



**GDAŃSK UNIVERSITY  
OF TECHNOLOGY**

FACULTY OF ELECTRONICS, TELECOMMUNICATIONS  
AND INFORMATICS



Student's name and surname: Gor Sahakyan  
ID: 164333  
First cycle studies  
Mode of study: Full-time studies  
Field of study: Electronics and  
Telecommunications  
Specialization: -

## **BACHELOR'S THESIS**

Title of thesis: hardware block for microelectronic system with PLB bus for controlling the audio amplifier

Title of thesis (in Polish): Blok sprzętowy systemu mikroelektronicznego z interfejsem magistrali PLB sterujący wzmacniaczem akustycznym

Supervisor	Head of Department
<i>signature</i>	<i>signature</i>
dr inż. Marek Wójcikowski	

Date of thesis submission to faculty office:



## STATEMENT

First name and surname: Gor Sahakyan  
Date and place of birth: 22.04.1992, Erevan-Arm  
ID: 164333  
Faculty: Faculty of Electronics, Telecommunications and Informatics  
Field of study: electronics and telecommunications  
Cycle of studies: undergraduate studies  
Mode of studies: Full-time studies

I, the undersigned, agree/do not agree\* that my diploma project entitled: hardware block for microelectronic system with PLB bus for controlling the audio amplifier may be used for scientific or didactic purposes.<sup>1</sup>

Gdańsk, .....

.....  
*signature of the student*

Aware of criminal liability for violations of the Act of 4<sup>th</sup> February 1994 on Copyright and Related Rights (Journal of Laws 2006, No. 90, item 631) and disciplinary actions set out in the Law on Higher Education (Journal of Laws 2012, item 572 with later amendments),<sup>2</sup> as well as civil liability, I declare that the submitted diploma project is my own work.

This diploma project has never before been the basis of an official procedure associated with the awarding of a professional title.

All the information contained in the above diploma project which is derived from written and electronic sources is documented in a list of relevant literature in accordance with art. 34 of the Copyright and Related Rights Act.

I confirm that this diploma project is identical to the attached electronic version.

Gdańsk, .....

.....  
*signature of the student*

I authorise the Gdańsk University of Technology to include an electronic version of the above diploma project in the open, institutional, digital repository of the Gdańsk University of Technology and for it to be submitted to the processes of verification and protection against misappropriation of authorship.

Gdańsk, .....

.....  
*signature of the student*

\*) delete where appropriate

---

<sup>1</sup> Decree of Rector of Gdańsk University of Technology No. 34/2009 of 9<sup>th</sup> November 2009, TUG archive instruction addendum No. 8.

<sup>2</sup> Act of 27<sup>th</sup> July 2005, Law on Higher Education:

Art. 214, section 4. Should a student be suspected of committing an act which involves the appropriation of the authorship of a major part or other elements of another person's work, the rector shall forthwith order an enquiry.

Art. 214 section 6. If the evidence collected during an enquiry confirms that the act referred to in section 4 has been committed, the rector shall suspend the procedure for the awarding of a professional title pending a judgement of the disciplinary committee and submit formal notice of the committed offence.

## INDEX

1.	INTRODUCTION .....	3
1.1	PROJECT AIMS AND OBJECTIVES .....	3
1.2	PREVIOUS NEEDED KNOWLEDGE .....	3
1.3	APPLICATION .....	3
2.	DESIGN ENVIRONMENT .....	3
2.1	XILINX EDK .....	3
3.	DEVICES .....	4
3.1	PMODAMP1 SPEAKER/HEADPHONE AMPLIFIER .....	4
3.2	PLB BUS .....	5
3.3	SPARTAN 3 STARTER BOARD.....	6
4.	CUSTOM HARDWARE/SOFTWARE BLOCK DESIGN .....	6
4.1	START THE BASE MICROELECTRONIC SYSTEM .....	6
4.2	CUSTOM IP .....	8
4.3	TRANSLATION OF MUSIC TO ELECTRONICS .....	11
4.4	CUSTOM SOFTWARE .....	15
4.5	SOFTWARE TO HARDWARE COMMUNICATION.....	17
4.6	OBTAINED SPECIFICATIONS.....	18
5.	CONCLUSIONS .....	19
6.	REFERENCES .....	20
	LIST OF FIGURES.....	21
	LIST OF TABLES.....	22
	APPENDIX A: USER GUIDE.....	23
	APPENDIX B: CD.....	26

## **ABSTRACT**

This paper covers the developing of a block of Intellectual Property (IP) module that supports PmodAMP1 Digilent audio amplifier. The project has been done in Xilinx Platform Studio. The designed block is equipped with the Processor Local Bus (PLB) on his 4.6 version and connected to a programmable microelectronic system with a MicroBlaze processor. A sample application has been made to demonstrate the proper operation of the designed block.

The programming languages used during the realization of this project were Verilog, VHDL and C/C++. The project has been build on the Spartan-3 Starter Board Field Programmable Gate Array (FPGA).

To demonstrate the operation of this block, a C code software has been written to communicate with hardware block and output a song through the audio amplifier.

## 1. INTRODUCTION

### 1.1 *Project aims and objectives*

The following document gathers everything necessary to create a hardware block for controlling the audio amplifier. This hardware block is based on audio amplifier PmodAMP1 connected to Spartan 3 Starter Board Field Programmable Gate Array (FPGA), so through Xilinx Project Studio (XPS), hardware specifications will be built on the embedded system, followed by the necessary code, written in C, to run a custom application. Knowledge of Verilog, VHDL and C/C++ languages is necessary for the proposed aim. The first objective is to create a microelectronic system with a MicroBlaze processor connected to the Processor Local Bus (PLB) in order to establish communication between the processor and the devices.

The second objective of this project, once the hardware block is designed, consists of creating a specific application to demonstrate the correct operation of the designed block.

