3 AN

The FPGA with which we will work is integrated in a plate Xilinx brand called Spartan 3 AN (Figure 4).

![Figure 4](image)

The good thing about working with this type of plate is that the device is surrounded by peripherals. Below show the characteristics of this board.

- **Clocks**
  - 50 MHz crystal oscillator on-board
  - Open slot for optional user-installed clock
- **Memory**
  - 4 Mbit Platform Flash PROM
  - 32M x 16 DDR2 SDRAM
  - 32 Mbit parallel Flash
  - 2-16 Mbit SPI Flash devices
- **Analog Interface Devices**
  - 4-channel D/A converter
  - 2-channel A/D converter
  - Signal amplifier
- **Connectors and Interfaces**
  - Ethernet 10/100 PHY
  - JTAG USB download port
  - Two 9-pin RS-232 serial port
- PS/2-style mouse/keyboard port
- 15-pin VGA connector capable of 4,096 colors
- One FX2 100-pin and two 6-pin expansion connectors
- 20 user I/O available on standard header pins
- Stereo mini-jack for PWM audio
- Rotary/push button function switch
- Eight individual LED outputs
- Four slider switches, four push-button switches.

**Display**
- 16 character, 2-Line LCD

Of all the peripherals that there we will use the A/D and amp regulator.

### 3.1 ANALOG CAPTURE CIRCUIT

The SPARTAN 3 AN board includes a two-channel analog capture circuit, consisting of a programmable scaling pre-amplifier and an analog-to-digital converter (ADC) (Figure 5).

![Figure 5](image)

The reference voltage for the amplifier and the ADC is 1.65V, generated via a voltage divider. Consequently, 1.65V is subtracted from the input voltage on VINA or VINB.

The maximum range of the ADC is ±1.25V, centred on the reference voltage, 1.65V. Hence, 1.25V appears in the denominator to scale the analog input accordingly.

Finally, the ADC presents a 14-bit, two’s complement digital output. A 14-bit, two’s complement number represents values between \(-2^{13}\) and \(2^{13}-1\).
3.1.1 PROGRAMMABLE PRE-AMPLIFIER

The preamplifier is used to scale the input of the A / D. On this scale plate is negative because it does not support large voltage values. The higher the level the greater the scales at the time of voltage transform.

Each analog channel has an associated programmable gain amplifier. Analog signals presented on the VINA or VINB inputs are amplified relative to 1.65V.

The amplifier interface is relatively slow, supporting only about a 10 MHz clock frequency.

3.1.2 A/D CONVERTER

After passing the signal through the preamp the signal goes to the A / D to be transformed.

The ADC presents the digital representation of the sampled analog values as a 14-bit, two’s complement binary value. Values are united in a pattern of 28 bits that have to be separated beforehand.

The maximum sample rate is approximately 1.5MHz.

3.2 SPI (Serial Peripheral Interface)

In this headland I would talk about the SPI protocol. The SPI protocol it’s a standard which use for communication with peripherals and devices.

This standard it's integrated on the SPARTAN 3 for communicate the FPGA with all the peripheral devices.

To begin a communication, the bus master first configures the clock, using a frequency less than or equal to the maximum frequency the slave device supports. Such frequencies are commonly in the range of 10 kHz–100 MHz. At case of the app, for example, the SPI SCK is declared only one time (2.5 Mhz) and is the same for all peripheral communications.

The basic structure of this protocol is the following.
The above picture shows a simple connection with SPI standard. That connection consists of 4 signals SPI CLK, MOSI, MISO and SS. Now, I will explain each signal:

- **SCLK**: serial clock (output from master);
- **MOSI**: master output, slave input (output from master);
- **MISO**: master input, slave output (output from slave);
- **SS**: Slaves select (active low, output from master).

That communication supports full duplex. That means that while the masters send the requests at the same time receives the information.

In SPARTAN3AN board the connection structure is a little different because don't need SS connection. Each device has one pin assigned of FPGA and the only thing that we can do is configure if it's input or output. Except the SPI_SCK that it's common in all devices.
4. POSSIBLE PROGRAMMING LANGUAGES

Usually, two types of language can be used to program an FPGA. Schematic or VHDL.

4.1 Schematic
The Schematic is a type of language that can create the application drawing directly the digital circuit using logic gates or boxes with functions written in code (Figure).

As it can be seen in the image its appearance is neater than a code that can be written in C or ASM.

In the case of this project we does not using this programming language so we will not go much deeper.

4.2 VHDL

When we refer to the VHDL language we mean written code. Within this code we can find two dialects or variants: the Verilog or VHDL.

4.2.1 Verilog

The Verilog code is a type of programming language that is located at a level closer to that of human interpretation; let us say that it is the highest level. While VHDL is a lower-level code and is which our application is based on.

4.2.2 VHDL

The VHDL, as discussed above, is the language where I do my application. It consists of in a code that is in the middle of the low-level language (As an example we can mention the introduction of bytes full in a variable) and a language of a slightly higher level (As an example we can mention the cycles or loops that can be used as the if, case, while, for, and so on..)
This language is structured in two parts that are entity and architecture (Figure).

4.2.2.1 Entity

In the entity section we declare the variables inputs and outputs of the project, or what the application relates to the configuration of the inputs and outputs of the FPGA.

As seen in the picture the declaration of the input or output is defined by the following steps:

1. Insert the name of the I/O.
2. Define whether input or output.
3. Define the variable type.

In the example in the picture above we can see how it is defining an entry named DU which is a 7-bit binary variable.
Also at the time of declaring the FPGA inputs and outputs, they need to be defined as input or output properly but you can also make them bi-directional. In addition you can also add different initial state defined variables.

4.2.2.2 Architecture

The architecture section it is divided into two subsections called, declarative and body.

```
architecture hamcorr of hamcorr is
  function syndrome (D:STD_LOGIC_VECTOR)
    return STD_LOGIC_VECTOR is
      variable SYN: STD_LOGIC_VECTOR (2 downto 0);
      begin
        SYN(0):= D(1) xor D(3) xor D(5) xor D(7);
        SYN(1):= D(2) xor D(3) xor D(6) xor D(7);
        SYN(2):= D(4) xor D(5) xor D(6) xor D(7);
      return SYN;
    end syndrome;
    begin
      process(DU)
      variable i : INTEGER;
      begin
        DC<=DU;
        i:=CONV_INTEGER(syndrome(DU));
        if i=0 then NOERROR <='1';
        else NOERROR <='0'; DC(i) <=not DU(i); end if;
      end process;
    end hamcorr;
```

4.2.2.2.1 Declarative

In the declarative, global variables are declared as the signed, the shared variables, or the constant. These variables will be used throughout the program and they can also take an initial value.

