Interfacing a processor core in FPGA to an audio system

Master thesis performed in Electronics Systems

by

José Ignacio Mateos Albiach

LiTH-ISY-EX--06/3896--SE

Linköping 2006
Interfacing a processor in FPGA to an audio system

Master thesis in Electronics systems
at Linköping Institute of Technology
by
José Ignacio Mateos Albiach
LiTH-ISY-EX--06/3896--SE

Supervisor: Kent Palmkvist
Examiner: Kent Palmkvist

Linköping 2006
Abstract

FPGA Implementation of a Nios II processor for an audio system.
By José Ignacio Mateos Albiach.

The thesis project consists on developing an interface for a Nios II processor integrated in a board of Altera (UP3- 2C35F672C6 Cyclone II). The main goal is show how the Nios II processor can interact with the other components of the board.

The Quartus II software has been used to create to vhdl code of the interfaces, compile it and download it into the board. The Nios II IDE tool is used to build the C/C++ files and download them into the processor.

It has been prepared an application for the audio codec integrated in the board (Wolfson WM8731 24-bit sigma-delta audio CODEC). The line input of the audio codec receives an analog signal from a laptop, this signal is managed by the control interface of the audio codec. The converters ADCs and DACs are stereo 24-bit sigma delta and they are used with oversampling digital interpolation and decimation filters.

The digital interface of the audio codec sends the digital signal to the Nios II processor and receives the data from the processor. After building the interfaces for the audio codec and the processor, it has been prepared an application in C++ language for the processor that modifies the volume of the signal. The signal come back to the audio codec and it is possible to check the results with headphones or speakers at the line output of the audio codec.
# Table of contents

List of Figures .................................................................................................................. iii
Introduction ....................................................................................................................... 1

Chapter I: Software and Tools ......................................................................................... 3
  1.1 - Quartus II ............................................................................................................ 3
  1.2 - SOPC Builder. .................................................................................................... 4
  1.3 - Nios II IDE. ...................................................................................................... 5
  1.4 - Modelsim. ......................................................................................................... 6

Chapter II: DE2 Development and education board ......................................................... 7
  2.1 - Components. ..................................................................................................... 7
  2.2 - Applications. .................................................................................................... 10

Chapter III: Audio codec interface ................................................................................. 13
  3.1 - Introduction to the Audio Codec WM8731. ..................................................... 13
  3.2 - Digital interface. ............................................................................................... 14
  3.3 - Control interface. ............................................................................................. 16
  3.4 - Loop back in the digital interface. .................................................................... 19
  3.5 - Volume control with the control interface. ....................................................... 21
  3.6 - Volume display. ............................................................................................... 22

Chapter IV: Nios II interface .......................................................................................... 23
  4.1 - Introduction to Nios II. ..................................................................................... 23
  4.2 - Nios II interface. ............................................................................................... 26
  4.3 - Loop back in the Nios II processor. .................................................................. 28
  4.4 - Volume control with the Nios II processor. ..................................................... 30

Chapter V: Results and conclusions ............................................................................... 35

Chapter VI: Conclusion .................................................................................................. 37
  6.1 - Conclusions. ..................................................................................................... 37
  6.2 - Future work. ..................................................................................................... 37

References ........................................................................................................................ 39
## List of figures

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Altera DE2 board</td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td>Block diagram of the DE2 board</td>
<td>8</td>
</tr>
<tr>
<td>3.</td>
<td>Example of application</td>
<td>11</td>
</tr>
<tr>
<td>4.</td>
<td>Block diagram of the audio codec</td>
<td>13</td>
</tr>
<tr>
<td>5.</td>
<td>Left justified mode</td>
<td>15</td>
</tr>
<tr>
<td>6.</td>
<td>Audio codec schematic</td>
<td>17</td>
</tr>
<tr>
<td>7.</td>
<td>Three-wire mode</td>
<td>17</td>
</tr>
<tr>
<td>8.</td>
<td>Two-wire mode</td>
<td>18</td>
</tr>
<tr>
<td>9.</td>
<td>Map of registers</td>
<td>20</td>
</tr>
<tr>
<td>10.</td>
<td>Nios II processor core block diagram</td>
<td>24</td>
</tr>
<tr>
<td>11.</td>
<td>Nios II system top</td>
<td>27</td>
</tr>
<tr>
<td>12.</td>
<td>Audio codec for Nios II</td>
<td>27</td>
</tr>
<tr>
<td>13.</td>
<td>NiosII_audio_codec.bdf</td>
<td>28</td>
</tr>
<tr>
<td>14.</td>
<td>Nios II volume top</td>
<td>30</td>
</tr>
<tr>
<td>15.</td>
<td>NiosII_volume_control.bdf</td>
<td>31</td>
</tr>
</tbody>
</table>
Introduction

In the first simulations, the board used was **UP3 - 1C12 Cyclone I** of Altera that has the Nios II chip integrated. The tutorials for this board and the interface for the audio chip were completed. This audio chip is located in an external board **TLV320AIC23B** and it is connected by the expansion connectors of the both boards.

Unfortunately, there were some problems in the board of the audio chip and it was needed to use another board of Altera (**UP3- 2C35F672C6 Cyclone II**). This board has the audio codec and the Nios II chip integrated with many other interesting components.

The language used for programming the components is **vhdl**, used in previous studies. The language used to program the Nios II chip is **C/C++**.

The first tasks were the tutorials. These tutorials represent the first contact with the software used in the thesis project. They consist on building simple designs that contains the chip Nios II with the Quartus II and SOPC Builder tools and download a simple C program to the chip.
Chapter 1 : Software and Tools

1.1 Quartus II:

This is the main program and it is used to build the projects. It has many tools to prepare vhdl or verilog structures and download them to the board.

It is used to build our own vhdl code but it is also possible to use the SOPC Builder tool that specifies the Nios II processor core, memory and other peripherals. In this case, a symbol is created that may be used in a block diagram to be completed with the inputs and outputs and with other symbols.

The assignment of the pins is done with the Quartus II Assignment Editor, after this the vhdl files can be compiled and downloaded to the board with the Programmer tool. In this project, JTAG programming (Joint Test Action Group) is used so the configuration bit stream is downloaded directly into the FPGA that will retain the configuration as long as the power is applied to the board.

The following steps describe the basic design flow for using the Quartus II graphical user interface:

1. To create a new project and specify a target device or device family, on the File menu, click New Project Wizard.

2. Use the Text Editor to create a Verilog HDL, VHDL, or Altera Hardware Description Language (AHDL) design. Use the Block Editor to create a block diagram with symbols that represent other design files, or to create a schematic.

3. Use the MegaWizard Plug-In Manager to generate custom variations of megafUNCTIONs and IP functions to instantiate in your design, or create a system-level design by using SOPC Builder or DSP Builder.

4. Specify any initial design constraints using the Assignment Editor, the Pin Planner, the Settings dialog box, the Floorplan Editor, or the Design Partitions window.

5. (Optional) Perform an early timing estimate to generate early estimates of timing results before fitting.

6. Synthesize the design with Analysis & Synthesis.

7. (Optional) If your design contains partitions and you are not performing a full compilation, merge the partitions with partition merge.