### 1.2 *Previous needed knowledge*

This project has been done after finishing Microelectronic Programmable Systems course and learning Xilinx Embedded Development Kit (EDK) system, C/C++ and Verilog languages are used. The MicroBlaze processor has been created through the Base System Builder (BSB) wizard using VHDL language, but while creating a custom Intellectual Property (IP) block, an option to create user logic in Verilog is available. For proper understanding of this project, a knowledge of Verilog, C/C++ and Xilinx Platform Studio is needed.

### 1.3 *Application*

In order to demonstrate that the design operates properly, an application needs to be designed. Its purpose is to cover all the functions that the block is designed for and give the project a task. The basic operation of an audio amplifier is to output an audio signal, so the designed application will play a song through the audio amplifier. To perform this task, some little knowledge of music staves or pentagrams is needed and it will be explained in the following chapters.

## 2. DESIGN ENVIRONMENT

### 2.1 *Xilinx EDK*

Xilinx EDK System is the development package for building embedded processor systems in Xilinx FPGA's. This development kit is separated into two different environments: Software Development Kit (SDK) and XPS.

Xilinx Platform Studio provides the user of a hardware description environment to configure and build the hardware specification of the embedded system, this includes

the processor core, the memory-controller, the I/O peripherals and many other blocks, so it can convert the designer's platform specification into a synthesizable Register Transfer Level (RTL) description (VHDL or Verilog).

The Software Development Kit allows the user to write, compile and debug C/C++ application for their designed embedded system, in other words, it handles the software that will be executed on the embedded system.

Also, Xilinx ISE Design Software is used in order to test and simulate the designed hardware block.

### 3. DEVICES

#### 3.1 PmodAMP1 Speaker/Headphone amplifier

The selected audio amplifier is the PmodAMP1 module by Digilent. It works with low power signals and it makes possible driving both, mono and stereo headphones/speakers [1]. Its features are as following:

- 1/8 inch stereo headphone jack
- 1/8 inch mono speaker jack
- 6-pin header for inputs
- Voltage range of operation 3V-5V

Connectivity to the Spartan 3 board has to be done through a 6 pin cable and its correspondent header. Because of the use of the expansion connectors described in the following chapter 3.2, the header connected to the cable needs to get crossed its first two pins.

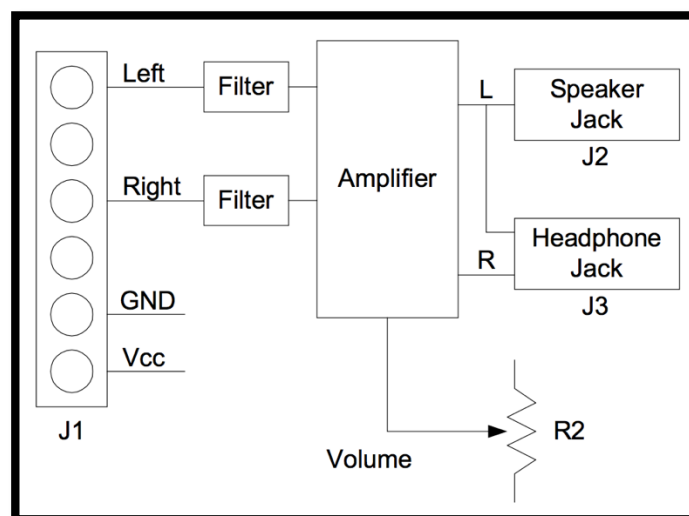


Fig.1. PmodAMP1 Block diagram [1]

As it is shown in Fig.2, the last two pins, need to be crossed by welding.

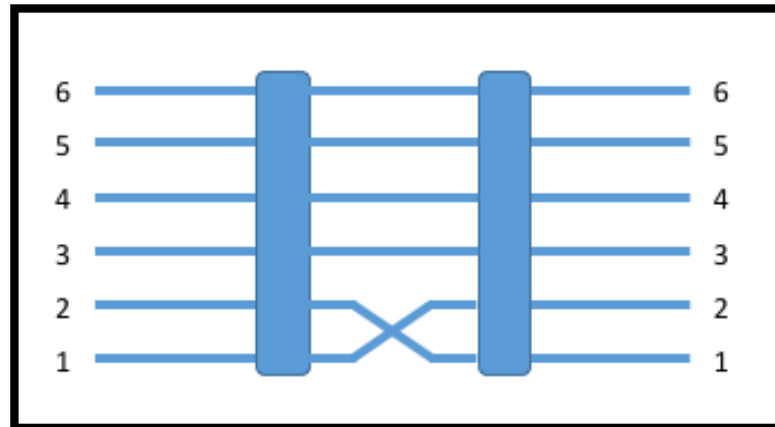


Fig.2. Schematic diagram

So the connection should be done like this:

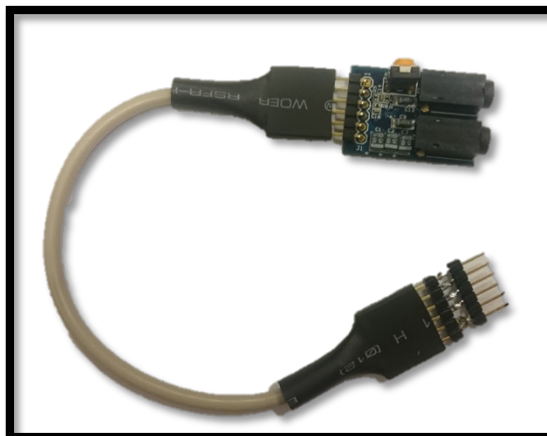


Fig.3. PmodAmp1 6-pin cable to crossed header

### 3.2 PLB Bus

CoreConnect is a microprocessor bus architecture from IBM System on a Chip (SoC) designs. Xilinx uses CoreConnect as the infrastructure for all their embedded designs. In this project, the Processor Local Bus PLB 4.6 is going to be used because it provides the infrastructure for connecting a different and optional number of Masters (16) and Slaves into a PLB system. *“It consists of a bus control unit, a watchdog timer, and separate address, write, and read data path units, as well as an optional DCR (Device Control Register) slave interface to provide access to its bus error status registers”* [9].

