Design, Test, and Integration Plan

Initial Report

MSc in Digital Systems Engineering

Department of Electronics

University of York

Group Project

Contents

Table of Figures

1 Introduction

1.1 Overview – Matt Reynolds

This document focusses on the design, test, and integration plan for the BitVia media player. The document is split into four main sections: high level design, low level design, test strategy, and integration plan. Each team member has contributed to each of these sections, detailing the components of the product for which they are working on. These components are relevant to their role within the company and have been clearly marked using the authors name in the associated headings.

High level design consists of the black box architecture of the product. Here a product overview is written, followed by more detailed subsections of the design. This is followed by the low level section, which expands upon the high level design by detailing the functioning of each of the system blocks. This should allow the reader to develop a clear understanding of the project's design. At this stage of the project, implementation has not been carried out, as part of the V-Model Process. This is discussed in more detail below.

As each stage of design is undertaken, relevant test strategies are being developed. These are to ensure that our product meets the design specification and therefore, the customer's requirements. These test strategies are explained in the test strategy section and relate to the two design sections already mentioned. This leads into the integration plan, where a step-by-step process is written, explaining how each subsection will be integrated.

Our product is aimed at making the music industry more accessible to amateur artists, by providing a low-cost recording platform with high-fidelity effects and audio processing. Therefore, FLAC audio compression has been used, with a 24-bit depth per sample. The effects are hardware accelerated to preserve quality, optimise performance, and reduce power consumption. Users are able to record over existing tracks by playing them from a microSD card, which plugs into the side of the device.

Live recordings can be saved to the microSD card as well. If a backing track is being played, the saved file will capture both of these. Audio will continue to be output even when not recording, allowing the user to use the device for effects and practice. All effects can be enabled and disabled using the pushbuttons interface. When disabled, the audio passes through the effect unprocessed.

All of the effects have adjustable parameters, the values for which are displayed on a 4.0" display. This features a simple text-menu interface, which should feel familiar to the user. Explanations on how to use the device are contained in the user manual, included in this document.

1.2 Project management – Matt Reynolds

1.2.1 V-Model – Matt Reynolds

Our design and testing process follows the V-Model, as explained in the Quality Assurance Manual. [1] An important aspect of this involves completing design stages, prior to implementation or testing. However, a test should be developed at each design stage, which will then be carried out following implementation. Figure 1 shows this process, graphically. [2]

Our project is currently at the Module Design stage, preparing to transition to the Coding stage. As depicted by Figure 1, this can be an iterative process. If at a stage of testing the device under test fails, the arrows should be followed back round to the relevant design stage. Working round the diagram in this way is intended to ensure successful delivery of a functioning product.

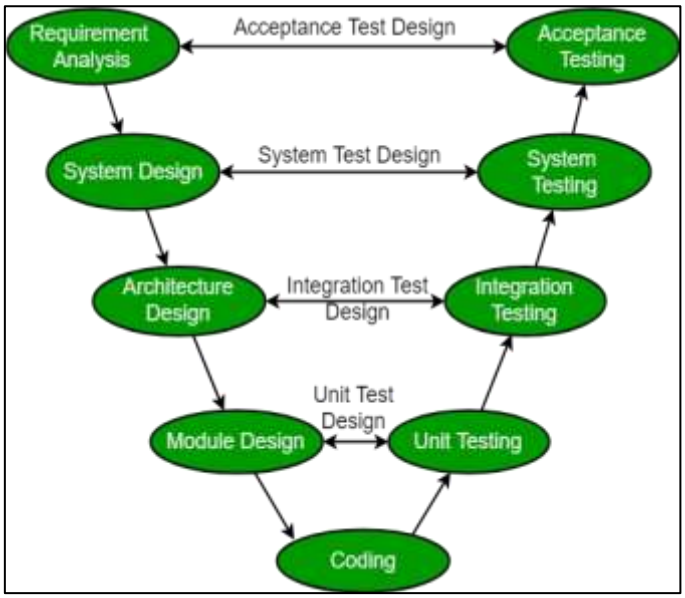

Figure 1 - V-Model Process

1.2.2 Gantt Chart – Matt Reynolds

The project has been scheduled around the V-Model process. As can be seen from Figure 2, we are now at the coding stage, which shows we are running according to schedule.

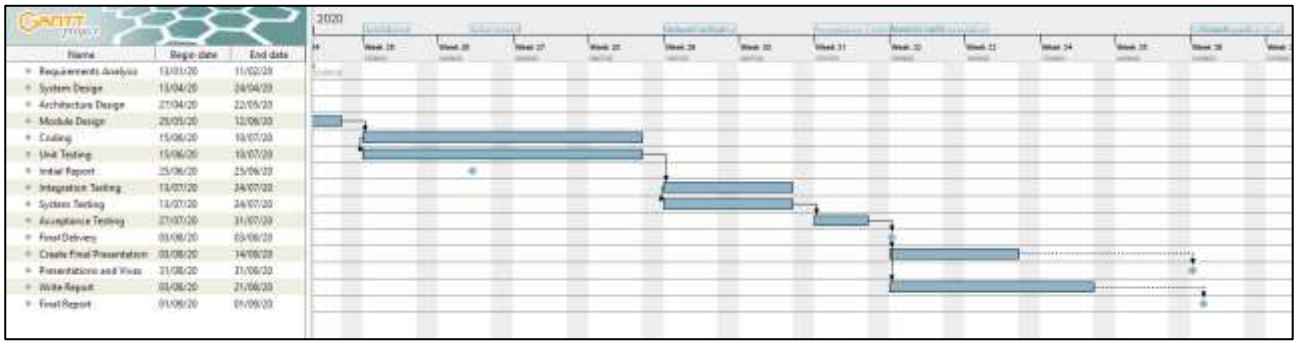

Figure 2 - Gantt Chart Overview

Assignment submissions and deadlines have been recorded as milestones. Certain stages have dependencies, especially in the testing stages. However, the project has been scheduled such that tasks can run in parallel. As our test manager is going to focus largely on testing, this role will become almost full time, allowing him to take on tests while development occurs.

This is necessary as the project has a compact timeline and coding and testing are time consuming processes. In doing this, we are still able to follow the V-Model, by completing a design, code, and test triangle before moving to the next layer; testing will occur from lowest to highest level. The only deviation from this is that integration and system testing will occur in parallel, where possible, to increase project velocity.

A small amount of buffer has been built into the schedule to accommodate unforeseen setbacks. These can often occur during the testing stages, when components which have already been developed, do not pass the tests to meet the specifications. By planning time into the schedule for such occurrences, it helps to mitigate the risk of a late delivery.

2 High Level Design

2.1 Product Overview – Matt Reynolds

2.1.1 Architecture – Matt Reynolds

The BitVia media player uses the built-in flash on the XC7Z020 SoC to store the boot files and bitstream. As part of the boot procedures, the bitstream is written to the programmable logic (PL) side of the SoC and a system reset is enabled by the processor system (PS) to initialise the hardware.

A microSD card is used as the main storage device. Here the user can save recordings and play already existing files. The storage system is accessed using the Xilffs APIs, which are implemented as bare metal functions in C. These will be encapsulated in a FreeRTOS task, allowing the kernel to perform storage operations in parallel with the rest of the system.

A Hyperpixel 4.0” touchscreen display is used to display text-based menus. The user can interact with these menus through the use of the pushbuttons and switches. An AXI GPIO IP passes the pushbuttons and switches values to the PS. An IRQ is placed each time a value is updated on this array. The interrupt handler updates the required values. Should any changes be made to the value of the effects chain, the FX values custom IP is written to. This then writes the stored values to the relevant FX entities in the PL. (Figure 3)

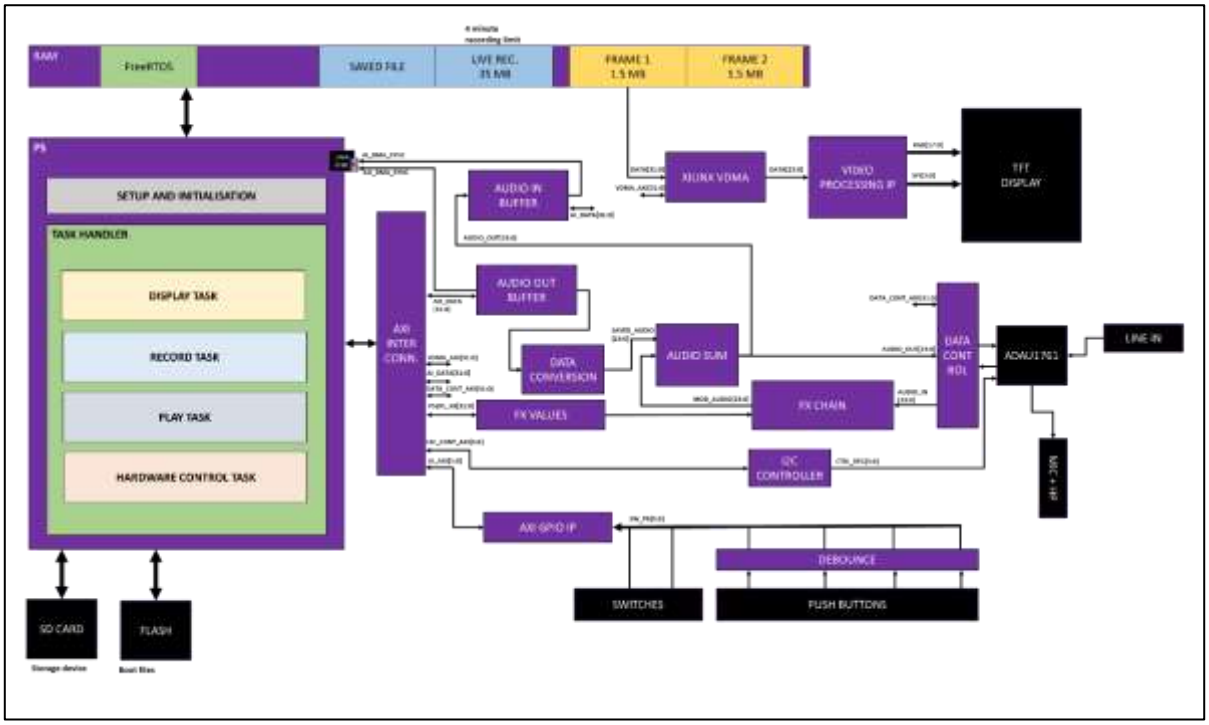

Figure 3 - Architecture Overview

A Xilinx soft-core Video Direct Memory Accessing IP (VDMA) is used to write data to the display. The VDMA uses a memory-mapped to stream (MM2S) interface, which allows memory-mapped data in the PS to be streamed to the PL. This is controlled by an AXI4-Lite interface. To prevent incorrect data read-out, a frame buffer is used. Two frames are stored in RAM, the PS writes to a frame, while the VDMA reads from the other. Interrupts are generated by user inputs, which run the VDMA, updating the frame buffers with new data, which will be streamed onto the display; preventing polling.

The analogue to digital converter (ADC) and digital to analogue converter (DAC) are located in the ADAU1761 IC. This audio codec is configured with a sampling frequency of 48 kHz; the rate at which audio input is sampled, and audio output is generated. The user may plug an instrument into the line in 3.5mm socket. Audio is converted and passed to the data control.

Data control deserialises the data and passes it to the effects chain. Here, digital signal processing (DSP) circuits manipulate the audio to create new waveforms. The effects are controlled by the user and updates are received from the PS via a custom AXI4-Lite interface peripheral. The audio output from the effects chain is passed through an audio summing block, where saved files may be mixed in if desired.

The resulting audio is stored in the on-board RAM via a hard-core DMA transaction. A buffer is used to store samples temporarily on the PL side, to make more efficient use of the AXI bus when transferring data from the PL to the RAM. A similar design is used for outputting audio, with the flow of data in the opposite direction. Both data transfer designs utilise the burst-write feature available on the full AXI interface.

Following the sequential initialisation to configure the product at startup, the software will enter the FreeRTOS task handler. Here, four main tasks will be called, which will run in parallel through the use of context switching. The record task will handle data transactions to the RAM and post-processing of the raw data into the FLAC format.

The play task handles data retrieval from the microSD into the RAM. It also manages decoding of the FLAC into RAW samples ready for the PL processing and audio output stream. A hardware control task will handle user inputs and ensure the PL is updated accordingly. Writing effects values, reading pushbutton and switches, configuring the ADAU1761, are all functions performed within the hardware control task.

Finally, there is the display task. Here, the functions required to ensure the menus are displayed correctly and are updated at the necessary times, are located. This task will interface with the VDMA, as it will be writing to the frame buffers.

2.1.2 Boot procedures – Matt Reynolds

Figure 4 shows a basic overview of the procedures the devices follows from power on to ready. Once the device is ready, the user can access the menus and begin to play audio through the system.

Reset and PL initialisation is controlled by the PS and occurs at the FSBL executed stage. Each time the device is powered off, user-saved settings will be deleted.

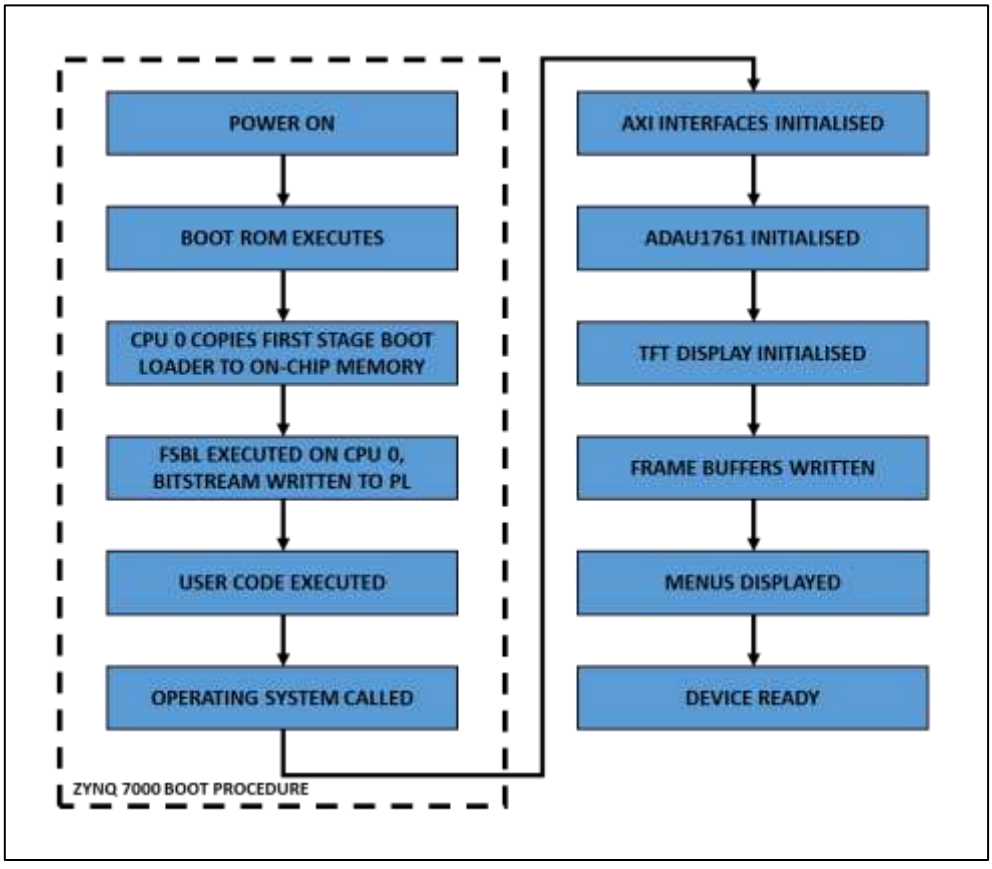

Figure 4 - Boot Procedure Flow Diagram

2.2 TFT Display – Matt Reynolds

The TFT Display is a Hyperpixel Pimoroni 4.0” Touchscreen LCD, which will be used as the front-end of the main user-interface. Here the user will be able to see the device menus, which allow them to edit the effects’ parameters and select the files they wish to play.

The drivers for the TFT display will be formed mostly from Xilinx soft-core IPs. Configuration of the display will be carried out using an AXI GPIO IP which will control the SPI interface on the display. A series of 16-bit values are written to the display, which initialise the pixels and prepare the RGB lines to receive data.

Menu data will be stored on the internal flash and accessed via the PS. User generated interrupts will update the data in the frame buffers, stored in RAM. (Figure 5) The VDMA will continuously read from the frame buffers, updating the display at the rate of 60 frames per second. This is required as the TFT display needs refreshing to maintain an image.

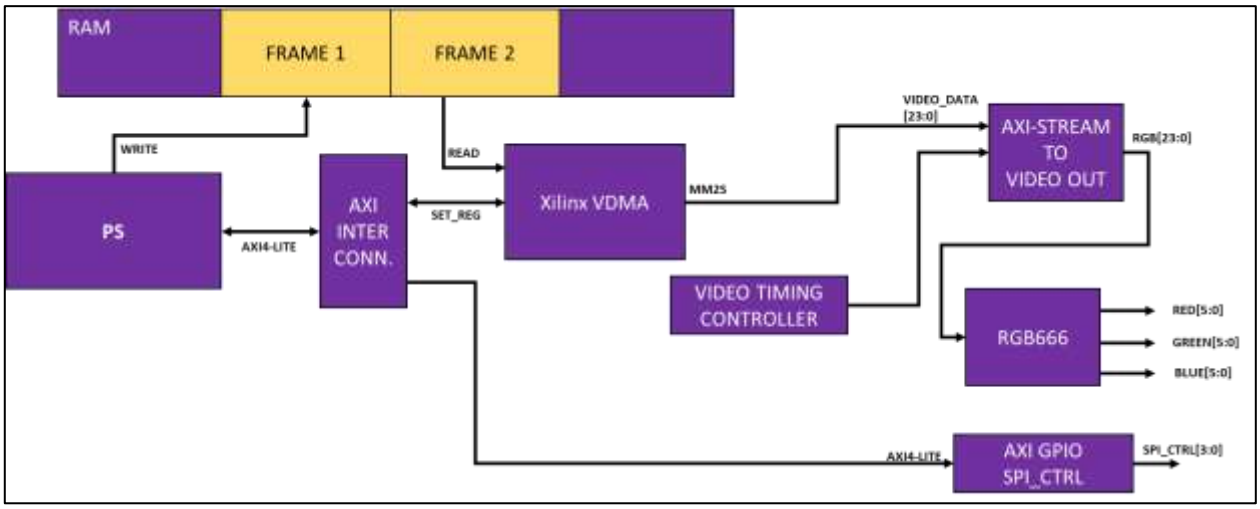

Figure 5 - TFT Display Hardware Overview

The PS will write to one frame buffer, while the VDMA reads from the other. The VDMA will output the image data using the MM2S (memory-mapped to stream) interface and pass this to the AXI-Stream to Video Out IP. From here, the data is passed to the RGB output and parallelised onto the RGB pins of the display.

The Video Timing Controller IP sends the required control signal data to the Video Out IP. This controls the timings for when data is read in from the AXI-Stream, as well as the DPI enable pin and vsync and hysnc signals. Thus ensuring the correct frame rate and frame alignment is maintained.

2.3 HW Block PL – Simone Ledda

In the following section the reader is given an overview of the audio path, the resources involved in its realization such as IP blocks designed discrete IC such as the audio codec. The board is equipped with the ADAU1761 audio codec, this integrated circuit is highly customizable and accessible. The codec is the entry and exit point of every piece of audio in this application.

The codec is linked to the FPGA via I2C bus and I2S bus; the Zynq chip is also feeding it the clock signal and power rails.

In Figure 6 in purple is shown the intended design for the audio datapath on the programmable logic side of the FPGA, while in black there’s the codec.

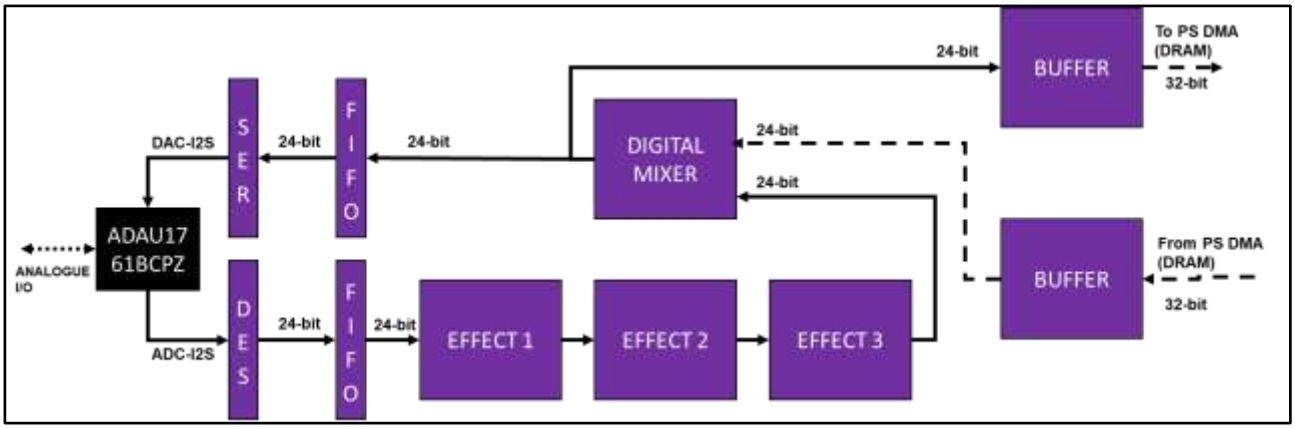

Figure 6 - Complete Datapath

The datapath takes into account the crossing of different clock domains, for example between the codec and the PL with two FIFOs banks, later on named RXFIFO and TXFIFO, as well as the crossing between PL and PS for data saving and playback.

As mentioned in the specifications, our device has a strict timing requirement to meet: audio signal must get in and out of the FPGA within 15ms. During that time window we have to condition the signal into the effect bank according to user's effect settings, eventually save it or ad a base to it, and finally serialize it for the codec to drive the headphones.