4.2.2.2 Body

In the part of the body is where local variables are declared and used to programme is developed.

The part of the body these code development processes can be divided. These processes will be executed in parallel and independently from each other (one of the great advantages offered by the FPGA compared to processors that run sequentially)

In conclusion, as we see in the example program, we can see the mixture of levels of language that offers this type of code. We can see actions more typical of a low level, as the `DU <= DE;`, or a higher level commands such as loops if, the process, etc..
5. MY APP

The purpose of this project is to obtain a PWM modulated signal, which represents a sine wave of frequency 50Hz and amplitude of 1 volt.

Now, I will explain briefly each part in which the project is divided.

Reading from left to right, first we find a sinusoidal signal with amplitude of 1 V and a frequency of 50 Hz to be introduced in the SPARTAN3AN and that will be our reference signal or signal to modulate.

Then, inside plate, we proceed to the conversion of the input signal. Passing first by a preamplifier, to adapt the signal to the converter, and then by the converter that will transform the analogical signal into binary signal in addition two formats. On parallel there is a formation of a virtual carrier generated by software with an editable form.

At the same time both signals (digital sinusoidal and carrier changed) will go through a comparator that will shape the signal pulse width modulated.

5.1 Analog capture circuit

As we already knew the Spartan3AN board includes two-channel analog capture circuit, consisting of a programmable scaling pre-amplifier and an analog-to-digital converter (ADC) (Figure).
As shown in the above image we can’t convert an analogical input without having gone before the preamp. For this reason we will first explain how to prepare or program the preamplifier.

### 5.1.1 Preamp

The preamplifier integrated into the board is a LTC6912. Is a family of dual channel, low noise, digitally programmable gain amplifiers (PGA) that are easy to use and take up little PC board space. The gains for both channels are independently programmable using a 3-wire SPI interface to select voltage gains of 0, 1, 2, 5, 10, 20, 50, and 100V/V (LTC6912-1).

In the next table you can see the interface signals between the FPGA and the amplifier. The SPI_MOSI and SPI_SCK signals are shared with other devices (D/A in this case) on the SPI bus. The AMP_CS signal is the active-Low slave select input to the amplifier.
Once known the signs with which I work the next step is to know on what scale I want to adjust the preamp. The table below shows the different levels of amplification and voltage ranges accepted.

<table>
<thead>
<tr>
<th>Signal</th>
<th>FPGA Pin</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI_MOSI</td>
<td>AB14</td>
<td>FPGA→AMP</td>
<td>Serial data: Master Output, Slave Input. Presents eight-bit programmable gain settings, as defined in Table 9-2.</td>
</tr>
<tr>
<td>AMP_CS</td>
<td>W6</td>
<td>FPGA→AMP</td>
<td>Active-Low chip select. The amplifier gain is set when the signal returns High.</td>
</tr>
<tr>
<td>SPI_SCK</td>
<td>AA20</td>
<td>FPGA→AMP</td>
<td>Clock</td>
</tr>
<tr>
<td>AMP_SHDN</td>
<td>W15</td>
<td>FPGA→AMP</td>
<td>Active-High shutdown, reset</td>
</tr>
<tr>
<td>AMP_DOUT</td>
<td>T7</td>
<td>FPGA←AMP</td>
<td>Serial data. Echoes previous amplifier gain settings. Can be ignored in most applications.</td>
</tr>
</tbody>
</table>

Considering that I want to introduce a signal with 1 Volt of amplitude I will choose -1, which is the widest range.

The next step is to know how I have to use the different signals to use by the device (FPGA) to set the preamp. The gain for each amplifier is sent as an eight-bit command word, consisting of two four-bit fields. The most-significant bit, B3, is sent first.
The SPI bus transaction starts when the FPGA assign AMP_CS Low. The amplifier captures serial data on SPI_MOSI on the rising edge of the SPI_SCK clock signal. The amplifier presents serial data on AMP_DOUT on the falling edge of SPI_SCK.

\[\text{AMP_CS}\]
\[\text{SPI_SCK}\]
\[\text{SPI_MOSI} \text{ (from FPGA)}\]

stop on the way to the configuration of the preamplifier to talk about the SPI clock devices. It is a step to consider because the clock operation of the FPGA cannot be at same frequency as the analogical conversion devices for physical issues. So I think it is important to comment that there has been a reduction of PCB clock to obtain the desired SPI clock signal.

Below is showed the simple code and the simulation of this reduction.

```
if clk'event AND clk='1' then
  reductor:= reductor+1; -- Increase the value in each rice
  if reductor = 10 then
    estado:='1';
  elsif reductor = 20 then
    estado:='0';
    reductor:=0;
  end if;
end if;

spi_sck<= estado; -- 2.5Mhz clock
```

As seen in the image where signals are represented SPI_CLK and clk simulated SPI have decreased the signal into a signal 20 slower. Clk =50 MHz and SPI_CLK = 2.5 MHz. Now we can continue with the preamp configuration.
5.1.1.1 Preamp code.

Now I'm explaining what I have written in each section of the program structure to configure the preamp.

In the entity declare the inputs and outputs signals of the FPGA settings which I need.

```vhdl
entity RELOJ is
    Port (      SPI_SCK : out STD_LOGIC:= '0';
                AMP_CS: OUT STD_LOGIC:= '0';
                SPI_MOSI: OUT STD_LOGIC:= '0';
                AMP_SHDN: OUT STD_LOGIC:= '0');
end RELOJ;
```

Then in the declarative declare the signal and constants variables needed.

```vhdl
architecture Behavioral of RELOJ is
    SIGNAL AMPLIPHASE : STD_LOGIC:= '0';--flag to enable the preamp phase
    CONSTANT GAIN_CONFIG: STD_LOGIC_VECTOR(7 DOWNTO 0):= "10001000";--GAIN SELECTED
begin
```

In the body describe the variables that I use.

```vhdl
amp_prog: process (clk,ad_dout)
    begin

        variable espera : integer :=0;--Slows the configuration starts
        variable counter_amp: integer range 0 to 342:=0;
        variable amp_cs_var: std_logic:= '0';
        variable amp_spi_mosi_var: std_logic :='0';
        variable amp_dout_var: std_logic_vector (0 to 7);
```
Below is a copy paste of the development of the code for the configuration, but is too long and I will explain the process I follow through an image which appears below. Also, the code is attached in the Annex.