8. (Optional) Generate a functional simulation netlist for your design and perform a functional simulation with the Simulator.

9. Place and route the design with the Fitter.
10. Perform a power estimation and analysis with the PowerPlay Power Analyzer.

11. Use the Simulator to perform timing simulation for the design. Use the TimeQuest Timing Analyzer or the Classic Timing Analyzer to analyze the timing of your design.

12. (Optional) Use physical synthesis, the Timing Closure floorplan, the LogicLock feature, and the Assignment Editor to correct timing problems.

13. Create programming files for your design with the Assembler and program the device with the Programmer and Altera programming hardware.

14. (Optional) Debug the design with the SignalTap® II Logic Analyzer, an external logic analyzer, the SignalProbe™ feature, or the Chip Editor.

15. (Optional) Manage engineering changes with the Chip Editor, the Resource Property Editor, and the Change Manager.

### 1.2 SOPC Builder:

This tool generates the Nios II processor system, adding the desire peripherals. It is possible to choose in which language (verilog or vhdl) is the code generated. It is also possible to specify the base addresses and interrupt request (IRQ) priorities for the components. The JTAG Uart provides a way to communicate the processor through the USB-Blaster. Many other components as timers, inputs, outputs, can be added and personal components and interfaces can be created.

SOPC Builder has an intuitive user interface that allows to select and parameterize components, select connections between components, generate a complete system including interconnect, and automatically generate memory map header files for use by software engineers.

An SOPC Builder component is a design module that SOPC Builder recognizes and can automatically integrate into a system. SOPC Builder connects multiple components together to create a top-level HDL file called the system module. SOPC Builder generates Avalon switch fabric that contains logic to manage the connectivity of all components in the system.

The Avalon switch fabric is the glue that binds SOPC Builder-generated systems together. The Avalon switch fabric is the collection of signals and logic that connects master and slave components, including address decoding, data-path multiplexing, wait-state generation, arbitration, interrupt controller, and data-width matching. SOPC Builder generates the Avalon switch fabric automatically, so that you do not have to manually perform the tedious, error-prone task of connecting hardware modules.

SOPC Builder provides an easy method for you to develop and connect your own components. With the Avalon interface, user-defined logic need only adhere to a simple interface based on address, data, readable, and write-enable signals. Once the SOPC Builder component is created, you can reuse the component in other SOPC Builder systems, and share the component with other design teams.
The different configurations for the Nios II processor core has to be chosen, they provide different levels of performance and resource utilization. It is possible to add different types of memories; on-chip, SRAM, SDRAM and FLASH memory, and select the width and total size.

In the connection panel you specify the master-slave connections between components. Each SOPC Builder component can have one or more master and slave ports. Each slave port must be connected to a master port. You can connect any master port to any slave port, as long as the ports use the same type of hardware interface. If they use different interfaces, they must communicate through a bridge component, such as the AHB-to-Avalon bridge provided with SOPC Builder.

In systems with one or more slave ports that generate IRQs, the Avalon switch fabric includes interrupt controller logic. A separate interrupt controller is generated for each master port that accepts interrupts. The interrupt controller aggregates IRQ signals from all slave ports, and maps slave IRQ outputs to user-specified values on the master IRQ inputs. Each slave port optionally produces an IRQ output signal. There are two master signals related to interrupts: irq and irq number. SOPC Builder generates the interrupt controller in one of two configurations, software priority or hardware priority, depending on the interrupt signals present on the master port.

The main feature of the Board Settings tab is the SOPC Builder pin mapper. The pin mapper simplifies the process of assigning FPGA pins. You use the pin mapper to map logical connections between system components and devices mounted on the PCB. Based on your pin mapper settings, SOPC Builder applies appropriate pin assignments to your Quartus II project.

After you have configured all system parameters and specified the desired generation options on this tab, clicking Generate starts the system generation process. Generate is available from all tabs in SOPC Builder. It is disabled whenever there are any errors displayed in the messages window. The middle area of the System Generation tab is a progress display, showing a time-stamped list of progress messages that occur during system generation. SOPC Builder executes multiple tools and scripts to generate the system.

1.3 Nios II IDE (Integrated Develop Environment):

This program compiles C language programs and downloads them into the Nios II chip. It is needed to select the SOPC system used, the memory, timer and many different options of compiling, debugging and running the C program. JTAG Uart is used to download the C file to the Nios II chip.

The Nios II IDE presents a new project wizard used to automate the set-up of the C/C++ application project and system library projects. In addition to the new project wizard, the Nios II IDE provides software code examples (in the form of project templates) to help software engineers bring up working systems as quickly as possible. Each template is a collection of software files and project settings. Designers can add their own source code to the project by placing the code in the project directory or importing the files into the project.
Based on the industry-standard GNU tool chain, the Nios II IDE provides a graphical user interface (GUI) to the GCC compiler. The Nios II IDE build environment is designed to facilitate software development for Altera's Nios II processors, providing an easy-to-use push-button flow, while also allowing designers to manipulate advanced build settings. The Nios II IDE build environment automatically produces a makefile based on the user's specific system configuration (the SOPC Builder-generated PTF file). Changes made in the Nios II IDE compiler/linker settings are automatically reflected in this auto-generated makefile. These settings can include options for the generation of memory initialization files (MIF), flash content, simulator initialization files (DAT/HEX), and profile summary files.

The Nios II IDE contains a robust software debugger based on the GNU debugger, GDB. The debugger provides many basic debug features, as well as several advanced debug features not usually available with low-cost processor development kits. Run and debug operations are available by right-clicking the project. The Nios II IDE allows you to run or debug the project either on a target board or the Nios II instruction set simulator (ISS).

Every application has its own library. This library contains the files referenced to the system built with the SOPC Builder. It is possible to select the memory, inputs, outputs and timers for the applications. There are many options of compiling and running. It is very useful for this thesis project the ‘Small C library option’. When the "Small C library" option is checked, the system library uses a reduced implementation of the Newlib ANSI C standard library. Notably, the printf() family of routines (printf(), fprintf(), sprintf(), etc.) does not support floating-point values when this option is checked. The reduced library is optimized for smaller memory footprint, although the implementation may be less time efficient.

1.4 Modelsim 6.0:

This is a program to simulate vhdl code. The vhdl files are complied, the input signals are forced to the correspondent values and the evolution of the signals is shown in waves and lists. The basic steps for simulating a design in ModelSim are:

1-Creating the working library
   In ModelSim, all designs are compiled into a library. You typically start a new simulation in ModelSim by creating a working library called "work". "Work" is the library name used by the compiler as the default destination for compiled design units.

2-Compiling your design
   After creating the working library, you compile your design units into it. The ModelSim library format is compatible across all supported platforms. You can simulate your design on any platform without having to recompile your design.

3-Running the simulation
   With the design compiled, you invoke the simulator on a top-level module (Verilog) or a configuration or entity/architecture pair (VHDL). Assuming the design loads successfully, the simulation time is set to zero, and you enter a run command to begin.