The following Figure 7 shows the Vivado block diagram with the IPs necessary for I2C communication with the audio codec. These block weren't shown in Figure 6 because they are part of the control block, omitted for readability purposes from the datapath. The SERDES interface is not present in the diagram as well since it's currently under development.

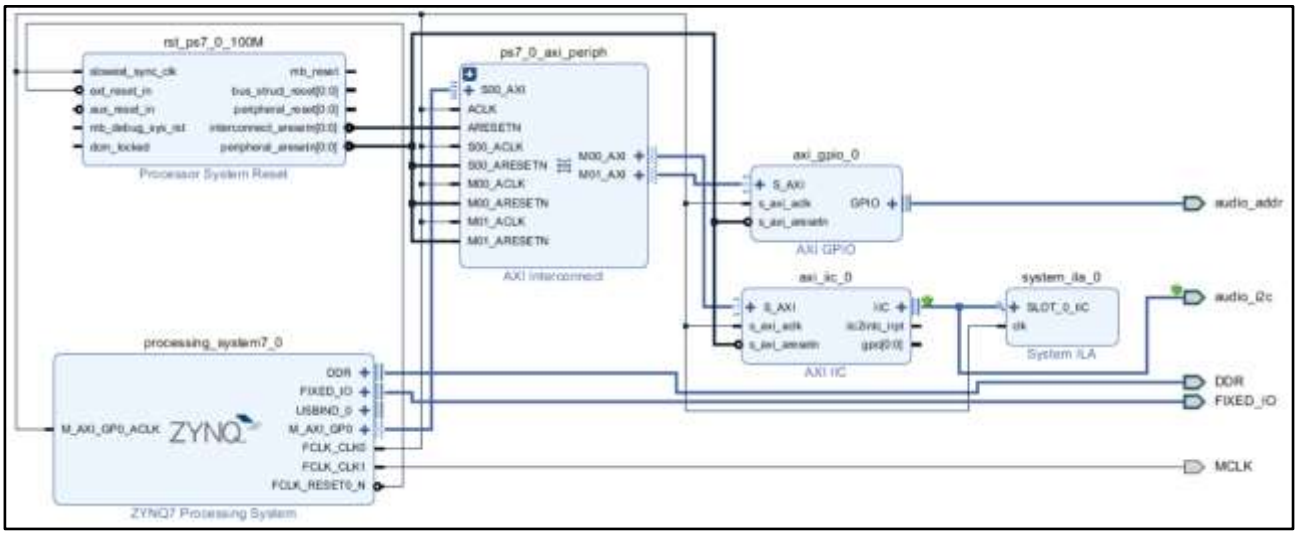

Figure 7 - Vivado I2C interface block diagram

It was chosen to adopt the default I2C interface over the SPI for the controls communication mainly because the Pynq board has pull-up resistors already in place on the SDA and SCL lines.

The AXI GPIO is used to set ADDR0 and ADDR1 bits of the audio codec chip address to 0s. This was an arbitrary choice as the I2C bus is a private bus, so no other chip is present on it apart from the codec. These two address bytes are used to differentiate the chip address so that it can be unique.

ADAU1761 I²C Address and Read/$\overline{\text{Write}}$ Byte Format

Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
0	1	1	1	0	ADDR1	ADDR0	R/$\overline{\text{W}}$

Figure 8 - ADAU1761 I2C chip address [3]

The I2C is working at 100kHz at the moment, further tests will take place to see if it's possible to push it to fast mode at 400kHz. Although audio data transfer will not benefit by this speed improvement, because they are travelling on different lines, 400kHz may reduce the initialization time at start up and could be used in high performance power mode. No higher frequency than 400kHz can be tested because the hardware block only shows these two as a valid option.

2.4 Pynq board audio overview – Simone Ledda

To fulfill the purpose of this project, we decided to rely on the Pynq-Z2 [4] board as it's versatile and equipped with all the necessary components needed.

Figure 9 shows how the audio codec and the audio filtering stage is layed out to the jacks. In black is shown the input path and in orange the output path.

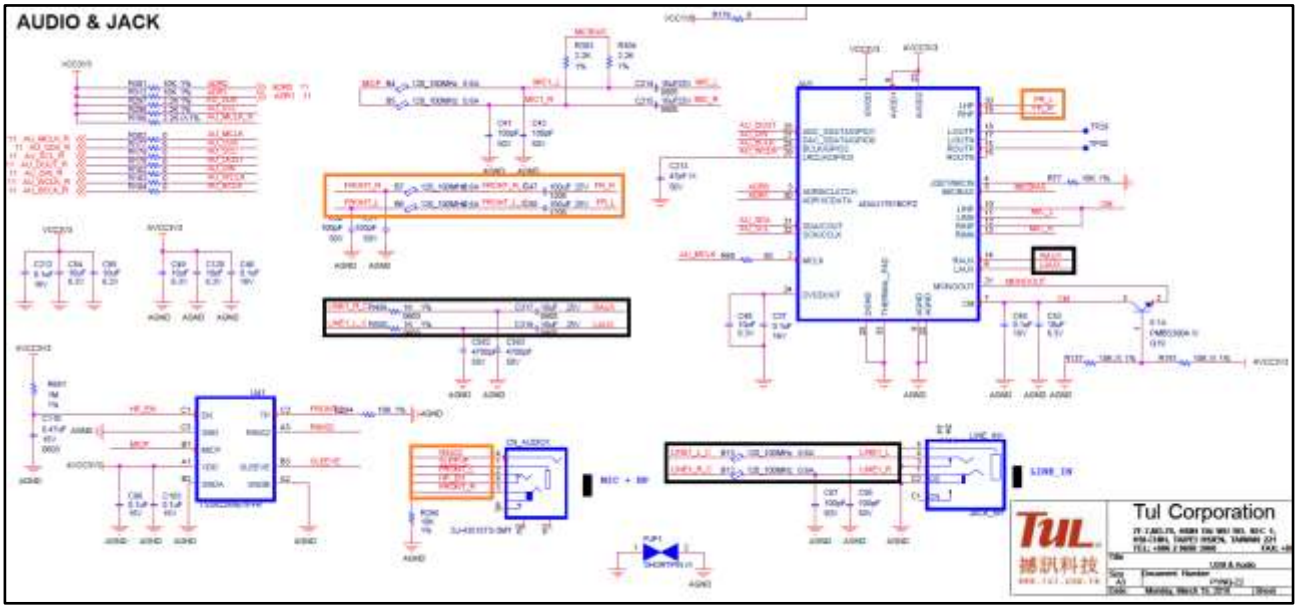

Figure 9 - PYNQ Audio & jack schematic [4]

The board features two 3.5 mm jacks labelled LINE IN and HP+MIC. It was chosen to use LINE IN as input and HP+MIC as output. LINE IN pinout connects left, right and ground lines, HP+MIC is used for its headphone output but the eventuality of using it as an input in case the user is a singer and would like to use a single headset to play is currently under evaluation and a possible stretch goal for this application.

2.5 Audio Effects – Matt Reynolds

The audio effects for this media player are hardware accelerated algorithms, designed to create popular audio manipulations used within the music industry. The project scope currently includes three effects: distortion, tremolo, delay. The effects have been implemented in the PL to reduce load on the CPU (ARM Cortex-A9) and meet the time delay constraints of the project. To ensure that the live audio recording process appears to happen in real-time, the audio input-output delay needs to be less than 15ms.

The effects will be chained together (Figure 10) and passed an enable signal, controlled by the UI, allowing the user to turn individual effects on and off. The incoming signal will pass straight through effects which have been switched off, without being modified. This mimics the true-bypass effects chain used by guitarists.

The order in which the effects are chained follows the conventional pattern used by most guitarists. [5] It is important to clarify that although the product is designed in a way which is familiar to guitarists, this is not the scope of the intended audience. Our product is aimed at all musicians who wish to record live instrumentation with the use of digital effects.

Figure 10 - Effects Chain

Distortion, tremolo, and delay were chosen as they are all popular within the music industry across many genres, helping our product to maintain versatility. They are also tried and tested in digital systems with positive outcomes. [6]

2.6 Memory Access – Gavin Johnston

Processor-centric memory access is a slow and processor intensive operation, requiring the processor to be blocked for large periods of time, in polling I/O, or have another task regularly interrupted, in interrupt I/O. Both are inefficient methods of transferring large blocks of data and cause delays for other pending tasks. [7]

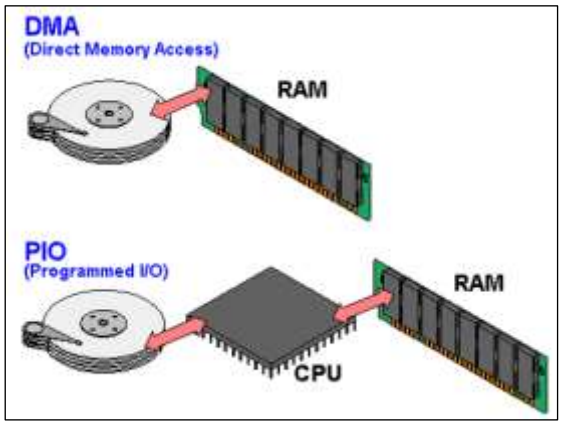

Figure 11 DMA vs Programmed I/O [8]

Direct Memory Access (DMA) is the process of accessing system memory without oversight from the processor, this is done with the DMA Controller (DMAC) which can control the data bus(es) and transfer data to and from memory with minimal interaction from the processor. This reduces the processor time spent on the transaction and avoids delays in the transaction, where the transaction must wait on the processor, i.e. context switching. Note, DMACs are commonly referred to as DMAs in Xilinx's tools so from hence forth; DMACs will be referred to as DMAs to match the toolset.

When designing a system on the Zynq SoC there are three main options to choose from, the hardcore DMA residing in the Processing System, PS, (henceforth, PS DMA), one of three general purpose softcore AXI DMAs which can be instanced in the Programmable Logic (PL) and CPU-centred memory access.

The PS DMA can process memory mapped to memory mapped requests between system memories and PL peripherals. It runs on the CPU_2x clock, which is the same clock which drives the PS

interconnects. Note, this does not mean the DMA runs twice as fast as the CPUs, as the CPUs run on the CPU_6x or CPU_4x clock depending on settings. [9]

The Zynq 7020 chip has three, general-use, soft DMA IPs available for instance. They are listed below, note that other DMA soft IPs exist, but are as part of specific subsystems:

- AXI Direct Memory Access (DMA)
- AXI Central Direct Memory Access (CDMA)
- AXI Multi-Channel Direct Memory Access (MCDMA)

The softcore DMAs can be thought of as the DMA IP and its variations. The DMA IP can handle transactions between a AXI-memory-mapped location and an AXI-streaming peripheral; CDMA can handle transactions between two AXI memory-mapped locations and MCDMA can handle transactions between several memory-mapped locations and AXI-streaming peripherals, in parallel, using multiple channels [2]. While the DMA IP does support multiple channels, it is more resource efficient to use MCDMA for multiple channels, AXI DMA's multi-channel options remain as the IP is the basis of other IPs and reference designs. [3]

CPU-centred memory access is most suitable when small volumes of data require transfer and infrequently. Our design produces small volumes of data, but does so relatively frequently, therefore it may be more suitable to store data in a buffer and transfer a larger quantity at once with a DMA to avoid the CPU overhead of memory access and the frequent blocking of the PS interconnects.

We have decided on using the PS DMA for this project as it will use fewer PL resources, reduce CPU overhead and the blocking of the interconnects. The PS DMA will be used to transfer blocks of the decoded backing track from the RAM to a PL buffer and the combined audio data from a PL buffer to RAM. The DMA will be programmed in Xilinx SDK using the DMAPS driver and will be triggered by the PL using the Peripheral Request Interfaces. We will buffer data on the PL side in blocks to avoid transferring single elements, which may be detrimental to performance.

2.7 Media Storage – Gavin Johnston

In our design, we require a media storage device to record and playback audio files from, we had two main options when selecting a media device; Universal Serial Bus (USB) or Secure Digital (SD).

Both USB and SD are natively supported by the MIO pins on the Zynq SoC and in the Xilinx SDK environment. They are both used and supported in many other devices such as laptops, however USB ports are more prevalent than SD ports.

USB has a larger form factor and would increase the overall footprint of the board drastically if a USB flash-memory drive was connected to the edge of the board, a large drive increases the risk of the user hitting the board and damaging the board or the USB port. Furthermore, rocking the USB drive back and forth, a common, damaging method of removing USB devices, increases the risk of damaging the port or board.

We selected the SD card format for use as mass storage, due to its smaller form factor and lower risk of damage with its spring-loaded port. It is noted that SD-to-USB conversion devices are available to buy from third party vendors, so users who do not have a SD port on their computer can still access the media disk.

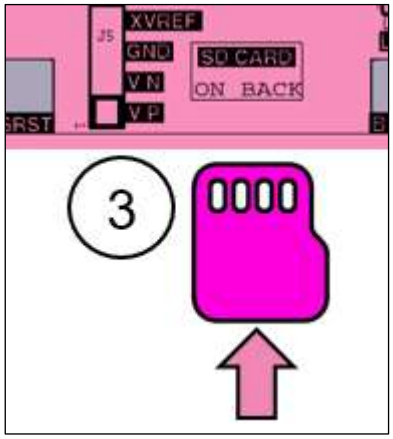

Figure 12 Pynq Board SD Card [4]

The SD card will need a file system to manage creation, reading and modification of files. Three file systems are common today; the New Technology File System (NTFS), the File Allocation Table (FAT) file system and the extended File Allocation Table (exFAT) file system.

NTFS is a proprietary file system licenced by Microsoft and is the default file system for all Windows machines. It contains many features such as file permission control, drive encryption and disk-level file compression. While it is a feature-rich file system, it is proprietary and not very compatible outside of windows machines. [10]

FAT32 is a free, widely supported file system with many USB drives and SD cards being sold pre-formatted with the file system; it is natively supported in Xilinx SDK. It is limited by size, as a single file cannot be larger than 4GB and a partition cannot be larger than 8TB. [10] It does not have the any additional features such as encryption or compression.

exFAT is an extension of the FAT file system standard, which effectively removes the file and partition size limits, it does not have any of the additional features present in NTFS, much like FAT32. In terms of compatibility it lies between FAT32 and NTFS, being fully compatible with more devices than NTFS but not supported by older devices such as Xbox 360 and PlayStation 3 gaming consoles. [10]

We chose FAT32 for our file system. FAT32 is compatible with most devices, is free and is natively supported in SDK. Furthermore, the library which supports FAT32 in Xilinx SDK, Xilffs, defaults to using the SD card, reducing development time. It does not contain as many features as NTFS; however, these features are superfluous for our project. The file and partition sizes limits are negligible, as audio files are relatively small; we are not expecting to reach either of these limits.

2.8 File Format – Gavin Johnston

One requirement of the project requires the use of a compressed audio file format. Two file formats were considered; the Moving Picture Experts Group, MPEG, Audio Layer-3 (MP3) and the Free Lossless audio Codec (FLaC).

MP3 is a widely used audio file format, it is free and features a high rate of compression. It uses lossy compression to reduce file sizes considerably. We were warned by our supervisor that implementation of MP3 is complicated and may be difficult to achieve in the timescale.

FLaC is not often used as an audio file format but is widely supported. FLaC is free and lossless. Lossless compression means FLaC will not have as high a compression rate as MP3, and therefore will create larger files; however, having the data which would otherwise be deleted by MP3, may be important to a user looking to adjust the audio later. FLaC is considered to be easier to implement than MP3 and the C source code is provided by the developers with instructions for embedded developers on how to remove unnecessary components.

We chose FLaC as it is free, lossless and less complex to implement than MP3. It also contains instructions for embedded developers and has a well-documented API.

2.9 User Interface Overview – Matt Reynolds

The users will be able to interact with the media player through a series of text-based menus. Figure 13 provides an overview of this menu system. The FX Groups allow user to save presets of the three effects and their parameters. At the push of a single button, all of these presets will be applied. The intended use case for these is for the user to setup and customise these effects banks prior to recording. Then during live recording, they can easily switch between them using the pushbuttons.

The Effects menu, takes the user directly to the three effects. They can then use Bank 1 of the pushbuttons to switch these effects on and off, in the same way that would be done with stomp-boxes. The backing tracks will typically be files already saved on the microSD card in the FLAC format. However, the user will also be able to play back their live recordings once they have been saved. This will allow them to create layers on the track and record fully-developed songs.

To support the Chinese language, Pinyin will be used. This to reduce project scope by avoiding the need to develop extended character sets for the font. For the benefit of the reader, Pinyin is the phonetic spelling of Chinese words using the English alphabet.

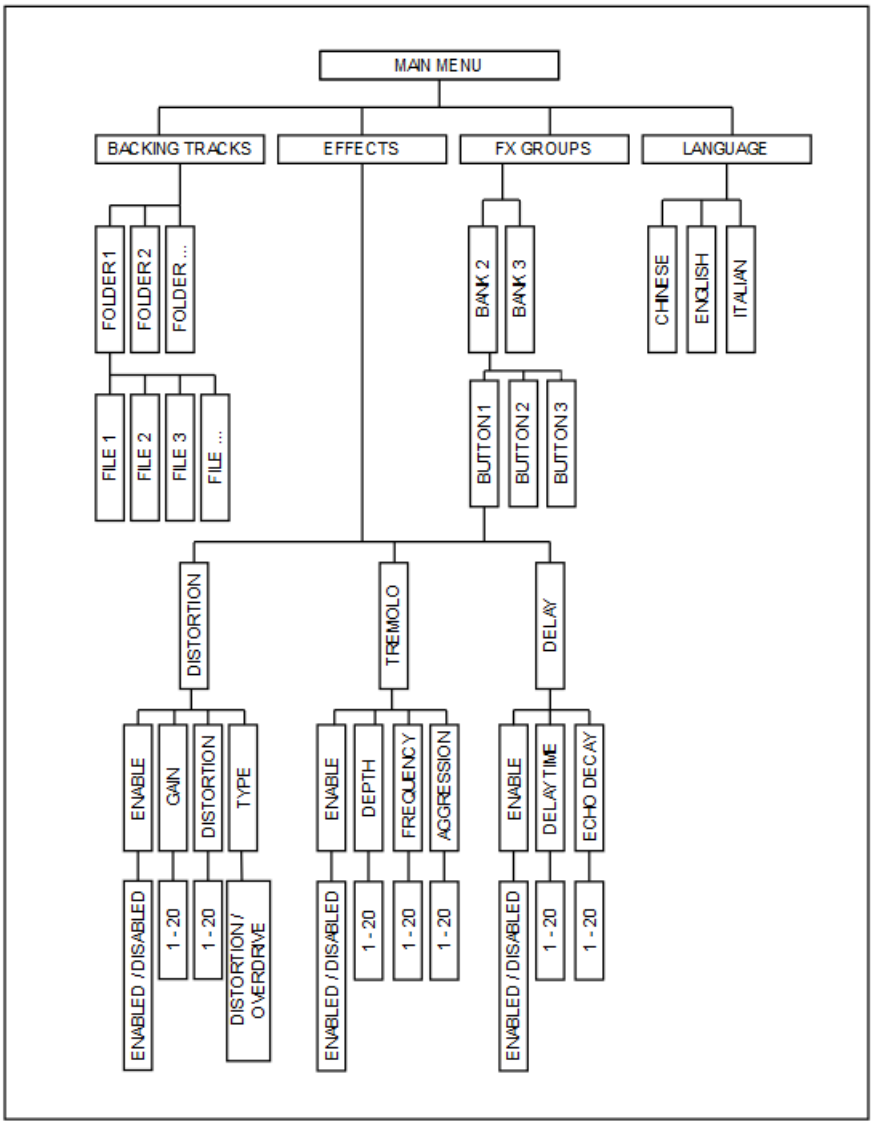

Figure 13 - User Interface Tree Diagram

2.10 User Interface Functionality - Gavin Johnston

Figure 14 shows which functions the user can call when they interact with the menu, the functions in each box represent which functions are available at that level.

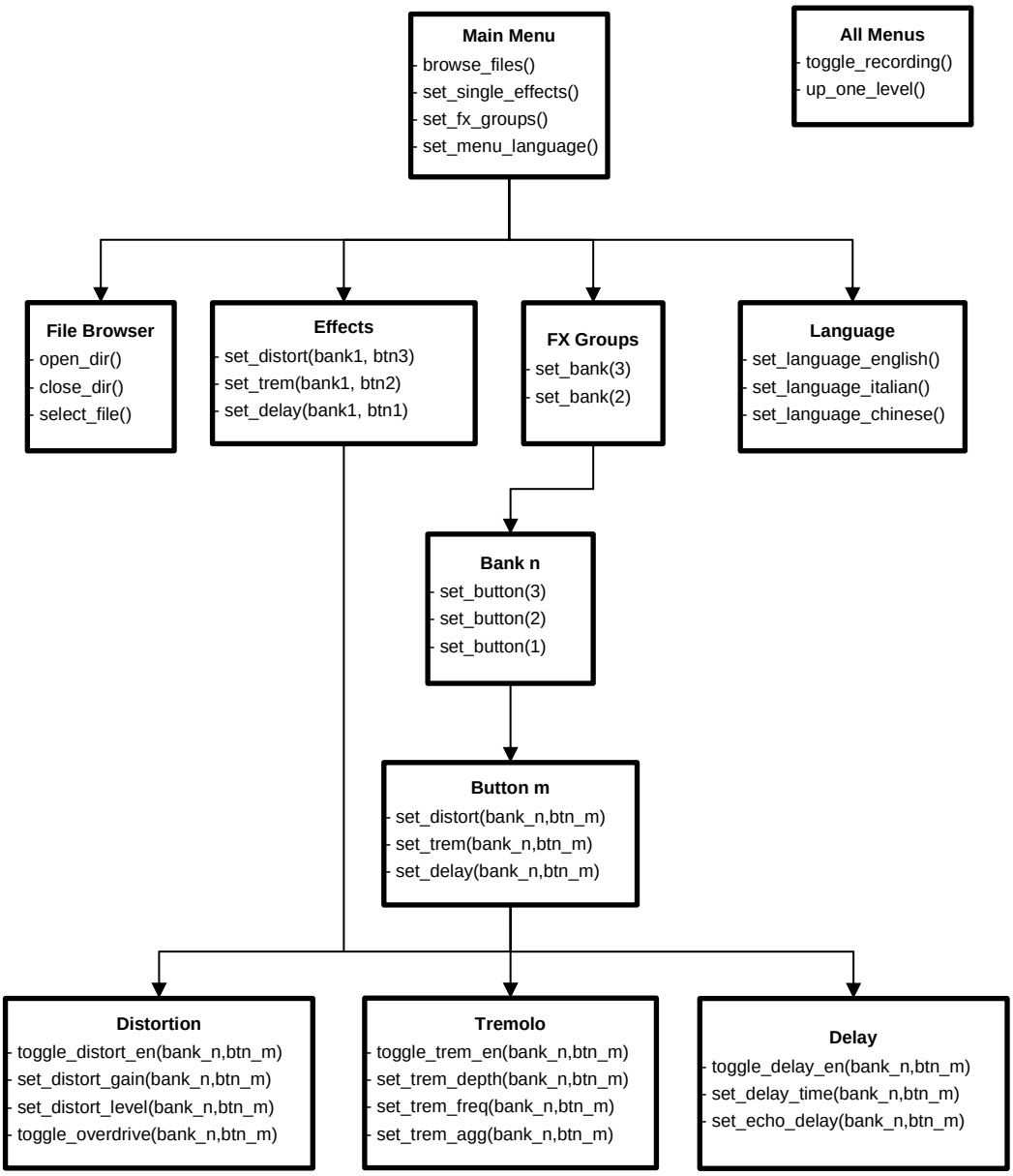

Figure 14 User Interface Functions

All Menus

These functions are available at every level of the menu tree.