The aim of the code that has been created for preamp configuration is to take the advantage of working at a higher frequency than device works. That way we can advance or delay the flanks of SPI_CLK known time constraints. Below there's an image where you can see the sequence generated by me in different outputs that go to the preamplifier.

Then, using a case-type loop has been set with an accuracy of 1/20 each of the input signals of the device and this generates the desired sequence for the configuration.

Now we show an image where we can see the real reaction using an oscilloscope preamplifier.
Explaining the above image we can see as our top of the screen signal is AMP_CS signal that is initiated by the flow of data input. The bottom of the screen signal is AMP_DOUT signal, this signal issue an echo sequence introduced previously. The constant value is introduce in the constant GAIN_CONFIG is equivalent to the gain -1 is "10001000".

*From the experience gained in the actual configuration is very advisable to introduce previously a sequence with values 0’s and after the sequence with the desired values.*

5.1.2 A/D.

Once the preamplifier is updated all the voltage I put on port J22, adapted to the A / D.

The A / D converter is a LTC1407A-1. It is a two-channel converter 14 bits. The LTC1407A contain two separate differential inputs that are sampled simultaneously on the rising edge of the CONV signal. These two sampled inputs are then converted at a rate of 1.5Msps per channel.

In the image below you can see the lists of signals between the FPGA and the ADC. The SPI_SCK signal is shared with other devices on the SPI bus. The active-High AD_CONV signal is the active-Low slave select input to the DAC. The DAC_CLR signal is the active-Low, asynchronous reset input to the DAC.

<table>
<thead>
<tr>
<th>Signal</th>
<th>FPGA Pin</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI_SCK</td>
<td>AA20</td>
<td>FPGA→ADC</td>
<td>Clock</td>
</tr>
<tr>
<td>AD_CONV</td>
<td>Y6</td>
<td>FPGA→ADC</td>
<td>Active-High, initiates conversion process.</td>
</tr>
<tr>
<td>ADC_OUT</td>
<td>D16</td>
<td>FPGA→ADC</td>
<td>Serial data. Presents the digital representation of the sampled analogical values as two 14-bit two’s complement binary values.</td>
</tr>
</tbody>
</table>

Once we know the signals we can proceed to view and control the conversion.

When the AD_CONV signal increases, the ADC simultaneously samples both analogical channels. The results of this conversion are not presented until the next time AD_CONV is asserted, a latency of one sample. The ADC presents the digital representation of the sampled analogical values as a 14-bit, two’s complement binary value.
The AD_CONV signal is not a traditional SPI slave select enable. Be sure to provide enough SPI_SCK clock cycles so that the ADC leaves the ADC_OUT signal in the high-impedance state. As shown two images above, use a 34-cycle communications sequence. The ADC 3-states its data output for two clock cycles before and after each 14-bit data transfer.

### 5.1.2.1 A/D code.

As in the previous code show step by step what I typed in the code to control the converter.

First show the inputs and outputs signals that refer to the A/D.

```vhdl
SPI_SCK: out STD_LOGIC:='0';
AD_CONV: OUT STD_LOGIC:='0';
AD_DOUT: IN STD_LOGIC;
```

Then in the declarative declare the signs and constants variables needed.

```vhdl
SIGNAL ADPHASE: STD_LOGIC:='0';--flag to start the converter
shared variable canalA: std_logic_vector (0 to 13);
shared variable canalAint: integer range -9000 to 9000:=0;
shared variable canalB: std_logic_vector (0 to 13);
shared variable canalBint: integer range -9000 to 9000:=0;
```

In the body describe the variables which I use.

```vhdl
----VARIABLES PARA EL CONVERSOR-------
variable counter_ad: integer range 0 to 668 :=0;
variable ad_conv_var: std_logic:='0';
variable ad_dout_var: std_logic_vector (0 to 27);
variable senal_captura: std_logic:='0';
variable reductormost: integer range 0 to 100:=0;
```
As in the previous use the same code to reproduce the sequence loop. As we work at a higher frequency than the capture device can place in the most optimal position to capture the bit of conversion (the code is in the appendix).

Thanks to the simulation tool we can see the sequence that we generated through the code. As you can see in the picture the first two signals are watches application SPI_CLK and CLK, as we see when CLK twenty times faster than SPI_CLK cannot appreciate the sequence of CLK. AD_CONV flag is the start of the conversion that starts when ADPHASE activated. The data received from the A/D is routed by the signal AD_DOUT and these are stored in the variables CANALA and CANALB.

Also with the help of another ISE tool pack (which is commented in another chapter) we see the capture of data during a given cycle time.

In the picture above we see the sinusoidal signal capture converted. The signal has no way of sinusoid because the data that is read from the converter are in addition two and the plotter reads the data as binary values, but you can see the amplitude and shape perfectly.
5.1.3 Carrier signal.

The carrier was from the beginning designed to be generated within the FPGA. Loops are formed on type 'if' that generate a signal period. They have a width of 14 bits, this dignified, 16384 values.

According to this application can be four possible forms of carrier:

- Carrier peak to peak

![Carrier peak to peak](image1)

Is a 1.5 kHz signal with an amplitude default values 16384.

- Sawtooth carrier

![Sawtooth carrier](image2)

Is a 3 KHz signal with an amplitude default values 16384.
• Face shift

It consists of two 3 kHz signals with amplitudes of 8192 values that are shifted 180 degrees and opposite polarity.

• Level shift

Consists of two 3 KHz signals with amplitudes of 8192 values and opposite polarity.