### 3.3 Spartan 3 Starter Board

In this project the Spartan 3 Starter board has been used. There are three 40-pin expansion connectors on this board: A1, A2 and B1, each one providing different features for some of the pins. Generally, pins for  $V_{cc}$  and GND are located in the first two pins of each connector. Most of the other pins are for general purpose but a few ones also provide extra features like additional logic on A1 connector or Master and Slave Parallel mode on A2 or B1 connectors [2]. This project does not need this features, so any of the three expansion connectors is valid, the A1 connector has been chosen for the presented project.

The table 1 describes the organization of the pins [2].

Table 1. Expansion connector A1 first 6 pins [2]

FPGA PIN	CONNECTOR		FPGA PIN
GND	1	2	N8
VCCO	3	4	L5 (SRAM)
N7	5	6	N3 (SRAM)
T8	7	8	M4 (SRAM)
R6	9	10	M3 (SRAM)
T5	11	12	L4 (SRAM)

Taking into the consideration this information, the PmodAMP1 needs to be connected to connector 1 to 6, so the pin for Left output (Speaker Jack) will be T5 and the pin for the Right output (Headphone Jack) will be T8.

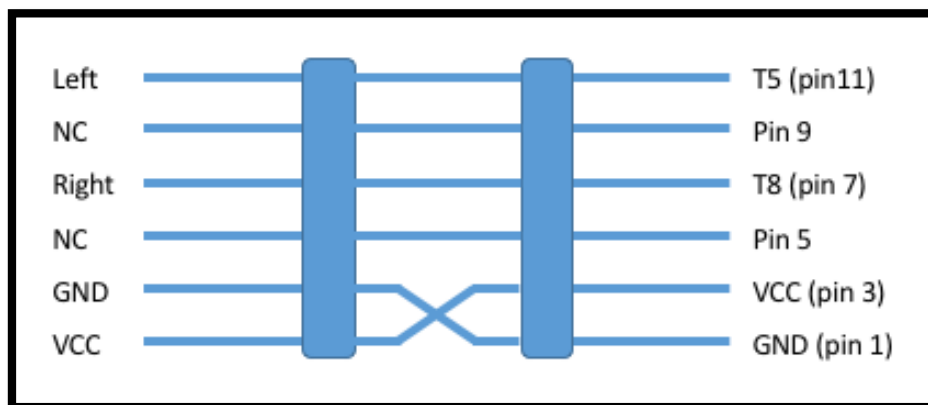


Fig.4. Amplifier to FPGA connections

## 4. CUSTOM HARDWARE/SOFTWARE BLOCK DESIGN

### 4.1 Start the base microelectronic system



First of all, it is needed to create the base system, by using the BSB wizard, with MicroBlaze processor with the following specifications:

- System clock frequency 50 MHz
- On-Chip Memory 16KB
- RS232, Push\_Buttons\_3bit and DIP\_Switches\_8bit IO devices selected.

Once the process is finished, the Block Diagram of the created system will be generated, as shown in the following figures Fig.5 and Fig.6:

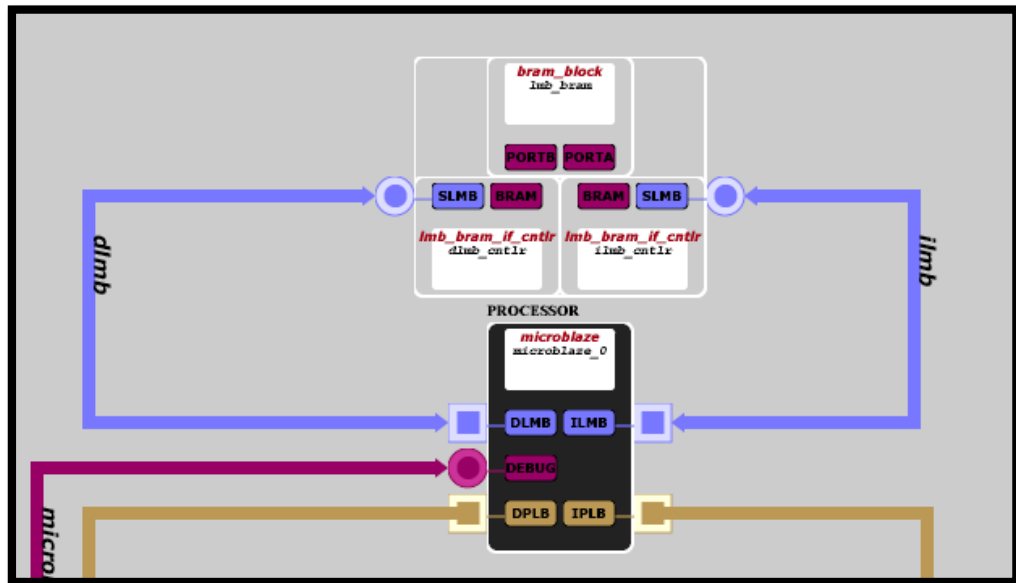


Fig.5. Block diagram I

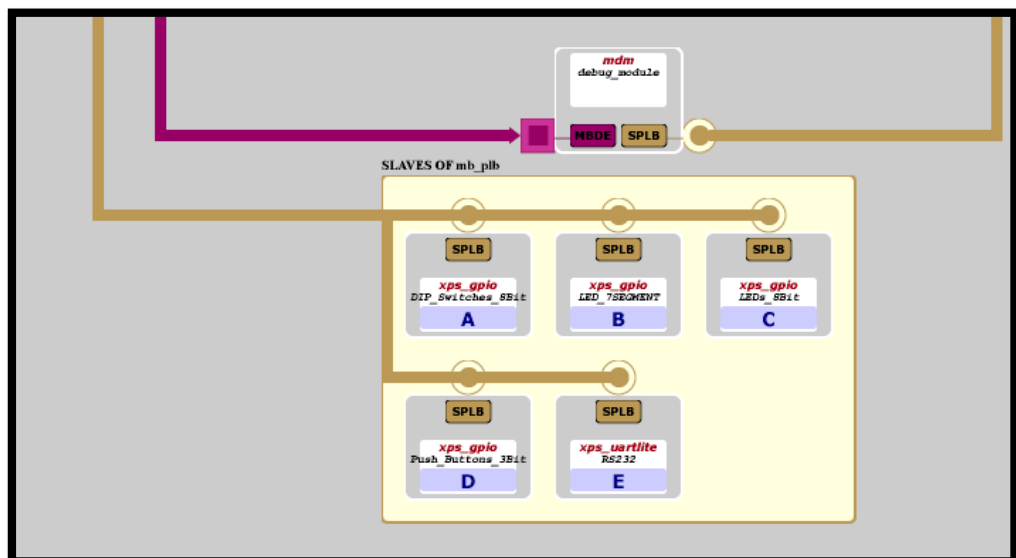


Fig.6. Block diagram II

Fig.5 contains the processor MicroBlaze and the BRAM, connections between them are made through Local Memory Bus. The connections between the processor and

the IO devices are done through Processor Local Bus as shown in Fig.6. The LMB is a fast, local bus for connecting MicroBlaze instruction and data ports to high-speed peripherals, primarily on-chip block RAM (BRAM). MicroBlaze uses this dedicated LMB bus in order to reduce the load on the other buses.

In System Assembly View, the same description but with hierarchical levels is shown, so the user can adapt the connections between each block to his own necessities.

While creating the project, the BSB wizard generates MHS (Microprocessor Hardware Specification) and MSS (Microprocessor Software Specification) files. The MHS defines the hardware component, in other words, defines the configuration of the embedded system, including bus architecture, peripherals, processor system connectivity and address space. On the other hand, the MSS file contains libraries, and directives for customizing Operating Systems and drivers [8].

## 4.2 Custom IP

Once the base system has been built, the next step is creating the custom Intellectual Property block, in other words, user's designed block. XPS allows to import or create new peripherals and add them to existing project. In this project, a custom IP block with 4 registers connected to the PLB bus has been created. While creating it, the program will give the choice of the desired programming language for the user logic, so Verilog has been chosen in this project.

This block is responsible to connect to the audio amplifier and output the desired tones that it will receive from the software, so basically Custom IP acts like a slave device to the processor.

The peripheral is based on two files, *custom\_ip.vhd*, and *user\_logic.v*. The first one defines the connectivity between the PLB interface and the peripheral IO ports and the second one contains the hardware specification. As it was mentioned before, four registers have been created so each of them has a dedicated task assigned. Data sent from the processor to the peripheral device travels through *ipif\_Bus2IP\_Data* and it is stored in the desired register, default code generated by XPS includes this operation, so there is no need to design a way to store data sent by the software to the registers.

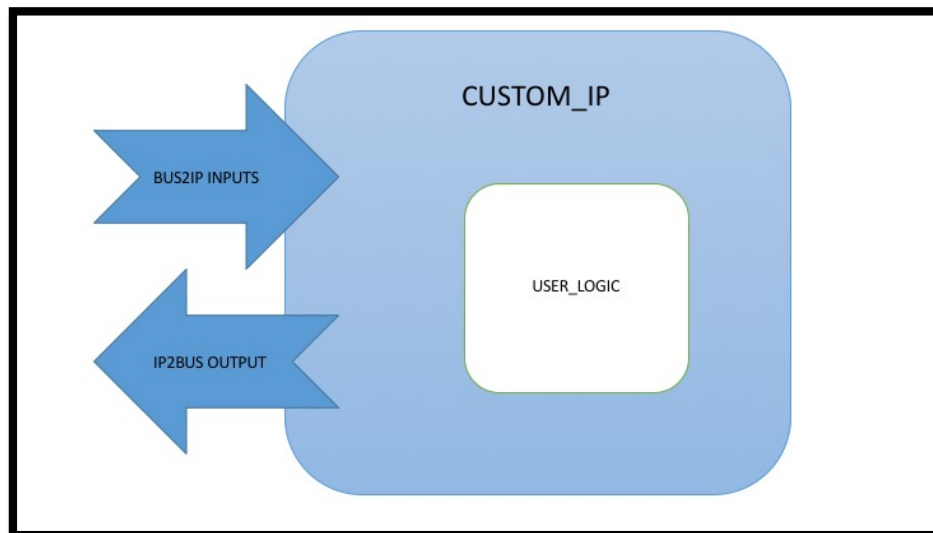


Fig.7. Custom IP diagram

The basic I/O interface transfers the information from software to hardware, but some additional ports need to be created in order to enable the hardware block to output the signal. In this project, two variables have been called *left*, as the left channel output, and *enable*, as the switch that enables Custom IP to start working. Both are external connections to their respective pins.

Information send to hardware is stored in different registers.