- toggle_recording() – start/stop recording
- up_one_level() – return up the menu tree one level (not available at top level)

Main Menu

- browse_files() – go to “File Browser” sub-menu
- set_single_effects() – go to “Single Effects” sub-menu

- set_fx_groups() – go to “Effects Groups” sub-menu
- set_menu_language() – go to “Language” sub-menu

File Browser

- open_dir() – open a folder
- close_dir() – close a folder
- select_file() – select a file as a backing track (output file generated automatically)

Language

- set_language_english() – set menu language to English
- set_language_italian() – set menu language to Italian
- set_language_chinese() – set menu language to Chinese

Single Effects

Facilitates the configuration of bank 1 (stomp-box) push buttons. The three functions are explained in the “button m” menu. Each button on bank only has one effect associated with it, similar to a stomp-box.

Effects Groups

Facilitates the configuration of bank 2 and 3 (grouped effects) buttons.

- set_bank(3) – stores “bank_3” as a parameter
- set_bank(2) – stores “bank_2” as a parameter

Bank n

Where n is the bank value selected in the “Effects Groups” menu

- set_button(3) – stores “button_3” as a parameter
- set_button(2) – stores “button_2” as a parameter
- set_button(1) – stores “button_1” as a parameter

Button m

Where m is the button value selected in the “Bank n” menu

- set_distort(bank_n,btn_m) – sets the distortion parameters for the bank and button selected
- set_trem(bank_n,btn_m) – sets the tremolo parameters for the bank and button selected
- set_delay(bank_n,btn_m) – sets the delay parameters for the bank and button selected

Distortion

- toggle_distort_en(bank_n,btn_m) – toggle the distortion effect
- set_distort_gain(bank_n,btn_m) – set distortion effect gain
- set_distort_level(bank_n,btn_m) – set distortion effect level
- toggle_overdrive(bank_n,btn_m) – toggle overdrive effect

Tremolo

- toggle_trem_en(bank_n,btn_m) – toggle the tremolo effect
- set_trem_depth(bank_n,btn_m) – set tremolo depth
- set_trem_freq(bank_n,btn_m) – set tremolo frequency
- set_trem_agg(bank_n,btn_m) – set tremolo aggression

Delay

- toggle_delay_en(bank_n,btn_m) – toggle delay effect

- set_delay_time(bank_n,btn_m) – set delay time
- set_echo_delay(bank_n,btn_m) – set echo delay

2.11 User Interface Graphics – Pengan Wang

User interface is very important for an electronic device. A clear user interface enables users to operate the device faster and more conveniently.

BitVia Media Player's user interface design is shown below:

When users open BitVia Media Player, they will see the MAIN MENU:

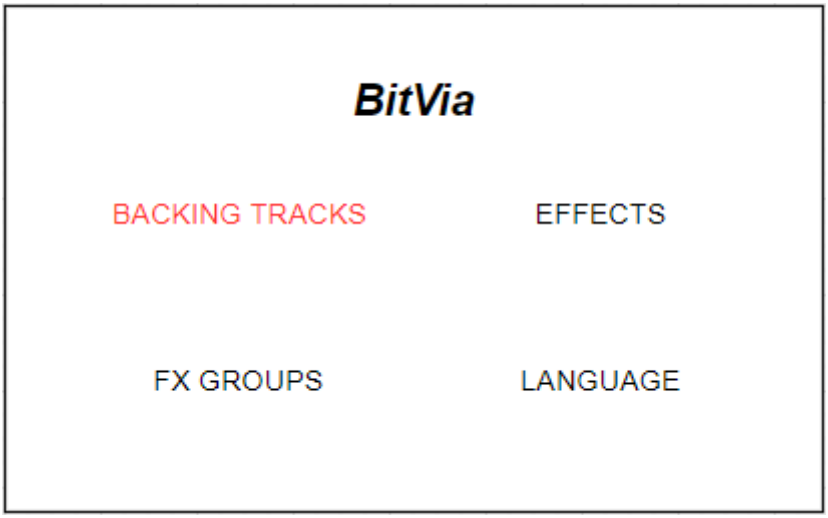

Figure 15 - Main Menu

When users choose the "BACKING TRACKS", the words will show in red, after pressing the OK button, the user will enter into the BACKING TRACKS user interface:

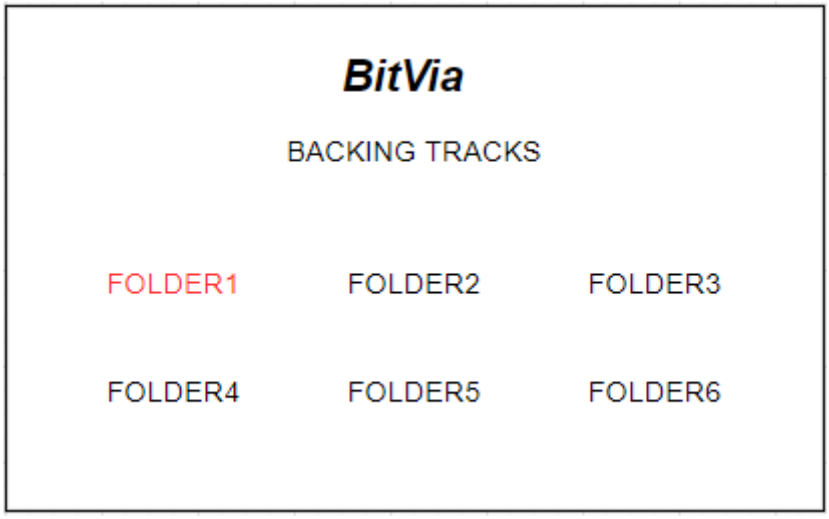

Figure 16 - Folder Sub-menu

When users choose the FOLDER1 and enter the next level user interface, the user interface will shown like:

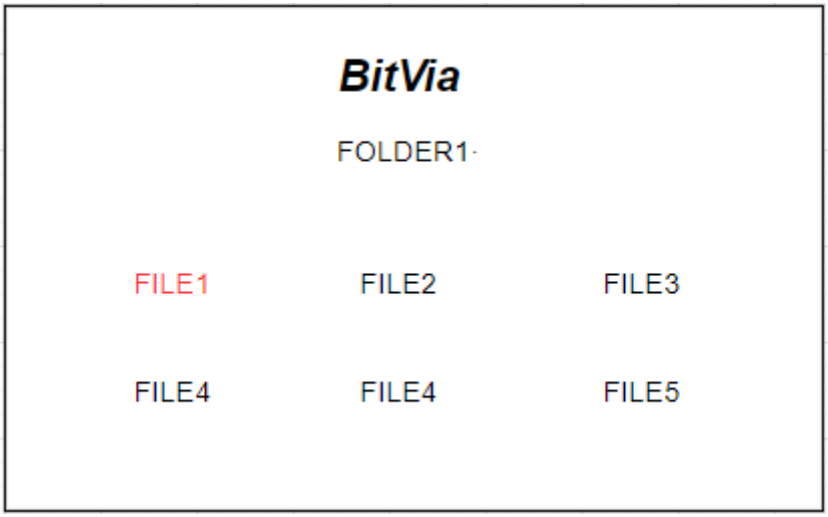

Figure 17 - Files Sub-menu

When users choose the “EFFECTS” in the MAIN MSNU, the word “EFFECTS” will show in red, after pressing the OK button, the user will enter into the EFFECTS user interface:

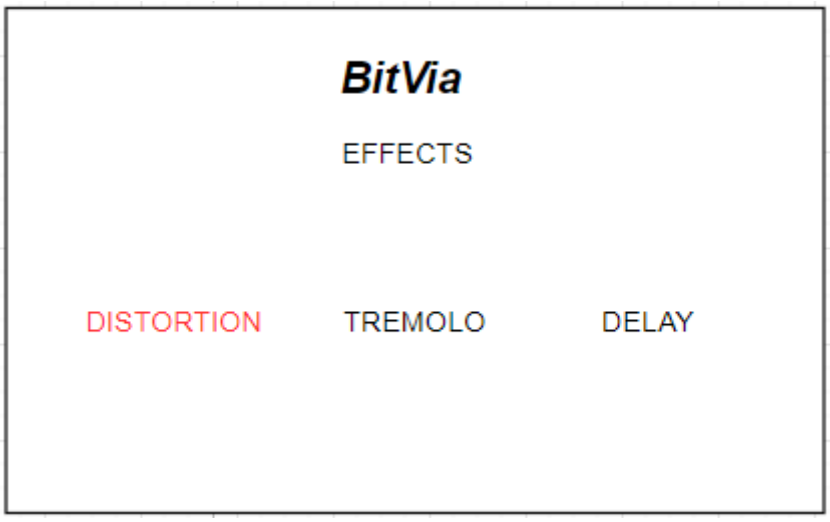

Figure 18 - Effects Sub-menu

The next level user interface for DISTORTION will show like:

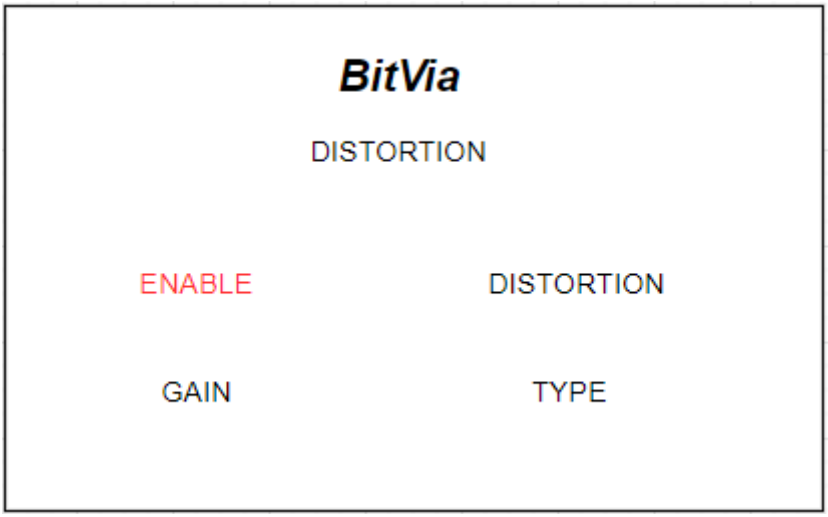

Figure 19 - Distortion Sub-menu

In the DISTORTION user interface, four next level user interfaces will show like:

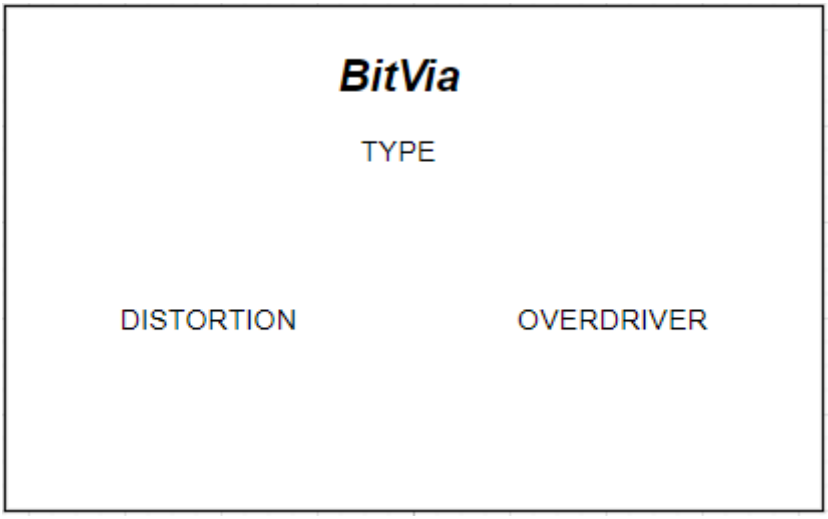

Figure 20 - Effects Type

For the ENABLE user interface, users can push button to choose to enable or disable this kind of effect. The enable user interfaces for tremolo effect and delay effect are same as this.

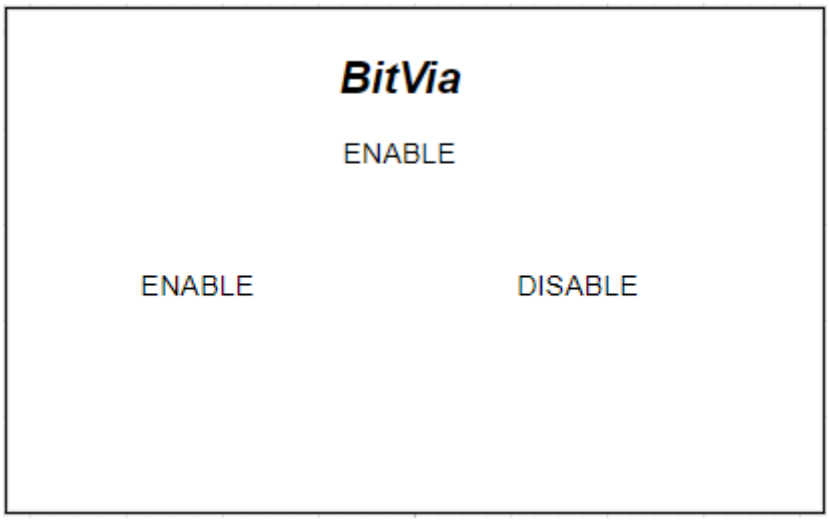

Figure 21 - Enable Sub-menu

In the “GAIN” user interface, users can push buttons to choose add or minus the number from 1-20, Other user interfaces for effects that need to tweak the values are similar to this user interface. (e.g. TREMOLO-DEPTH, TREMOLO-FREQUENCY, TREMOLO-AGGRESSION etc.)

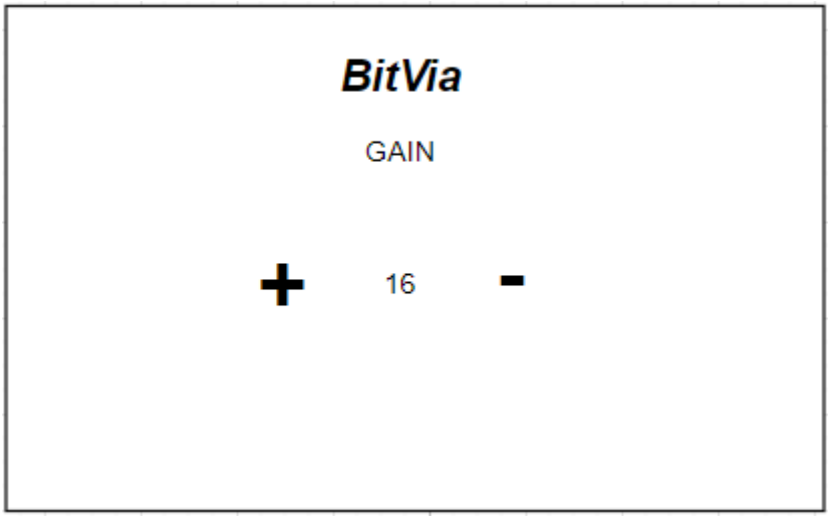

Figure 22 - Gain Sub-menu

The next level user interface for TREMOLO in EFFECTS will show like:

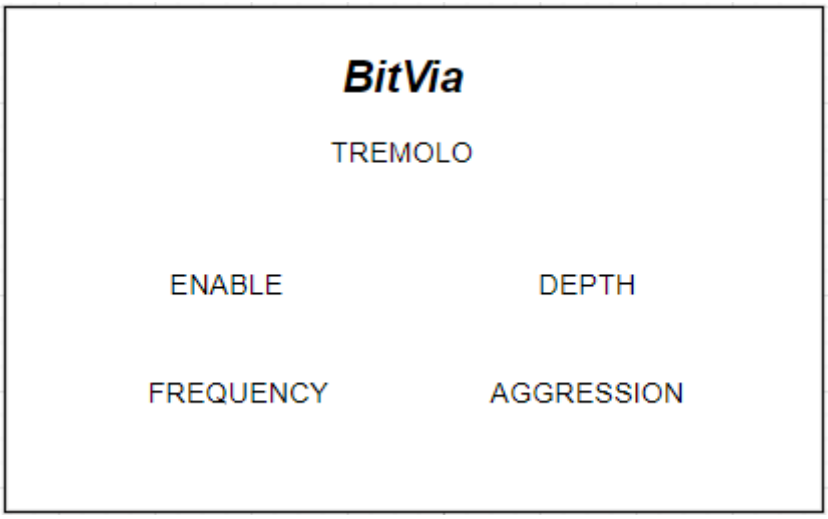

Figure 23 - Tremolo Sub-menu

The next level user interface for DELAY in EFFECTS will show like:

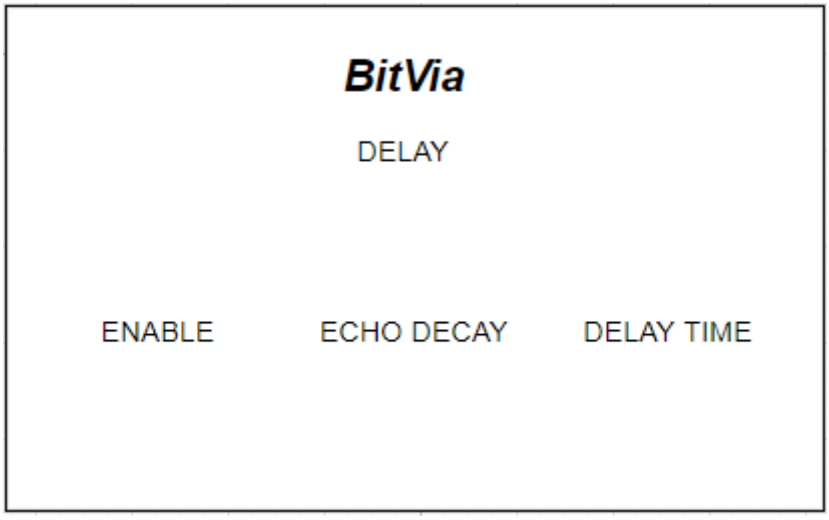

Figure 24 - Delay Sub-menu

When users choose the “FX GROUPS”, the words will show in red, after pressing the OK button, the user will enter into the FX GROUPS user interface:

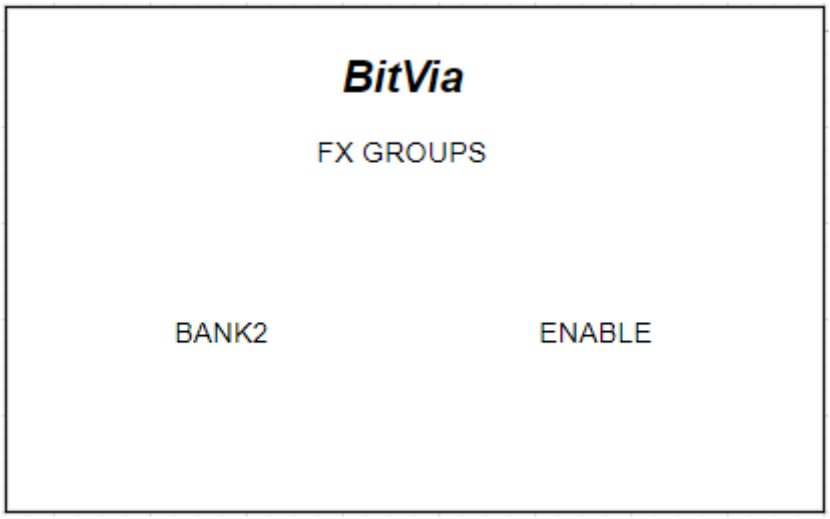

Figure 25 - FX Groups Sub-menu

Users also can choose the system language from Chinese, English and Italian.