These carriers can be selected by the first two switches, starting from the left, which are located in the board. These switches are linked to a variable named 'canalsel'.
5.1.3.1 Carrier code.

As the carrier is generated internally in the entity section there any high resolution variable related to this block.

In the architecture section is where I have done the basic variables of the application.

```verbatim
shared variable canalA: std_logic_vector ( 0 to 13);
shared variable canalAint:integer range -9000 to 9000:=0;
shared variable canalB: std_logic_vector ( 0 to 13);
shared variable canalBint:integer range -9000 to 9000:=0;
shared variable porta_share: integer range -9000 to 9000;
shared variable portap: integer range -9000 to 9000;
shared variable portan: integer range -9000 to 9000;
```

In the section of the body it will appear the variables needed to be build.

```verbatim
variable portadora: integer :=0;
variable portapos: integer:=0;
variable portaneg: integer:=12500;---para face shift
variable portaposi: integer:=0;
variable portanega: integer:=0;---para level shift
variable portasierra: integer:=12500;--para señal de sierra
variable flag: std_logic:='0';
variable contcarry: integer:=0; --reduccion para la frecuencia de la portadora
variable avisoport: std_logic:='0';
```

The code which generates the carrier is attached in the Annex. Each of the carriers is generated through a loop-type 'if'.
5.1.4 Comparator

The comparison phase is the last phase of the application. As its name suggests compares the values obtained from the conversion to the values of the carrier. If the carrying value is less than the value of the catch signals' PWM1 'or' PWM2 '(depending on the selected carrier) are equal to 1 if not it will be equal to '0'. The variables' PWM1 'and PWM2' are linked to 'pwmsource1' and 'pwmsource2' which are the outputs of the FPGA.

```vhdl
entity signals is

  -- Entity signals.
  attribute visibility : string := "public";

  -- Variables for comparison
  variable pwm1 : STD_LOGIC := '0';
  variable pwm2 : STD_LOGIC := '0';

end signals;
```

This code is also attached to Annex.
6. XILINX ISE DESIGN TOOLS.

In this chapter I will talk about the ISE tools that I have used to do my application.

6.1 ISim

In my opinion is one of the most fundamental applications of ISE.

This program can:

- Change the clock frequency.
- Enter or force input signals.
- Simulate signals whose shape changes over time.
- Breakpoints.
- Make time calculations.
- Simulate for a specific period of time.
- Observe values obtained from the program.

In my experience with the project it has been vital to observe and respect the time required by the board peripherals.

6.2 Chip Scope Pro

ChipScope Pro is a program for monitoring and displaying variables in our application to the FPGA. This program captures over a limited number of machine cycles storage of values.

Below I will make a brief tutorial on how this application works.

1. First, the main step is to have a code that can compile properly. For that first used the previous tool ISim.
2. Once the first steps now let the application environment of ISE and click the right mouse button where we will open a small window.

3. Within this small window click on the operation 'New source …' and we open a window.

4. This window sounded like when we introduce vhd file but now we click on the option 'ChipScope Definition and Connection file'. Furthermore, in the same window, enter the name of our file (ej. test). We click next.

5. Then windows where we give information about the file, at the moment you click on the Finish button.

6. At this point, just below the vhd file appears document 'test'.
7. The next step is to double click on the file created. At that moment a window from ChipScope program. This information appears where the file because it belongs board. We click Next.

8. At next window press directly ‘next’.
9. In the next window began to set the variables we will want to capture. First select the number of variables (in the example two). Then introduce the variable size and type (in the example consists of a 14-bit variable). Once configured press next.

10. In the next window you can set the number of machine cycles or warning signs that you can configure a maximum of 16384. We can also enable captured variables.
11. In the last window we link configured variables to real variables of the FPGA. In addition to introducing an input variable, the program calls clock to know when to capture the value of the variable.

12. If you double click on one of the variables to fill we see a new window. It is a search engine of our code vhd that link our variables to the set. In the window enter the name of the variable (in the example canal *). Once you have found the desired variable link this with the button 'Make connections'.
13. Once you press the button linked all 'OK' and return to the previous window. In this window click on the button 'Return to Project Navigator'. The window will close and return to ISE programming environment.

14. Once back in the programming environment click on the vhd file and below the window double click on 'Analyze Design Use ChipScope'.

15. ChipScope Pro opens.

16. With SPARTAN 3AN connected click on the button 'Open Cable'.
17. We waited a moment and a small window that describes the peripheral that is connected. Push OK button.

18. The next step is to click with the right mouse button 'MyDevice0' and click on the configure option.

19. Then a window will appear where you will have to enter the bit file that was created in the compilation. Also activate the 'Clean previous project settings' to erase the previous configuration.
20. Once you add the file and pressed the button ‘ok’ appear a small window where you will return to our private information about SPARTAN.

21. After pressing 'ok' we will see in the window of 'project' appears some options.
22. One of the options is 'Trigger Setup' where you can set the number of catches to execute, its value and start or capture.

![Trigger Setup](image)

23. Another option is 'Waveform' that will make a collection of all the values captured during that time interval.

![Waveform](image)

24. 'Listing' is an option that does the same as 'Waveform' but vertically collects data to be exported for example.

25. 'Bus plot' makes a graph of the data on time

![Bus plot](image)
Personally this program has helped me to see step by step what was really going on in the code I've done. I hope this little tutorial helps for future applications.
7. APPLICATION TEST

In this chapter I will show some screen captures of the app.

7.1 Peak to peak carrier.

The above picture shows the input and output of the application when the carrier mode is peak to peak. The top signal is the input and the lower signal is the output.

7.2 Level shifted carrier.

The following captures refer to the level shifted carrier. The output of this PWM is double because it's separated in two output signals (PWM1 and PWM2). PWM1 represents the positive part of input signal and PWM2 represents the negative part.
7.3 Face shifted carrier.

The following captures refer to the face shift carrier. The output of this PWM is double because it’s separated in two output signals (PWM1 and PWM2). PWM1 represents the positive part of input signal and PWM2 represents the negative part.
7.4 Saw tooth carrier

Last capture represents the input signal and output signal of saw tooth carrier mode.
8. EXPANSION OF THE APP

Due to lack of time, the possible improvements or adaptations have been cut off. For this reason I will write some improvements and modifications just in case any school mate wants to continue the evolution of it.