Fig.8 shows code block that works to create the note, it is just a simple clock divider. The first register of Custom IP is *slv\_reg0*, the software will store in this register the counter value to create a digital signal with specific frequency, so this is the way to create all the different musical notes. Verilog does not understand the frequency, but clock cycles, so the different musical notes have to be traduced from frequencies to time in seconds and later to clock cycles. This will be explained in detail in section 4.3.

```

always @(posedge Bus2IP_Clk)
begin
    if (Bus2IP_Reset)
        Q <= 17'b0;
    else if (Q == (slv_reg0-1))
        Q <= 17'b0;
    else
        Q <= Q + 1;
end

always @(posedge Bus2IP_Clk)
begin
    if (Bus2IP_Reset)
        left_aux <= 0;
    else if (Q == (slv_reg0-1))
        left_aux <= ~left_aux;
    else
        left_aux <= left_aux;
end

```

*Fig.8. Clock divider Verilog code*

The part of the code presented in Fig.9 is responsible for capturing the created signal and outputting it, checking previously, if it has to be sent to the output (*enable* and *start*).

```

always @(posedge Bus2IP_Clk)
begin
    case(slv_reg3 && enable)
    1:
        begin
            if (delay < slv_reg1)
                begin
                    delay <= delay + 1;
                    left <= left_aux;
                    slv_reg2 <= 1;
                end
            else
                slv_reg2 <= 0;
        end
    0:
        delay <= 0;
    endcase
end

```

*Fig.9. Part of User Logic Verilog code*

Registers *slv\_reg1* and *slv\_reg3* store *delay* and *start* values. Register three *slv\_reg2* is a special one, its default code has been edited to allow hardware to write to it, but not software. So every time that Custom IP finishes outputting a note it writes a logical one to *slv\_reg2*, so software can know, by reading it, the exact moment when it has to send the next note to play.

### 4.3 Translation of music to electronics

In this section explanations of how to take a music stave, analyze and traduce it to electronics language will be done.

To output a tone, a clock divider will be used, so depending on its frequency the output signal can produce different tones. For example, the frequency of the musical note *La* is 440 Hz in the 4th octave [3], considering that the systems clock frequency is 50 MHz:

$$\text{counter} = \frac{\text{processor clocks frequency}}{\text{tone frequency}}$$

counter for *La* is 113636, this means that the digital pulse width is 113636 processor clock cycles.

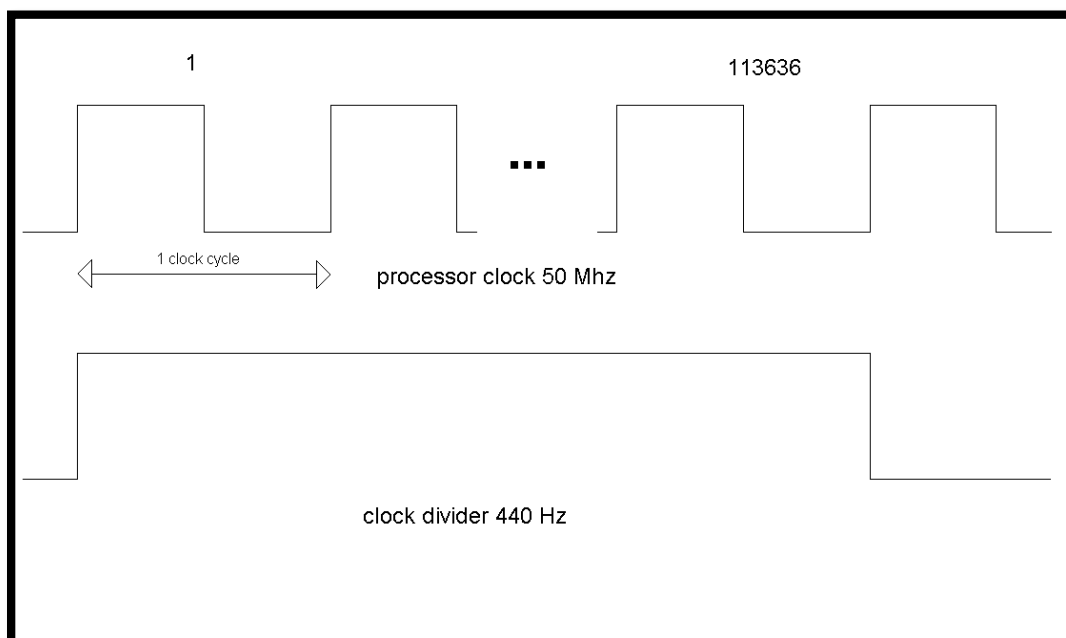


Fig.10. Processor clock VS output signal clock for La note

Table 2 contains the different musical tones used in this project [3]:

*Table 2. Frequency and counter values for musical notes [3]*

Musical Tone	Frequency (Hz)	Counter value
<b>Do octave 4</b>	261,625565	191113
<b>Re4</b>	293,664768	170262
<b>Mi4</b>	329,627557	151686
<b>Fa4</b>	349,228231	143173
<b>Sol4</b>	391,995436	127552
<b>La4</b>	440,000000	113636
<b>Si 4</b>	493,883301	101238
<b>Do octave 5</b>	523,251131	95556
<b>Re5</b>	587,329536	85131
<b>Mi5</b>	659,255114	75843
<b>Fa5</b>	698,456463	71586
<b>Sol5</b>	783,990872	63776
<b>La5</b>	880,000000	56818

Considering this information, Verilog code is just a counter that creates a signal and inverts its value every counter value cycles (clock divider), depending on which tone is outputted.









But for playing a song this is not enough, the program should also know how long it has to output this signal thus, a delay and an enable signal will be needed. In music staves, there are many different durations of the notes, but they are represented in tempos and they have to be traduced to seconds, firstly, and later to clock cycles. The following table represents the different tempos and how much half notes they contain per each minute:

*Table 3. Tempos [4]*

Half notes per minute	Italian expression
<b>40-43</b>	Grave
<b>44-47</b>	Largo
<b>48-51</b>	Larghetto
<b>52-54</b>	Adagio
<b>55-65</b>	Andante
<b>66-69</b>	Andantino
<b>70-95</b>	Moderato
<b>96-112</b>	Allegretto
<b>113-120</b>	Allegro
<b>121-140</b>	Vivace
<b>141-175</b>	Presto
<b>176-208</b>	Prestissimo

If tempo Allegro is chosen, then each half note duration is  $\frac{113}{60} = 0,53$  seconds [4]. The processor clock is 50MHz this means that there are  $50 * 10^6$  clock cycles in 1 second, so the duration in tempo and in clock cycles have the values presented in Table 4.