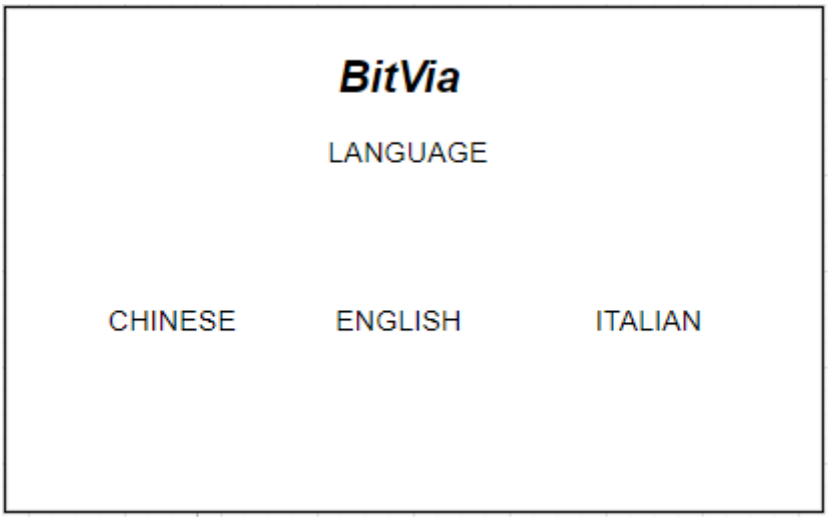

Figure 26 - Languages Sub-menu

3 Low Level Design

3.1 Codec settings – Simone Ledda

The audio codec's register were accessed via I2C protocol and register settings have been altered to match the red lined internal configuration.

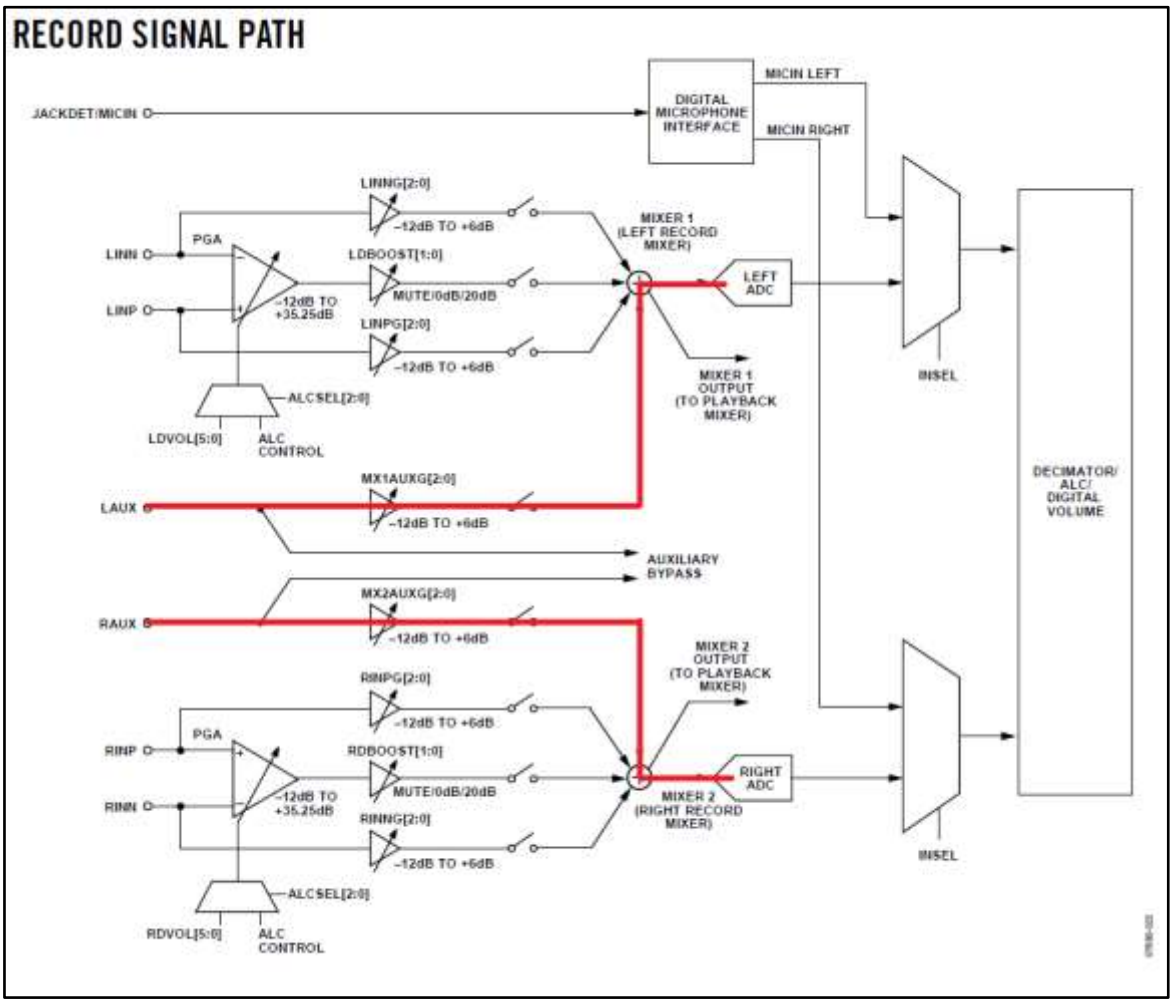

Figure 27 - ADAU1761 Record signal path [3]

As you can see from Figure 27 the AUX INPUT line, where the audio from LINE IN is travelling, cannot be amplified further by the PGA stage, instead, is directly converted by the ADC.

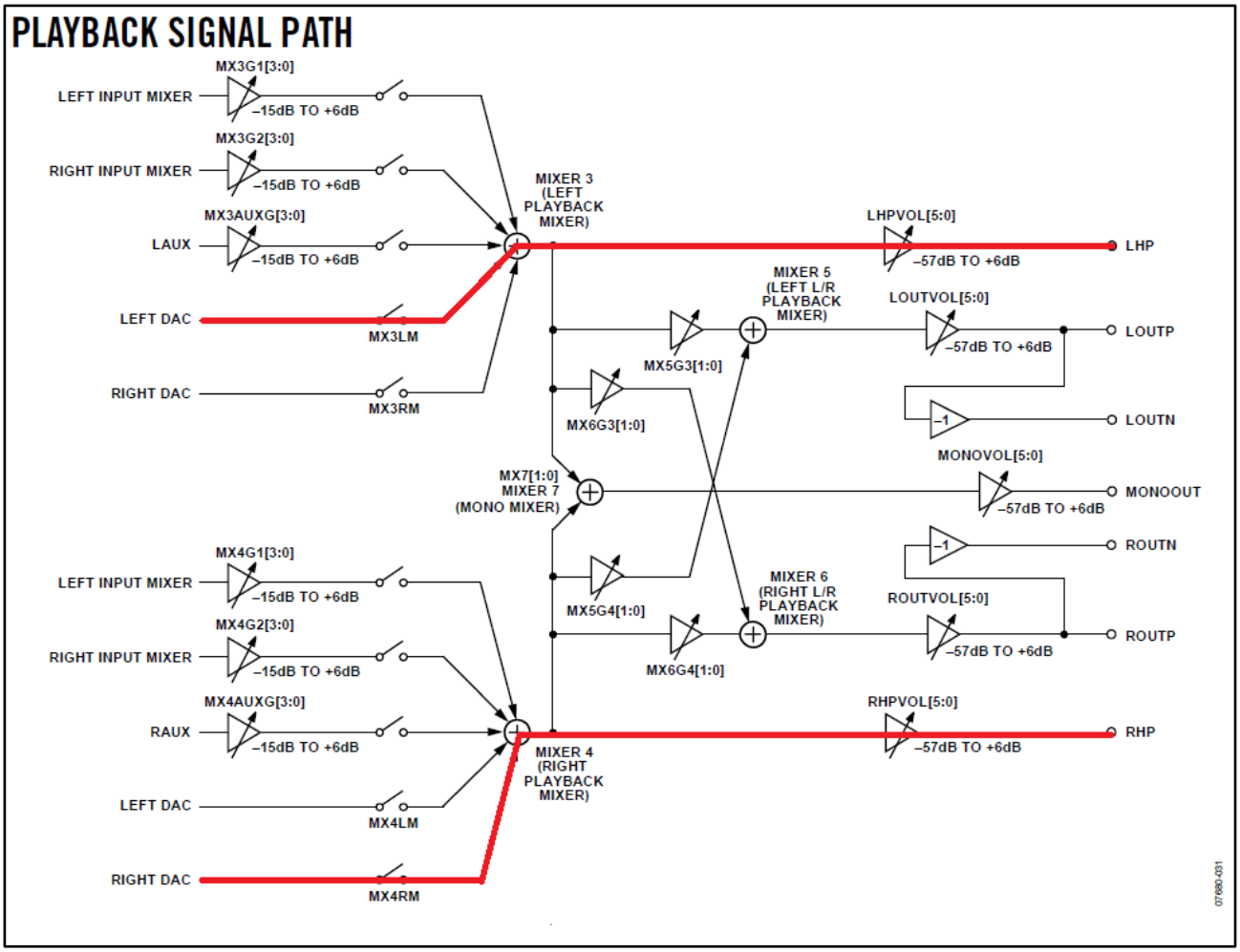

Figure 28 - ADAU1761 Playback signal path [3]

Figure 28 shows the output path of the codec.

The output path uses MIX 3 for LEFT and MIX 4 for RIGHT channel, all gains are boosted to the maximum value available. Boosting the gain doesn't introduce any distortion effect, on the other hand was necessary due to the losses on the AUX cable that appeared during the early testing stage. A further GAIN stage is being developed from the PL side to increase the volume. On the other hand, when turning the volume down, first the GAIN stage control parameter will be turned to minimum, then the volume registers in the ADAU can be changed accordingly to meet user's choice.

3.2 Registers setting and sampling rate – Simone Ledda

The high customizability of the codec allows us to change its parameters easily, other than the datapath changes shown above, some crucial parameters were written in the registers, here is a commented summary of the choices made:

- Fractional PLL enabled to generate a 48kHz sampling frequency from an 8MHz clock

  The PLL register, written in burst mode via I2C protocol, enables the codec to self-generate internal clocking references for every use, either it's for sampling or for serial communications. Due to the fact that a clock source had to be fed to the codec for it to function, it was chosen an 8 MHz clock reference. This choice was driven by two factors, the first one is that Vivado can set the Zynq chip to output precisely an 8MHz clock, at least on paper, the latter reason is that 8MHz is the lowest frequency tabled the chip can use as an external clocking reference with a known locking period for the PLL. Since the routing of the board wasn't done by us we would like to avoid sending high frequency signals around internal layers for EMC purposes.

The PLL in this case uses 3,5ms to lock, which is the longest amid locking times, though it's a once only event that happens during power up and initialization so it doesn't affect performance.

- Mixers 1,2,3,4 enabled, any other mixer disabled

  These four mixers are the only mixers the signal needs to encounter during its acquisition and output, so any other mixer that could be used is instead turned off to save power.

- Every gain stage on the red lined path has been set to its maximum value (see mirror testing)

  This choice was driven mainly because during the mirroring test the sound reaching the headphones was very feeble. Causes of this are still under investigation, the main cause may be the aux cable length that introduces power losses, but to fully answer this question we need to observe data samples to understand whether we have some dynamic room to further amplify the audio signal.

- I2C communication set for controls and I2S for data transfers

  It was chosen to use the I2C because the board's manufacturer equipped I2C lines with pull up resistors, so forcing SPI on those lines, even though it is possible, was a discarded option. This also represented a challenge because the I2C protocol wasn't encountered during the studies so far, while SPI was very familiar.
  I2S protocol is an industry standard for audio format. Since we are not using more than two channels, we discarded the TDM audio transfer option that was also available with the codec.

- Data transfers make use of dual channel, 24 bits per channel, 48kHz sampling frequency

  The recording system was thought to record on MONO but this option wasn't available with the given codec. It was chosen to record in stereo and then strip away one channel with the SERDES interface. To guarantee high fidelity we chose 24-bit conversion, that's the highest bit depth achievable with the internal Sigma-Delta ADCs.
  We also chose 48kHz sampling frequency because of the signal audio nature, hence we expect the signal to have a maximum bandwidth of 22kHz; In accordance with Nyquist's sampling frequency theorem we are sampling with a sampling frequency at least double the maximum signal's frequency.

3.3 I2C Protocol – Simone Ledda

Philips semiconductors standard is used, the hardware IP is available in Vivado 2017.4 as shown from the previous block diagram. The protocol is synchronous and uses two physical lines named SCL and SDA, respectively clock and data. It features handshaking signals such as ACK and NACK (acknowledge or miss acknowledge from slave) as well as start, restart and stop conditions. The following table is a summary of the protocol and its critical signal showing how the codec is expecting the communication to be like.

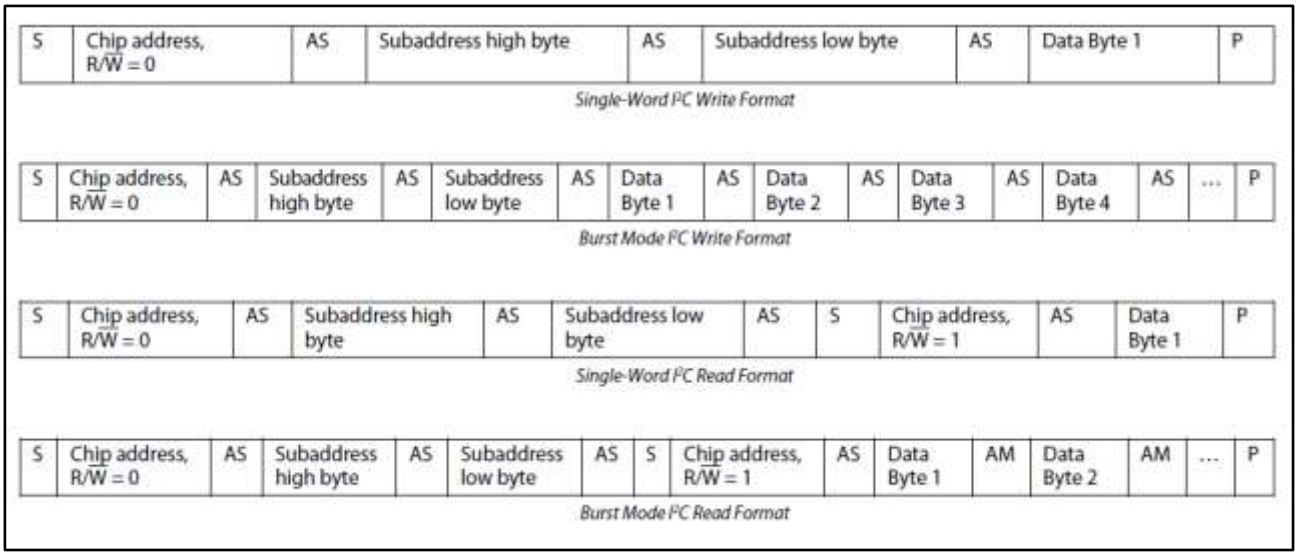

Figure 29 - ADAU1761 I2C Protocol for read and write operations [3]

3.4 Audio codec drivers – Simone Ledda

A C code draft of the drivers for the I2C bus is ready and is being tested. At this stage it is possible to use them to read and write a single register as well as read and write in burst mode. Burst mode is when the chip address and register location are called only at the beginning and every data byte following finds its own place in the receiver register because the receiver self-increments the destination register allowing the user to access multiple adjacent registers without spending three bytes to address them. The PLL register, for instance, is a register that is 6 bytes long and can be written only in burst mode.

Two sets of custom reading and writing functions have been developed, the first set reads and writes only one byte, which is suitable for the majority of the registers in the codec, the second set is for burts read and burst writes, it allows actions on multiple bytes in a row.

The codec also supports burst writes across adjacent registers, meaning that it's sufficient to address the lowest register address and begin to write data for as many register as pleased there and the address register number will be self-incremented by the I2C receiver in the codec to fit the incoming data bytes. It's important to mention that occasionally there are some register numbers missing because some registers are double the size, hence this must be taken into account when preparing the initialization unsigned-eight array.

3.4.1 Pseudo Code – Simone Ledda

At the moment there are some printf functions embedded in the driver that show in the serial interface which bytes are being sent in order to verify that the order and the values are correct, these bits of code will be removed further on because they slow down the communication. Status register are also polled during the activity to show the TXFIFO and RXFIFO status, that are later printed on the serial interface, as well as ACK received or missed. These status registers are accessed at runtime by Xilinx's driver (DynamicSend) which is also responsible to clear the interrupt flag and handle the communication.

I2C-Write

```
In write mode R/W bit is 0 (write)

R/W bit begins as a 0

TX is the transmit buffer pointer

XTC_SUCCESS = 0

XTC_FAIL = 1


// Build the message to send as an u8 array TX
TX[0] = Chip address + R/W bit
TX[1] = High subaddress
TX[2] = Low subaddress
For index = 0 to index = #databytes     // supports burst writes
       Copy databyte in TX[2+i]
Bytesent = DynamicSend(TX) // DynamicSend returns the value of bytes sent
If Bytesent -3 = #bytes to send
       Return XTC_SUCCESS
Else   Return XTC_FAIL
```

Here follows an example of communication where the first register is written with 0x07 data

Chip address: 0x70

Register address: 0x4000 (High sub 0x40, Low sub 0x00)

Data: 0x07

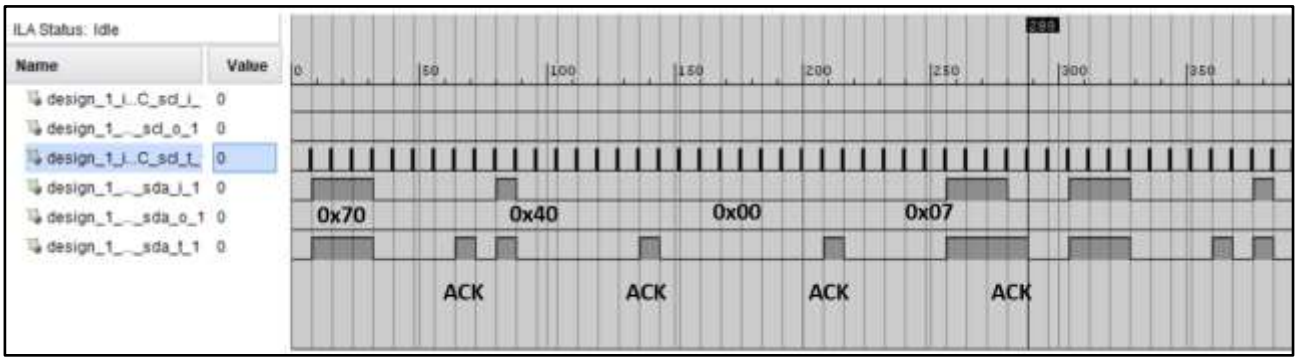

Figure 30 - System ILA I2C write operation capture

I2C-Read

Read function is more complex than the write because the master has to hand over the bus control to the slave who writes the requested data.

The read operation begins as a write except no data is written, but a stop condition is issued by the master reversing the transfer direction upon a further read&restart call on the chip address followed by the register location.

```
R/W bit begins as a 0

TX is the transmit buffer pointer

RX is the receive buffer pointer passed as an argument of read function

TX[0] = Chip address + R/W
TX[1] = High subaddress
TX[2] = Low subaddress
DynamicSend(TX, start condition)
TX[0] = Chip address + R/W // R/W bit is now 1
Dynamic Receive (TX,RX,stop)
Return RX buffer pointer
```

Here follows a System ILA's capture where a reading is attempted right after the previous write, retrieving the data byte successfully.

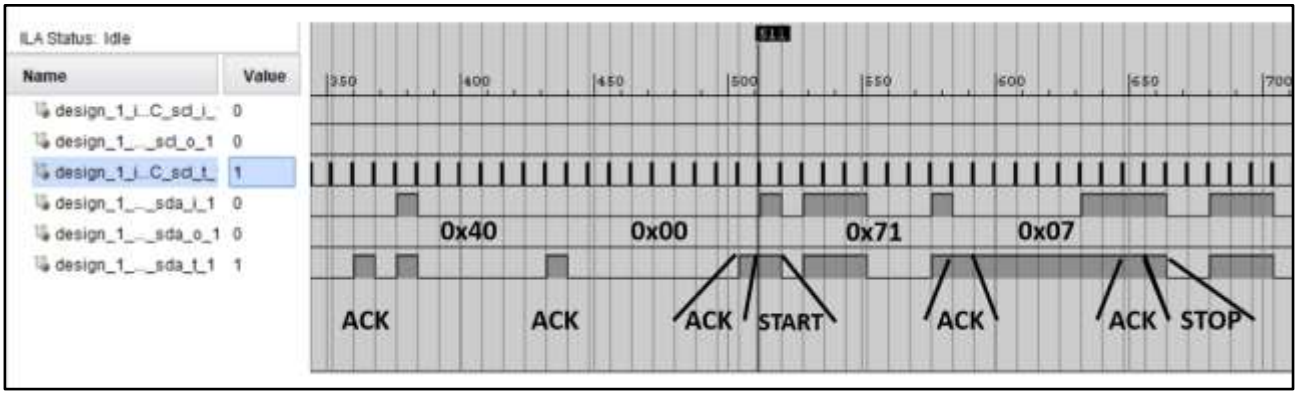

Figure 31 - System ILA I2C read operation capture

It's possible to see how the master tri-state line is driven high during acks but is asserted low in the sda_i which is the input line, meaning that the master is not driving the line, the pull up resistor is driving that high but the slave is pulling it low to flag the ack, no electrical short is made because the pull up is weaker than the tri-state driver from both sides of the bus.

3.5 SERDES Interface IP – Simone Ledda

A SERDES interface should be the first and last IP block audio data encounters on the PL side of the FPGA. The codec is sending the converted audio channels through I2S protocol. I2S protocol is particularly suited for audio transfers as it packages audio data interleaved. A summary picture is shown below.