8.1 Adaptation to three phase structure.

At the moment the app is only set up for a single-phase installation but with some modification it can be expanded to a three-phase installation. Let us see some examples:

8.1.1 3 carriers

This modification consists of increasing the number of carriers up to three. The way of doing this is to create three virtual carriers with 120 degree relative shifting. We also need three comparison loops to create the three PWM signals.

8.1.2 External device

Due to the limitation of A/D converter there are only two analog inputs and this is a problem for the three-phase structure. But luckily, we can have a lot of bidirectional ports in SPARTAN 3 AN. We can use one of these ports to connect an multichannel external converter or three single external converter. With that solution the three-phase control can be implemented.
8.2 Carrier modifications

Another possible modification is changing the shape or the frequency of the carriers. Below I will explain some possible changes.

8.1.2.1 Frequency carrier increase.

In the current app there is a limitation referring to the frequency of carrier. Because the carrier signal is generated with a 14 bit D/A converter, if we want to use 16384 steps to generate a full reference period, the maximum frequency of the reference signal would be 1.5 kHz due to the limitation in the timing resources of the device.

For this reason one change we could do is decrease the number of steps use to generate the carrier signal to increase your frequency. That seems too easy but the difficulty of this is that the value of analogical capture must be decreased also and this reduces the accuracy of signal.

8.1.2.2 Increasing the number of carriers.

Use SPARTAN peripherals to select the number of carriers use to modulate the reference signal.
8.1.2.3 Signal offset

Use SPARTAN peripherals to select the phase of each carrier.

Phase disposition or phase opposite disposition.

8.3 Work with more SPARTAN peripherals.

Use the built in peripherals (LCD screen, rotary knob, buttons...) to help built a user friendly interface.
9. CONCLUSION

The main objectives of the project have been achieved. A versatile modulator capable of modulating reference signal with for possible carrier dispositions has been implemented. A good command on the tools required to deal with FPGA devices has been acquired too. Mastering these skills has been almost compulsory in order to develop the current project.

Some guidelines on possible ways to expand this project have been included as reference for future students.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
--use ieee.std_logic_arith.all ;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity RELOJ is
    Port (      SPI_SCK : out STD_LOGIC:= '0';
        AMP_CS: OUT STD_LOGIC:= '0';
        SPI_MOSI: OUT STD_LOGIC:= '0';
        AMP_SHDN: OUT STD_LOGIC:= '0';
        AD_CONV: OUT STD_LOGIC:= '0';
        AD_DOUT: IN STD_LOGIC;
        clk : in STD_LOGIC;
        LEDS : OUT STD_LOGIC_VECTOR (0 TO 7);
        inter: in std_logic;
        canalsel: in std_logic;
        pwm1source: out std_logic;
        pwm2source: out std_logic;
        portmod0: in STD_LOGIC:= '0';
        portmod1: in STD_LOGIC:= '0';
          flagout: out STD_LOGIC);
end RELOJ;

architecture Behavioral of RELOJ is
    SIGNAL AMPLIPHASE : STD_LOGIC:= '0';--flag to enable the amplifier
    SIGNAL ADPHASE: STD_LOGIC:= '0';--flag to start conversion
    shared variable CARRIER: STD_LOGIC:= '0';
    shared variable COMPHASE: STD_LOGIC:= '0';--flag to start comparison
    CONSTANT GAIN_CONFIG: STD_LOGIC_VECTOR(7 DOWNTO 0):="10001000";--select gain
    signal amp_dout_var: std_logic_vector (0 to 7);
    shared variable canalA: signed ( 0 to 13);
    shared variable canalB: std_logic_vector ( 0 to 13);
    shared variable porta_shareb: signed (0 to 13);
    shared variable portap: signed (0 to 13);
    shared variable portan: signed (0 to 13);
    shared variable stopcarry: std_logic := '1';

    begin

    spi_clock:process(clk)

            variable reductor : integer :=0;--REDUCE THE FREQUENCY WATCH FOR PREAMP
            variable estado: std_logic;--DISCRIMINATION LEVEL HIGH
            AND LOW

            begin
if clk'event AND clk='1' then
    reductor:= reductor+1;--Increase the value of every
    flank gear up
    if reductor = 10 then
        estado:='1';
    elsif reductor = 20 then
        estado:='0';
        reductor:=0;
    end if;
end if;

spi_sck<= estado;-- 2.5Mhz watch SPI
end process;

amp_prog: process (clk)
begin
    if clk'event AND clk='1' then
        espera:= espera + 1;
        if (espera = 40) then
...
AMPLIPHASE<='1';--to stabilize the clock cycles 
communications wait to initiate communication
END IF;

if (AMPLIPHASE = '1') THEN

  case counter_amp is
  when 0 to 14 =>
  amp_cs_var:='0';
  amp_spi_mosi_var:='0';
  AMPLIPHASE<='1';
  counter_amp:=counter_amp+1;
  --CUSTOMIZE YOUR VALUE EVERY 20 ns
  when 15 to 23 =>
  amp_cs_var:='1';
  amp_spi_mosi_var:='0';
  AMPLIPHASE<='1';
  counter_amp:=counter_amp+1;
  --CUSTOMIZE YOUR VALUE EVERY 20 ns
  when 24 to 173 =>
  amp_cs_var:='0';
  amp_spi_mosi_var:='0';
  AMPLIPHASE<='1';
  counter_amp:=counter_amp+1;
  --CUSTOMIZE YOUR VALUE EVERY 20 ns
  when 174 to 182 =>
  amp_cs_var:='1';
  amp_spi_mosi_var:='0';
  AMPLIPHASE<='1';
  counter_amp:=counter_amp+1;
  --CUSTOMIZE YOUR VALUE EVERY 20 ns
  when 183 =>
  amp_cs_var:='0';
  amp_spi_mosi_var:='0';
  AMPLIPHASE<='1';
  counter_amp:=counter_amp+1;
  --CUSTOMIZE YOUR VALUE EVERY 20 ns
  when 184 to 203 =>
  amp_cs_var:='0';
  amp_spi_mosi_var:=GAIN_CONFIG(0);
  AMPLIPHASE<='1';
  counter_amp:=counter_amp+1;
  --CUSTOMIZE YOUR VALUE EVERY 20 ns
  when 204 to 223 =>
  amp_cs_var:='0';