4-Debugging your results
   If you don't get the results you expect, you can use ModelSim’s robust debugging environment to track down the cause of the problem.
Chapter 2 : DE2 Development and education board

2.1 Components:

The information about the board and its components has been taken from the ‘DE_user_manual.pdf’ that is included in the CD of the DE2 package. The DE2 board has many features that allow the user to implement a wide range of designed circuits, from simple circuits to various multimedia projects.

The following hardware is provided on the DE2 board:

- Altera Cyclone® II 2C35 FPGA device
- Altera Serial Configuration device - EPCS16
- USB Blaster (on board) for programming and user API control; both JTAG and Active Serial (AS) programming modes are supported
- 512-Kbyte SRAM
- 8-Mbyte SDRAM
- 4-Mbyte Flash memory (1 Mbyte on some boards)
- SD Card socket
- 4 pushbutton switches
- 18 toggle switches
- 18 red user LEDs
- 9 green user LEDs
- 50-MHz oscillator and 27-MHz oscillator for clock sources
- 24-bit CD-quality audio CODEC with line-in, line-out, and microphone-in jacks
- VGA DAC (10-bit high-speed triple DACs) with VGA-out connector
- TV Decoder (NTSC/PAL) and TV-in connector
- 10/100 Ethernet Controller with a connector
- USB Host/Slave Controller with USB type A and type B connectors
- RS-232 transceiver and 9-pin connector
- PS/2 mouse/keyboard connector
- IrDA transceiver
- Two 40-pin Expansion Headers with diode protection

In addition to these hardware features, the DE2 board has software support for standard I/O interfaces and a control panel facility for accessing various components. Also, software is provided for a number of demonstrations that illustrate the advanced capabilities of the DE2 board.
This is the block diagram of the DE2 board:

![Block diagram of the DE2 board](image)

**Figure 2 – Block diagram of the DE2 board [2]**

More detailed information about the blocks:

**Cyclone II 2C35 FPGA**
- 33,216 LEs
- 105 M4K RAM blocks
- 483,840 total RAM bits
- 35 embedded multipliers
- 4 PLLs
- 475 user I/O pins
- FineLine BGA 672-pin package

**Serial Configuration device and USB Blaster circuit**
- Altera’s EPCS16 Serial Configuration device
- On-board USB Blaster for programming and user API control
- JTAG and AS programming modes are supported

**SRAM**
- 512-Kbyte Static RAM memory chip
- Organized as 256K x 16 bits
- Accessible as memory for the Nios II processor and by the DE2 Control Panel

**SDRAM**
- 8-Mbyte Single Data Rate Synchronous Dynamic RAM memory chip
- Organized as 1M x 16 bits x 4 banks
- Accessible as memory for the Nios II processor and by the DE2 Control Panel
Flash memory
- 4-Mbyte NAND Flash memory (1 Mbyte on some boards)
- 8-bit data bus
- Accessible as memory for the Nios II processor and by the DE2 Control Panel

SD card socket
- Provides SPI mode for SD Card access
- Accessible as memory for the Nios II processor with the DE2 SD Card Driver

Pushbutton switches
- 4 pushbutton switches
- Debounced by a Schmitt trigger circuit
- Normally high; generates one active-low pulse when the switch is pressed

Toggle switches
- 18 toggle switches for user inputs
- A switch causes logic 0 when in the DOWN (closest to the edge of the DE2 board) position and logic 1 when in the UP position

Clock inputs
- 50-MHz oscillator
- 27-MHz oscillator
- SMA external clock input

Audio CODEC
- Wolfson WM8731 24-bit sigma-delta audio CODEC
- Line-level input, line-level output, and microphone input jacks
- Sampling frequency: 8 to 96 KHz
- Applications for MP3 players and recorders, PDAs, smart phones, voice recorders, etc.

VGA output
- Uses the ADV7123 240-MHz triple 10-bit high-speed video DAC
- With 15-pin high-density D-sub connector
- Supports up to 1600 x 1200 at 100-Hz refresh rate
- Can be used with the Cyclone II FPGA to implement a high-performance TV Encoder

NTSC/PAL TV decoder circuit
- Uses ADV7181B Multi-format SDTV Video Decoder
- Supports NTSC-(M,J,4.43), PAL-(B/D/G/H/I/M/N), SECAM
- Integrates three 54-MHz 9-bit ADCs
- Clocked from a single 27-MHz oscillator input
- Multiple programmable analog input formats: Composite video (CVBS), S-Video(Y/C), and YPrPb components
- Supports digital output formats (8-bit/16-bit): ITU-R BT.656 YCrCb 4:2:2 output + HS, VS, and FIELD
- Applications: DVD recorders, LCD TV, Set-top boxes, Digital TV, Portable video devices
10/100 Ethernet controller
- Integrated MAC and PHY with a general processor interface
- Supports 100Base-T and 10Base-T applications
- Supports full-duplex operation at 10 Mb/s and 100 Mb/s, with auto-MDIX
- Fully compliant with the IEEE 802.3u Specification
- Supports IP/TCP/UDP checksum generation and checking
- Supports back-pressure mode for half-duplex mode flow control

USB Host/Slave controller
- Complies fully with Universal Serial Bus Specification Rev. 2.0
- Supports data transfer at full-speed and low-speed
- Supports both USB host and device
- Two USB ports (one type A for a host and one type B for a device)
- Provides a high-speed parallel interface to most available processors; supports Nios II with a Terasic driver
- Supports Programmed I/O (PIO) and Direct Memory Access (DMA)

Serial ports
- One RS-232 port
- One PS/2 port
- DB-9 serial connector for the RS-232 port
- PS/2 connector for connecting a PS2 mouse or keyboard to the DE2 board

IrDA transceiver
- Contains a 115.2-kb/s infrared transceiver
- 32 mA LED drive current
- Integrated EMI shield
- IEC825-1 Class 1 eye safe
- Edge detection input

Two 40-pin expansion headers
- 72 Cyclone II I/O pins, as well as 8 power and ground lines, are brought out to two 40-pin expansion connectors
- 40-pin header is designed to accept a standard 40-pin ribbon cable used for IDE hard drives
- Diode and resistor protection is provided

2.1 Applications:

The board has many audio and video capabilities. It also has many possibilities for external connections, USB, ethernet, RS.32, PS/2, IrDA, expansion headers… Some examples of applications are:

- Play video and audio input form a DVD player using the VGA output and the Audio codec on the board.
- Use the Philips ISP1362 chip and the Nios II processor to implement a USB mouse movement detector
- A karaoke machine that uses the Audio codec and the microphone-in, line-in and line-out ports of the board.
- Nios II processor that sends and receives Ethernet packets using the DN9000A Ethernet PHY/MAC controller.
- Nios II processor that reads music data stored in the SD-card and uses the audio codec to play the music.

The goal for this thesis project is to show how the Nios II processor can interact with the other components of the board. The application prepared in this thesis project use the 24-bit CD-quality audio CODEC.

Many other components, such as clocks, memory or USB connection, are needed to implement the application and it is very important to understand its functionality. Other components like the displays or switches are used to show the results and check the functionality of the systems.

The software was installed in a laptop; Quartus II, Nios II IDE and Modelsim. The license to compile and download files with Quartus II is provided by a USB stick of Altera (Rainbow Super Pro T000099476). The laptop is connected to the board using a USB connection and Jtag programming. The line input signal was provided also by the laptop to the board and the line output was received by headphones or speakers.

After this, the hardware is ready to program the board with vhdl files and program the Nios II processor with C/C++ files.
Chapter 3. Audio Codec Interface

3.1 Introduction to the Audio Codec WM8731:

This is a low power stereo CODEC with an integrated headphone driver. It is designed for portable MP3 players and recorders, CD and minidisc recorders, PDAs and smartphones.

The digital audio inputs **word lengths** are from 16 bits to 32 bits and the **sample rate** from 8 kHz to 96 kHz. The default values are used for this project, 24 bits for the input audio data bit length and 48 kHz for the sample rate.

There is a stereo line input and a mono microphone input. They are provided with volume control and mute function from the control interface. The converters ADCs and DACs are stereo 24-bit sigma delta and they are used with oversampling digital interpolation and decimation filters.

The external clock input **XCK** has different values of frequency depending on the sample rate. In the default mode, the frequency is 12.288MHz. This signal is an input of the audio codec so it has to be built. The clock signal taken form the board is CLOCK_50 (pin_N2) that has 50 MHz, so it is needed to divide it by four to get the XCK signal, f=50/4=12.5MHz. This is not the exact value of frequency required but it is enough for the purpose of the project. The accuracy is decreased with this but it is not possible to notice any change while checking the results.

![Figure 4 – Block diagram of the audio codec [4]](image-url)
3.2 Digital interface:

The digital interface controls the digital input and output of the audio codec (DACDAT and ADCDAT). There are four modes to operate; left justified, right justified, I2S and DSP mode. All these modes are MSB first and operate with data from 16 to 32 bits.

The signals that control the digital audio interface are:

- **ADCLRC**, an alignment clock that control whether left or right channel is present on the ADCDAT line.

- **DACLRC**, an alignment clock that control whether left or right channel is present on the DACDAT line.

- **BCLK**, it is an input in slave mode, the data is ready each rising edge is this signal. The frequency of this signal is f(BCLK)=f(XCK)/4 so it is needed to divide this signal by four.

```verbatim
--CLOCK_50 has f=50MHz => XCK=CLK/4
if (CNT1=4) then
  CNT1:=1;
  XCK<='1';
  CNT2:=CNT2+1;
elsif (CNT1=2) then
  XCK<='0';
  CNT1:=CNT1+1;
elsif (CNT1=0) then
  XCK<='1';
  CNT1:=CNT1+1;
else CNT1:=CNT1+1;
end if;

--BCLK: in normal mode BCLK=XCK/4 for all sample rates except for 88.2-96kHz
if (CNT2=5) then
  CNT2:=1;
  BCLK<=0';
  CNT3:=CNT3+1;
elsif (CNT2=3) then
  BCLK<='1';
elsif (CNT2=0) then
  BCLK<=0';
  CNT2:=CNT2+1;
end if;
```
--LRCIN, LRCOUT: Sample rate=48 KHz => LRCIN=BCLK/32
if (CNT3=65) then
    CNT3:=0;
    LRCIN<='1';
    LRCOUT<='1';
elseif (CNT3=33) then
    LRCIN<='0';
    LRCOUT<='0';
elseif (CNT3=0) then
    LRCIN<='1';
    LRCOUT<='1';
    CNT3:=CNT3+1;
end if;

The **left justified mode** is implemented in this project. This is not the default mode so it is need to change it at the control interface. This mode is the default mode in the board where the digital interface was designed at the beginning (UP31C12 Cyclone I). This is the easier mode to program and it is enough for the purpose of the thesis project.

In this mode, the MSB is available on the first rising edge of the BCLK following an ADCLRC or DACLRC transition.

Figure 5 – Left justified mode [5]
3.3 Control interface:

The clock for the control serial data input is SCLK. The maximum frequency for this signal is 400 kHz, the frequency used in this project is \( f = \frac{50 \text{ MHz}}{128} = 390.625 \text{ kHz} \). The data input signal is SDIN, it contains the information for the control interface.

genSCLK : process(CLOCK)
variable CNT4 : natural;
variable c4 : std_logic;
begin

if (CLOCK'event and CLOCK='1') then

    if (r='1' or l='1' or h='1') then
        c4:='1';
    end if;

    if (r='0' and l='0' and h='0' and c4='1') then
        CNT4:=0;
        c4:='0';
    end if;

    if (CNT4=256) then
        SCLK<='1';
        CNT4:=1;
    elsif (CNT4=128)then
        SCLK<='0';
        CNT4:=CNT4+1;
    elsif (CNT4=0) then
        SCLK<='1';
        CNT4:=CNT4+1;
    else CNT4:=CNT4+1;
    end if;

end if;
end process;
The control interface may be operated with 3-wire or 2-wire interface. It is possible to select the serial control mode by setting the state of the MODE pin. This pin is set to ground for this audio codec so it is needed to use the **2-wire interface**.

The first simulations were built with 3-wire interface. This is the default mode for the TLV320AIC23B external audio chip that was used with the UP31C12 Cyclone I. This mode is easier to implement due to the CSB signal is used to latch the data. It is not needed to set start and finish conditions.
The control interface was reprogrammed using the 2-wire interface when the original board was replaced by the UP3- 2C35F672C6 Cyclone II board. The **start condition** is a falling edge on SDIN while SCLK is high. The following seven bits determines which device receives the data, the **address** depends on the CBS state (set to ground in this codec) so it is “0011010”. After this address, the **bit R/W** determines the direction of data transfer, in this case a ‘0’ indicates ‘write’.

The device recognizes the address and R/W by pulling SDIN low during the ninth clock cycle, **acknowledging** the data transfer. The control follows with two bit blocks (separated with another acknowledge). The first block B[15:9] contains the **control address bits**, and the second block B[8-0] contains the **control data bits**.

The **stop condition** after the data transfer is a rising edge on SDIN when SCLK is high. If a start condition is detected out of the sequence at any point in the data transfer then the device will jump to the idle condition. After a complete control operation, the audio codec returns to the idle state and waits for another start condition.

The code that generates the SDIN signal in vhdl is:

```vhdl
case CNT5 is
  when 0 => SDIN<=start;
  when 64 => SDIN<=ack;

  when 192 => SDIN<=address(6);
  when 448 => SDIN<=address(5);
  when 704 => SDIN<=address(4);
  when 960 => SDIN<=address(3);
  when 1216 => SDIN<=address(2);
  when 1472 => SDIN<=address(1);
  when 1728 => SDIN<=address(0);

  when 1984 => SDIN<=rw;

  when 2240 => SDIN<=ack;
end case;
```

---

**Figure 8 – Two-wire mode [8]**
3.4 Loop back in the digital interface:

The first contact with the audio codec interface consisted on preparing a loop back of the sound signal. The analog signal enters in the line input of the audio chip and the digital output signal (ADCDAT) is put it directly back in the digital input signal (DACDAT).