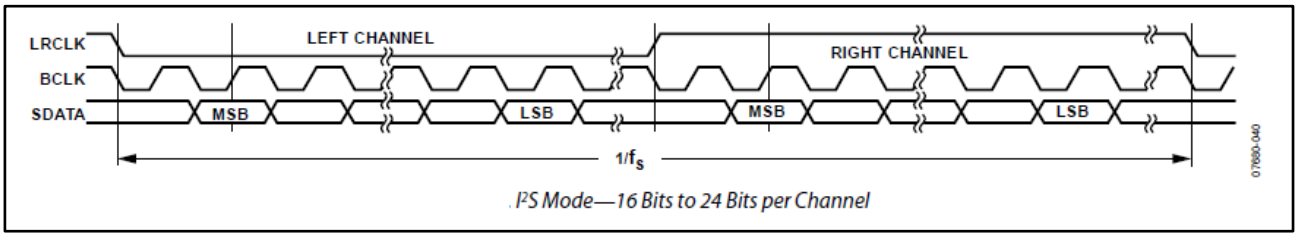

Figure 32 - ADAU1761 I2S Timing diagram [3]

The codec is connected to the FPGA through four lines that are LRCLK, BCLK, ADC-SDATA and DAC-SDATA. ADC-SDATA is converted audio samples that need to be manipulated through the effect bank, DAC-SDATA is data that is ready to be dispatched to the DAC and later to the headphones. ADC-SDATA and DAC-SDATA are serialized data.

The SERDES interface has three main tasks:

- Parallelize ADC-SDATA into 24-bit samples
- Serialize DAC-SDATA for the codec
- Control the I2S interface in order to send data in the correct time window
- Strip and duplicate channels

It's important to mention that the codec is the master of every I2S transfer since it's sourcing the clock and data on those lines. It was chosen to set the codec as a master because this way no FPGA slice is dedicated to sourcing the bus clocking, instead we can rely on the internal codec clocking system for it.

The first two points are the basics of the SERDES interface, to achieve this the logic will be clocked with BCLK clock frequency of 48kHz, to be synchronous with the codec. Data converted and data ready to be sent out will be stored in two separated FIFOs since the remainder of the datapath is clocked at 100MHz, so crossing clock domains is inevitable.

The third point regards the fact that the I2S is a full-duplex interface, hence left data should be sent only when left data is being received, same happens with the right channel.

A key feature of this device is that music instrument is recorded in MONO not in STEREO, hence it was chosen to strip one of the incoming channel because the information content is the same as its twin. This operation will be carried out after parallelization, only left channel content will be written into the RX FIFO, the right channel will be discarded.

This operation allows us twice the computing time to manipulate every sample, but we also have to duplicate samples once they are ready to be sent, plus some delicate operation before audio compression from the PS side. Every data read from the TX FIFO will be sent twice, once on the left and once on the right channel to fulfil the mono output standard.

The SERDES input data throughput is 2.3Mb/s (48kHz*2channel*24bit), its output is 1.15Mb/s since we are stripping away the right channel.

3.6 Data Conversion – Matt Reynolds

The output from the audio CODEC is in the format of fixed-point 1.23 binary notation. Our project uses the FLAC file format for storing audio data, which requires integer values. Currently, FLAC does not support floating point values. Furthermore, the effects chain being implemented is designed to work with signed 24-bit integer values.

Therefore, the data must be converted when exiting the CODEC (Figure 33). As fixed and floating point math operations inside PCs typically require more resources (either more PL fabric area or more PS operations), it is more efficient to convert the data as soon as it leaves the CODEC and only convert it back just before it enters the CODEC. The only stage between the CODEC and the data conversion is the SERDES and FIFO data control.

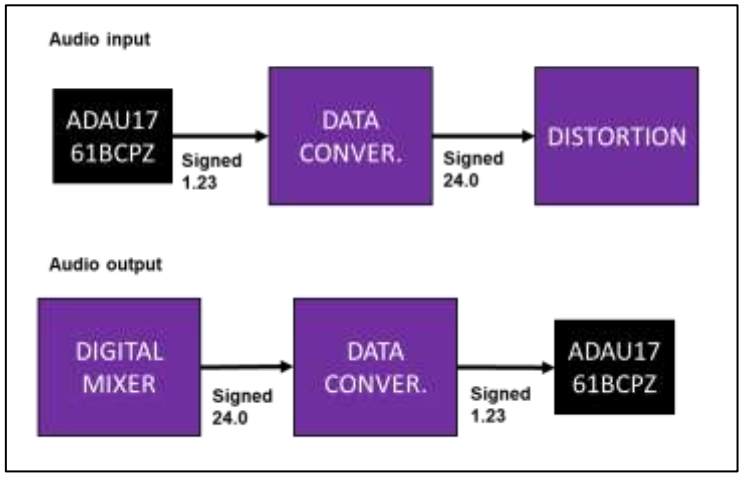

Figure 33 - Audio Data Conversion

The data conversion utilises the IEEE_PROPOSED.FIXED_PKG library. This is supported by IEEE and includes math operations for fixed point numbers. The data is stored in a register, which is then multiplied by 2^{24}, shifting the data from 1.23 to 24.0. The magnitude has been linearly increased, while preserving the sign of the sample. At this point it is converted to a signed integer value of type SIGNED (IEEE.NUMERIC_STD), following which it is passed into the audio effects chain.

The reverse process occurs when data is transferred from the digital mixer to the output FIFO.

3.7 Audio Effects – Matt Reynolds

3.7.1 Distortion – Matt Reynolds

There are two main types of distortion, mathematically speaking: hard-clipping and soft-clipping. The latter of these has many forms of implementation and can be customised almost infinitely. There are also three main names within the music industry for distortion effects: distortion, overdrive, and fuzz. It is difficult to objectively clarify the differences between these when implementing them in the digital domain, as the naming stems from the characteristics of analogue circuits.

However, for the purpose of this project, distortion will refer to hard-clipping and overdrive will refer to soft-clipping. The project's current scope includes distortion implemented as hard-clipping. There are different forms of hard-clipping. One way it can be achieved is producing sound which exceeds the maximum range of the CODEC. This results in samples being "cut-off" by the CODEC and is typically considered undesirable.

An alternative method is to intentionally cut-off the samples which exceed a controlled value, which lies within the CODEC's range. This can be expressed as:

$$f(x) = \begin{cases} -c, & x \leq -c \\ x & -c < x < c \\ c, & x \geq c \end{cases}$$

This is a common method of implementing a desirable version of distortion. It gives the user control over the amount of distortion, while generally not affecting the perception of volume. The output of a hard-clipping algorithm can be seen in Figure 34, where a green sine wave is clipped to create the purple waveform.

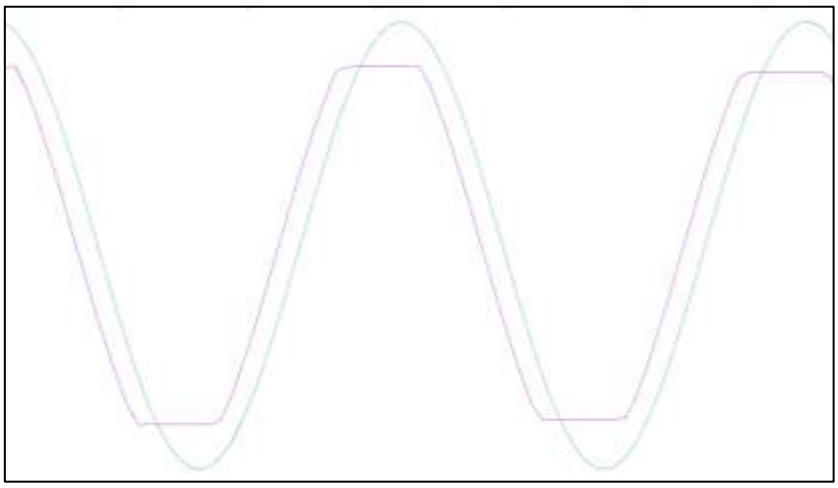

Figure 34 - Hard-clipping Waveform

For the BitVia media player this hard-clipping implementation also creates control interfaces for the user: gain, distortion level. Gain control, allows the user to control signal amplification prior to entry to the distortion DSP. Increased gain will result in higher (and earlier) levels of distortion. This may be useful for quieter signals.

Distortion level controls the value at which clipping occurs. Increasing the control value, decreases the clipping value, which creates for more aggressive distortion. These two control implementations

mimic those found on most analogue distortion interfaces and have therefore been implemented in this way to create a familiar user-interface; as per the specification.

3.7.2 Tremolo – Matt Reynolds

The tremolo effect is created by modulating the input signal such that the amplitude is regularly decreased and increased, creating rapid audible changes in volume. One way in which this effect can be achieved is to multiply the incoming signal with a modulating waveform. The incoming signal preserves its frequency spectrum (i.e. there is no pitch shift), but the amplitude is changed at the rate of the modulating wave. (Figure 35)

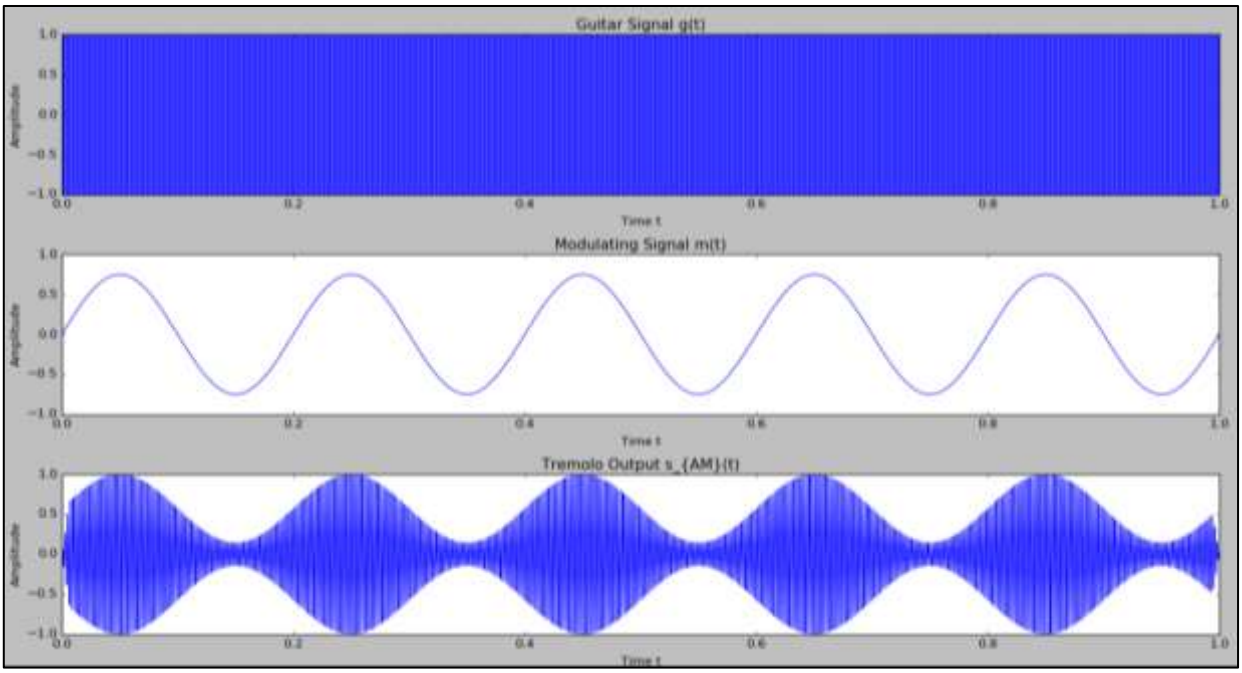

Figure 35 - Tremolo Modulation

Different types of wave can be used as the modulating waveform, depending on the severity of the effect which is desired. For the BitVia media player a triangle waveform has been chosen to create a slightly more intense tremolo effect than that of a sinewave modulator. Also included is an '*aggression*' control, which allows the user to clip the modulating wave before it is multiplied with the incoming signal. The clipping of the triangle-waveform creates a near-square wave, resulting in a more aggressive tremolo effect.

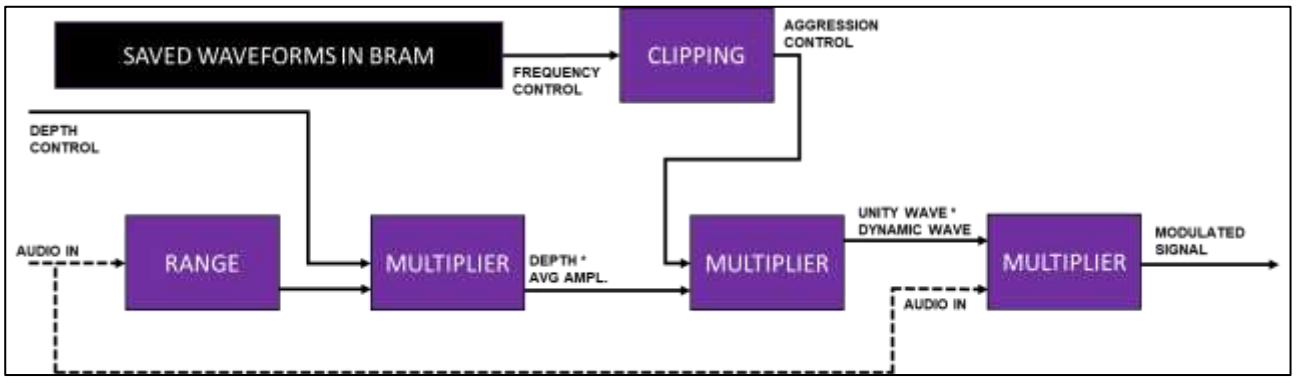

Figure 36 - Tremolo Block Diagram

Figure 36 shows the block diagram for the tremolo effect design. A range block is included to allow the effect to respond to dynamic changes in volume of the input signal. This prevents the effect causing the output to become dramatically louder or quieter than the input signal, which would be

undesirable. The depth control allows the user to increase or decrease the amplitude of the modulating waveform. If the modulating waveform is equal to the input signal in amplitude, then it is said to be modulating at 100%. Reducing the modulating waveform's amplitude decreases this value. The depth control allows for modulation to be greater than 100%, which creates interesting audio features and is often a desirable characteristic.

To reduce the implemented area of the tremolo effect on the PL fabric and to reduce the computation required, a sample waveform will be stored as the modulating wave, as opposed to the generation of one. From this, the user will be able to control the frequency of the wave, its shape, and its amplitude; each of these parameters will be discrete and finite.

A single period of a triangle waveform will be stored in a block RAM (BRAM) on the PL fabric, which will behave as a read-only memory. Typical modulation frequencies for a tremolo effect are between 1 Hz – 20 Hz. The user will be able to select at every integer of one between, and including, these values.

A waveform of 1 Hz requires 48000 samples to capture a complete wave period for a sampling frequency of 48 kHz.

$$n = \frac{{F_w}^{-1}}{{F_s}^{-1}}$$

n = Number of sample, F_w = Frequency of wave, F_s = Sampling Frequency

Each sample is 24-bits, resulting in 1152000 bits ≈ 1152kb. A BRAM on the XCZ7020 is 36kb, therefore a total of 32 BRAMs would be required. There are 140 available on the CLG400 package.

Given a sample wave-period of a fixed frequency, the output frequency can be controlled by altering the speed at which the waveform is read out. To create a higher frequency wave, the sample wave would be read out faster, resulting in a full wave-period being read out in a shorter period of time, where $F_w = {}^1/_t$. The reason for which the lowest frequency wave desired is used as the sample waveform is for audio fidelity.

The lowest frequency requires the greatest number of samples and therefore consumes the most resource. A similar effect could have been achieved by saving a higher frequency waveform and reading it out slower. However, this would diminish the quality of the audio and therefore, contradict a key selling point of our product.

The sample rate for our media player is set at 48 kHz. This means that the wave cannot be read out 'faster' as computations are to occur only on each passing of a new sample, and are thus fixed. To overcome this, samples will be skipped when reading out the waveform. (Figure 37) Thus, giving the same effect. If a user should desire a higher frequency modulation wave, the circuit will skip more samples.

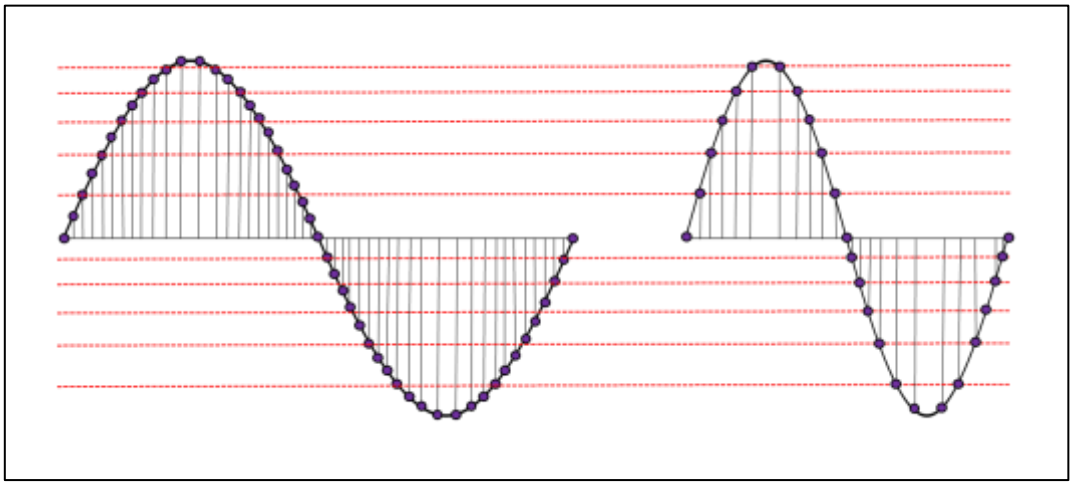

Figure 37 - Skipping Samples Illustration

3.7.3 Delay – Matt Reynolds

The delay effect has been designed to loop a decaying repeat of the input signal. (Figure 38) If the user were to play a single note, they would hear this note repeated after a short period of time (10s of ms). It would then continue to repeat, each time getting quieter, until becoming inaudible. The user is given control over the length of the loop, by controlling how fast the sample decays.

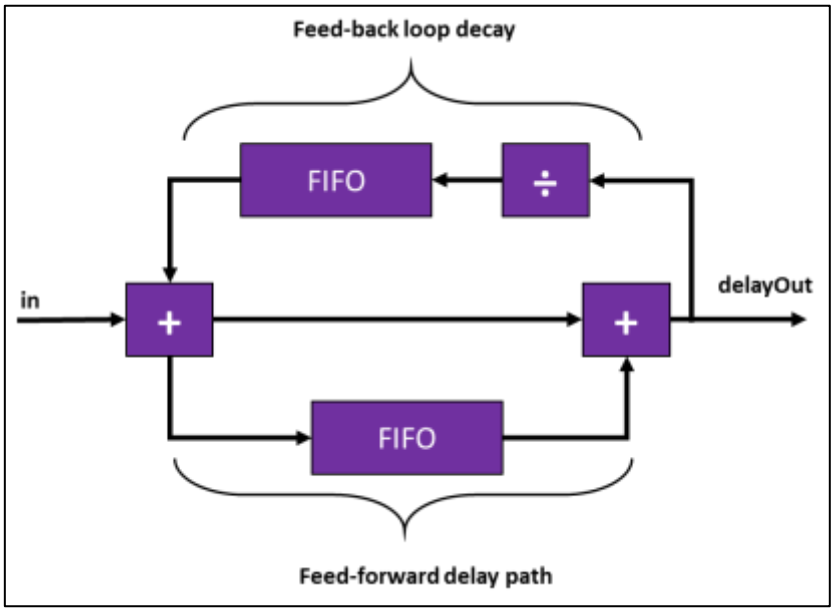

Figure 38 - Delay Block Diagram

They are also able to control the time period for the delay. For very short delay periods this can have the effect of reverb. These are delay paths which are too short for the human ear to identify as a separate echo, but create depth to the sound (similar to playing in a large empty room). Allowing the user to select these short delay times as well as longer delays, adds versatility to the effect for a relatively small area penalty in the PL fabric.

To control the delay time the user will set the value of a counter, which will be compared with the count value of the number of samples in the feed-forward FIFO. When the values match, the control circuit will toggle the read signal high, causing the sample to be read out of the FIFO. By increasing the counter value, the user will increase the delay time, as a higher number of samples will be read into the FIFO before being read out. Of course this means a finite delay limit will be implemented.

A minimum delay of 10ms would allow for the reverb effect to take place. A maximum delay of 1s would provide the user with a desirable range for the delay effect. To make the effect easy to use the discrete values between the range will be split into 20 options. This will continue the interface already designed into the distortion and tremolo effects.

To create a delay of 1s, 48000 samples are required. This equates to 32 BRAMs per FIFO. The feedback loop will require the same FIFO depth as the feed-forward loop, therefore 64 BRAMs will be required for the delay effect. This brings the current total of BRAMs used by the effects chain to 96, out of an available 140 on the XC7Z020.

In the feedback loop, which will create the decaying echo, a divide by shift operation is performed on the incoming signal to reduce the amplitude. The user will set the value of the shift operation, determining the rate of reduction of amplitude. Again this will be set into 20 discrete values for which the user can choose, ranging from 2^0 to 2^{24}, where 2^0 will have a very slow decay time and therefore the note will loop for longer; 2^{24} will be no loop at all.

3.8 TFT Display – Matt Reynolds

3.8.1 Hardware – Matt Reynolds

The drivers for the TFT display are situated in the PL fabric. This hardware interface drives the GPIO pins connected to the 40-pin header of the PYNQ-Z2 board, where the display is connected.

There are two main operations which must occur when using the display, configuration and writing data. Configuration uses the SPI interface of the display. However, the display's pinout only makes available a single SPI data line, MOSI; there is no MISO used in the screen configuration.

Due to the lack of a datasheet or supporting documentation, existing configuration software has had to be ported and reverse engineered to complete the setup steps required. To align with the current software drivers, the use of an AXI GPIO IP to control the SPI interface is required. Although, the design could be further optimised by implementing a dedicated SPI IP and redesigning the software to interface with this. Currently, this is not in the scope of work as it is low priority.

AXI-Stream to Video Out is an existing and tested IP, making it a reliable choice for outputting data to the display. This requires an AXI-Stream input and outputs RGB data. The RGB data of the IP is 8-bits per channel, however the screen operates with 6-bits per channel. A common way of interfacing these two RGB formats (which is also supported by the labelling of the display pins), is to use bits 6:1 of 7:0, when outputting image data. This has been done for each channel, taking 18-bits from the concatenated channels, and converting the 24-bit output of the Video Out IP.