GAIN_CONFIG(1);

amp_spi_mosi_var:=
AMPLICPHASE<='1';
counter_amp:=counter_amp+1;

-CUSTOMIZE YOUR VALUE EVERY 20 ns

when 224 to 243 =>
amp_cs_var:='0';
amp_spi_mosi_var:=
GAIN_CONFIG(2);

AMPLICPHASE<='1';
counter_amp:=counter_amp+1;

-CUSTOMIZE YOUR VALUE EVERY 20 ns

when 244 to 263 =>
amp_cs_var:='0';
amp_spi_mosi_var:=
GAIN_CONFIG(3);

AMPLICPHASE<='1';
counter_amp:=counter_amp+1;

-CUSTOMIZE YOUR VALUE EVERY 20 ns

when 264 to 283 =>
amp_cs_var:='0';
amp_spi_mosi_var:=
GAIN_CONFIG(4);

AMPLICPHASE<='1';
counter_amp:=counter_amp+1;

-CUSTOMIZE YOUR VALUE EVERY 20 ns

when 284 to 303 =>
amp_cs_var:='0';
amp_spi_mosi_var:=
GAIN_CONFIG(5);

AMPLICPHASE<='1';
counter_amp:=counter_amp+1;

-CUSTOMIZE YOUR VALUE EVERY 20 ns

when 304 to 323 =>
amp_cs_var:='0';
amp_spi_mosi_var:=
GAIN_CONFIG(6);

AMPLICPHASE<='1';
counter_amp:=counter_amp+1;

-CUSTOMIZE YOUR VALUE EVERY 20 ns

when 324 to 332 =>
amp_cs_var:='0';
amp_spi_mosi_var:=
GAIN_CONFIG(7);

AMPLICPHASE<='1';
counter_amp:=counter_amp+1;
when 333 to 341 =>
    amp_cs_var:='1';
    amp_spi_mosi_var:='0';
    AMPLIPHASE<='1';
    counter_amp:=counter_amp+1;--CUSTOMIZE YOUR VALUE EVERY 20 ns

when 342 =>
    amp_cs_var:='0';
    amp_spi_mosi_var:='0';
    AMPLIPHASE<='0';
    ADPHASE<='1';
    counter_amp:=342;--CUSTOMIZE YOUR VALUE EVERY 20 ns

end case;

end if;--Amplifier communication ends

if ( ADPHASE ='1') THEN
    case counter_ad is
        when 0 to 3 =>
            ad_conv_var:='0';
            --ad_dout_var:='0';
            ADPHASE<='1';
            counter_ad := counter_ad+1;
        when 4 to 11 =>
            ad_conv_var:='1';
            --ad_dout_var:='0';
            ADPHASE<='1';
            counter_ad := counter_ad+1;
        when 12 to 63 =>
            ad_conv_var:='0';
            --ad_dout_var:='0';
            ADPHASE<='1';
            counter_ad := counter_ad+1;
        when 64 =>
            ad_conv_var:='0';
            --ad_dout_var:='0';
            canalA(13):=AD_DOUT;--1
            ADPHASE<='1';
            counter_ad := counter_ad+1;
        when 65 to 83 =>
            ad_conv_var:='0';
            --ad_dout_var:='0';
            senal_captura:='0';

        end case;

end if;
ADPHASE<='1';
counter_ad := counter_ad+1;

when 84 =>
   ad_conv_var:='0';
   --ad_dout_var:='0';
canalA(12):=AD_DOUT;--2
   ADPHASE<='1';
counter_ad := counter_ad+1;

when 85 to 103 =>
   ad_conv_var:='0';
   --ad_dout_var:='0';
   senal_captura:='0';
   ADPHASE<='1';
counter_ad := counter_ad+1;

when 104 =>
   ad_conv_var:='0';
   --ad_dout_var:='0';
canalA(11):=AD_DOUT;--3
   ADPHASE<='1';
counter_ad := counter_ad+1;

when 105 to 123 =>
   ad_conv_var:='0';
   --ad_dout_var:='0';
   senal_captura:='0';
   ADPHASE<='1';
counter_ad := counter_ad+1;

when 124 =>
   ad_conv_var:='0';
   --ad_dout_var:='0';
canalA(10):=AD_DOUT;--4
   ADPHASE<='1';
counter_ad := counter_ad+1;

when 125 to 143 =>
   ad_conv_var:='0';
   --ad_dout_var:='0';
   senal_captura:='0';
   ADPHASE<='1';
counter_ad := counter_ad+1;

when 144 =>
   ad_conv_var:='0';
   --ad_dout_var:='0';
canalA(9):=AD_DOUT;--5
   ADPHASE<='1';
counter_ad := counter_ad+1;

when 145 to 163 =>
   ad_conv_var:='0';
   --ad_dout_var:='0';
   senal_captura:='0';
   ADPHASE<='1';
counter_ad := counter_ad+1;
when 164 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  canalA(8):=AD_DOUT;--6
  ADPHASE<='1';
  counter_ad := counter_ad+1;

when 165 to 183 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  senal_captura:='0';
  ADPHASE<='1';
  counter_ad := counter_ad+1;

when 184 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  canalA(7):=AD_DOUT;--7
  ADPHASE<='1';
  counter_ad := counter_ad+1;

when 185 to 203 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  senal_captura:='0';
  ADPHASE<='1';
  counter_ad := counter_ad+1;

when 204 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  canalA(6):=AD_DOUT;--8
  ADPHASE<='1';
  counter_ad := counter_ad+1;

when 205 to 223 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  senal_captura:='0';
  ADPHASE<='1';
  counter_ad := counter_ad+1;

when 224 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  canalA(5):=AD_DOUT;--9
  ADPHASE<='1';
  counter_ad := counter_ad+1;

when 225 to 243 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  senal_captura:='0';
  ADPHASE<='1';
  counter_ad := counter_ad+1;

when 244 =>

54
ad_conv_var:='0';
--ad_dout_var:='0';
canalA(4):=AD_DOUT;--10
ADPHASE<='1';
counter_ad := counter_ad+1;