Table 4. Musical note values [5]

Note	Delay	Name	Duration (Tempo)	Duration (clock cycles)
		whole note	1	53000000
		half note	1/2	26500000
	 or ۲	quarter note	1/4	13250000
		eighth note	1/8	6625000

So this information is enough to traduce a stave to electronic language.

In this project Lost Woods songs has been chosen to play, its stave is presented in Fig.11.

**Lost Woods**  
From The Legend of Zelda: Ocarina of Time

<http://www.gamemusicthemes.com/> Koji Kondo  
Transcribed by BLUESCD

**Allegro**

Piano *mf*

5

9

13

16

*Repeat Forever*

Fig.11. Lost Woods Piano Stave [3]



From the point of view of the C code, tempos are traduced for whole notes as “r”, half notes as “b” and quarter notes as “n”. The translation of the different notes is shown in Table 5.

Table 5. Musical notes in C code

Note	Name in C file
<b>Octave 4</b>	
Do	Z
Re	X
Mi	C
Fa	F
Sol	G
La	H
Si	J
<b>Octave 5</b>	
Do	K
Re	Y
Mi	U
Fa	I
Sol	O
La	P
Si	0

#### 4.4 Custom Software

Once the hardware block is created, software can be written as C/C++ code. Custom IP includes the generated default functions to write and read [8], they are written in a C file located in:

“project directory”/microblaze\_0/libsrc/custom\_ip/custom\_ip\_v1\_00\_a.

These two functions are:

→void CUSTOM\_IP\_mWriteSlaveReg“number of register 0,1,2...”(Xuint32 BaseAddress, unsigned RegOffset, Xuint32 Value).

→Xuint32 CUSTOM\_IP\_mReadSlaveReg“number of register 0,1,2...”(Xuint32 BaseAddress, unsigned RegOffset).

*Xuint32 BaseAddress*, from mentioned functions is the base address of Custom IP block, there are two options: the use of numeric address or to use of the macro that is representing it. This can be checked in *xparameters.h* library file located in *microblaze\_0/includes directory*. Depending on the situation base address could change so it’s recommended to use the macro. *Unsigned RegOffset* is the slave register offset, in this case, it’s 0, and finally, *Xuint32Value*, for write function, is the desired value to be written to the register.

Musical notes and tempos are traduced, as shown in Table 5 and stored in two different variables. The program will take the information of those variables and translate them to the values of counter and delay. For example, if the first musical note of the stave is “Octave 4 La”, it should be stored as “H”, in this way the program can know through a switch statement which counter value corresponds to this musical note. The same happens with the delay.

```
char note[]={
    'F','H','J','F','H','J','F','H', ...
};
char tempo[]={
    'n','n','b','n','n','b','n','n', ...
};
```

Fig.12. Stored notes

The code is not independent of the hardware; this means that checking the status of played notes is necessary in order to know when to send the following ones. As mentioned before, this information will be obtained from *slv\_reg2*, and checking has to be done through a loop, so the software will wait until the hardware finishes.

```
for (i=0; i<N; i++){
    play (note[i],tempo[i],1);
    while (readReg2() == 1){}
    next ();
}
```

Fig.13. Main loop

The loop in Fig.13 will run until there are no more notes to be played, so variable “N” contains the total number of notes. Counter values are known by a “switch statement” as shown in Fig.14.

```
switch (note){
    case 'Z': counter=191113; /*OCATAVA 4*/
              break;      /*DO*/
    case 'X': counter=170262; /*RE*/
              break;
    case 'C': counter=151686; /*MI*/
              break;
              .
              .
              .
    default : counter=0; }
```

Fig.14. Selection of the counter values depending on the selected note

#### 4.5 Software to Hardware communication

Read and write functions described in *custom\_ip.h* (the header file that contains high-level functions for Custom IP) are based on other functions that the system has included: *xbasic\_types.h*, *xstatus.h* and *xio.h*.

Device drivers are computer programs working with operating systems or applications and hardware devices, so the software does not communicate directly with the hardware but it invokes a routine in the driver and this one communicates with the hardware. The main purpose of a driver is enabling access from software to hardware without knowing details of it. Communication with driver is usually made through the bus it is connected to, the PLB in this case. Drivers are also capable of invoking routines in the software if they get this request from software [10].

Read and write functions are high-level functions, and when they are called, they invoke other low-level functions. A “.mdd” file is a microprocessor device description file, every device is required to have such file. An example of “.mdd” file is shown in Fig.14.

```
OPTION psf_version = 2.1.0;

BEGIN DRIVER custom_ip

    OPTION supported_peripherals = (custom_ip);
    OPTION depends = (common_v1_00_a);
    OPTION copyfiles = all;

END DRIVER
```

Fig.15. Custom IP MDD

Option “supported\_peripherals” indicates which devices are supported by this driver, obviously, this one is Custom IP. “Option depends” specifies that the driver depends on the sources of a directory named “common\_v1\_00\_a”, this directory contains all the needed low-level functions that high-level ones will invoke to read from or write to the registers.

#### 4.6 Obtained Specifications

The Tables 6 and 7 contain the obtained specifications of this project.

Table 6. Logic Utilization

Logic Utilization		
<b>Total Number Slice Registers</b>	1,600 out of 3,840	(41%)
<b>Number used as Flip Flops:</b>	1,599	
<b>Number used as Latches:</b>	1	
<b>Number of 4 input LUTs:</b>	2,107 out of 3,840	(54%)

Logic Utilization describes the resources that the Synthesis tool uses to build the combinatorial functions. These resources (Flip-Flops (FF), Latches, Look-Up Tables(LUTs)) will get placed in slices, so there are multiple resources in a slice. Total number of slices indicates how much slices have at least one element used in them. Synthesis also estimates, how the design will be packed and placed in the target device.

Table 7. Logic Distribution

Logic Distribution		
<b>Number of occupied Slices:</b>	1,381 out of 1,920	(71%)
<b>Number of Slices containing only related logic</b>	1,381 out of 1,381	(100%)
<b>Number of Slices containing unrelated logic</b>	0 out of 1,381	(8%)
<b>Total Number of 4 input LUTs</b>	2,242 out of 3,840	(58%)
<b>Number used as logic</b>	1,697	
<b>Number used as a route-thru</b>	135	
<b>Number used for Dual Port RAMs (Two LUTs used per Dual Port RAM)</b>	256	
<b>Number used as Shift registers</b>	154	

## 5. CONCLUSIONS

The final implicit aim of the study is to learn how to create and customize hardware blocks, how to communicate with them, understand how the drivers work, etc. The difficult task is understanding the default files provided from the beginning by XPS and using this information for creating the required new blocks.

Future users can try to achieve also the keyboard to play notes through the hardware; this can be done by two different ways, firstly, using UartLite and Hyperterminal, so written characters will be translated, by software, to notes and outputted by hardware, or secondly, using the PS/2 port from Spartan 3 Board to connect keyboard and program a new hardware block to communicate with it and interpret the sent information.

## 6. REFERENCES

- [1]. PmodAMP1 Speaker/Headphone Amplifier reference manual :  
<http://store.digilentinc.com/pmodamp1-speaker-headphone-amplifier-retired/>.
- [2]. Spartan-3 Starter Kit Board user guide:  
[http://www.xilinx.com/support/documentation/boards\\_and\\_kits/ug130.pdf](http://www.xilinx.com/support/documentation/boards_and_kits/ug130.pdf).
- [3]. Frequencies of musical notes:  
<http://latecladeescape.com/h/2015/08/frecuencia-de-las-notas-musicales>.
- [4]. Wikipedia: <https://es.wikipedia.org/wiki/Tempo>.
- [5]. Wikipedia: [https://en.wikipedia.org/wiki/Note\\_value](https://en.wikipedia.org/wiki/Note_value).
- [6]. J.Ganssle, The Art of Designing Embedded Systems, Elsevier, 2008.
- [7]. J. Catsoulis, Designing Embedded Hardware, O'Reilly Media, 2005.
- [8]. Xilinx Community Forums: <https://forums.xilinx.com/>
- [9]. Xilinx Processor Local Bus product specification:  
[http://www.xilinx.com/support/documentation/ip\\_documentation/plb\\_v46.pdf](http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf).
- [10]. Wikipedia: [https://en.wikipedia.org/wiki/Device\\_driver](https://en.wikipedia.org/wiki/Device_driver).

## LIST OF FIGURES

<b>Fig.1.</b> PmodAMP1 Block diagram .....	3
<b>Fig.2.</b> Schematic diagram.....	4
<b>Fig.3.</b> PmodAmp1 6-pin cable to crossed header.....	4
<b>Fig.4.</b> Amplifier to FPGA connections.....	5
<b>Fig.5.</b> Block diagram I.....	6
<b>Fig.6.</b> Block diagram II.....	6
<b>Fig.7.</b> Custom IP diagram.....	8
<b>Fig.8.</b> Clock divider Verilog code.....	9
<b>Fig.9.</b> Part of User Logic Verilog code.....	9
<b>Fig.10.</b> Processor clock VS output signal clock for La note.....	10
<b>Fig.11.</b> Lost Woods Piano Stave.....	13
<b>Fig.12.</b> Stored notes.....	15
<b>Fig.13.</b> Main loop.....	15
<b>Fig.14.</b> Selection of the counter values depending on the selected note.....	16
<b>Fig.15.</b> Custom IP MDD.....	16

**LIST OF TABLES**

<b>Table 1.</b> Expansion connector A1 first 6 pins.....	5
<b>Table 2.</b> Frequency and counter values for musical notes.....	11
<b>Table 3.</b> Tempos .....	11
<b>Table 4.</b> Musical note values .....	12
<b>Table 5.</b> Musical notes in C code.....	14
<b>Table 6.</b> Logic Utilization.....	17
<b>Table 7.</b> Logic Distribution.....	17



## APPENDIX A: Custom IP Block User Guide

### Introduction

This document describes the specifications for the Custom IP core attached to PLBv4.6. It has been created in order to control an audio amplifier with digital input. The core has been tested with PmodAmp1 Digilent audio amplifier connected to Spartan-3 Starter Kit Board.

### Features

- Connects as a 32-bit slave on PLBv4.6.
- Digital output

Core Specifics			
Supported Device	Spartan-3	Starter	Kit Board
Version of core	v1.00a		
Used Resources			
Slices	1381		
FFs	1599		
LUTs	2242		
Block RAMs	8		
Provided with Core			
Documentation	Product Specification		
Design File Formats	Verilog, VHDL		
Design Tool Requirements			
Xilinx Implementation Tools	Xilinx Platform Studio		
Synthesis	XST		

### Functional description

The designed core consists of 4 registers, containing different data needed to control the audio amplifier. The contents of this registers are *counter*, *delay*, *next* and *start* as shown in the following table:

Table 1. Registers

Register description	Register name
<b>Counter</b>	slv_reg0
<b>Delay</b>	slv_reg1
<b>Next</b>	slv_reg2
<b>Start</b>	slv_reg3

The registers *counter* and *delay* must contain a value expressed in clock cycles. A clock divider will run creating a signal depending on counter value. This digital signal is the output tone, so core will work only if the selected amplifier accepts digital inputs. The output duration is determined by *delay* value and output will happen only if the content of *start* register is a logical "1". *Next* register is a closed one, this means that only the core is allowed to write on it if a determined tone has already finished being outputted, *next* will show it by writing a logical "1", if not, its content will be a logical "0". There exist also an external *enable* input allowing the core to run only if its value is a logical "1". As shown in the following Fig.1, the digital output signal is *left*.

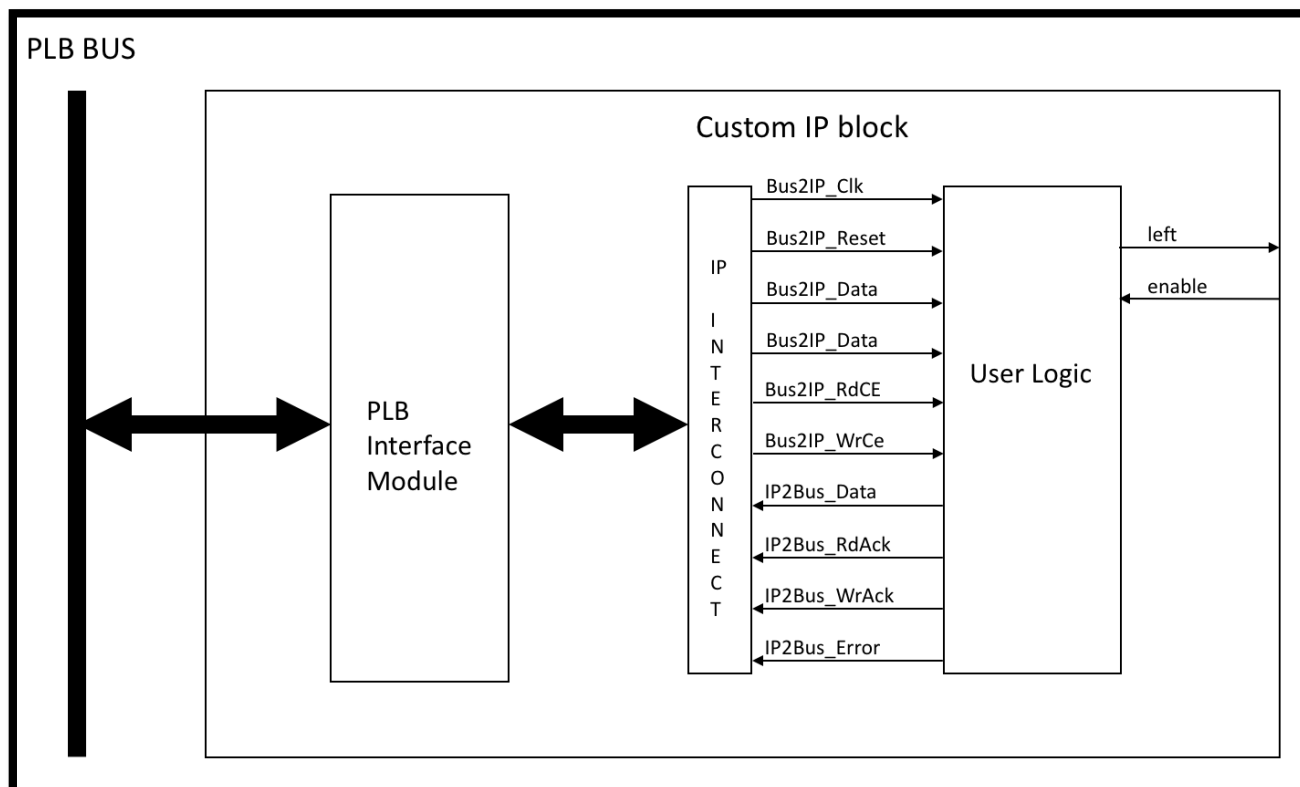


Fig.1. Custom IP diagram block

Signal Name	I/O	Description
Bus2IP_Clk	Input	System clock
Bus2IP_Reset	Input	External system reset
Bus2IP_Data	Input	Bus to IP data bus
Bus2IP_RdCE	Input	Read Chip Enable
Bus2IP_WrCE	Input	Write Chip Enable
IP2Bus_Data	Output	IP to bus data bus
IP2Bus_RdAck	Output	Read Acknowledgement
IP2Bus_WrAck	Output	Write Acknowledgement
IP2Bus_Error	Output	Error Response
left	Output	Output signal
enable	Input	External system enable

Table 2. I/O Signals

Data from *Bus2IP\_Data* will be stored in the register if its correspondent write chip enable, *Bus2IP\_WrCE* is activated. The registers are selected depending on *Bus2IP\_WrCE* value, it consists of an 8-bit variable, each bit selects one register, so, for example if data needs to be stored in register *slv\_reg0*, *Bus2IP\_WrCE* should contain "0001000". Exactly the same happens with read operation, "10000000" value in *Bus2IP\_RdCE*, for example, indicates that register "slv\_reg3" is going to be read. *IP2Bus\_RdAck* and *IP2Bus\_WrAck* provide signals to the bus to inform when data from *IP2Bus\_Data* is ready to be read or

written. *Enable* and *left* are external signals, if *enable* value is logical "1" it will send to output *left* signal, if it is not, output will be 0.

If the core is programmed to work with software, there are provided two functions to write or read the registers:

→void CUSTOM\_IP\_mWriteSlaveReg("number of register 0,1,2...")(Xuint32 BaseAddress, unsigned RegOffset, Xuint32 Value).

→Xuint32 CUSTOM\_IP\_mReadSlaveReg("number of register 0,1,2...")(Xuint32 BaseAddress, unsigned RegOffset).

Where "Xuint32 BaseAddress", from mentioned functions, is the base address of Custom IP block, "Unsigned RegOffset" is the slave register offset to, in this case, it's 0, and finally, "Xuint32Value", for write function, is the desired value to write to the register. These functions header file can be found in the directory: *microblaze\_0/libsrc/custom\_ip\_v1\_00\_a/src/custom\_ip.h*

**APPENDIX B: CD**

A CD containing all the necessary files to run properly this project is attached to it.