Data will be streamed to the Video Out IP using the AXI VDMA soft-core IP. An AXI4-Lite interface allows the VDMA to be configured via a memory-mapped interface. This is also used to set the start address for the frame buffers and control the run/stop bit. To prevent data corruption, two frame buffers will be used. These are located in RAM. The VDMA will read from one buffer while the PS writes to another buffer. Each frame buffer will be 864 kB.

It is possible to reduce the frame buffer size by using colour look-up tables. However, the total memory consumed by the two buffers is acceptable for the amount of RAM available in the system.

Additional information is required to output images to a screen, these include vsync, hsync, DPI_En, and DPI_clk. These are all related to the timing of the image and ensure that each frame is displayed correctly. These are controlled by the Video Timing Controller IP and are configured using the GUI during design. Once implemented these timings are fixed into the PL fabric. A separate clock is generated by the PS to control these timings and meet the required frame rate of the display (60 FPS). DPI_clk receives the inverted clock signal.

3.8.2 Software – Matt Reynolds

Initialisation procedures are carried out using the XGpio APIs and the ported code from the Pimoroni Git repo. The touchscreen and the display setups will be called by the main function. These will write to the AXI interconnect which will control the AXI GPIO for these GPIO pins.

Image data will be written to the screen via the VDMA. This will be controlled by the software on the PS. The VDMA requires several steps in order to be configured and set into a run state:

1. *"Write control information to the channel VDMACR register (Offset 0x00 for MM2S and 0x30 for S2MM) to set interrupt enables if desired, and set VDMACR.RS=1 to start the AXI VDMA channel running.*
2. *Write a valid video frame buffer start address to the channel START_ADDRESS register 1 to N where N equals Frame Buffers (Offset 0x5C up to 0x98 for MM2S and 0xAC up to 0xE8 for S2MM). Set the REG_INDEX register if required.*
3. *Write a valid Frame Delay (valid only for Genlock Slave) and Stride to the channel FRMDLY_STRIDE register (Offset 0x58 for MM2S and 0xA8 for S2MM).*
4. *Write a valid Horizontal Size to the channel HSIZE register (Offset 0x54 for MM2S and 0xA4 for S2MM).*
5. *Write a valid Vertical Size to the channel VSIZE register (Offset 0x50 for MM2S and 0xA0 for S2MM). This starts the channel transferring video data. "* [11]

The VDMA control register (VDMACR) is accessed via the AXI4-Lite interface to the VDMA IP. This is a 32-bit register with an offset of 0x00 from the IP's base address on the AXI bus. The values in Table 1 will be assigned to the register during initialisation.

Table 1 - VDMA Control Register Values

Bits	Field Name	Value	Description
31-24	IRQDelayCount	0x00	Generates interrupt after a delay period. Not used.
23-16	IRQFrameCount	0x00	Counts down the number of frames when transfer starts. Not used.
15	Repeat_En	0x00	Only applicable in Genlock Master/Dynamic Master mode. Not used.
14	Err_IrqEn	0x01	Status register generates an interrupt out when error occurs.
13	DlyCnt_IrqEn	0x00	Generates interrupt out for error on delay count. Not used.
12	FrmCnt_IrqEn	0x00	Frame count not used. Disabled.
11-8	RdPntrNum	0x00	Used with multiple entities. Set to default value; controlling entity 1.
7	GenlockSrc	0x00	Enable external Genlock.
6-5	Reserved	-	-
4	FrameCntEn	0x00	Halts transfers after frame count complete. Not used.
3	GenlockEn	0x01	MM2S synchronized to Genlock frame input.
2	Reset	0x00	May be used during runtime if error occurs, otherwise 0.
1	Circular_Park	0x01	Circular mode – continuously circle through frame buffers.
0	RS	0x01	Set to 1 during setup – run mode.

During initialisation, the malloc function will be called to allocate memory for the two frame buffers. The start address for both of these will be passed to the START_ADDRESS register, with the first address written to offset 0x5C and the second, 0x60.

The frame delay indicates the minimum number of frame buffers for which the slave should be behind the master, in this project this value is one as there are two buffers. The stride value indicates the number of bytes between the first pixels of each video line, or the horizontal size of the image/resolution. As our display has a width of 800pixels, with each pixel being stored as a 32-bit word, this results in 200 bytes. Therefore, a value of 0x010000C8 will be written to FRMDLY_STRIDE.

The MM2S_HSIZE value must match the stride value. The value 0x000000C8 will be written to HSIZE register at offset 0x54 for MM2S. Our display has a vertical depth of 480 pixels. The value 0x000001E0, equating to 480 lines, at the offset 0x50 for MM2S.

Following the completion of these steps, the VDMA will begin streaming data to the Video Out IP.

The VDMA IP also offers a frame sync setting (FSYNC). It is recommended that in a read-channel-only design, FSYNC is set to none to avoid race conditions. This means the Video Out IP will control the frame sync to the VDMA via the AXI-Stream interface, which in turn is controlled by the Video Timing Controller.

To prevent the VDMA reading from the frame buffer which is being written to, it will be placed into Dynamic Genlock Slave mode. This makes the PS the master, which will out put the frame number being written to and pass this to the mm2s_frame_ptr_in port. This port is sampled by the VDMA to determined which frame to read from.

To enable this the following steps must be taken:

1. Set GenlockEn: MM2S_VDMACR[3] = 1 to enable Genlock synchronization between master and slave.
2. Set GenlockSrc: MM2S_VDMACR[7] = 0 to enable External Genlock mode.

Figure 39 shows the software diagram for the TFT drivers, located in the PS. The display management will be wrapped into a FreeRTOS task, which will call configure the VDMA on first run. A finite state machine will be developed as the display control. When the user presses a push button or switch, this will call an interrupt handler which will update values in the FSM.

The text output, which contains parameter values from multiple tasks, will be generated into an image. The text will be output on a black background, with the selected text a different font colour to the rest of the menu items. Based on these two specifications, the output image will be generated and stored into the frame buffers.

When updating the display, both frame buffers will be written to with the new image, as the image is mostly stationary. Therefore, there is not a live stream of moving data. The VDMA readout will continuously cycle through the frame buffers, so they must have the same data to create the correct image.

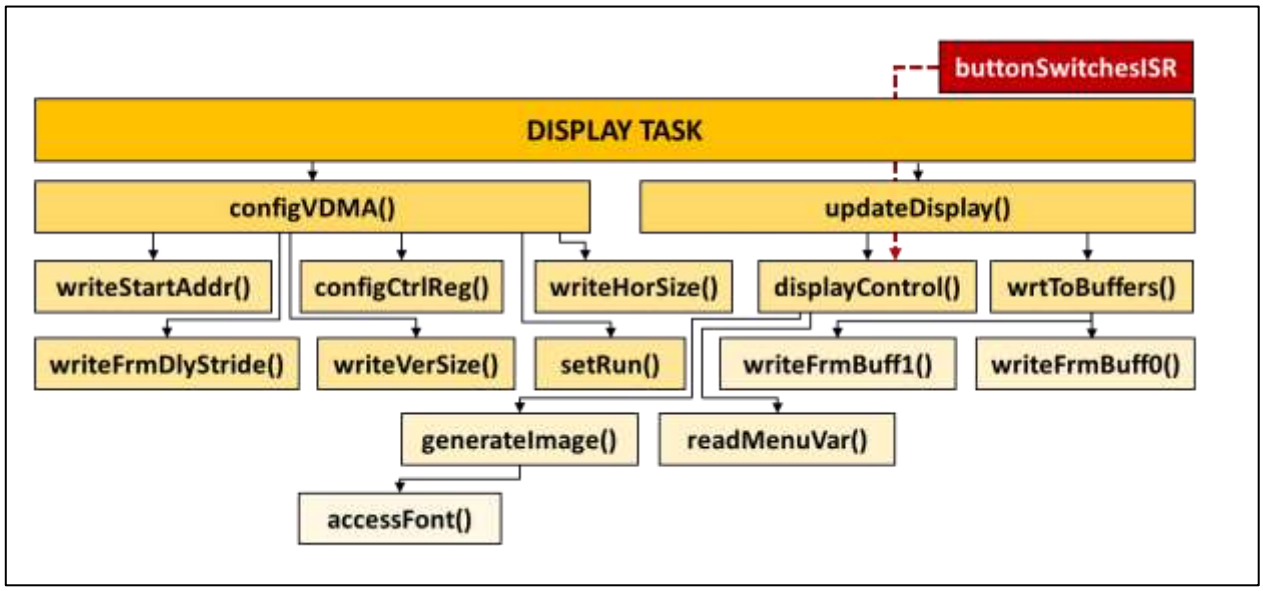

Figure 39 - TFT Driver Software Diagram

3.9 DMA – Gavin Johnston

The datapath of our design is shown in Figure 40, it has three distinct segments; the backing track reading and decoding path; the combined audio encoding and writing path and; the instrument codec and effects I/O path.

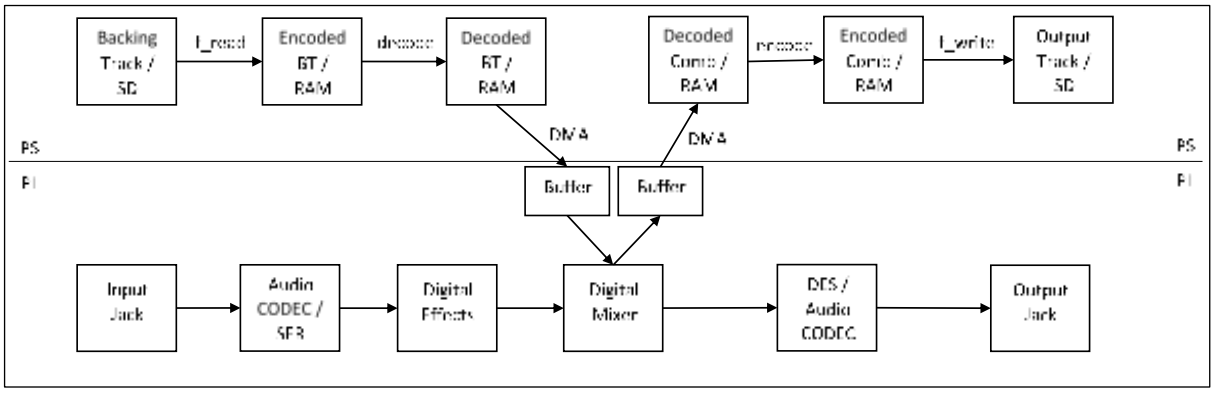

Figure 40 Data Flow Diagram

The backing track reading and decoding path starts with the backing track (if selected) being read, from a file on the SD card, into an intermediate buffer in RAM. The backing track is then decoded using the FLaC library functions into another RAM buffer. The backing track is then sent to a buffer on the PL side, through the PS DMA. The data is then sent to the Digital mixer, sample-wise, as required.

The instrument codec and effects path starts with the instrument being sampled from the input audio jack by the codec. It is then serialised and passed through the digital effects chain, which adds the effects to the instrument data. It is then mixed with the backing track (if selected) in the digital mixer, de-serialised, decoded and output on the output audio jack.

The combined audio encoding and writing path starts with data being written to a PL buffer, from the digital mixer, as it is generated. Once the buffer is full, a DMA transfer is triggered to transfer the buffer contents into the combined audio buffer in RAM. The combined audio in this buffer is then encoded using the FLaC library functions into an intermediate buffer. The encoded combined audio is then written to an output file.

This design features two distinct areas where data is transferred between the PS and PL using DMA. There are represented in Figure 40 as the two arrows marked DMA. To implement a DMA transfer, you need to program and then trigger a channel. We have two distinct transfer requirements therefore we will implement two channels.

The PS DMA is programmed in SDK using the Xilinx DMAPS library. Pseudocode for generating a DMA (microcode) program, which will control channel n, is shown below:

```
DMA_commmand_structure_type DmaCmd
DmaCmd <- source burst size (bytes)
DmaCmd <- source burst length
DmaCmd <- source address incrementation
DmaCmd <- destination burst size (bytes)
DmaCmd <- destination burst length
DmaCmd <- destination address incrementation
DmaCmd <- starting source address
DmaCmd <- starting destination address
DmaCmd <- transfer size (bytes)
Initialise DMA
Initialise interrupt System
Attach fault ISR
Attach done ISRs
Enable fault interrupt
Enable done interrupts
Generate DMA program for channel n using DmaCmd structure
```

Figure 41 - DMA program generation pseudocode

Once the code has been generated it is triggered with a Peripheral Request Interface (PRI) in the PL. PRIs are simple AXI handshaking interfaces which can be used to interface directly to the PS DMA (DMAC in Figure 42) and are used to trigger the DMA transfer.

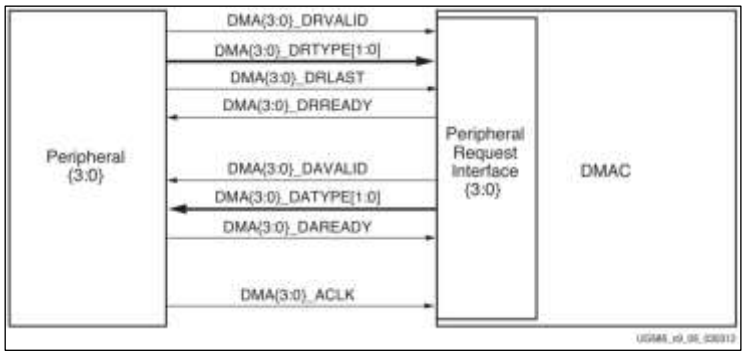

Figure 42 Peripheral Request Interface [9]

Up to four PRIs can be implemented (indexed in Figure 42 with {} brackets) which can each trigger a separate channel. They use standard AXI ready/valid handshaking protocol and can request a burst or single fire transfer from the DMA. We will use two PRIs, one for each of our channels, they will trigger their respective channels when the backing track PL buffer is almost empty or when the combined audio PL buffer is almost full. Figure 43 and Figure 44 illustrate the hardware for the backing track and the combined audio DMA transfers respectively.

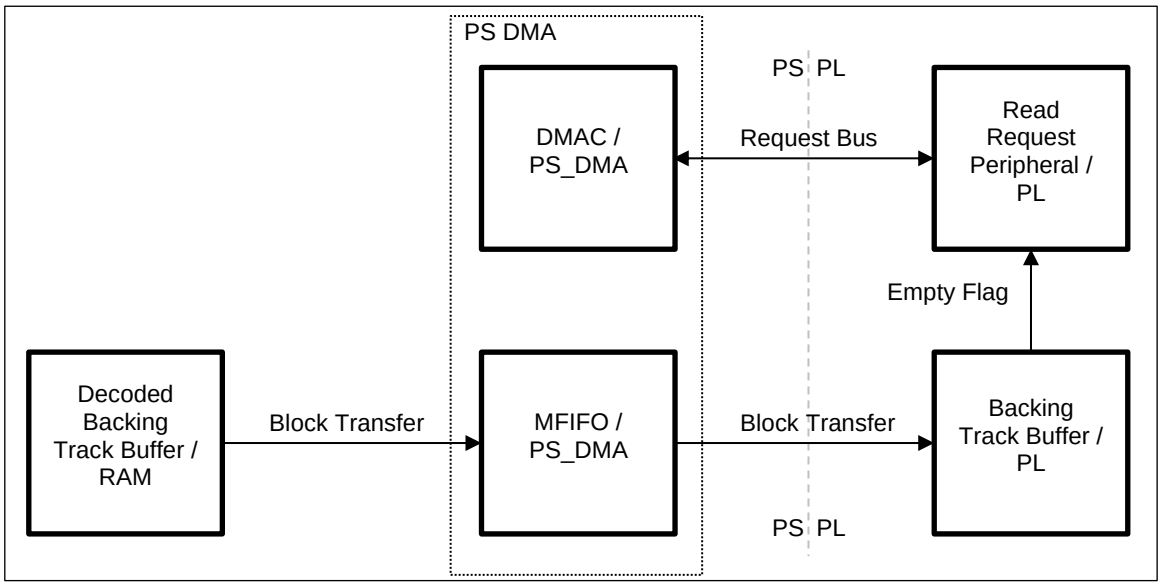

Figure 43 Backing Track RAM to PL buffer transfer

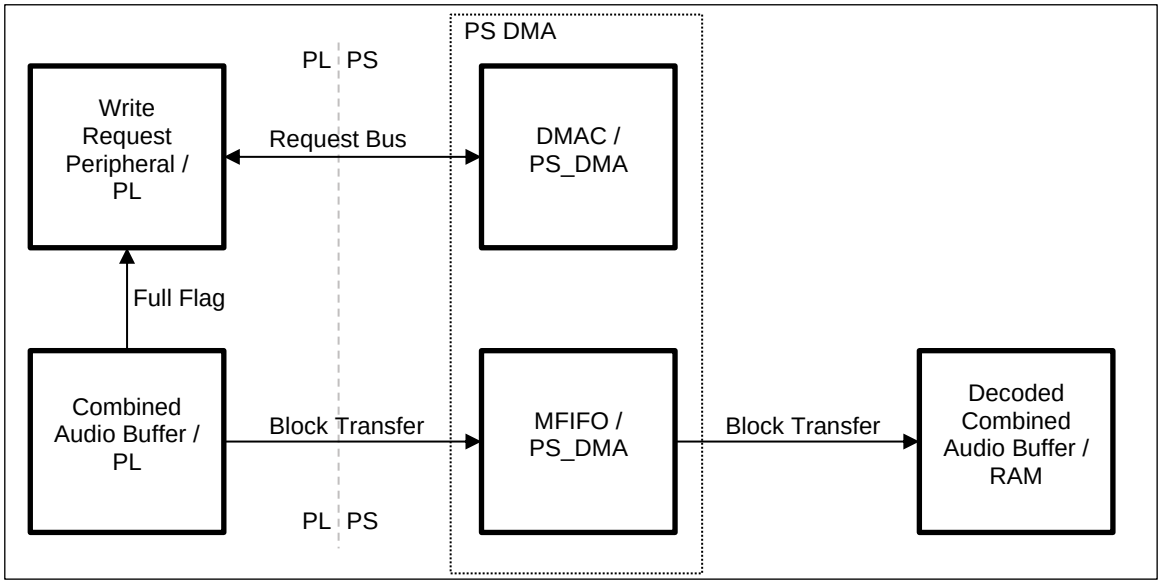

Figure 44 Combined Audio PL to RAM buffer transfer

The Write/Read Request Peripheral and DMAC blocks represents the control logic and the rest of the blocks are the datapath. Note, the Write and Read Request Peripheral are identical, the channel set-up is completed in software so the only jobs for the request peripheral is to trigger the channel and handle the acknowledgements.

3.10 SD Card – Gavin Johnston

As stated previously, we are using an SD card as mass media storage; it will be formatted with the FAT32 file system using the Xilffs library. Before using a drive, the drive must be mounted. Pseudocode for the wrapper function to mount the SD card is shown below.

```
Attempt to mount SD card

If mount failed as no card is present

	Report card is missing

Else if mount failed because no FAT drive was found

	Format SD card with FAT32

Else if mount failed for another reason

	Report error
```

Figure 45 - SD card initialisation wrapper function pseudocode

The initialisation wrapper function attempts to mount the card and reacts to any error that occurs; if the card is missing the user is notified, if a card is present but no FAT drive was found then the drive is reformatted, any other errors are returned. If the drive mounts successfully then it is ready to use.

For this project we require the ability to open, close, create, read and write files; as well as opening and closing directories. Most functions exist in the library and are suitable for our use cases. Creating a file requires a custom function and we want a wrapper around the write file function to avoid overwriting.

Creating a new file is achieved by attempting to open a file that doesn't exist and using the "always open" option, this will be encapsulated in its own function along with validation that the file doesn't already exist and an option to overwrite if it does. The create function is shown in Figure 46.

```
Attempt to open file of given name

If succeeds // because file exists

	Prompt to overwrite

	If prompt is denied

		Close file and exit function

	// else everything is already set up for overwrite

Else // file does not exist

	Open file with given name using the always open option //create file
```

Figure 46 - File create function pseudocode

The write function will have a simple wrapper function (Figure 47) that moves the cursor to the end of the file before writing to prevent corruption. A file must be open before the write wrapper function is called.

```
Move cursor to the end of the file

Write to file
```

Figure 47 - File write wrapper function pseudocode

3.11 FLaC Codec – Gavin Johnston

We have chosen to encode our files in the FLaC file format, to implement this we will use the libFLaC library provided by the Xiph.org Foundation. [12] The FLaC codec is used to decode the backing track and encode the combined audio (See the encode and decode arrows in Figure 40).

The backing track will be decoded using the FLaC stream decoder, the pseudocode to set up and use the stream decoder is shown below.

```
Generate new stream decoder instance

Initialise the stream

Run decoder until end of metadata

While backing track data is waiting to be decoded

	If decoded backing track RAM buffer is not full

		Decode a single audio frame

	Else

		Wait for buffer to empty

Finish the stream decoder

Delete the stream decoder
```

Figure 48 - Stream decoder pseudocode

The combined audio data will be encoded using the FLaC stream encoder, the pseudocode to set up and use the stream encoder is shown below.

```
Generate new stream encoder instance

Set number of channels

Set bits per sample

Set sample rate

Initialise the stream

While combined audio data is waiting to be encoded

	If encoded combined audio buffer is not full

		Encode a single audio frame

	Else

		Wait for buffer to empty

Finish the stream encoder

Delete the stream encoder
```

Figure 49 - Stream encoder pseudocode

4 Testing

4.1 Audio Datapath Testing – Simone Ledda

System testing will be carried out according to the V-model. This way we ensure that every IP joining the datapath on the PL is reliable and has passed the individual testbench.

The V-model is a testing model used in industry; it requires thorough testing of every hardware component along with its corresponding software block, when testing is complete the system is enhanced by the tested block and the next block can undergo testing like for the waterfall model.