when 245 to 263=>
ad_conv_var:='0';
--ad_dout_var:='0';
senal_captura:='0';
ADPHASE<='1';
counter_ad := counter_ad+1;

when 264 =>
ad_conv_var:='0';
--ad_dout_var:='0';
canalA(3):=AD_DOUT;--11
ADPHASE<='1';
counter_ad := counter_ad+1;

when 265 to 283 =>
ad_conv_var:='0';
--ad_dout_var:='0';
senal_captura:='0';
ADPHASE<='1';
counter_ad := counter_ad+1;

when 284 =>
ad_conv_var:='0';
--ad_dout_var:='0';
canalA(2):=AD_DOUT;--12
ADPHASE<='1';
counter_ad := counter_ad+1;

when 285 to 303 =>
ad_conv_var:='0';
--ad_dout_var:='0';
senal_captura:='0';
ADPHASE<='1';
counter_ad := counter_ad+1;

when 304 =>
ad_conv_var:='0';
--ad_dout_var:='0';
canalA(1):=AD_DOUT;--13
ADPHASE<='1';
counter_ad := counter_ad+1;

when 305 to 323 =>
ad_conv_var:='0';
--ad_dout_var:='0';
senal_captura:='0';
ADPHASE<='1';
counter_ad := counter_ad+1;

when 324 =>
ad_conv_var:='0';
--ad_dout_var:='0';
  canalA(0):=AD_DOUT;--14
  ADPHASE<='1';
  counter_ad := counter_ad+1;

when 325 to 383 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  senal_captura:='0';
  ADPHASE<='1';
  counter_ad := counter_ad+1;

when 384 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  canalB(13):=AD_DOUT;--1
  ADPHASE<='1';
  counter_ad := counter_ad+1;

when 385 to 403 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  senal_captura:='0';
  ADPHASE<='1';
  counter_ad := counter_ad+1;

when 404 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  canalB(12):=AD_DOUT;--2
  ADPHASE<='1';
  counter_ad := counter_ad+1;

when 405 to 423 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  senal_captura:='0';
  ADPHASE<='1';
  counter_ad := counter_ad+1;

when 424 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  canalB(11):=AD_DOUT;--3
  ADPHASE<='1';
  counter_ad := counter_ad+1;

when 425 to 443 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  senal_captura:='0';
  ADPHASE<='1';
  counter_ad := counter_ad+1;

when 444 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  canalB(10):=AD_DOUT;--4
ADPHASE<='1';
counter_ad := counter_ad+1;

when 445 to 463 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  senal_captura:='0';
  ADPHASE<='1';
counter_ad := counter_ad+1;

when 464 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  canalB(9):=AD_DOUT;--5
  ADPHASE<='1';
counter_ad := counter_ad+1;

when 465 to 483 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  senal_captura:='0';
  ADPHASE<='1';
counter_ad := counter_ad+1;

when 484 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  canalB(8):=AD_DOUT;--6
  ADPHASE<='1';
counter_ad := counter_ad+1;

when 485 to 503 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  senal_captura:='0';
  ADPHASE<='1';
counter_ad := counter_ad+1;

when 504 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  canalB(7):=AD_DOUT;--7
  ADPHASE<='1';
counter_ad := counter_ad+1;

when 505 to 523 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  senal_captura:='0';
  ADPHASE<='1';
counter_ad := counter_ad+1;

when 524 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  canalB(6):=AD_DOUT;--8
  ADPHASE<='1';
counter_ad := counter_ad+1;
when 525 to 543 =>
  ad_conv_var:='0';
--ad_dout_var:='0';
senal_captura:='0';
ADPHASE<='1';
counter_ad := counter_ad + 1;
when 544 =>
  ad_conv_var:='0';
--ad_dout_var:='0';
  canalB(5):=AD_DOUT;--9
ADPHASE<='1';
counter_ad := counter_ad + 1;
when 545 to 563 =>
  ad_conv_var:='0';
--ad_dout_var:='0';
senal_captura:='0';
ADPHASE<='1';
counter_ad := counter_ad + 1;
when 564 =>
  ad_conv_var:='0';
--ad_dout_var:='0';
  canalB(4):=AD_DOUT;--10
ADPHASE<='1';
counter_ad := counter_ad + 1;
when 565 to 583 =>
  ad_conv_var:='0';
--ad_dout_var:='0';
senal_captura:='0';
ADPHASE<='1';
counter_ad := counter_ad + 1;
when 584 =>
  ad_conv_var:='0';
--ad_dout_var:='0';
  canalB(3):=AD_DOUT;--11
ADPHASE<='1';
counter_ad := counter_ad + 1;
when 585 to 603 =>
  ad_conv_var:='0';
--ad_dout_var:='0';
senal_captura:='0';
ADPHASE<='1';
counter_ad := counter_ad + 1;
when 604 =>
  ad_conv_var:='0';
--ad_dout_var:='0';
  canalB(2):=AD_DOUT;--12
ADPHASE<='1';
counter_ad := counter_ad + 1;
when 605 to 623 =>
ad_conv_var:='0';
--ad_dout_var:='0';
senal_captura:='0';
ADPHASE<='1';
counter_ad := counter_ad+1;

when 624 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
canalB(1):=AD_DOUT;--13
ADPHASE<='1';
counter_ad := counter_ad+1;

when 625 to 643 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
senal_captura:='0';
ADPHASE<='1';
counter_ad := counter_ad+1;

when 644 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
  senal_captura:='1';
  canalB(0):=AD_DOUT;--14
ADPHASE<='1';
counter_ad := counter_ad+1;

when 645 to 667 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
senal_captura:='0';
ADPHASE<='1';
counter_ad := counter_ad+1;

when 668 =>
  ad_conv_var:='0';
  --ad_dout_var:='0';
senal_captura:='0';
  CARRIER:='1';
  COMPHASE:='1';
  counter_ad := 0;
  ADPHASE<='0';

end case;
end if;--conversion finish

if (comphase = '1') then
  --
  if portmod0 = '0' and portmod1 = '0' then
    --
    if canalA >
      porta_shareb then
      pwm1:='1';
      elsif canalA < porta_shareb then

pwm1:='0';
end if;

elsif portmod0 = '1' and portmod1 = '0'
then

if canalA > "00000000000000"
then

if canalA > portap then
    pwm1:='1';
    pwm2:='0';
elsif canalA < portap then
    pwm1:='0';
    pwm2:='0';
end if;
end if;

if canalA < "00000000000000"
then

if canalA > portan then
    pwm2:='1';
    pwm1:='0';
elsif canalA < portan then
    pwm2:='0';
    pwm1:='0';
end if;
end if;---POLARITY SWITCH