A new project named ‘loop_back.qpf’ was created with Quartus II software. For this application, the vhdl files are compiled and downloaded by the program into the board. It is needed to change some default values in the control interface to prepare this loop back.

There are 11 registers with 16 bits per register (7 bits of address and 9 bits of data). It is needed to change some of the default options for these registers. The register map is shown in the next figure.
First, the ‘mute’ option has to be deactivated and the ‘left to right channel line input volume and mute data load control’ has to be activated in the ‘left line in’. The register address is R0 (‘00000000’) that corresponds to B[15:9] in the 2-wire process previously related. To change the default values, B7 has to be set to ‘0’ and B8 has to be set to ‘1’, the other data bits (related to volume) must be the same. With this, the mute option is turned off and the ‘right line in’ volume and mute options will have the same configuration than the ‘left line in’ options.

control:="0000001000100111";

There is an internal bypass activated in the default options so it is needed to deactivate it. The ‘DAC select’ option has to be activated for this register too. The ‘analog audio path’ register address is R4 (‘00001000’) and it is needed to switch B3 to ‘0’ and B4 to ‘1’.

control:="0000100000010010";

The power down control address has to be changed too. The register is R6 (‘00001100’), it is needed to deactivate the power off mode (B7=’0’), activate line input, ADC, DAC and outputs power down (B0=B2=B3=B4=’0’).

control:="000011000000100010";

The last change is the ‘digital audio interface format’ (‘00001110’) where the ‘left justified mode’ is selected switching B0 to’1’ and B1 to’0’.

control:="000011100000100010";

<table>
<thead>
<tr>
<th>REGISTER</th>
<th>B 15</th>
<th>B 14</th>
<th>B 13</th>
<th>B 12</th>
<th>B 11</th>
<th>B 10</th>
<th>B 9</th>
<th>B 8</th>
<th>B 7</th>
<th>B 6</th>
<th>B 5</th>
<th>B 4</th>
<th>B 3</th>
<th>B 2</th>
<th>B 1</th>
<th>B 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 (00h)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R1 (02h)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R2 (04h)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R3 (06h)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R4 (08h)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R5 (0Ah)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R6 (0Ch)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R7 (0Eh)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R8 (10h)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R9 (12h)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R15(1Eh)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 9 – Map register [9]**
After this, the ‘digital_interface.vhd’ and ‘control_interface.vhd’ files are built. They will be the base for the rest of the project. The file ‘loop_back.vhd’ links both components and is the main file. The part of the code where the components are linked is:

\[
\begin{align*}
LB_0 & : \text{digital}\_\text{interface} \text{ port map (CLK,RST,ADCDAT,DACDAT,DACLRCK,ADCLRCK,BCLKC,XCK);} \\
LB_1 & : \text{control}\_\text{interface} \text{ port map (CLK,RST,VOL\text{\_H},VOL\text{\_L},MUT,SDAT,VOL,SCLKC);} \\
LB_2 & : \text{volume}\_\text{display} \text{ port map (CLK,VOL,RST,HEX1,HEX0);} \\
\end{align*}
\]

The ‘digital_interface.vhd’ file prepares the XCK, BCLK, ADCLR and DACLRC signals for the audio codec and takes care of the loop back.

\[
\begin{align*}
\text{DACDAT} & \leftarrow \text{ADCDAT}
\end{align*}
\]

The ‘control_interface.vhd’ file prepares the SCLK and SDIN signals for the control interface. The reset signal that manages these files is set in the switch button KEY0 (pin_G26). This switch is debounced so there are no problems of glitches, it provides a high logic level when it is not pressed.

Before the compilation of these files, the pin names are set with the pin assignment editor. The pin names for the audio codec interfaces are shown in the next table.

<table>
<thead>
<tr>
<th>Signal name</th>
<th>FPGA pin number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADCLRCK</td>
<td>PIN C5</td>
<td>Audio codec ADC LR clock</td>
</tr>
<tr>
<td>ADCDAT</td>
<td>PIN B5</td>
<td>Audio codec ADC data</td>
</tr>
<tr>
<td>DACLRCK</td>
<td>PIN C6</td>
<td>Audio codec DAC LR clock</td>
</tr>
<tr>
<td>DACDAT</td>
<td>PIN A4</td>
<td>Audio codec DAC data</td>
</tr>
<tr>
<td>XCK</td>
<td>PIN A5</td>
<td>Audio codec chip clock</td>
</tr>
<tr>
<td>BCLK</td>
<td>PIN B4</td>
<td>Audio codec Bit-Stream clock</td>
</tr>
<tr>
<td>SCLK</td>
<td>PIN A6</td>
<td>Audio codec I2C data</td>
</tr>
<tr>
<td>SDAT</td>
<td>PIN B6</td>
<td>Audio codec I2C clock</td>
</tr>
</tbody>
</table>

After this, the project is compiled generating the ‘loop_back.sof’ file that will be downloaded with the programmer. Before the reset button is pressed, the internal loop back is working due to the bypass is not deactivated and it is possible to listen the music at the speakers. After the reset button is pressed, the control interface changes the default options and the same output is got but the loop back is done in the digital interface.

3.5 Volume control with the control interface:

Next step consist on modifying the volume with control interface and, with this, the functionality of the loop back created will be checked. For this, it is needed to prepare some changes in the ‘control_interface.vhd’ file. The ‘volume_control.vhd’ file links the components and the new project name is ‘volume_control.qpf’.
The switch button \textit{KEY1} (pin \textit{N23}) is used for the ‘\textbf{volume low}’ control, the \textit{KEY2} (pin \textit{P23}) is used for the ‘volume high’ control and \textit{KEY3} (pin \textit{W26}) is used for the ‘\textbf{mute}’ control. These switches are debounced like the reset button previously related.

When one of these buttons is pressed, it starts a new control operation that modifies the ‘\textbf{left line in}’ (0000000). The variable \texttt{volume[4:0]} increases or decreases when the buttons are pressed and set the volume at B[4:0] in the SDIN signal. It is not needed to take care of the right channel volume due to the ‘left to right load’ is activated.

\begin{verbatim}
control:="0000001000"&volu;
\end{verbatim}

After this, the new pins for the switches are set, the new vhdl files are compiled to generate the ‘\texttt{volume_control.pof}’ file that is downloaded into the board. The results are as expected, the volume of the line input can be modified by the switch buttons.

\textbf{3.6 Volume display:}

One of the options of the board is to use the \textbf{7-segment displays} to show the volume value. For this, another component named ‘\texttt{volume_display.vhd}’ is created. The volume value is the input, this value is decoded to get the values of the 7-segment display for the outputs. Displays \texttt{HEX0} and \texttt{HEX1} are used to show the numbers. The initial value for the volume is 23 and it is changing from 0 to 31 (5 bits)
Chapter 4 . Nios II Interface

4.1 Introduction to Nios II:

The Nios II processor is a general-purpose RISC processor core. Some information about the characteristics of this processor has been taken from this file: ‘http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf’

- Full 32-bit instruction set, data path, and address space
- 32 general-purpose registers
- 32 external interrupt sources
- Single-instruction 32 × 32 multiply and divide producing a 32-bit result
- Dedicated instructions for computing 64-bit and 128-bit products of multiplication
- Floating-point instructions for single-precision floating-point operations
- Single-instruction barrel shifter
- Access to a variety of on-chip peripherals, and interfaces to off-chip memories and peripherals
- Hardware-assisted debug module enabling processor start, stop, step and trace under integrated development environment (IDE) control
- Software development environment based on the GNU C/C++ tool chain and Eclipse IDE
- Integration with Altera's SignalTap(r) II logic analyzer, enabling realtime analysis of instructions and data along with other signals in the FPGA design
- Instruction set architecture (ISA) compatible across all Nios II processor systems
- Performance up to 250 DMIPS

A Nios II processor system is equivalent to a microcontroller or “computer on a chip” that includes a CPU and a combination of peripherals and memory on a single chip. The term “Nios II processor system” refers to a Nios II processor core, a set of on-chip peripherals, onchip memory, and interfaces to off-chip memory, all implemented on a single Altera chip. Like a microcontroller family, all Nios II processor systems use a consistent instruction set and programming model.

The Nios II architecture describes an instruction set architecture (ISA). The ISA in turn necessitates a set of functional units that implement the instructions. A Nios II processor core is a hardware design that implements the Nios II instruction set and supports the functional units described previously. The processor core does not include peripherals or the connection logic to the outside world. It includes only the circuits required to implement the Nios II architecture.
The Nios II architecture supports a flat register file, consisting of thirty two 32-bit general-purpose integer registers, and six 32-bit control registers. The architecture supports supervisor and user modes that allow system code to protect the control registers from errant applications. The Nios II architecture allows for the future addition of floating point registers.

The Nios II arithmetic logic unit (ALU) operates on data stored in general-purpose registers. ALU operations take one or two inputs from registers, and store a result back in a register. The ALU supports the next data operations; addition, subtraction, multiplication and division. To implement any other operation, software computes the result by performing a combination of the fundamental operations.

Nios II processor systems are configurable, the memories and peripherals vary from system to system. As a result, the memory and I/O organization varies from system to system. A Nios II core uses one or more of the following to provide memory and I/O access:

- Instruction master port - An Avalon master port that connects to instruction memory via Avalon switch fabric
- Instruction cache - Fast cache memory internal to the Nios II core
- Data master port - An Avalon master port that connects to data memory and peripherals via Avalon switch fabric
- Data cache - Fast cache memory internal to the Nios II core
- Tightly coupled instruction or data memory port - Interface to fast memory outside the Nios II core

The Nios II architecture supports separate instruction and data buses. Both the instruction and data buses are implemented as Avalon master ports that adhere to the Avalon interface.
specification. The data master port connects to both memory and peripheral components, while the instruction master port connects only to memory components.

The Nios II architecture does not specify anything about the existence of memory and peripherals; the quantity, type, and connection of memory and peripherals are system-dependent. Typically, Nios II processor systems contain a mix of fast on-chip memory and slower off-chip memory. Peripherals typically reside on-chip, although interfaces to off-chip peripherals also exist.

Instruction Master Port

The Nios II instruction bus is implemented as a 32-bit Avalon master port. The instruction master port performs a single function: Fetch instructions that will be executed by the processor. The instruction master port does not perform any write operations. The instruction master port is a pipelined Avalon master port. Support for pipelined Avalon transfers minimizes the impact of synchronous memory with pipeline latency and increases the overall $f_{\text{MAX}}$ of the system. The instruction master port can issue successive read requests before data has returned from prior requests. The Nios II processor can prefetch sequential instructions and perform branch prediction to keep the instruction pipe as active as possible.

The instruction master port always retrieves 32 bits of data. The instruction master port relies on dynamic bus-sizing logic contained in the Avalon switch fabric. By virtue of dynamic bus sizing, every instruction fetch returns a full instruction word, regardless of the width of the target memory. Consequently, programs do not need to be aware of the widths of memory in the Nios II processor system.

Data Master Port

The Nios II data bus is implemented as a 32-bit Avalon master port. The data master port performs two functions:

- Read data from memory or a peripheral when the processor executes a load instruction
- Write data to memory or a peripheral when the processor executes a store instruction

Jtag programming

The Nios II architecture supports a JTAG debug module that provides onchip emulation features to control the processor remotely from a host PC. PC-based software debugging tools communicate with the JTAG debug module and provide facilities, such as:

- Downloading programs to memory
- Starting and stopping execution
- Setting breakpoints and watchpoints
- Analyzing registers and memory
- Collecting real-time execution trace data
4.2 Nios II interface:

The investigation and development part of the thesis project starts at this point. The previous work consisted on theoretical issues and functionality of the board and audio codec. A project named ‘niosII_audio_interface.qpf’ is created with the Quartus II software. It is needed to use the SOPC builder tool to implement a design with the Nios II processor core.

As it is explained in the ‘Software and tools’ chapter, this tool generates the Nios II processor system, adding the desire peripherals. This program generates a block symbol that may be used in a block diagram later. The name for this symbol is ‘niosII_system_top’ and the file generated is ‘niosII_system_top.ptf’. The language selected is vhdl for the generated code.

First, the Nios II processor is added selecting the Nios II/f core, 4Kbytes of instruction cache, 2 Kbytes of data cache and level 1 for the Jtag debug module. Then, the Jtag Uart component is added with the default configuration, data depth 64 and IRQ threshold 8.

After this, a timer is chosen with the default configuration too. Processor systems require at least one memory for data and instructions. The memory added is 30 Kbyte on-chip memory, Ram type and 32 bits of memory width.

After this, the audio codec interface has to be added. There are two possibilities to do this:

1-Create a new component in the SOPC builder in order to have all the components in the same symbol.
2-Prepare another symbol outside the niosII_system_top.

The second option has been implemented. It is easier to build and easier to understand the functionality when several blocks are shown. To link both blocks, it is needed one signal for the data control (‘ready’) and two 20-bit signals for the data flow: ‘data_for_niosII’ and ‘dacdat’. These signals are added as PIO components (Parallel Input Output).

<table>
<thead>
<tr>
<th>Module name</th>
<th>Description</th>
<th>Base</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>niosII</td>
<td>Nios II Processor - Altera Corporation</td>
<td>0x00008000</td>
<td>0x000087FF</td>
</tr>
<tr>
<td>Jtag_uart</td>
<td>JTAG UART</td>
<td>0x00008840</td>
<td>0x00008847</td>
</tr>
<tr>
<td>Sys_clk</td>
<td>Interval timer</td>
<td>0x00008800</td>
<td>0x0000881F</td>
</tr>
<tr>
<td>onchip_memory</td>
<td>On-Chip Memory (RAM or ROM)</td>
<td>0x00000000</td>
<td>0x000077FF</td>
</tr>
<tr>
<td>Dacdat</td>
<td>PIO (Parallel I/O)</td>
<td>0x00008820</td>
<td>0x0000882F</td>
</tr>
<tr>
<td>Data_for_niosII</td>
<td>PIO (Parallel I/O)</td>
<td>0x00008830</td>
<td>0x0000883F</td>
</tr>
<tr>
<td>Ready</td>
<td>PIO (Parallel I/O)</td>
<td>0x00008850</td>
<td>0x0000885F</td>
</tr>
</tbody>
</table>

The base address and IRQs are assigned with the ‘auto-assign’ option. After this, the system can be generated in the ‘niosII_system_top.ptf’ file. At this point, the system_top symbol is ready to be used in a block diagram. The ‘niosII_audio_interface.bdf’ block diagram is created and, then the symbol is added.
After this, the ‘audio_codec_for_niosII.bsf’ symbol is created, this is a symbol that contains the information for the audio codec interface. The symbol was created adding the vhdl files of the audio codec interface; ‘digital_interface.vhdl’, ‘control_interface.vhdl’, ‘volume_display.vhdl’. The file that links all the components was renamed to ‘audio_codec_for_niosII.vhd’.

The loop back prepared in the ‘digital_interface.vhdl’ file is deleted and two 20-bits registers are prepared to send and receive the signal. The new signals used are:
- The ‘rdy’ signal is ‘1’ for the first 20 bits of data in the left channel and right channel.
- The ‘data’ variable is ‘1’ one clock cycle before the BCLK rising edge.
- The ‘REGO’ signal sends the data to the ‘data_for_niosII’ signal.
- The ‘REGI’ signal receives the data from the ‘dacdat’ signal.

```vhdl
if (rdy='1') then
  if (data='1') then
    data:='0';
    REGO(19-reg)<=DOUT;
    DIN<=REGI(19-reg);
  end if;
end if;
```

`audio_codec_for_niosII` inst | `clk` | `reset_n` | `out_port_from_the_dacdat[19..0]` | `in_port_to_the_data_for_niosII[19..0]` | `in_port_to_the_ready`

Figure 11 – Nios II system top

Figure 12 – Audio codec for Nios II
After this, both symbols are linked and the inputs and outputs pins are prepared. The final ‘niosII_audio_codec.bdf’ is:

![Diagram of niosII_audio_codec.bdf]

After checking the functionality in Modelsim, the project is compiled and downloaded into the board. Then the Nios II interface is ready.

### 4.3 Loop back in the Nios II processor:

The interface is ready and it is possible to program the Nios II processor with C/C++ files now using the Nios II IDE (Integrated Develop Environment).

First, a new C/C++ application of the ‘hello_world.c’ is created, this is an example to check the functionality of the processor and the other components designed. The memory size is not enough to compile this program so it is needed to use the ‘small C library’. This reduces the size of the ‘Hello world’ application from 50kbytes to 4Kbytes (the memory selected is 30 kbytes on-chip memory). The code for this simple file is:

```c
#include <stdio.h>

int main()
{
    printf("Hello from Nios II!\n");
    return 0;
}
```
After building the file, the Jtag and USB devices are prepared to run the program into the processor. The results were correct, the program was downloaded and the console showed the proper message.

After this, the ‘loop_back.c’ file is prepared using ‘small C library’ too. The functions and variables implemented in the header files ‘system.h’ and ‘altera_avalon_pio_regs.h’ are used to get the values of the input signals and assign value to the output signals. With this, the file is prepared to read the value of ‘ready’ every clock cycle, read the value of data from ‘data_for_niosII’ every falling edge of the ready signal and puts it back to ‘dacdat’ every falling edge of the ready signal.

The code is:

```c
#include <stdio.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"

int main()
{
    alt_u32 data;
    int ready;
    int rdy;

    data=0;
    rdy=1;

    while(1)
    {
        ready = IORD.AlterA_AVALONPIO_DATA(READY_BASE);

        if (ready==1)
            {rdy=1;}
        if ((ready==0) & (rdy==1))
            {
                rdy=0;
                data = IORD.AlterA_AVALONPIO_DATA(DATA_FOR_NIOSII_BASE);
                IOWR.AlterA_AVALONPIO_DATA(DACDAT_BASE, data);
            }
    }
    return 0;
}
```

After downloading the C file, the results were as expected. When reset button is pressed the internal loop back of the board is deactivated, the digital signal enters in the Nios II processor and it is directly put back to the audio codec. The volume control of the control interface that change the volume of the line input is still working and, also, the volume display.
4.4 Volume control the Nios II processor:

Next step is modifying the volume with Nios II processor instead that using the control interface. For this, it is needed to modify the Nios II interface and the audio codec interface. The new project name is ‘niosII_volume_contro.qpf’.

The mute, volume high and volume low buttons are connected now to the Nios II processor core so they are included in the SOPC Builder. The value of the signal volume is now managed by the niosII processor too. The ‘niosII_volume_top.ptf’ generated is:

<table>
<thead>
<tr>
<th>Module name</th>
<th>Description</th>
<th>Base</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>niosII</td>
<td>Nios II Processor - Altera Corporation</td>
<td>0x00008000</td>
<td>0x000087FF</td>
</tr>
<tr>
<td>Jtag_uart</td>
<td>JTAG UART</td>
<td>0x00008890</td>
<td>0x00008897</td>
</tr>
<tr>
<td>onchip_memory</td>
<td>On-Chip Memory (RAM or ROM)</td>
<td>0x00000000</td>
<td>0x000077FF</td>
</tr>
<tr>
<td>sys_clk</td>
<td>Interval timer</td>
<td>0x00008800</td>
<td>0x0000881F</td>
</tr>
<tr>
<td>Mute</td>
<td>PIO (Parallel I/O)</td>
<td>0x00008820</td>
<td>0x0000882F</td>
</tr>
<tr>
<td>vol_high</td>
<td>PIO (Parallel I/O)</td>
<td>0x00008830</td>
<td>0x0000883F</td>
</tr>
<tr>
<td>vol_low</td>
<td>PIO (Parallel I/O)</td>
<td>0x00008840</td>
<td>0x0000884F</td>
</tr>
<tr>
<td>Data_for_niosII</td>
<td>PIO (Parallel I/O)</td>
<td>0x00008850</td>
<td>0x0000885F</td>
</tr>
<tr>
<td>Dacdat</td>
<td>PIO (Parallel I/O)</td>
<td>0x00008860</td>
<td>0x0000886F</td>
</tr>
<tr>
<td>Ready</td>
<td>PIO (Parallel I/O)</td>
<td>0x00008870</td>
<td>0x0000887F</td>
</tr>
<tr>
<td>Volume</td>
<td>PIO (Parallel I/O)</td>
<td>0x00008880</td>
<td>0x0000888F</td>
</tr>
</tbody>
</table>

The symbol block is:

```
clk
reset_n

in_port_to_the_data_for_niosII[19..0]
in_port_to_the_mute
in_port_to_the_ready
in_port_to_the_vol_high
in_port_to_the_vol_low

out_port_from_the_dacdat[19..0]
out_port_from_the_volume[4..0]
```

Figure 14 – Nios II volume top
The ‘audio_codec_niosII.bdf’ symbol is changed too. The digital interface is the same but the digital output goes to the Nios II processor and the digital input enters into the audio codec from the processor (after being modified).

The control interface takes care of changing the default values of bypass, power down and data format but not the volume control. The ‘volume_display.vhdl’ receives the volume value from the Nios II processor instead from the control interface.
The interface is ready and it is time to program the Nios II processor with C/C++ files now using the **Nios II IDE** (Integrated Develop Environment). After checking the functionality with a ‘hello_world.c’ application, the ‘volume_control_with_niosII.c’ file is prepared, this file manages the volume and has the same functionality than the ‘loop_back.c’ file when the volume and mute buttons are not pressed.

The program reads ‘data’ value from ‘data_for_niosII’ every falling edge of the ready signal.

```c
data = IORD_ALTERA_AVALON_PIO_DATA(DATA_FOR_NIOSII_BASE);
```

If the volume is not changed by the switches, the loop back is still working.

```c
else
{
    IOWR_ALTERA_AVALON_PIO_DATA(DACDAT_BASE, data);
}
```

When the volume is decreased, the data bits are **shifted to the right** except the signed bit that is the same. If the signed bit is ‘1’, it is needed to fill with ‘1’s the empty spaces in order to get the proper number. When the mute option is activated, all the bits are shifted so the final digital signal is full of ‘0’s.

```c
if (init_vol_low==1)
{
    if (data > 262114) //MSB=1
    {
        data = data >> shift;
        data = data + 262114;
        if (shift>1)
        {
            data = data + 131072;
        }
        if (shift>2)
        {
            data = data + 65536;
        }
    }
    else
    {
        //MSB=0
        data = data >> shift;
        IOWR_ALTERA_AVALON_PIO_DATA(DACDAT_BASE, data);
    }
}
```

---

If’ conditions untill shift>20

---
When the volume is increased, the data bits are shifted to the left except the signed bit. When the original sound is very loud then it is not possible to increase the volume and noise appears.

```c
else if (init_vol_high==1)
{
    if (data > 262114)    //MSB=1
    {
        data = data << shift;
        data = data - 524288;
    }
    if (shift>0)
    {
        if (data < 262114)
        {
            data = data + 262114;
        }
    }
    IOWR_ALTERA_AVALONPIO_DATA(DACDAT_BASE, data);
}
else
{    //MSB=0
    data = data << shift;
    if (shift>0)
    {
        if (data > 131072)
        {
            data = data - 262114;
        }
    }
    IOWR_ALTERA_AVALONPIO_DATA(DACDAT_BASE, data);
}
```

The program works as expected when it is downloaded into the board. When the reset button is pressed the digital signal enters into the Nios II processor, after this, when the volume or mute buttons are pushed the program modify the digital signal to increase or decrease the volume at the speakers.
Chapter 5 . Results and Problems

The results of the simulations are difficult to be shown in a report, actually the best way to do it is listening to the speakers.

After checking the functionality with Modelsim, the synthesis of the vhdl files may not work always as expected. It is very important to know how the file is synthesised in order to solve these problems.

The final simulation check the falling edges of the signal ‘vol_high’, ‘vol_low’ and ‘mute’ that come from the switch buttons every clock cycle. It produces same noise when these three signal are working at the same time due to the Nios II system is not able to manage it. A solution for this problem is use only one variable (short int) that manages the three signals at the same time. This change increases the difficulty of the program in C but it is needed for complex structures.

The printf instruction of the C files also produces a lot of noise if it is used every clock cycle so it is used only for the welcome message and everytime the volume is changed.

There are problems when the volume is increased and the line input signal is very high. In this case the shift procedure produces same noise.

The on-chip memory is easier to implement that the other memories but the capacity is reduced so, for further applications, it is need to use SRAM, SDRAM or FLASH memory.

Despite these problems, the main goal is completed. The Nios II manages a digital signal and interact with the other components of the board.
Chapter 6 . Conclusions

6.1 Conclusions:

The main conclusion for this thesis project are:

- Quartus II software is good a tool to prepare and download vhdl code and it has many possibilities. It is not difficult to prepare very complex designs.

- It is important to write the vhdl code in the right way in order to minimize the size of the design and get the correct results after synthesis.

- Modelsim tool is a great help to simulate the vhdl files but it does not take care about the synthesis issues. Sometime, the code is working in Modelsim but not in the board.

- The on-chip memory is enough to manage reduced applications in C/C++ using the ‘small C library’ but it is need to implement other memory components for complex systems.

- The Nios II processor is able to receive, manage and send digital signals to other components of the board and the interface is not difficult to implement with SOPC Builder tool.

6.2 Future work:

There are many possibilities to continue this thesis project:

- When The Nios II interface was built, the audio codec interface was prepared as a different symbol block separated from the system top block generated with the SOPC builder. It will be an interesting work to prepare the audio codec interface as another component of the SOPC builder in order to get only one symbol for the complete system. This may be implemented using the component editor of the SOPC builder tool.

- The digital signals that sends and receive the Nios II processor from the audio codec are 20-bits vectors but the word length of the data is 24 bits. It is needed to implement these signals as 24-bits in order to get more accuracy. It will be needed to change the timing of the system when these signals are changed.
References

The webpage of Altera has been visited many times. There are many documents to understand the behaviour of the board and its components: http://www.altera.com

Documents related to the board and components:

- DE2_User_manual.pdf ------ In the CD of the DE2 package
- WM8731_WM8731L.pdf ------ In the CD of the DE2 package

Bibliography:


Figures:

- [1] – Figure 2.1 in DE2_User_manual.pdf
- [2] – Figure 2.2 in DE2_User_manual.pdf
- [3] – Figure 5.11 in DE2_User_manual.pdf
- [5] – Figure 26 in WM8731_WM8731L.pdf
- [6] – Figure 4.10 in DE2_User_manual.pdf
- [7] – Figure 32 in WM8731_WM8731L.pdf
- [8] – Figure 33 in WM8731_WM8731L.pdf
- [9] – Table 29 in WM8731_WM8731L.pdf
- [10] – Figure 2.1 in n2cpu_nii5v1.pdf
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår. Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art. Upphovsmannens ideella rätt innefattar rätt att bli nämd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances. The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© José Ignacio Mateos Albiach
Interfacing a processor core in FPGA to an audio system

José Ignacio Mateos Albiach

The thesis project consists on developing an interface for a Nios II processor integrated in a board of Altera (UP3-2C35F672C6 Cyclone II). The main goal is show how the Nios II processor can interact with the other components of the board.

The Quartus II software has been used to create vhdl code of the interfaces, compile it and download it into the board. The Nios II IDE tool is used to build the C/C++ files and download them into the processor.

It has been prepared an application for the audio codec integrated in the board (Wolfson WM8731 24-bit sigma-delta audio CODEC). The line input of the audio codec receives an analog signal from a laptop, this signal is managed by the control interface of the audio codec. The converters ADCs and DACs are stereo 24-bit sigma delta and they are used with oversampling digital interpolation and decimation filters.

The digital interface of the audio codec sends the digital signal to the Nios II processor and receives the data from the processor. After building the interfaces for the audio codec and the processor, it has been prepared an application in C++ language for the processor that modifies the volume of the signal. The signal come back to the audio codec and it is possible to check the results with headphones or speakers at the line output of the audio codec.

Keywords
Nios, audio CODEC, Quartus, SOPC bulider, IDE