The testing involves laying out subsets of the entire datapath and run tests to see their reliability and their eventual issues or delays.

The ideal test equipment involves a function generator to plug into the input jack and an FFT oscilloscope or a spectrum analyser in the output jack. The test purpose is that to observe the same input function on the output jack, and also to make sure that eventual noise is outside the audio bandwidth (20-20kHz). Unfortunately, due to the lack of laboratory equipment we had to change the testing for the entire architecture. Our input generator will be a known audio track and the output oscilloscope will just be a standard headphone output.

We will make extensive use of the System ILA IP because it's the main debugging tool, due also to our current situation where we are forced to work without the adequate lab equipment needed for testing. In test reports its use will be extensively documented and debugging images will be shown as captured screenshots, evidences of the work carried out.

The starting subset will be a simple VHDL buffer responsible to mirror I2S-ADC data to I2S-ADC line. This test aims to check the correct configuration of the internal registers of the audio codec as well as the I2S-full duplex feature of the protocol, as it wasn't mentioned on the codec datasheet. This test has been carried out already and a gain loss has been reported, meaning that we will necessarily add a gain stage to our datapath as sound is very feeble even when driven with maximum gains wherever possible.

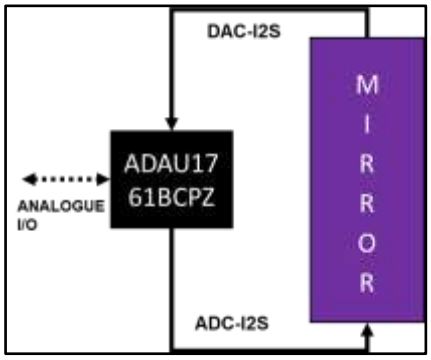

Figure 50 - Block diagram of mirroring test

The next step is to carry out a test for the SERDES interface, testing its features and operations. The thorough testing in this phase will concern the timings of events as the interface being full-duplex, the SERDES will have to output the right channel and the right bit every BCLK period.

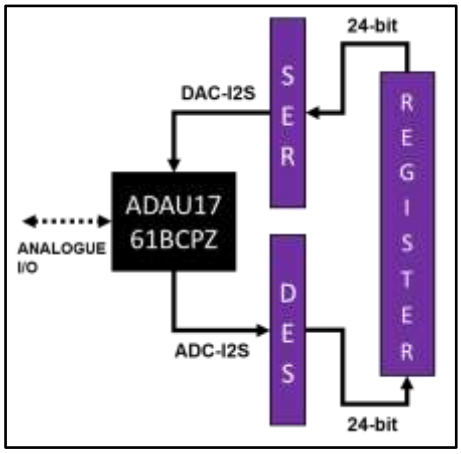

Figure 51 - Block diagram of SERDES interface

Once the input and output rings of datapath chain are set, it's the time to test the effects. Effects IP will be a single IP accessible with registers through the AXI-lite interface. The hardware manager will be able to trimmer the effects as he pleases and there will also be the chance to apply eventual corrections to math operations and filters involved.

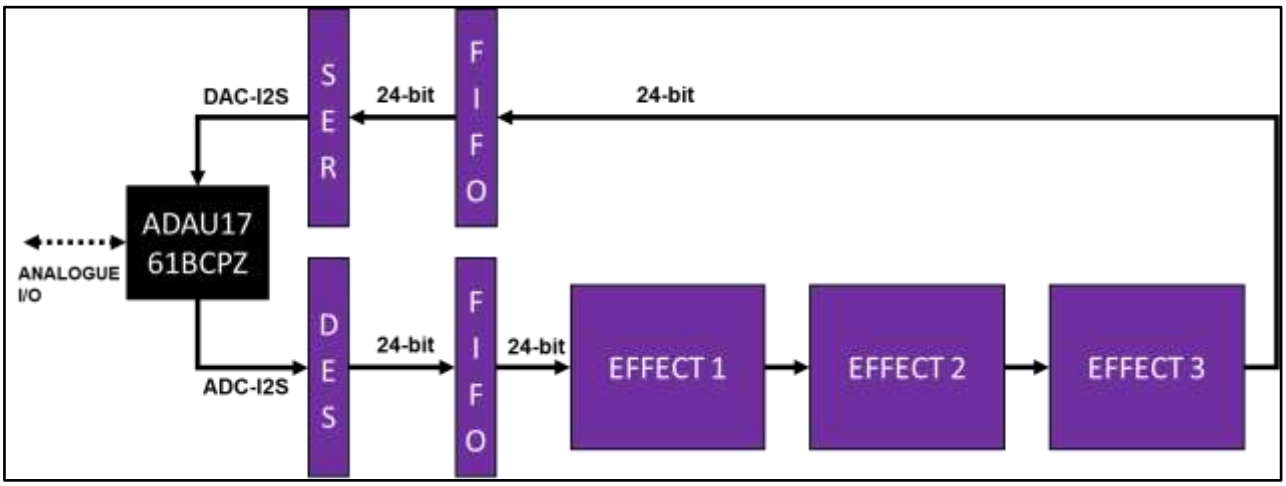

Figure 52 - Block digram of STAGE 1

Once effects are set, the system is enhanced with the DMA block, enabling us to save files that are currently being played. This crucial feature will start also the software testing for main software blocks such as the FLaC compression and the SD data saving.

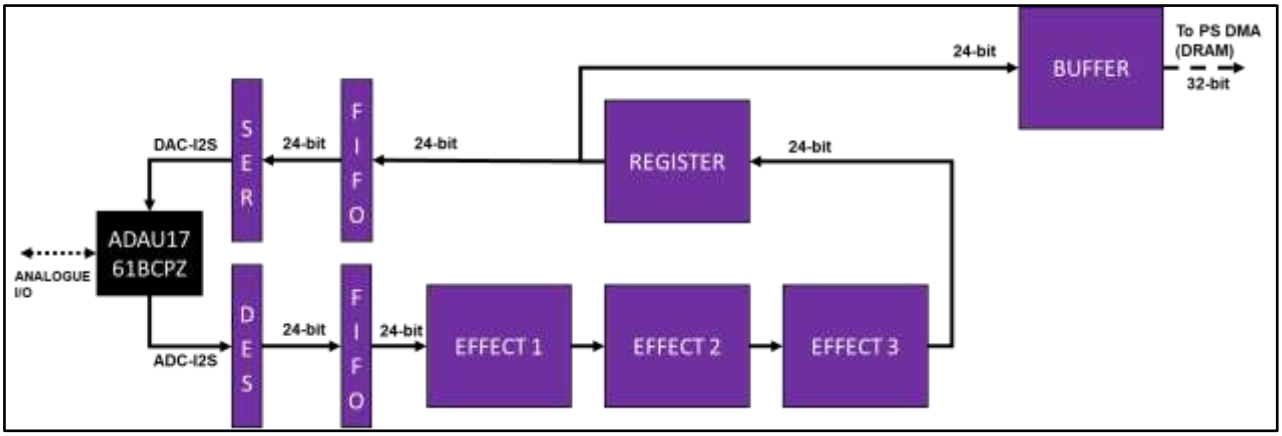

Figure 53 - Block diagram of STAGE 2

The final block to join the datapath will be the digital mixer. This block is able to merge two data stream, one coming from the effects bank and the other one coming from the PS side, which represents the base track the user wants to play upon.

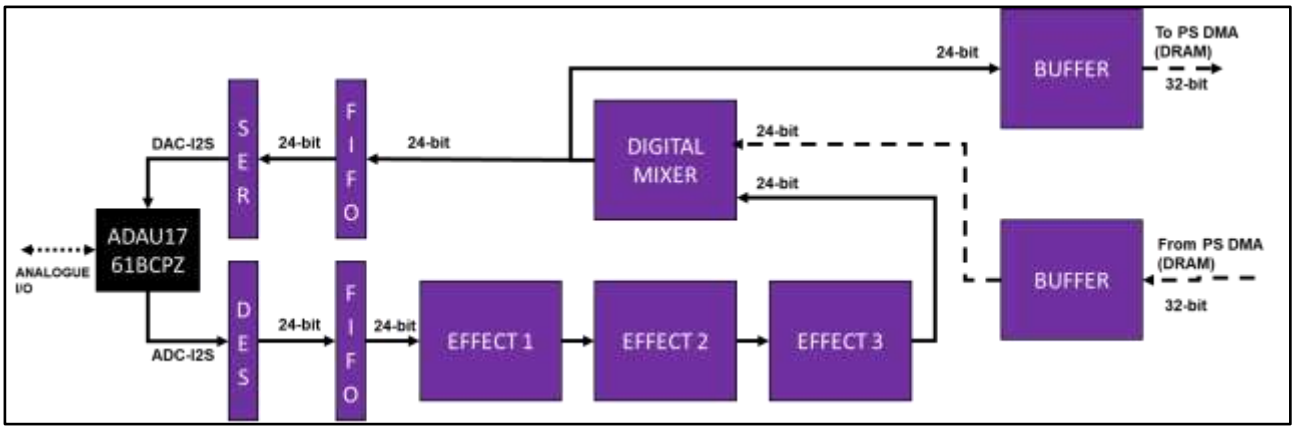

Figure 54 - Block diagram of STAGE 3

4.2 Data Conversion – Matt Reynolds

Known values are sent into the data conversion circuit using a VHDL testbench, where the output can be monitored to identify correct values. This also allows access to the internal signals of the circuit, where errors can be easily identified.

This format of testing can be used to test a wide range of values and for a large test sample, the FILEIO library can be used in conjunction with the ASSERT/REPORT/SEVERITY statements to create a comprehensive automated testbench. It is important that overflow cases are tested to ensure that the signed information is maintained.

Furthermore, linearity of numbers must be tested. In the absence of the conversion, an increase in value from the CODEC may not be observed as an increase in value from the perspective of the effects chain. This is best explained with an example, as shown in Table 2.

Table 2 - Data Conversion Example

Binary	Unsigned	Signed	1.3
1001	9	-7	-0.125
1011	11	-5	-0.375

From this it can be seen that an increase in binary value, results in an increase in value in both the unsigned and signed domains, yet the 1.3 format decreases in value. Although 1.3 has been used in the example and 1.23 is used by the CODEC, the principles remain the same but the data is larger. This is the reason that the data conversion circuit is needed and therefore, the aspect which should be heavily tested.

4.3 Distortion Effect – Matt Reynolds

An input waveform (sine wave) will be generated using a custom C programme and placed in a constant array within a VHDL testbench. This will then be passed through the block, altering the control parameters during the simulation and capturing the data in an output file. This file will then be plotted on an XY graph along with the original sine wave. The two will be compared to identify: has clipping occurred, have the controls operated correctly (clipping increased/decreased), is the period of the wave preserved?

Second, a hardware-in-the-loop test will take place, where audio will be played through the codec and the distortion effect will be implemented on the PL. An assessment of the audio will be made by the tester to identify whether the effect is desirable. The tester should listen for undesirable elements such as, phase shifting, popping or sharp sounds, crackling noises, low-fidelity audio.

Ideally, in a lab environment, a tone generator would be used to sweep through the range of audible frequencies, while an oscilloscope captures the FFT plot of the output. This would allow the tester to identify undesirable harmonics and ensure any undesirable changes in amplitude or phase are negligible. However, this equipment is not available in the current situation.

4.4 Tremolo Effect – Matt Reynolds

The tremolo effect is more complex than the distortion effect and requires several stages of testing. A sine wave will be passed through the moving average block and the output will be measured against the known value generated in a C programme. It is expected that a waveform of amplitude +/-10, will return an average value of ~6, which is approximately the RMS of the triangle waveform. This value is useful as the amplitude multiplier for the modulation waveform, as it creates modulation below 100%, which allows for a more versatile range of depth. [13]

The second part that will be tested independently is the reading out of the stored waveform at different frequencies (number of samples skipped), this test can be combined with the clipping of the waveform. The waveform should be read out at each of the 20 discrete frequencies identified within the design and the output data stored in a file. This can then be plotted to visually identify the waveform, which when unclipped should represent a triangle wave, and the wave periods can be measured to verify the correct frequencies have been obtained. This process can be repeated with different values of the *aggression control* to test the clipping of the triangle wave.

Finally, all subcomponents must be tested together. A test tone (constant sine wave of 10 kHz) will be looped as the input data to the effect. The modulated output data will be captured in an output file. This process will be repeated for varying parameters of the control variables. The final output data will be plotted and compared with the data created in software for which this circuit implements.

Again, a hardware-in-the-loop test will be required with live audio being used to test the tremolo effect. Simulations and data verification are useful for identifying if the mathematical theorems have been correctly implemented, however the subjectivity of sound requires a tester to listen to the effect. Audible clarity is one component of the test for which the listener should be observing, but this must also be supplemented with perceived desirability; does the effect sound good?

4.5 Delay Effect – Matt Reynolds

The delay effect cannot make use of the visual waveform testing in quite the same way as the tremolo and distortion effects, as the output waveform becomes much less clear and results in complex waves. Therefore the VHDL testbench will rely on identifying delayed samples in the numerical format. This is best viewed in the Vivado Waveform Viewer.

A test tone will be passed into the effect through the use of the FILEIO functions in VHDL. This will consist of a series of samples. The numerical values will be clocked in by the circuit and stored into a FIFO. After a number of clock cycles (which must match the delay parameter) the sample should appear on the output again, this will be visible numerically.

As the delay period is extended during testing, new samples will be mixed with old samples and the numerical data may be harder to verify. To assist in this, the ASSERT/REPORT/SEVERITY statement will be applied in the testbench, to automate the test. An efficient way to carry out the test would be to use generics in the module entity. This is because, a one second delay period on a 100 MHz clock, will take a very long time to simulate. It is likely that the test will be carried out with smaller simulation periods and verified to show that it adjusts accordingly with the updating of the generic. If successful, it will be assumed that the circuit is able to exert the same behaviour for the larger delay times used in implementation.

Hardware-in-the-loop testing is going to be most beneficial here. Testing the ranges of the control parameters to check they are appropriate, ensuring signal clarity is maintained, listening for the delay period of the feedback loop, these must all be measured by the tester.

4.6 TFT Display – Matt Reynolds

The Video Test Pattern Generator IP will be used to test the hardware drivers of the display. This will also test the configuration functions written in software. If all test patterns are displayed correctly then the tester can verify that the configuration files are correct and that the drivers have been implemented appropriately. Follow this, several stages of testing need to be carried out.

First, a block colour will be written into the frame buffers. The debug tools in the SDK will be used to identify that the correct values have been written to the correct memory locations by the write frame buffers function. This will be repeated for red, blue, and green. The tester should identify that the correct number of bytes is being written between the lower and upper address boundaries of the frames.

Second, the text output which should be displayed on the screen, will also be printed over the serial port using the Xil_printf function. This will identify that the display controller is calling the correct menus in response to the relevant pushbutton/switch interaction.

To try force data corruption, frame 0 and frame 1 will be continuously written with alternating colours (red and blue). The VDMA should always follow the MM2S frame pointer. If an error occurs the VDMA will output an error IRQ (as set by bit 15 in the VDMACR).

The pushbuttons/switches interrupt handler will be tested by first being connected to an LED on the PYNQ board. An LED should light up when a button is pressed or a switch changes value. Once confirmed this can be rewritten to interact with the display control FSM. Setting a flag to indicate a change in the value of the buttons and reading in the new value. This value will be printed over the serial port during testing.

To confirm the correct horizontal and vertical sizes have been set in the control register, a red box, one pixel wide, will be drawn around the boundaries of the test image of a black screen. This will be manually generated by writing the required values to the output array from an external C programme.

4.7 DMA – Gavin Johnston

Figure 43 and Figure 44 show the hardware involved for the two DMA transfers. Both transfer paths must be tested before integration.

To test the backing track RAM to PL buffer transfer, the decoded backing track buffer present in RAM will be filled with a known pattern of data. A transfer will be triggered by resetting the PL buffer; causing the empty flag to be raised. Once the transfer is complete, the contents of the RAM buffer and the PL buffer will be compared to ensure all data was transferred correctly and without corruption.

To test]the combined audio PL to RAM buffer transfer, the combined audio PL buffer is filled with a known pattern of data which will trigger the full flag, triggering the transfer. Once the transfer is complete, the contents of the PL buffer and the RAM buffer will be compared to ensure all data was transferred correctly and without corruption.

4.8 SD Card – Gavin Johnston

To test the SD Card interface, all the functions in the Xilffs library, and wrapper functions, that we plan to use will be tested. Note, extensive testing of Xilffs functions are not necessary as they have already been tested as part of a library, our tests are only to confirm the file system is set up correctly and the functions can operate correctly (i.e. file system is not read only etc.).

To test the initialisation function (Figure 45), we will start by having no SD card inserted into the board and attempting to mount an SD card, ensuring the function correctly registers the lack of card and reports the fact. We will then insert an NTFS-formatted SD Card into the board and attempt to mount, the software should register the card is not FAT formatted and reformat the card. Finally, a FAT32 SD card, formatted externally, will be inserted into the board to ensure the function can mount an externally formatted SD card.

The file create function (Figure 46) will be tested in two parts, creating a new file and overwriting an existing file, this test will also test the open file function. First, we will attempt to create a text file with a name that was not previously used; this should create an empty file with that name. Then we will attempt to create a file that already exists and has data. This should cause a prompt for overwrite and the function should successfully overwrite the existing file with a new blank file with the same name.

The write wrapper function (Figure 47) will be tested by creating a new file (with the file create function), writing data, closing the file, opening the file and writing more data. When examined this file should have the new data appended to the old data. This also tests the open file read/write option, write and file close functions.

The read function will be tested by opening a file created and populated from another computer and reading its contents into a buffer, the contents of the buffer and the file will be compared.

The open and close directory functions are tested by attempting to navigate through an externally-made folder structure with multiple levels; if we can successfully traverse through the folder structure and open files in different folders then we can consider the functions operational.

4.9 FLaC Codec – Gavin Johnston

The FLaC codec has two streams, decode and encode, we will test these individually.

To test the decoder stream, we will generate or source a FLaC encoded audio test track, write it onto the SD card externally then decode it on the device. The output of the decoder will be compared to an external decoder to ensure integrity.

To test the encoder stream, we will place raw data in the decoded combined signal buffer in RAM, encode it then write the output to file. The file will be decoded externally, and the data will be compared to the raw data input to ensure integrity.

4.10 Software testing – Simone Ledda

Software testing will make use of the black-box testing strategy, this testing system will check the code functionality against its specs making sure it fulfils the purpose.

During development phase two main branches of software will be lined out, one is the drivers, the other one is high level functions.

Drivers are IP-specific designed and their testing is carried on in parallel with the hardware development. Among the drivers the reader can find I2C, effect bank, DMA, Digital mixer and control interface drivers. Drivers are written in C code and may be modified as the integration test plan proceeds to further stages.

High level software functions involve heavy software tasks that will make extensive use of the A9-Cortex. Among these functions there are DMA transfers to and from PL, data compression and decompression, data saving to SD card and data reading from SD card.

4.11 Test failure mitigation – Simone Ledda

Failure can happen that's why testing is carried out. We plan to catch the majority of the eventual failures by early testing every component as it's ready and by reducing the number of variables every test carries in. That's why every main IP block will be tested individually before joining the datapath,

according to the V-model. We will also make use of the Pareto rule, by testing more thoroughly fewer parts of the system that we estimate can carry the majority of troubles. We believe the DMA and the digital mixer will represent a peak complexity IPs, hence further tests will be carried out on them, mostly related to tight timings available to carry out the operations and coding issues.

Our efforts are towards passing every test and, in case this may not be possible a group action will carried out to find a solution to the issue found, possibly by avoiding re-designing major parts of the architecture.

4.12 Test reports – Simone Ledda

Every team member will test the IPs they made and write a test report according to the testing templates available in appendix to this document. Every IP will be tested in hardware as well as in software, where available. Test blocks and main coding functions will undergo just the software testing using the black-box testing methodology. Test reports will be attached to the final report as evidence of the work done, giving the reader the chance to replicate the tests.

5 Integration Plan

5.1 Overview – Simone Ledda

The integration plan involves merging hardware blocks along with their respective drivers and software main operations involving the datapath. The integration plan will necessarily follow the testing plan for the involved IPs and drivers.

The integration plan consists of two main tasks than can be run in parallel not to waste valuable time. The two tasks involve the integration of the user interface with pushbuttons and switches, and the integration of the entire datapath with the processing system and the SD card. These two tasks are independent until the late stages of the project.

The user interface consists of the TFT display, the LEDs, pushbuttons, switches and the Cortex A9. Its integration revolves around the display drivers, pushbuttons interrupts and software menus. The plan is to obtain a surfable menus interface that can later on be populated with the low level datapath drivers according to the software function outlined.

On the other hand, the datapath integration is going to be more time consuming because of the number of IPs involved and the eventuality of mistakes that may occur.

This operation will see three main subsets (stages) of the datapath, the first integration step will see the SERDES interface and the effect bank together. As mentioned in the testing plan at this stage, the effects bank is equipped with registers that will be used to change its settings. A datapath controller is introduced in this stage, by wrapping every IP on the PL in a single IP block featuring also status registers, mapping FIFOs states in case data is blocked somewhere.

This overlay structure will be used for testing and debugging on both sides either software and hardware.

An early stage timing analysis will also carried out because of our tight timing constraint of a maximum input-output delay of 15ms.