PROCESS ENDS

elsif portmod0 = '0' and portmod1 = '1'
then

if canalA > "00000000000000"
then

if canalA > portap then
    pwm1:='1';
    pwm2:='0';
elsif canalA < portap then
    pwm1:='0';
    pwm2:='0';
end if;
end if;

if canalA < "00000000000000"
then

if canalA > portan then
    pwm2:='1';
    pwm1:='0';
elsif canalA < portan then
    pwm2:='0';
    pwm1:='0';
end if;
end if;---POLARITY SWITCH

PROCESS ENDS

elsif portmod0 = '1' and portmod1 = '1'
then

if canalA > "00000000000000"
then

if canalA > portap then
    pwm1:='1';
    pwm2:='0';
elsif canalA < portap then
    pwm1:='0';
    pwm2:='0';
end if;
end if;

if canalA < "00000000000000"
then

if canalA > portan then
    pwm2:='1';
    pwm1:='0';
elsif canalA < portan then
    pwm2:='0';
    pwm1:='0';
end if;
end if;---POLARITY SWITCH

PROCESS ENDS
if canalA > porta_shareb then
  pwm1:= '1';
elsif canalA < porta_shareb then
  pwm1:= '0';
end if;

end if; --carrier select finish

end if; --FINALIZA COMPARACION

pwm1source<=pwm1;
pwm2source<=pwm2;

if (CARRIER = '1') then

  if portmod0 = '0' and portmod1 = '0' then
    if porta <= -8192 then
      flag:= '1';

elseif porta >= 8191 then
      flag:= '0';

  avisoport:= '1';

  end if;

end if;

if flag = '1' then
  porta:=porta +1;
elsif flag = '0' then
  porta:=porta -1;

end if;

porta_shareb:= porta;

if porta = -8192 and avisoport = '1' then

  avisoport:= '0';
ADPHASE<= '1';
CARRIER:= '0';

end if;

end if;
if portmod0 = '1' and portmod1 = '0' then
  if portapos <= 0 then
    flag:='0';
  elseif portapos >= 8191 then
    flag:='1';
    avisoport:='1';
    end if;
  '1' then
    portapos:=portapos -1;
    portaneg:=portaneg -1;
  end if;

  if flag = '1' then
    portapos:=portapos +1;
    portaneg:=portaneg +1;
  else
    portapos;
    portaneg;
  end if;

  if flag = '1' and portapos = 0 and avisoport = '1' then
    avisoport:='0';
    ADPHASE<='1';
    comphase:='0';
    CARRIER:='0';
    flag := '0';
  end if;

end if;

if portmod0 = '0' and portmod1 = '1' then
if portaposi <= 0 then
  flag:='0';
elsif portaposi >= 8191 then
  flag:='1';
avisoport:='1';
end if;
if flag = '1' then
  portaposi:=portaposi -1;
  portanega:=portanega +1;
elsif flag = '0' then
  portaposi:=portaposi +1;
  portanega:=portanega -1;
end if;
portap:= portaposi;
portan:= portanega;
if flag = '1'
  and portaposi = 0 and avisoport = '1' then
    comphase:='0';
    CARRIER:='0';
    avisoport:='0';
    ADPHASE<='1';
    flag := '0';
  end if;
end if;
end if;
if portmod0 = '1' and portmod1 = '1' then
  if portasierra >= 8191 then
    avisoport:='1';
  end if;
  if portasierra >= 8191 then
    portasierra:=portasierra +1;
  end if;
  portasierra:=portasierra +1;
porta_shareb:= portasierra;
  if avisoport = '1' then
    portasierra:="10000000000000";
  end if;
end if;
comphase:='0';
CARRIER:='0';
avisoport:='0';
ADPHASE<='1';

end if;
end if;

end if;

end if;

amp_cs<= amp_cs_var;
spi_mosi<= amp_spi_mosi_var;
ad_conv <= ad_conv_var;
amp_shdn<= '0';
flagout<=flag;

if canalsel='1' then

if inter='1' then

leds (7)<= '0';
leds (6)<= '0';
leds (5)<= canalA(13);
leds (4)<= canalA(12);
leds (3)<= canalA(11);
leds (2)<= canalA(10);
leds (1)<= canalA(9);
leds (0)<= canalA(8);

elsif inter = '0' then

leds (7)<= canalA(7);
leds (6)<= canalA(6);
leds (5)<= canalA(5);
leds (4)<= canalA(4);
leds (3)<= canalA(3);
leds (2)<= canalA(2);
leds (1)<= canalA(1);
leds (0)<= canalA(0);

end if;
elsif canalsel ='0' then

if inter='1' then

leds (7)<= '0';
leds (6)<= '0';
leds (5)<= canalB(13);
leds (4)<= canalB(12);
leds (3)<= canalB(11);
leds (2)<= canalB(10);
leds (1)<= canalB(9);
leds (0)<= canalB(8);

end if;

else
elsif inter = '0' then

  leds (7)<= canalB(7);
  leds (6)<= canalB(6);
  leds (5)<= canalB(5);
  leds (4)<= canalB(4);
  leds (3)<= canalB(3);
  leds (2)<= canalB(2);
  leds (1)<= canalB(1);
  leds (0)<= canalB(0);

  end if;

end if;

end process;

end Behavioral;
INFORMATION


BIBLIOGRAPHY

TITLE: VHDL, LENGUAJE ESTANDAR DE DISEÑO ELECTRONICO

EDITORIAL: MCGRAW-HILL

YEAR: 1997

WEBGRAPHHY

http://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm
http://www.xilinx.com/fpga/index.htm
http://embeddedmicro.com/tutorials/mojo/what-is-an-fpga/
https://en.wikipedia.org/wiki/Field-programmable_gate_array