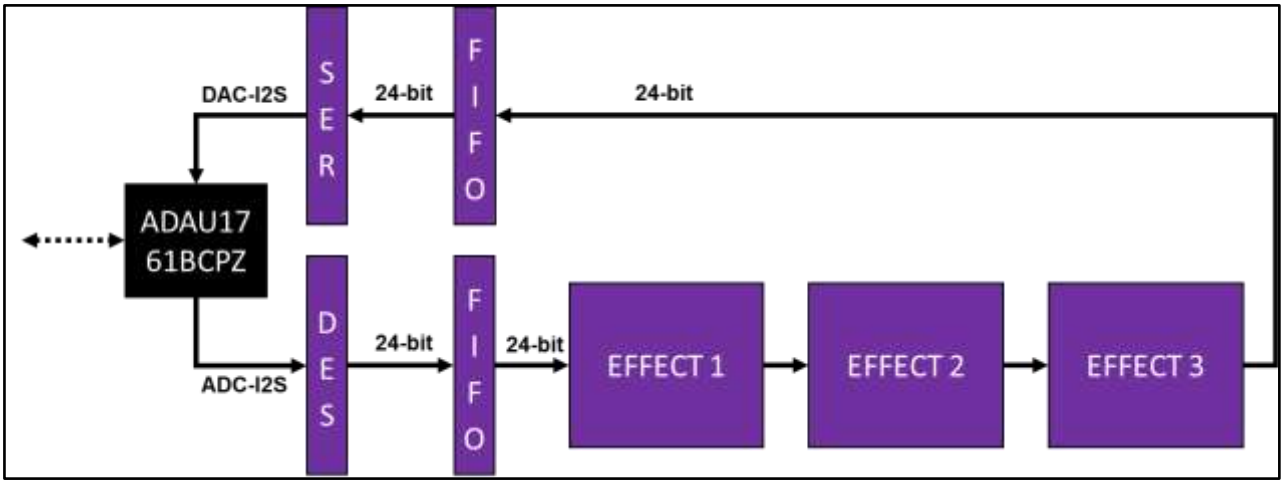

Figure 55 - Block diagram of STAGE 1

The second stage will see stage one plus the DMA for transfers to DRAM.

The higher hierarchical IP control block developed in the previous integration stage will be modified making room for the DMA control register and status register. DMA will be accessed through the AXI-Lite interface and software testing can be started.

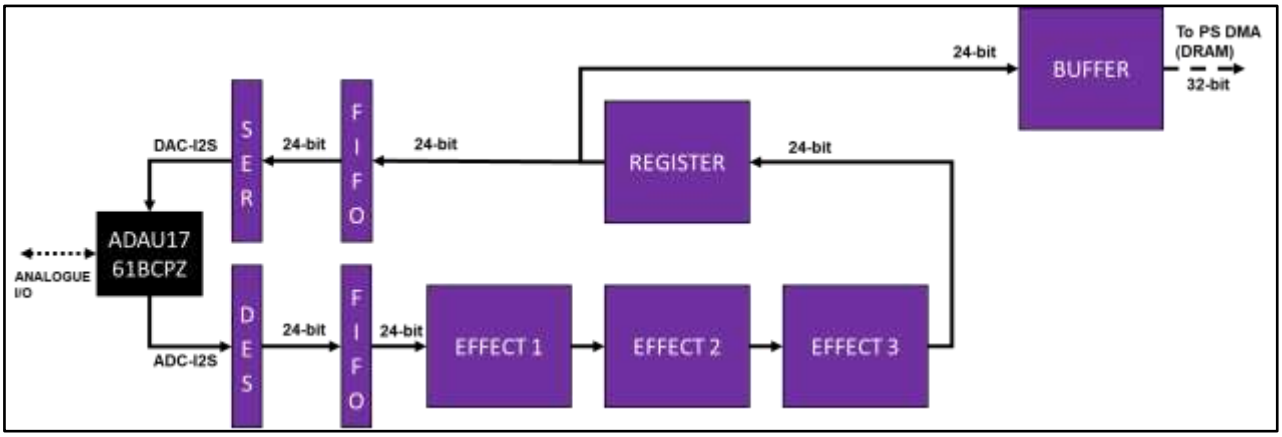

Figure 56 - Block diagram of STAGE 2

Stage two is extremely important for software testing as it can finally run with live streaming data that can be captured and saved to the SD card. We will also have to check timing constraints such as how big should the buffer be and how often is better to transfer its content to DRAM, even though we have already an estimation of its value, in needs to be verified at this stage.

The third and final integration stage will see stage two plus the digital mixer block, allowing to play stored files out to the headphones. Here we will run the same buffer estimation we ran in stage two but the other way around, from PS to PL. Also the time constraint of 15ms will be examined for the last time and awarded a pass/fail grade. In case of a fail grade we may consider to introduce further pipelining stage or to clock the PL at a higher speed, according to the amount of resources left on the programmable logic.

Once reached this stage the datapath and the developed controls should be complete, allowing us to test the entire hardware and software system.

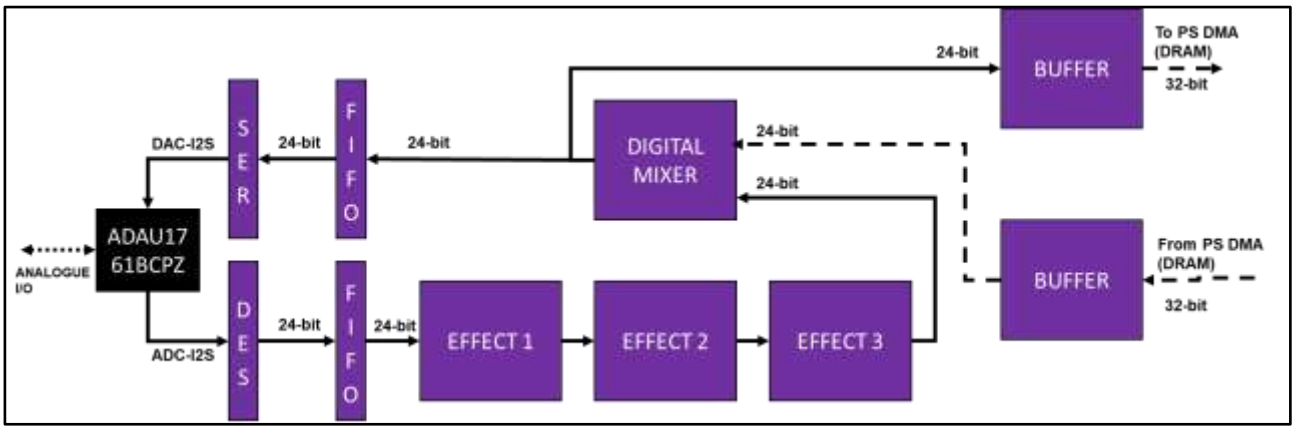

Figure 57 - Block diagram of STAGE 3

Among the most delicate tests there are those involving stress tests such as adding a base track to the incoming stream and save the file at the same time.

Then quality controls will be addressed such as suppression of pops and click sounds that may occur due to register changing in the datapath architecture that need to be muted in the codec. We will also test effect transitions from on to off an vice versa.

The following step will be the matrimony between STAGE 3 and the integrated user interface and its peripherals. This may be time consuming from the synthesis and implementation point of view because of the usage of the PL, but we are confident that everything will fit on the Zynq.

Software integration can be eased by grouping drivers in uniquely named headers and C files that are then saved together with the IP. This way every team member can have access to the resources

and import them easily, avoiding conflicts. For the sake of readability we are going to uniform the developed C code via adopting stake case style.

Once reached this point we can evaluate to develop also some stretch goals such as touch TFT display or dual power settings (high performance and low power).

5.2 Timings – Simone Ledda

A crucial aspect that has to be evaluated during the integration stages is timing, meaning when certain drivers will be called to fulfil the wanted action by the user. Though we are trimming some project details we are confident about the main tasks of the software, for sure the file compression will occur in a later stage rather than live as the user is recording. Other important aspects may be the display transitions and menus updates upon selection, effect sharp enable or disable and many other. Power-on and other main UI interactions will be adjusted to guarantee the smoothest user experience possible.

5.3 Logistic – Simone Ledda

Logistic involved in the integration of the various parts of the project wasn't mentioned too far. Every team member is working independently on the assigned parts, but the IPs are tested and stored on Google Drive. This platform allows us to work flexibly and also to keep version control over the developed files together with weekly meetings updates and documentation.

6 System Acceptance – Matt Reynolds

The media player will undergo system acceptance testing, which will indicate to the group whether it is fit for market and meets the customer's requirements. This test will be based on the product specifications, which have already been agreed with the customer. Test equipment will involve only the product, as the user would receive it.

System acceptance will be passed if the product meets the following:

- Input-output audio delay is less than 15 ms
- The audio effects enable and disable in real-time
- Live recordings successfully save to the microSD card
- Saved files can be played back from the microSD card
- The TFT display outputs text-based menus
- Menu response time is less than 0.5 s
- All audio effects control parameters are fully operational

Furthermore, the user-interface should feel smooth and consistent. Each menu and sub-menu should have a similar response time and be free of noticeable delay. The audio should be clear on the headphones output, without undesired distortion or noise. The volume levels should be appropriate for the product, ensuring the audio is clearly audible. Live recordings should be accurately captured by the media player and when played back should appear to the user as an exact recording of what they have just played.

While some of these features are difficult to categorise numerically, it will be down to the expertise of the development team to agree whether these features have been met. There contains some subjectivity to these measurables and it is expected that the team would negotiate with the customer, when delivering the final product, on the successfulness of meeting the requirements.

BitVia remain a company who wish to uphold a reputation for high-standards and as such, feel that should this product pass their system acceptance tests, it will certainly meet and exceed the customer's requirements.

7 References

[1] M. Reynolds, G. Johnston, S. Ledda and P. Wang, “Quality Assurance Manual,” University of York, York, 2020.

[2] D. Kumar, “Software Engineering | SDLC V-Model,” Geeks for Geeks, 17 June 2020. [Online]. Available: https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/. [Accessed 17 June 2020].

[3] Analog Devices, Inc., “Analog Devices, Inc.,” 2018. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ADAU1761.pdf. [Accessed 23 06 2020].

[4] TUL Technology Unlimited, “TUL Technology Unlimited,” 19 03 2018. [Online]. Available: http://www.tul.com.tw/download/TUL_PYNQ%20Schematic_R12.pdf. [Accessed 23 06 2020].

[5] Total Guitar, “Guitar Pedal Order: How to organise your pedal board,” Music Radar, 10 06 2020. [Online]. Available: https://www.musicradar.com/how-to/guitar-pedal-order-how-to-organise-your-pedalboard-signal-chain. [Accessed 17 06 2020].

[6] A. Sharma, “The 10 best delay pedals 2020: our pick of the best delay guitar effects from Boss, Strymon, Electro-Harmonix and more,” Music Radar, 19 02 2020. [Online]. Available: https://www.musicradar.com/news/best-delay-pedals. [Accessed 17 06 2020].

[7] N. T, “Direct Memory Access (DMA),” Binary Terms, 10 October 2019. [Online]. Available: https://binaryterms.com/direct-memory-access-dma.html. [Accessed 20 June 2020].

[8] Farlex Inc., “Direct Memory Access,” [Online]. Available: https://encyclopedia2.thefreedictionary.com/Direct+Memory+Access. [Accessed 21 June 2020].

[9] Xilinx, “Zynq-7000 SoC Technical Reference Manual,” 1 July 2018. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf. [Accessed 20 July 2020].

[10] C. Hoffman, “What's the difference between FAT32, ExFAT and NTFS,” 30 March 2018. [Online]. Available: https://www.howtogeek.com/235596/whats-the-difference-between-fat32-exfat-and-ntfs/. [Accessed 21 June 2020].

[11] Xilinx, “AXI Video Direct Memory Access v6.3,” Xilinx, San Jose, 2017.

[12] Xiph.org Foundation, “flac,” 2014. [Online]. Available: https://xiph.org/flac/. [Accessed 21 June 2020].

[13] A. S. Nastase, “How to Derive the RMS Value of a Triangle Waveform,” Mastering Electronics Design, 1 September 2014. [Online]. Available: https://masteringelectronicsdesign.com/how-to-derive-the-rms-value-of-a-triangle-waveform/. [Accessed 23 June 2020].

8 Appendix A

8.1 User Manual – Pengan Wang

Dear users:

Thank you for choosing BitVia products. In order to make it easy for you to use this product as soon as possible, please read this user manual carefully before using this machine.

Cautions for Users:

Do not disassemble the product without permission, which may cause electric shock or equipment damage.

Do not wash the product or product parts with water.

This product may not work normally when it is affected by the magnetic field, radio wave, etc.

Product Features:

Support for multiple languages.

Supports multiple sound effects selection.

Supports recording in various formats.

Support simultaneous recording of vocals and instruments.

The input/output audio is in mono.

Recording Settings:

Users can adjust three different sound effects: DISTORTION, TREMOLO and DELAY.

Distortion:

Users can choose to turn this effect on or off at the next level interface.

Users can choose the Distortion type: Distortion or Overdrive.

Users can adjust the value of Gain and Distortion from 1 to 20.

TREMOLO:

Users can choose to turn this effect on or off at the next level interface.

Users can adjust the value of Depth, Frequency and Aggression from 1 to 20.

DELAY:

Users can choose to turn this effect on or off at the next level interface.

Users can adjust the value of Delay time and Echo decay from 1 to 20.

Switches and Button setting:

Users can push PB4 and PB3 to change which option they want to choose. Then push PB2 to select into the next user interface.

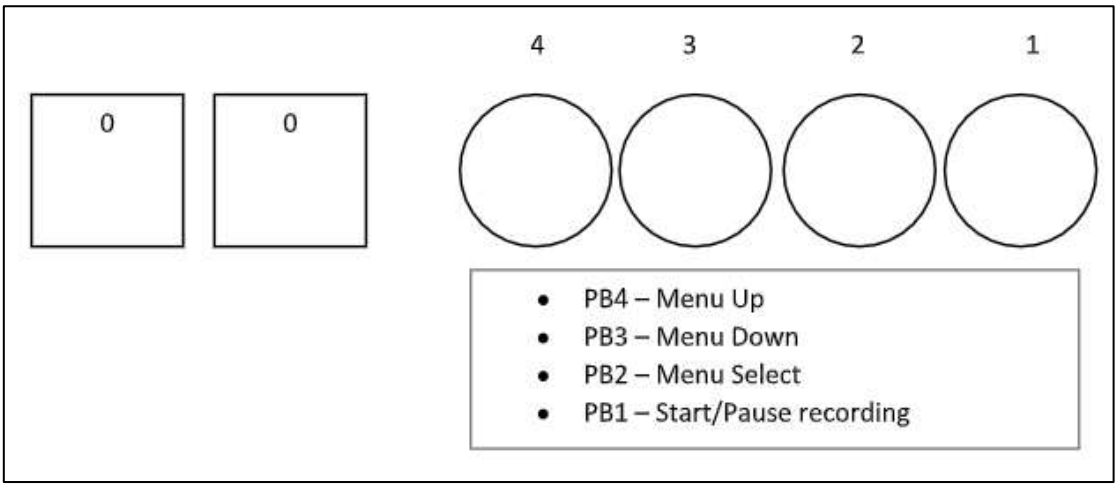

Figure 58 - Push buttons / Switches Interface

Users can push PB4, PB3, PB2 and PB1 to choose different effect groups in the FX group.

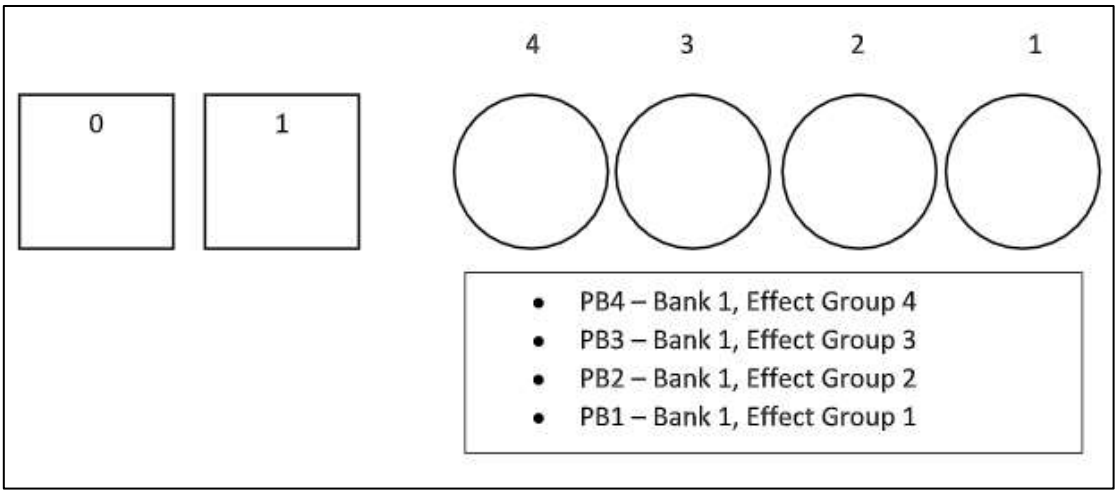

Figure 59 - Push buttons / Switches Interface

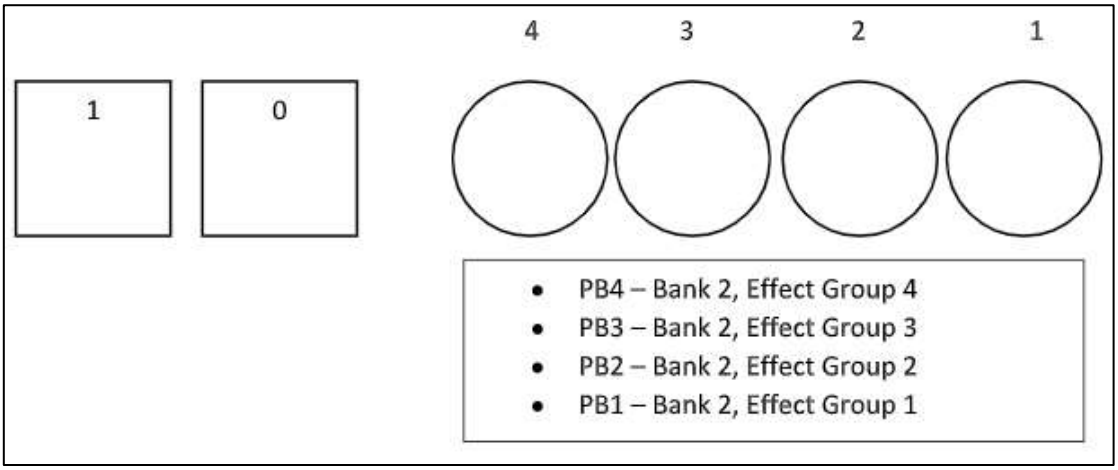

Figure 60 - Push buttons / Switches Interface

When users finish their record, they can push PB4, PB3, PB2 and PB1 to store their recorded file. Then, users can find their recorded files in the “BACKING TRACKS” option in the MAIN MENU. (NOTE: the format of the microSD must be in FAT32 and that the audio files must be FLAC.)

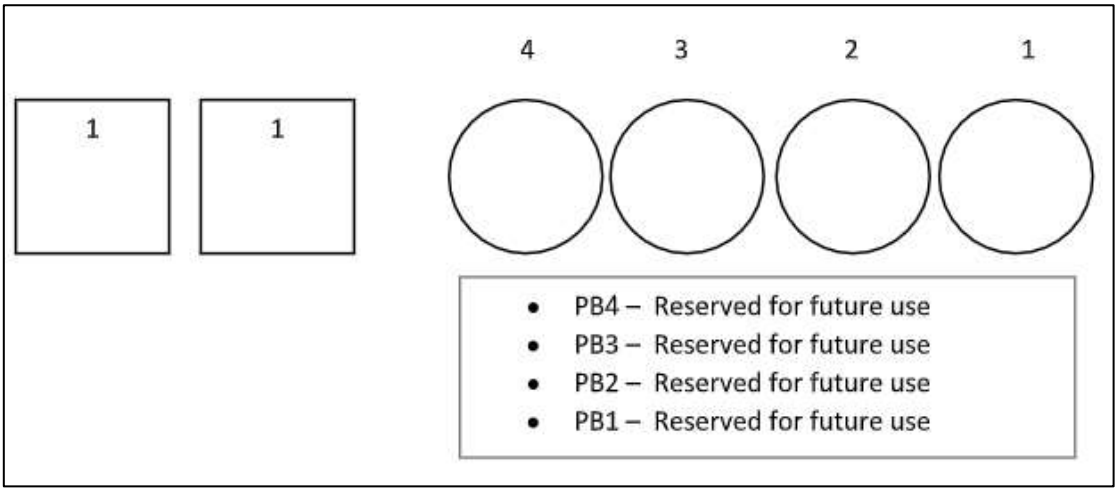

Figure 61 - Push buttons / Switches Interface

Store Settings:

Users will be able to find the audio files they have already recorded in the Backing Tracks folder.

Users have the option of deleting recorded audio files.

Language Settings:

In the language user interface, the user can choose three different languages as the system language.-Chinese English Italian.

After-sales service and maintenance:

If the product fails due to the components of the machine, the user may choose to return the product, replace it or maintain it within seven days after purchasing the machine. Within 15 days of purchase, users can choose to replace the machine or repair it. Users can enjoy free maintenance service within 6 months after purchase.

9 Appendix B

9.1 Software Test Template – Simone Ledda

Software Test Sheet

MSc in Digital Systems Engineering
Department of Electronics
University of York

Group Project

Contents

1 Test

2 Tester name

3 Test date start

4 Test date finish

5 UUT typology (VHDL entity / custom IP / subsystem)

6 UUT name

7 Software/driver functionality description

8 Objective of test

1.
2.
3.

The black-box testing strategy is used for this test.

9 Test results

10 Observations

11 Grade (pass/fail)

12 Software manager approval signature

9.2 Hardware Test Template – Simone Ledda

Hardware Test Sheet

MSc in Digital Systems Engineering
Department of Electronics
University of York

Group Project

Contents

1 Test

2 Tester name:

3 Test date start:

4 Test date finish:

5 UUT typology (VHDL entity / custom IP / subsystem)

6 UUT name

7 Hardware block design screenshot

8 Objective of test

1.
2.
3.

The following test is ran in compliance with the V-model testing strategy

9 Testbench description / test strategy overview:

10 Test results:

11 Observations:

12 Grade (pass/fail):

13 Hardware manager approval signature: