Institutionen för systemteknik
Department of Electrical Engineering

Examensarbete

USB 2.0 Audio device

Examensarbete utfört i Elektronikkonstruktion
vid Tekniska högskolan vid Linköpings universitet
av

Johan Österberg, Carl-David Ekblom

LiTH-ISY-EX-ET--12/0389--SE

Linköping 2012

Linköpings universitet
TEKNISKA HÖGSKOLAN

Department of Electrical Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings tekniska högskola
Linköpings universitet
581 83 Linköping

USB 2.0 Audio device

Examensarbete utfört i Elektronikkonstruktion
vid Tekniska högskolan i Linköping
av

Johan Österberg, Carl-David Ekblom

LiTH-ISY-EX-ET--12/0389--SE

Handledare: **Jonny Lindgren**
ISY, Linköpings universitet
Markus Svensson
Syncore Technologies

Examinator: **Jonny Lindgren**
ISY, Linköpings universitet

Linköping, 04 November, 2012

Avdelning, Institution
Division, Department

Division of Automatic Control
Department of Electrical Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

Datum
Date

2012-11-004

Språk
Language

□ Svenska/Swedish
☒ Engelska/English

□ ________________

Rapporttyp
Report category

□ Licentiatavhandling
☒ Examensarbete
□ C-uppsats
□ D-uppsats
□ Övrig rapport
□ ________________

URL för elektronisk version

`http://www.control.isy.liu.se`
`http://www.ep.liu.se`

ISBN
—

ISRN
LiTH-ISY-EX-ET--12/0389--SE

Serietitel och serienummer
Title of series, numbering

ISSN
—

Titel
Title

USB 2.0 Ljudenhet
USB 2.0 Audio device

Författare
Author

Johan Österberg, Carl-David Ekblom

Sammanfattning
Abstract

The main task of this project were to develop, hardware and software that could stream audio data via USB 2.0. This project were based on XMOS, USB 2.0 design. In this project we have brought an idea to reality in the form of a finished product. This with verification help from engineers on Syncore technologies. Under the development process the functionality surrounding component databases, provided by Altium designer, were to be evaluated. To be mentioned is that Altium designer was the software used to develop the PCB in this project.

After many hours spent developing, we finally got the hardware and software to behave in the way it was suppose to do. That is, to be able to stream audio data from a high-resolution source(PC/MAC/unit with S/PDIF out, maximum resolution 24-bit 192 kHz). This to both S/PDIF and analog stereo out via RCA-connectors. The sound quality from a possible subjective point of view is very good and we are happy with the result.

We think that the functionality surrounding component databases are convenient in many applications. Not just the fact that you easily can generate an up to date pricing of all components used in a project, you can also shorten the development process. This because the developer don't have to recreate schematic symbols and footprints that has already been created. Which of course was the fundamental idea behind the database functionality. These are just a few examples of its advantages. To be considered is the fact that the administration surrounding the component databases can be very time consuming. To take full advantage of Altium designers functionalities we think that it needs a dedicated administrator that maintains the database repository.

Nyckelord
Keywords USB 2.0, Audio, PCB

Abstract

The main task of this project were to develop, hardware and software that could stream audio data via USB 2.0. This project were based on XMOS, USB 2.0 design. In this project we have brought an idea to reality in the form of a finished product. This with verification help from engineers on Syncore technologies. Under the development process the functionality surrounding component databases, provided by Altium designer, were to be evaluated. To be mentioned is that Altium designer was the software used to develop the PCB in this project.

After many hours spent developing, we finally got the hardware and software to behave in the way it was suppose to do. That is, to be able to stream audio data from a high-resolution source(PC/MAC/unit with S/PDIF out, maximum resolution 24-bit 192 kHz). This to both S/PDIF and analog stereo out via RCA-connectors. The sound quality from a possible subjective point of view is very good and we are happy with the result.

We think that the functionality surrounding component databases are convenient in many applications. Not just the fact that you easily can generate an up to date pricing of all components used in a project, you can also shorten the development process. This because the developer don't have to recreate schematic symbols and footprints that has already been created. Which of course was the fundamental idea behind the database functionality. These are just a few examples of its advantages. To be considered is the fact that the administration surrounding the component databases can be very time consuming. To take full advantage of Altium designers functionalities we think that it needs a dedicated administrator that maintains the database repository.

Sammanfattning

I detta projekt har hårdvara samt mjukvara utvecklats som möjliggjort överföring av ljuddata över USB 2.0. Ett projekt som byggt på XMOS, USB 2.0 design. Denna utvecklingsprocess har utgått från en idé till färdig produkt med verifieringshjälp av ingenjörer på Syncore technologies. Under utvecklingsprocessen har även Altium designers funktionalitet gällande databashantering av komponenter utvärderats. Altium designer är det utvecklingsverktyg som använts för att producera PCB-kortet under detta projekt.

Efter många timmars utvecklande började hårdvara samt mjukvara utföra det den var tänkt att göra. Det vill säga, leverera ljud från en högupplöst ljudkälla, PC/MAC(över USB 2.0) eller S/PDIF. Detta med en maximal upplösning på 24-bit 192 kHz. Ljuddatat har sedan behandlats och vidarebefodrats till både

S/PDIF samt analog stereo ut via RCA-kontakter. Ljudkvaleliten från en möjligen subjektiv synvinkel låter mycket bra.

Funktionaliteten gällande databashanteringen anser vi vara mycket användbar i många avseenden. Inte minst när det gäller att snabbt ta fram en kostnadskalkyl för komponenter i ett projekt, utan framförallt i sparad utvecklingstid. Detta då man slipper reproducera schemasymboler samt footprints om de skapats förut. Något som givetvis var grundidén kring detta konsept. Faktum kvarstår att det krävs ett visst underhåll av databaserna. Detta resulterar i merarbete och i vissa fall kan det krävas en heltidstjänst för att få det att fungera på ett önskvärt sätt.

Acknowledgments

We would like to thank: the Syncore technologies corporation along with its employees for making this project possible. We would like to give special thanks to our supervisor Markus Svensson for all the help surrounding hardware and software. Last but not least we want to thank our examiner Jonny Lindgren.

Contents

Chapter 1

Introduction

The main goal behind this project were to build a USB 2.0 audio device and to test some of Altium designers functionalities surrounding component databases. Altium designer is a PCB development environment which have been used in this project. The version of Altum designer used in this project is 10. This project uses technology provided by XMOS to stream USB Audio class 2.0 and 1.0 data via their XS1-L01A-TQ128-C5 chip. Their default driver supports OS X version 10.6.3 and above. The Windows drivers are provided by Thesycon, CEntrance and Polytec which supports WDM/Direct X and ASIO 2.1. The OS X drivers are free of charge whereas the Windows drivers are not.

This documentation will start with a brief introduction to some fundamental building blocks used when developing PCB cards. This part is included in this document to simplify the explanation of Altium designers provided functional surrounding component databases. The database created during this project includes all used components in the USB 2.0 project.

In section 4 on page 13 the actual PCB design is discussed. This documentation have focused on the interesting parts of the PCB design and left the most fundamental parts out. This document ends up with a discussion of the project result and our opinion regarding the component database tools.

The schematic sheets used in this project are attached in the appendix part of the document for more detailed information surrounding the interconnections.

Chapter 2

Component representation

When designing electronic devices there must be a way to represent components and their behaviour in different contexts in the design process. Altium designer provides a variety of ways to do this and will be explained in this chapter. The way Altium designer represents components are extremely usual for similar tools.

2.1 Simulation model

Altium has a mixed signal simulator supporting PSpice models. This representation of a component is associated with the schematic symbol[9]. This type of representation is used for verifying signals and there behaviour.

2.2 Schematic symbol

A schematic symbol is a logical representation of a component, for example a resistor. The schematic symbol is used when creating connectivity drawings. This component representation is used early in the design process and defines the interconnections on the PCB. A schematic symbol usually contains all pins of the physical component that it represents. Each pin has a associated designator. The connectivity between components are defined by nets connected to schematic symbol pins. This information is later exported to the PCB environment in the design process. When doing this export, the schematic symbol maps to its corresponding footprint. The footprint pad-designators must have the same designators as the pins used in the schematic symbol to maintain desired connectivity.

To a schematic symbol one can add parameters[9]. Some examples are supplier links, part numbers and voltage ratings. Schematic symbols and their associated parameters will be discussed later in chapter 3 when these parameters are to be stored in databases.

2.3 Footprint

When developing electronic devices the components used in a project must have some kind of representation on the actual PCB. This is done via so called footprints. A footprint contains information of including pads/vias used as electrical terminals between the component and the interconnections on the PCB[6]. When designing a footprint there are many aspects that must be taken into consideration. The IPC(Association Connecting Electronic Industries) standards has recommendations how components should be soldered. A good design guideline is to use the IPC standards.

Footprints can both be of SMD(surface mount device) or through-hole type. Components of SMD type are mounted directly on the surface of the PCB. The counterpart, through-hole components requires drill holes in the PCB where the component pins can be inserted. The component manufactures has agreed on some standards surrounding footprints to decrease the development time. Some examples of commonly used footprints are 0603, TO-92 and SOIC-8.

2.3.1 Designing a footprint

There are some guidelines regarding size and shape of the footprint, pads/holes provided by the component manufacturer. These recommendations are often found in the component datasheet. These recommendations are usually based on the IPC standards. If there are no guidelines the designer has to take the size and shape decisions of pads/holes depending on the application.

One important part of a footprint is the silk-screen. This is an informative layer that often contains the outline of the component. If orientation is important a small dot is often used to mark pin number 1. To this layer other guidelines can be added to help the assembly process. Another important part of a footprint is the pads/holes. Pads are shapes of conductive material which the component usually will be soldered onto during manufacturing. Pads can be of SMD or through-hole type discussed earlier in section 2.3. The pads are designed to be larger then the actual size of the component legs or contact surface of the component. This to allow solder to flow out, giving larger connection area and a stronger connection between the component and the PCB. This is a important aspect if the PCB are to be placed in a environment with a lot of vibrations.

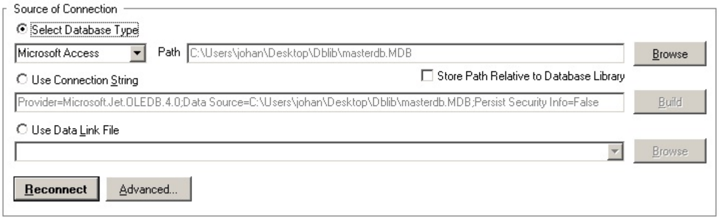
Chapter 3

Component database

This chapter will go into the details of the tools provided by Altium designer used to simplify the usage of component databases. The way of representing components were discussion in chapter 2. The data types used to represent components in different contexts are the information to be stored in the databases.

3.1 Database linking

To make use of external data collections of components, Altium designer must provide some sort of linkage to the data source. To establish the linkage Altium have introduced two file types, *.DBLib and *.DBLink[2]. There exists an extension to the *.DBLib file called *.SVNDBLib which have added functionality for subversion treatment. Both file-types has the capability of linking to a database either on a local or network hard drive. The only restriction on the connection is that the database has to support OLEDB or ODBC interface[12]. Altium uses MicrosoftÂ´s OLEDB as linkage layer to the databases as default.



Source of Connection

● Select Database Type

Microsoft Access

Path C:\Users\johan\Desktop\Dblib\masterdb.MDB

○ Use Connection String

□ Store Path Relative to Database Library

Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Users\johan\Desktop\Dblib\masterdb.MDB;Persist Security Info=False

○ Use Data Link File

Figure 3.1: This image explains the database connection opportunities provided by Altium designer.

In figure 3.1 the connection dialogue starts by providing functionality to browse for a specific database. It has to be of either Microsoft Access or excel type. When selecting a database source Altium automatically creates a connection string in the second text-field(OLEDB syntax). If the database source are neither of Microsoft

Access or excel type this field could be used to write a custom connection string. The last connection alternative is to use a Microsoft Data Link file (*.udl). This is simply a vessel for a connection string. Figure 3.2 gives an overview of how multiple users could connect to the same network database. However there could not be more than one database connection established per *.DBLib or *.DBLink file.

3.1.1 *.DBLib and *.DBLink

Both the *.DBLib and *.DBLink file uses the connection methodology to connect to the external or local component database found in figure 3.2. The connection could be established using one of the alternatives in section 3.1. The main thing that distinguishes these file types from each other are the fact that the *.DBLib file could be used to place components directly from the database into a schematic file[10]. This is not possible for the *.DBLink file. The *.DBLink file is used to synchronize specific component parameters with the matching database record. A property that the *.DBLib file also holds. One example of a commonly associated parameter to components are the current pricing. A parameter that could be used to calculate the costs of the whole project, often refereed as BOM(Bill of material). When placing components directly into a schematic using the .*DBLib connection the schematic symbol location have to be known by the database. This implies that the database has to store information where to find the specific schematic symbol. Another big difference between the *.BDLink and the *.DBLib file when used in a PCB project is that the *.DBLib file has only to be included as a library whereas the *.DBLink file has to be included into the project as a physical document part.

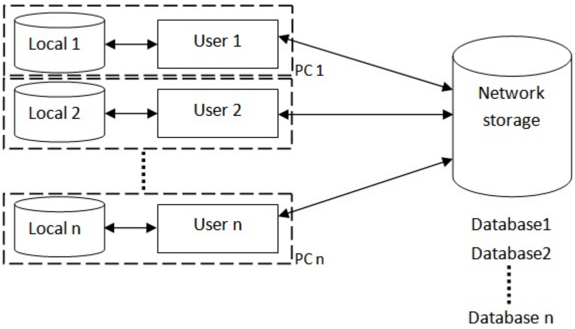
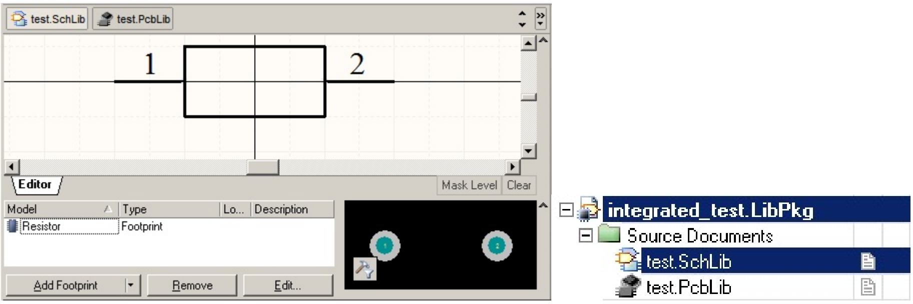


Figure 3.2: This image gives an overview of the connection opportunities to databases.

3.2 Database creation

Altium provides functionality to create databases from integrated libraries. The tool used to create these databases is called ”integrated library to database library translator wizard”. To be mentioned is that databases could also be created manually without using the wizard. This is just a way to simplify the database creation process and a good way to illustrate the oncoming sections using the database. A integrated library combines schematic libraries with their related PCB footprints and/or SPICE and signal integrity models. In this example a simple component named resistor was created with a footprint for its PCB layout.

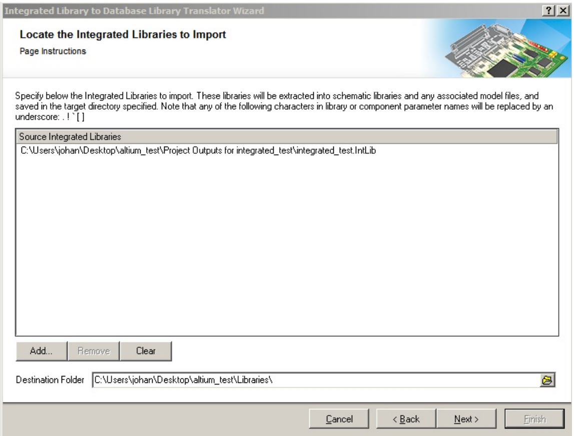


Model	Type	Lo...	Description
Resistor	Footprint		

(a) Component example.

(b) Integrated library hierarchy.

In figure 3.3b an integrated library file was created and the schematic and PCB library was imported to that project. When right clicking at the integrated_test.LibPkg file the opportunity to
”Compile Integrated Library integrated_test.LibPkg” comes up. When compiling the project a new file is created, named integrated_test.IntLib which could be used in a project without having to include the schematic and PCB representation of the component. It is now possible to use the earlier mentioned wizard to convert this integrated library to an Access database. First a *.DbLib file has to be created and from there start the wizard via tools»”Import From Integrated Libraries”. The fist window that appears gives you the opportunity to create a new database file or append an existing one. When passing this step you are about to choose which *.IntLib files you want to include in the database.



Integrated Library to Database Library Translator Wizard

Locate the Integrated Libraries to Import

Page Instructions

Specify below the Integrated Libraries to import. These libraries will be extracted into schematic libraries and any associated model files, and saved in the target directory specified. Note that any of the following characters in library or component parameter names will be replaced by an underscore: . ! ` [ ]

Source Integrated Libraries

C:\Users\johan\Desktop\altium_test\Project Outputs for integrated_test\integrated_test.IntLib

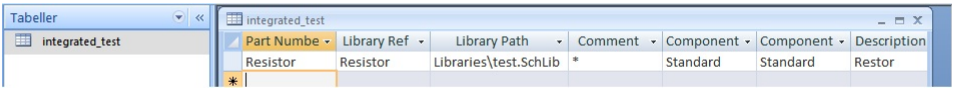
Destination Folder: C:\Users\johan\Desktop\altium_test\Libraries\

Figure 3.3: This is the part of the database creation wizard in which the integrated libraries to import are chosen.

In figure 3.3 the created integrated library is chosen as source to base the new database on. When this is done it is just to finish the database creation wizard. The *.DbLib file used to create the database has automatically created a link to the newly created database. This is done using the connection dialogue in figure 3.1.

3.3 Database structure

This section will explain the structure of the component databases. The database created in the previous section 3.2 are viewed in figure 3.4.



Tabeller

integrated_test

integrated_test

Part Numbe	Library Ref	Library Path	Comment	Component	Component	Description
Resistor	Resistor	Libraries\test.SchLib	*	Standard	Standard	Restor
*						

Figure 3.4: This figure displays the created Microsoft Access test database *.MDB

On the left side of the picture there is a area called ”Tabeller” in which the name of the integrated library recently created in section 3.2 are listed. If we in the database creation wizard would have included more integrated libraries they would appear as separate tables in this area. At the right side the content of the selected table is listed. Each row in the table represents a component and each column the parameters associated with each component. Common parameters are manufacture part number, schematic symbols, footprints, pricing and value to mention a few. The parameters attached to schematic symbols was discussed in chapter 2 on page 3. The parameters associated to each component in the database will be added to the schematic symbol if the component is placed onto a schematic via the *.DBLib or *.SVNDBLib file. This procedure are discussed in section 3.1.1. To be able to distinguish a specific component from the database each component has to have a unique number associated with it[10]. It is up to the administrator to select this number. In this test project the manufacturer part-number was chosen as the unique identification number for each component.

3.4 Database administration

This section will describe how databases can be administrated. As mentioned in section 3.1 the files used to establish linkage between databases have in common that they all use the ODBC or OLEDB(default) layer to communicate with the database.

3.4.1 Administration via Altium

To change parameter values, delete and add components in the database. Altium designer provides a GUI with a common structure as the one used in Microsoft Access. The GUI window is found in figure 3.5. This interface appears when exploring the *.DBLib or *.DBLink file.

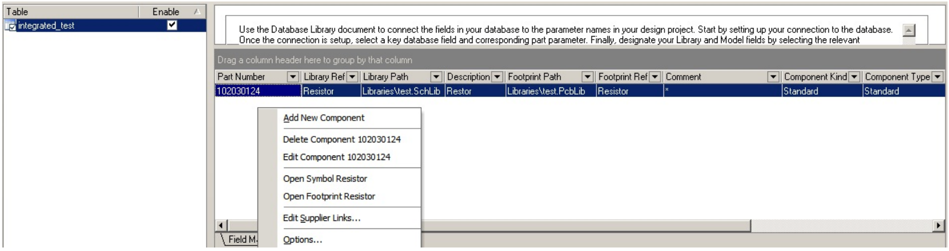


Table	Enable
☑ integrated_test	☑

Use the Database Library document to connect the fields in your database to the parameter names in your design project. Start by setting up your connection to the database. Once the connection is setup, select a key database field and corresponding part parameter. Finally, designate your Library and Model fields by selecting the relevant

Drag a column header here to group by that column

Part Number	Library Ref	Library Path	Description	Footprint Path	Footprint Ref	Comment	Component Kind	Component Type
102030124	Resistor	Libraries\test.SchLib	Restor	Libraries\test.PcbLib	Resistor	*	Standard	Standard

Figure 3.5: This image views the database administration GUI provided by the Altium designer environment.

This administration environment gives the user the possibility of changing all fields in the database records within Altium. This GUI also gives the user the possibility to add and delete components(rows) and parameters(columns) in the associated database[12]. This holds true for both the *.DBLink and *.DBLib file. An change in the GUI window gives an immediate change in the corresponding database. Then a question arise, what happens if there are multiple users connected to the same network database? This bad way of using a database could lead to race conditions and non-deterministic results. This problem are resolved by Altium via the earlier mentioned file-extension *.SVNDBLib, mentioned in section 3.1. This brings us to the next section, 3.5.

3.5 *.SVNDBLib

This file type are provided by Altium to give the user the capability to treat the database under subversion. This by either the built-in subversion tool in Altium designer or a third-party subversion tool. Under this test project, TortoiseSVN was used. Which is a commonly used subversion software. There exists two user-modes when working with version-controlled database libraries. These two modes differs in there permissions. These are mentioned and described in the table below.

Mode:	Description:
Designer	As the mode depicts this role are created for the designers in a working group. This role gives the user the rights to checkout, open and modify library content, but is prevented from committing the change to the repository[2].
Librarian	This mode has all the access rights as the designer, with the added permission to commit changes to the repository[2].

One big difference between the *.DBLib and the *.SVNDBLib are that the schematic and footprint symbols has to be split into separate libraries when using the *.SVNDBLib file. This is not the case using the *.DBLib file where all schematic symbols and footprints could be contained in one schematic respectively one pcb library. Altium provides a library splitting tool that divide schematic- and pcb-libraries including multiple schematic symbols and footprints into separate schematic- and pcb-libraries. In order to be able to manipulate the schematic symbols and footprints when using a subversion database, a local copy of the database repository has to be checked out. The copy is placed in a specific folder called ”working folder” in Altium[2]. When changes to existing schematic symbols or footprints are made, or additional data are added. These updated parts has to be checked in. This to give other users of the database the capability to use the newly modified or added schematic symbol or footprint. The subversion treatment also applies to parameters associated with the components in the database. Altium has an additional tool that are used to report the data status of all components in the database. This tool displays the last modification-date and the current reversion of each component. The tool either uses the built in or a third-party subversion handler to compare the component data in the repository with the local copy. The comparison could give three outcomes which are listed below.

- Conflict
- No modification
- Out of date

The ”conflict” state appear when an change to any schematic symbol or footprint has been made on the local copy that has the same reversion as the one in the repository. To resolve this conflict the modifications has to be committed to the repository. The ”no modification” indicates that the local copy has the same reversion as the counterpart in the repository. If there exists multiple librarians in the administrative group there will be times when the reversion of the local copy for some users will be ”out of date” which is the last state. This state as discussed appears when there have been a changes committed to the main repository. In order to modify this ”Out of date” schematic symbol or footprint the local repository copy has to be updated. Otherwise an commit command would be ignored. When subversion controlling the repository containing the database the earlier mentioned non-deterministic race hazard is resolved.

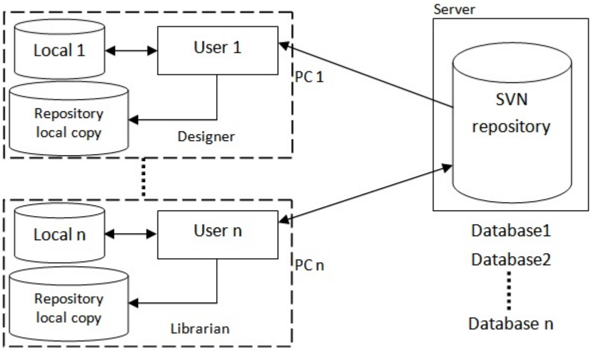


Figure 3.6: This image views the SVN repository administration overview.

Chapter 4

USB Audio 2.0 device

This chapter will explain the interesting hardware and software parts in the USB audio 2.0 device project. All schematic files used in this project are attached in the appendix part of this document for a total overview of the design. The schematic sheets starts at page 47. The datasheets for all components could be found at the manufacturers web-pages for additional information if needed. The audio chain is discussed in the same order as the data flow through the system, starting from the host computer. The data flow is found in figure 4.1.

4.1 Audio data

Figure 4.1 gives an overview of how audio data are distributed over the system and in which format it is transmitted. Thickened arrows represents bus-connection whereas thin arrows represents single transmission lines.

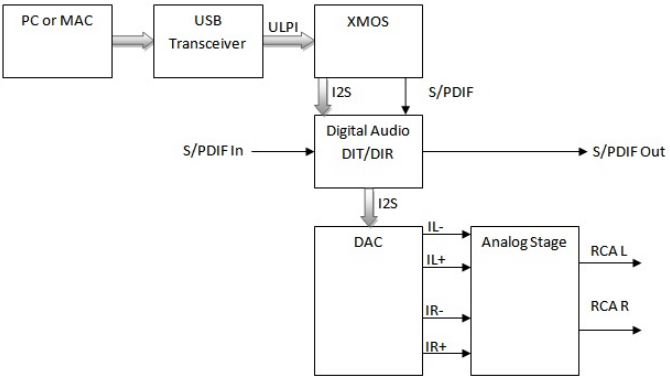


Figure 4.1: Overview of the audio data connections between subsections.

4.1.1 USB3318

This hi-speed USB 2.0 transceiver is used to convert serial USB-data from PC or MAC to the industry standard UTML+ low pin interface(ULPI). This to distribute the data to the XS1-L01A-TQ128-C5 processor. This interface uses a method of in-band signalling where data and status byte transfers are sent on the same bus. The configuration of the USB3318 chip was setup from the information given in the datasheet. Table 4.1.1 gives an explanation why this setup was chosen. Not all pins are mentioned because they are either self explanatory or belongs to communication protocols that are not discussed in this document. The data rate between the USB3318 chip and the connected PC/MAC is 480Mb/s.

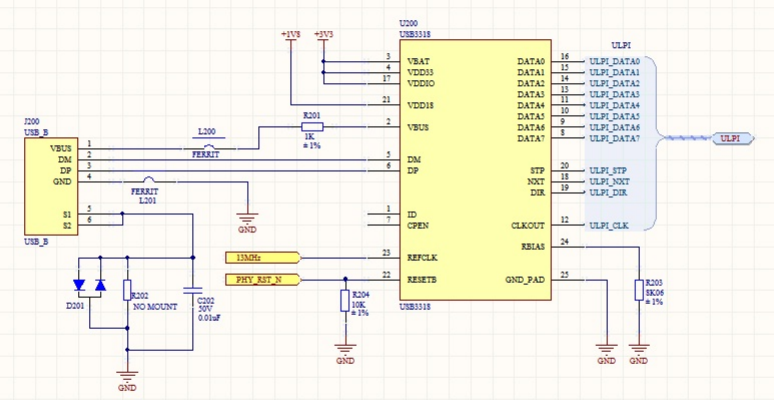


Figure 4.2: Schematic drawing of the USB 2.0 transceiver.

[Pin:name]	Property:
1:ID	If this chip would be used as an OTG sytem(On-The-Go) where it could act as a ”master”, an ID is required. In this project no such functionality has to be supported. This pin is left floating.
2:VBUS	This is the +5V provided by the attached ”master”. Resistor R201 and a ferrit bead L200 are added to protect against transients and to increase the impedance at high frequencies. This to filter out high noise components which is common in these applications.
7:CPEN	This signal is used when a VBUS switch is present. In this design no such switch is used. This pin is left floating.
17:VDDIO	Sets the ULPI interface supply voltage. Supported range is $+1.8V \geq$ VDDIO $\leq +3.3V$, $+3.3V$ was selected in this design.
22:RESETB	This pin is active high. For that reason a pull down resistor is added to ensure that it is in a well defined low state until the XS1-L01A-TQ128-C5 processor activates it.

4.2 XS1-L01A-TQ128-C5

This processor is the fundamental building block in this design. It handles all USB 2.0 audio data from the host computer. The XS1-L01A-TQ128-C5 chip is a event driven, multi threaded, single core processor with 64 kB internal RAM and 8 kB OTP. The software provided by XMOS supports all sample rates listed blow. This in either 16- or 24-bit format.

- 44.1 kHz
- 48 kHz
- 88.2 kHz
- 96 kHz
- 176.4 kHz
- 192 kHz

4.2.1 Hardware

This section will dig deeper into the hardware configurations of the XS1-L01A-TQ128-C5 processor. This chip will sometimes be referred as XMOS from now on. The interesting and important hardware configurations are highlighted. In figure 4.3 on page 15 the ”Config”-block represents the configuration pins of the XMOS processor. The highlighted signal-paths to this block goes to the MODE[2], MODE[3] and TRST_N pin. Table 4.1 gives information of there properties. In figure 4.3 a burn-out circuit named NCP303 is used to detect a voltage drop below a threshold voltage of +0.9V. This circuit has a active low output at pin 1 which is connected to the NC7WZ07 circuit. This transmission line has a pull-up resistor to keep the signal path in a well defined high state. When the voltage is lower then the threshold voltage the NC7WZ07 forces the TRST_N pin low which resets the XMOS processor. The NC7WZ07 circuit is a dual buffer with open drain outputs.

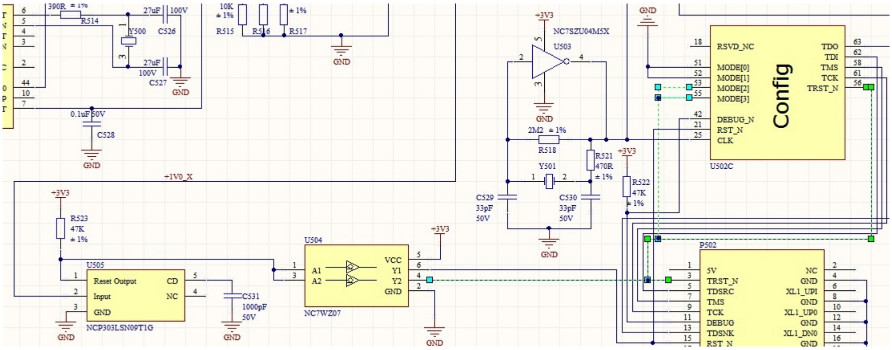


Figure 4.3: Overview of the XMOS configuration block, NCP303L and NC7WZ07 circuits.

MODE[2]	MODE [3]	This pin pair determines which boot source that should be used for the XMOS processor.
0	0	Booted via JTAG.
0	1	Reserved
1	0	Booted via XMOS Link B
1	1	Booted via SPI

Table 4.1: This table describes the functionality of the MODE[2] and MODE[3] pins.

If the XSYS2 adapter is inserted into the P502 header in figure 4.3 the processor will boot via JTAG if TRST_N is asserted low. This pin is asserted low by the programming software run on the programming host computer. If the signal is high the open drain buffer will set the MODE[2], MODE[3] and TRST_N high which forces the processor to boot from external SPI flash memory. These two earlier mentioned alternatives are the one used in this project. To be mentioned is that the XSYS2 adapter is a debug-, programming-tool provided by XMOS. The MODE[0] and MODE[1] in figure 4.3 determine the PLL multiplier value. The external clock feeding the processor is running at 13 MHz. When tying MODE[0] and MODE[1] to ground this configures the multiplier to be (13×30.75)MHz. This implies that the system clock will run at a frequency of ≈ 400 MHz.

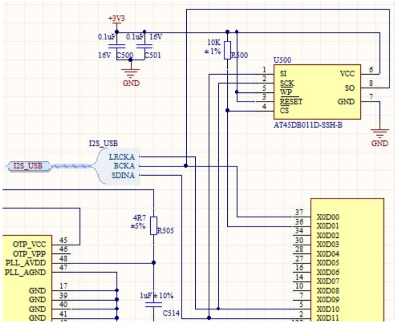


Figure 4.4: Overview of the XMOS external SPI flash memory.

The external SPI flash containing the executable binaries to be run at the XMOS processor are shown in figure 4.4. The pins used to connect the XMOS processor to its external SPI flash are preconfigured by the chip manufacturer. To minimize the number of used pins the I^2S communication to the DIX chip are attached to the same pins used to read and write data to and from the SPI flash. This is possible because there are no more data read from the external SPI flash after booting the XMOS processor.

4.2.2 Software

This section will explain how the software architecture are structured. The code is written in C and XMOS own extension, XC. This extension provides smart functions that utilizes XMOS great way of dealing with concurrency, I/O and time operations. This software is written by the XMOS corporation. In this design only small modifications were introduced to be compatible with the design. In the software provided by XMOS there exists six threads that is used to handle the USB data transmissions. These threads communicate via channels, which are interconnected as in figure 4.5.

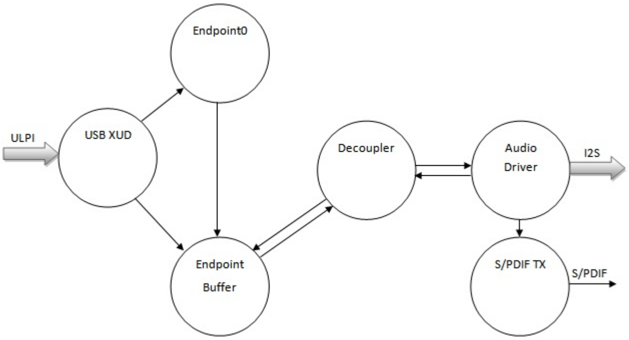
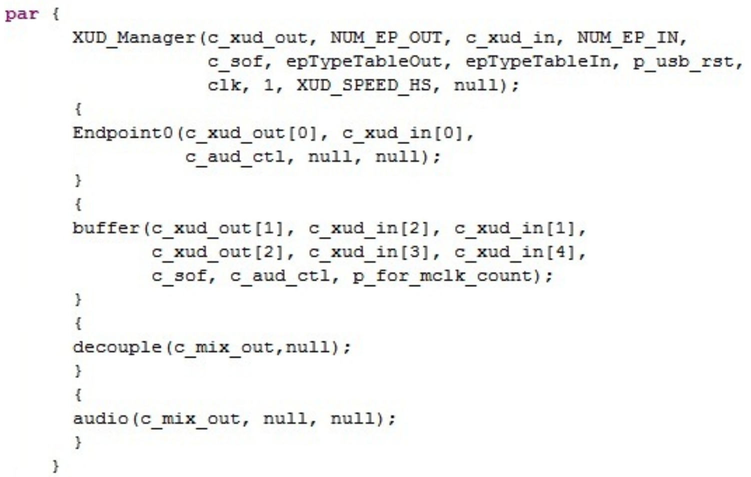


Figure 4.5: This figure gives an overview of the communication paths between the software threads.

In figure 4.6 the code snippet used to initialize and start the threads in figure 4.5 are displayed. The ”par” statement tells the compiler that the functions in the scope encapsulated with curly brackets should be run in separate threads. In figure 4.5 there are six threads displayed but the code in figure 4.6 only starts 5 threads. That is because the S/PDIF thread is started in an similar statement inside the audio thread. This way of initialize the thread hierarchy makes the audio and S/PDIF thread to share $\frac{1}{5}$ of the execution time. The XMOS processor guarantee each thread to be run at a minimum of 80 MIPS when five threads are used. This is a requirement from the USB-XUD thread to operate properly when the system clock speed is 400MHz. A brief description of each treads task are mentioned below.



```
par {
      XUD_Manager(c_xud_out, NUM_EP_OUT, c_xud_in, NUM_EP_IN,
                  c_sof, epTypeTableOut, epTypeTableIn, p_usb_rst,
                  clk, 1, XUD_SPEED_HS, null);
      {
      Endpoint0(c_xud_out[0], c_xud_in[0],
                c_aud_ctl, null, null);
      }
      {
      buffer(c_xud_out[1], c_xud_in[2], c_xud_in[1],
             c_xud_out[2], c_xud_in[3], c_xud_in[4],
             c_sof, c_aud_ctl, p_for_mclk_count);
      }
      {
      decouple(c_mix_out,null);
      }
      {
      audio(c_mix_out, null, null);
      }
    }
```

Figure 4.6: This code snippet starts all threads mentioned in 4.5.

USB XUD thread

This thread controls the low level USB I/O operations.

Endpoint0 thread

This thread controls the management of the USB device. The main part of its tasks are to handle enumerations, resets, audio configuration settings and firmware upgrades.

Endpoint Buffer thread

This thread transmits and receives buffers passed by the decoupler thread.

Decoupler thread

As mentioned in previous section 4.2.2 this thread creates buffers to transmit or receive data through. This thread also determines the size of each packet of audio data sent to the host used to match the audio rate and the USB packet rate.

4.3 STM32F100C4T6B

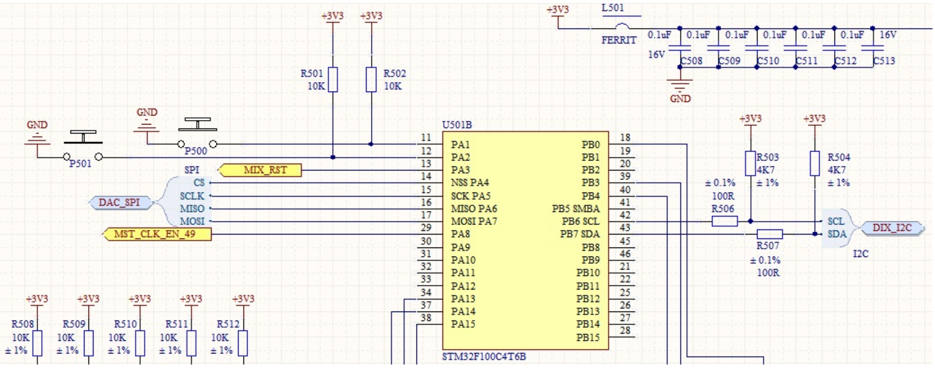


Figure 4.7: Overview of the Cortex chip.

This section will describe the interesting parts of the software and hardware used to run the STM32F100C4T6B processor. This processor has a 16 kB flash memory and 4 kB of SRAM. The processor runs with an internal PLL at 24 MHz, which is three times the external clocks connected to the chip. This processor will be referred as Cortex in some part of this document. The main task of this processor is to configure the digital audio DIT/DIR(DIX4192) and DAC(PCM1792A) circuits. In the schematic the Cortex circuit is divided into two parts consisting of a power/configuration together with an I/O segment. Figure 4.8 displays the power/configuration part whereas figure 4.7 shows the other.

4.3.1 Hardware

Figure 4.7 gives an overview of the I/O pins and what they are connected to. The table below describes the connected pins of interest and what they are used for. The pins that are not in the table are connected to the JTAG header used to write the binaries to the internal flash.

[Pin:Name]	Description:
[11,12]:[PA1,PA2]	To these pins the push buttons are connected. P500 is used to mute the DAC chip and P501 to change input source at the audio multiplexer DIT/DIR.
13:PA3	This pin resets the DAC and DIX chip on demand of the XMOS processor.
[15-17]:[SPI]	This is the SPI-bus used to transfer configuration setting to the DAC chip.
29:PA8	This pin is used to start a 49.152 MHz clock connected to the DAC. This clock is used to achieve the maximum sigma-delta frequency that the hardware of the DAC supports.
18:PB0	This pin is connected to the XMOS processor. This pin goes high when the XMOS processor changes the sampling frequency on demand of the host computer.
[42,43]:[I2C]	These pins are used to send configuration data to the DIX chip via the I^2C protocol.

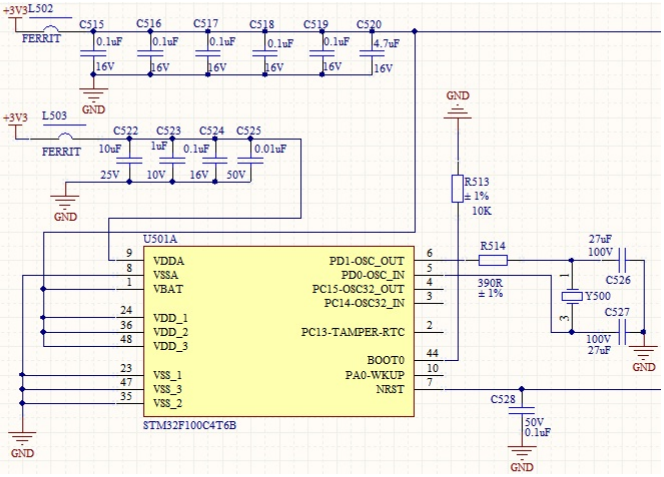
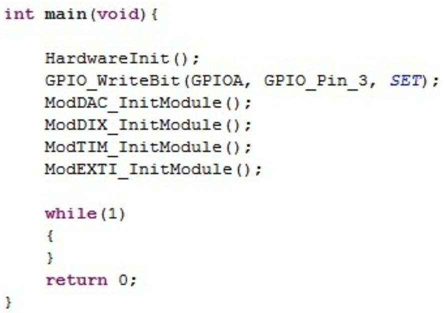


Figure 4.8: This figure shows the power/configuration part of the Cortex processor.

In figure 4.8 pin 44 named BOOT0 are tied to ground to ensure that the processor boots from main flash memory. This is the location where the boot image is located.

4.3.2 Software

This section will explain the interesting part of the software running on the STM32F100C4T6B processor. The code are written in C with use of STMicroelectronics library provided for all their stm32f10x chips. In figure 4.9 the code of the main.c file is displayed. The functions in the main file are executed in sequential order. These code modules are discussed in the oncoming subsections. These subsections will be discussed in the same order as they are called in the main file.

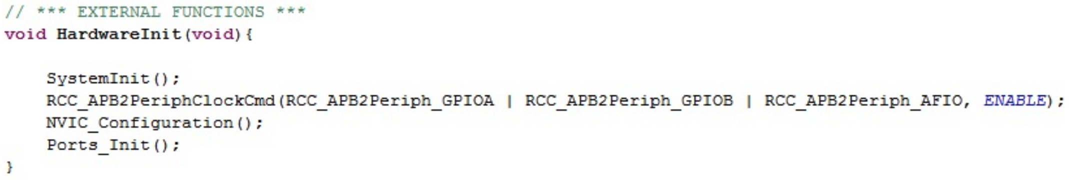


```
int main(void){

    HardwareInit();
    GPIO_WriteBit(GPIOA, GPIO_Pin_3, SET);
    ModDAC_InitModule();
    ModDIX_InitModule();
    ModTIM_InitModule();
    ModEXTI_InitModule();

    while(1)
    {
    }
    return 0;
}
```

Figure 4.9: This figure shows the main function running at the Cortex processor.

HardwareInit()

Figure 4.10 displays the code that executes when calling the HardwareInit() function in the main.c file displayed in 4.9.



```
// *** EXTERNAL FUNCTIONS ***
void HardwareInit(void){

    SystemInit();
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB | RCC_APB2Periph_AFIO, ENABLE);
    NVIC_Configuration();
    Ports_Init();
}
```

Figure 4.10: This figure displays the code that executes when running the HardwareInit() function.

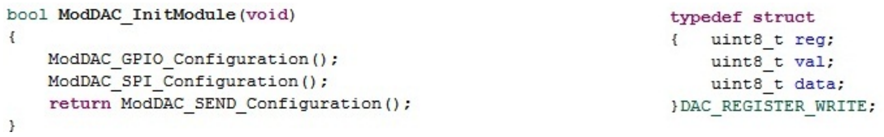
This module starts by calling the SystemInit() function which initialize the embedded flash interface, PLL and updates the system clock. This function is a part of the STMicroelectronics library. When this function has returned the execution proceeds by calling the RCC_APB2PeriphClockCmd() function which enables port A, B and the AFIO peripheral clocks. After this initialization the NVIC_Configuration() function is called. This function initializes the interrupt vector table to be located at the start address 0x8000000. The HardwareInit() function ends by calling Ports_Init() which initializes PA8, PA3 and PB0 pins with its correct performance parameters. These pins are found in figure 4.7.

Table 4.2: This table describes the interesting data bytes sent to the DAC chip.

Register:	Value: [Hex]	Property:
18	0x51	This register tells the DAC chip that the received data will be in 24-bit I2S format and initially mutes the system.
20	0x01	Selects the delta sigma modulator oversampling rate to be 32 times f_S and tells the DAC-chip to work in stereo mode.
21	0x00	This disables the output zero pins

ModDAC_InitModule()

When the main function calls ModDAC_InitModule() the code in figure 4.11 is executed. It starts by calling ModDAC_GPIO_Configuraion() which configures the pins used to communicate to the DAC chip. Then it calls ModDAC_SPI_Configuration() that configures the SPI parameters used to enable the communication with the DIX chip.



```
bool ModDAC_InitModule(void)
{
    ModDAC_GPIO_Configuration();
    ModDAC_SPI_Configuration();
    return ModDAC_SEND_Configuration();
}
```

```
typedef struct
{   uint8_t reg;
    uint8_t val;
    uint8_t data;
}DAC_REGISTER_WRITE;
```

Figure 4.11: This figure displays the code that executes when running the ModDAC_InitModule() function.

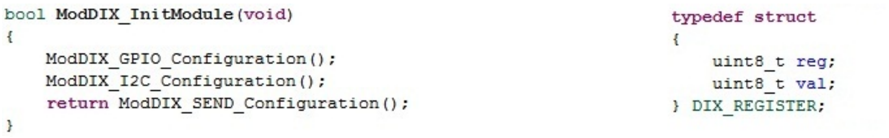
The last function to be called in this module is ModDAC_SEND_Configuration(). This function sends the data used to configure the DAC-chip. An array of a data structure named DAC_REGISTER_WRITE found in figure 4.11 is used to store the configuration data. The structure contains three unsigned bytes named reg, val and data. The val and reg byte contains the integer value of the register to be configured, and the data byte contains the actual data to be sent via SPI to the DAC circuit. Table 4.2 explains the interesting data bytes in the array of DAC_REGISTER_WRITE structures and the impact it has to the DAC configurations.

ModDIX_InitModule()

In figure 4.9 the main function calls the ModDIX_InitModule() module. This module is used to configure the DIX chip. The code executed by this module is found in figure 4.12.

Table 4.3: This table describes the interesting data bytes used to configure the DIX circuit.

Register:	Value: [Hex]	Property:
3	0x21	This register initializes the DIX chip port A to receive 24-bit I^2S data.
5	0x19	This register initializes the DIX chip port B to send 24-bit I^2S data. Selects port A to be the source of the output data on port B. This means that the initial output source to the analog stage and the S/PDIF(out) signal are the USB 2.0 audio data.
6	0x00	This initializes the LRCKB clock output from port B to be $\frac{MCLK}{128}$. MCLK is the input clock to the DIX chip. This clock is either 24.576 MHz or 11.2896 MHz, which is selected by the Cortex processor. The 24.576 MHz clock is used for the 192, 96, 48 MHz input I^2S data, whereas the 11.2896 MHz is used for the 176.4, 88.2, 44.1 MHz inputs. This implies that the output LRCKB signal is 192 KHz when the 24.576 MHz clock is selected, else 88.2 KHz. The LRCK signal in the I^2S protocol represents the sample rate of the signal.
7	0x10	Selects DIR(Digital audio interface receiver) to be the data source for the DIT(Digital audio interface transmitter) function block.
13	0x08	Selects S/PDIF(in, RX1) to be the input of the DIR.



```
bool ModDIX_InitModule(void)
{
    ModDIX_GPIO_Configuration();
    ModDIX_I2C_Configuration();
    return ModDIX_SEND_Configuration();
}
```

```
typedef struct
{
    uint8_t reg;
    uint8_t val;
} DIX_REGISTER;
```

Figure 4.12: This figure displays the code executed when calling the ModDIX_InitModule() module.

The first function executed in the ModDIX_InitModule is ModDIX_GPIO_Configuration which configures PB6 and PB7 pins to be used when communicating with the DIX circuit via I^2C. These pins are found in figure 4.7. When this function returns ModDIX_I2C_Configuration() are called. This function configures the I^2C parameters to enable the communication. The last function to be called is the ModDIX_SEND_Configuration() which sends the data stored in the array of DIX_REGISTER structures. The table 4.3 describes the interesting data bytes to be sent and their impact in the DIX chip.

4.4 Clock domains

In this part of the document all clock domains used in the USB Audio 2.0 project are discussed and what they are used for. Figure 4.13 gives an overview of the clock domains and which circuits that uses them. The components used in the oscillator circuits and how their values were chosen are also stated in this part of the document.

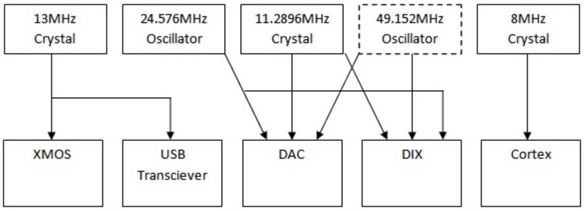
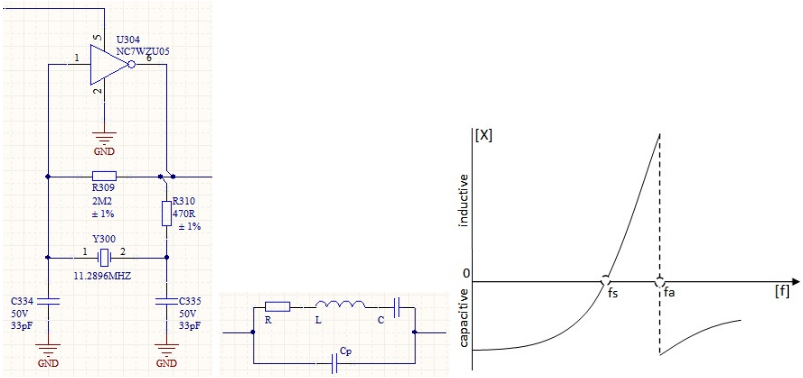


Figure 4.13: Overview of the clock domains and users. The dashed rectangle symbolizes a clock domain which is controlled by the Cortex processor.

In this project both crystals and oscillators are used. All crystals in this project are used in a parallel resonant oscillator circuit fashion viewed in figure 4.14a.



(a) Parallel oscillator circuit. (b) Crystal equivalent. (c) Impedance curve.

Figure 4.14: Crystal images.

A crystal can be represented by a equivalent circuit shown in figure 4.14b. L and C are the motional inductance and capacitance of the crystal[5]. C_p is the shunt capacitance due to crystal electrodes. In the 11.2896MHz crystal C_p is 7pF. Capacitive and inductive reactance is determined as X_C respectively X_L in equation 4.1. In 4.1 L corresponds to inductance in Henry, C as capacitance in Farads and f as frequency.

$$X_C = \frac{V_C}{I_C} = \frac{1}{2\pi f C} \quad [\Omega], \qquad X_L = \frac{V_L}{I_L} = 2\pi f L \quad [\Omega] \tag{4.1}$$

Freq [MHz]	Usage
13	System clock for the USB-transceiver and the XMOS processor.
24.576	This clock are used by the DAC- and DIX-chip to be able to handle the 192, 96, and 48 kHz audio data frequency.
11.2896	This clock domain is used by the DAC- and DIX-chip to handle the 176.4, 88.2, 44.1 kHz audio data frequency.
49.152	This clock were used in this project to be able to achieve the maximum oversampling rate possible by the delta-sigma modulator in the DAC chip. Used only for performance tests.
8	This domain is used by the Cortex processor as system clock.

Table 4.4: This table describes the clock domains used in the project.

$$f_s = \frac{1}{2\pi\sqrt{LC}} \;\; [Hz], \qquad f_a = \frac{1}{2\pi\sqrt{L\frac{C_L C_P}{C_L + C_P}}} \;\; [Hz] \tag{4.2}$$

$$C_L = \frac{C_1 C_2}{C_1 + C_2} \;\; [F] \tag{4.3}$$

The circuit equivalent in figure 4.14b can operate in either series or parallel resonance. This is determined by the calibration of the crystal which is done by the schematic in figure 4.14a. The oscillator frequency depends on the load capacitance which fa shows. This load capacitance is determined by the discrete load, stray board(C_S), and miller capacitances [5]. In equation 4.2 fs represents the series resonance frequency whereas fa represents the parallel resonance frequency. An perfect parallel resonance circuit would make an infinite impedance. The resonance frequency could be placed in the range, $fs - fa$. As figure 4.14a shows there are two capacitors added C334 and C335, both at a value of 33pF. This circuit will generate a C_L value determined by equation 4.3. In this design $C_L = \frac{33*33*10^{-24}}{(33+33)*10^{-12}} + C_S \approx 16pF + C_S$. The data sheet for the crystal used in 4.14a has a recommended C_L of 18pF. A lower value was selected due to expected stray capacitances. The same calculations were done for the 13 MHz clock used by the XMOS processor.

Two theorems are stated for oscillator circuits to obtain oscillation and these are [13].

- 1) The closed loop gain must be ≤ 1.
- 2) The phase shift around the loop must be $n * 360°, \;\; \forall n \in \mathbb{N}$

The inverter is used to produce a 180° phase shift. In this crystal circuit the π network ensures a $n * 360°$ phase shift around the loop [5].

The table 4.4 describes what all clock domains are used for in this project.

4.5 DIX4192

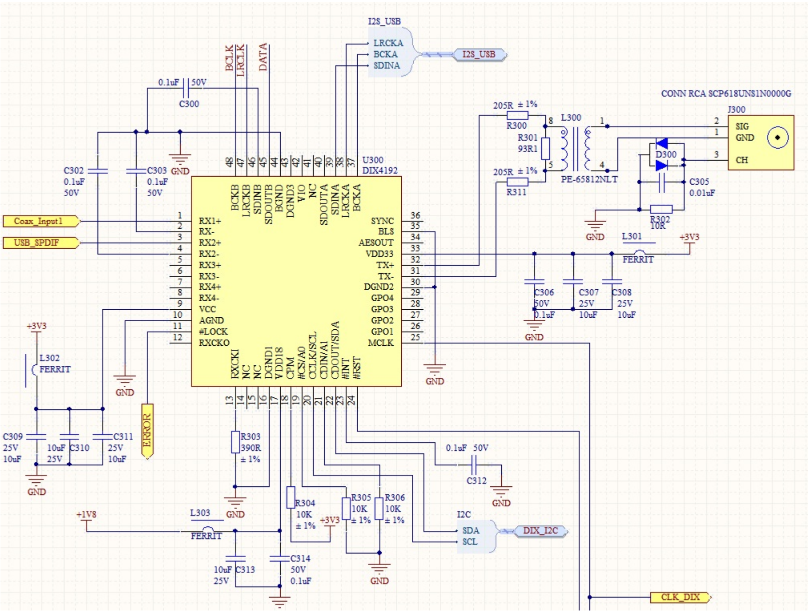


Figure 4.15: This figure displays the DIX-chip in the project schematic.

This circuit is a digital audio receiver and transmitter(DIR/DIT) used in the project to multiplex audio data sources. As mentioned in section 4.3 this chip is configured by the cortex processor via the I^2C protocol. The data bytes transmitted to this circuit is listed and explained in table 4.3.

4.5.1 Hardware

This section will explain interesting hardware configuration of the chip and its impacts. The DIX chip supports either I^2C or SPI as configuration protocols. This is hardware configured by the value of pin18 (High=I^2C, Low=SPI). In this project it is tied to a pull-up resistor R304 to +3.3V. Pin19(A0) and Pin21(A1) when not used for SPI these pins determines the slave address for the $I2C$ protocol. A0 represents the LSB in the configurable address. These are tied via respective pull down resistors, R305 and R306 to ground. This gives the opportunity to have four DIX devices connected to the same bus, if wanted. In this project this is of no interest so any address could have been used. Port A is used to receive data from the XMOS circuit via the I^2S protocol. As figure 4.1 on page 13, describes the DIX chip has three audio inputs and two outputs in this design. The table 4.5 describes the audio data direction and used transfer protocol. What audio source that the DIX chip will provide at the outputs are determined by the Cortex microcontroller. This is done by push button P501. The button is used to toggle between the possible audio sources.

Direction	Protocol	Description
In	S/PDIF	This audio media is delivered from an attached external S/PDIF source(if any).
In	[S/PDIF,I^2S]	These signals are provided by the XMOS processor. The media is the same at bout signal types, it is only the transfer protocols that differs.
Out	S/PDIF	The DIX chip has the capability to create and transmit an differential S/PDIF signal created from the selected source. A functionality that is utilized in this project.
Out	I^2S	This output is used to transport audio data to the analog part of this project via the DAC chip.

Table 4.5: This table describes each I/O from the DIX chip used in this project.

The TX+ and TX- outputs from the DIX chip represents one differential S/PDIF pair. These signals goes to a transformer with a 1:1 relationship between the primary and secondary side. The transformer is used to improve common-mode noise performance and to isolate the DIX chip from the attached device. The S/PDIF standard uses a 75Ω cable to transmit the audio data signal. For maximal power transfer the characteristic impedance at each terminating end should be 75Ω. A good impedance matching minimize the problem of the transmitted signal to be reflected back and thus introduce error to the signal. The resistor network on the primary side of the transformer form a 75Ω impedance. This is calculated by shorting the TX+ and TX- terminals, which gives $Z_{out} = \frac{410*93}{410+93} \approx 75\Omega$.

4.6 PCM1792A

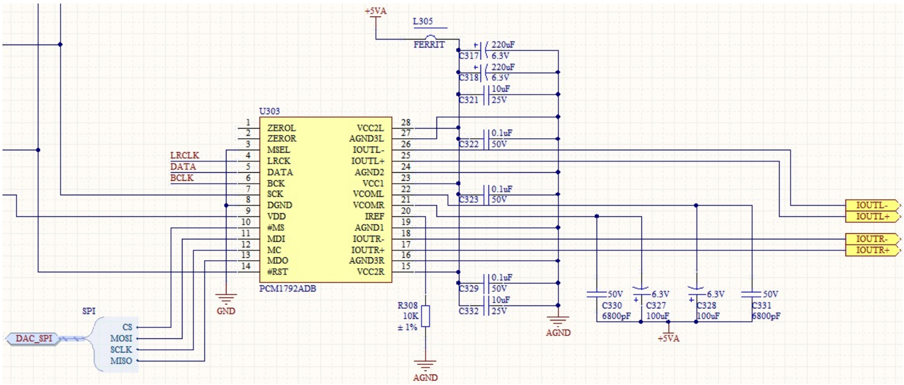


Figure 4.16: This figure shows the DAC chip in the project schematic.

This DAC is a high performance Bur Brown circuit.

4.6.1 Hardware

This section describes the hardware configurations of the DAC chip. All references in this section refer to figure 4.16. The DAC chip is configured as mentioned in section 4.3 via SPI, although it also supports I^2C. Which protocol to use is determined by pin 3(MSEL). A high value on this pin selects SPI whereas a low value selects I^2C. The capacitor bank on the right side of the DAC in figure 4.16 are determined by the datasheet. The DAC outputs consists of two differential pair representing the right respectively left channel. These signals are connected to the analog stage that converts the differential current output provided by the DAC to a suitable output voltage for the line-out RCA-connector. In figure 4.17 the output current as a function of input data are displayed.

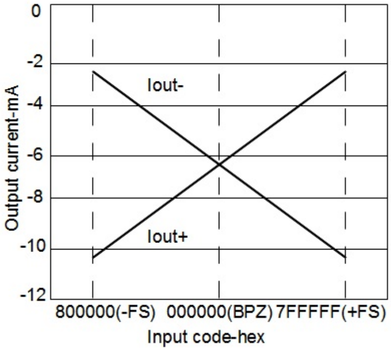


Figure 4.17: This figure shows the output current characteristics vs input code.

	800000(-FS)	000000(BPZ)	7FFFFF(+FS)
IOUT-[mA]	-2.3	-6.2	-10.1
IOUT+[mA]	-10.1	-6.2	-2.3
VOUT-[V]	-1.725	-4.650	-7.575
VOUT+[V]	-7.575	-4.650	-1.725

Table 4.6: This table describes the output/input data dependencies.

Table 4.6 describes the voltages and currents applied to the differential outputs.

4.7 Analog stage

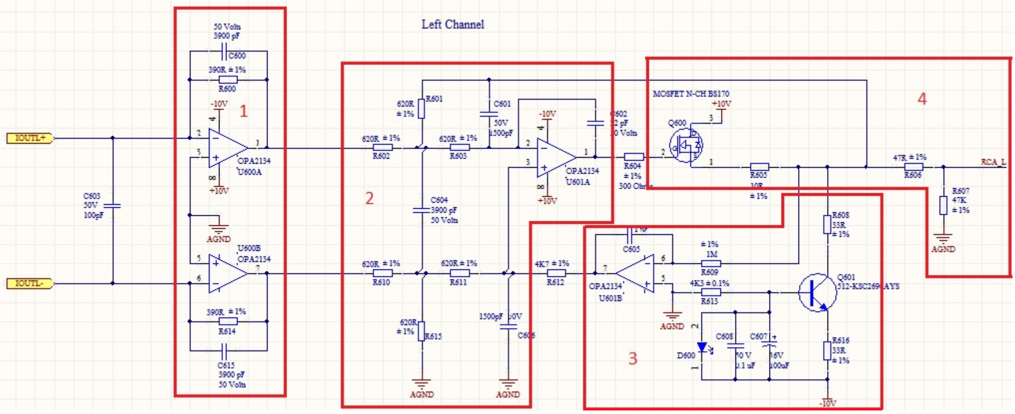


Figure 4.18: This figure shows the analog stage that is used to convert the current output from the DAC-chip to a desired output voltage.

This section will explain how the analog stage used to amplify the output signal from the DAC to a desired output level works. This analog stage is found in figure 4.18. To escape time-consuming and complex hand calculations, all except from the transimpedance circuit are simulated with LTspive IV. The circuit schematic used to simulate the analog stage is found in figure 4.20.

The first rectangular box with index one is two so called transimpdance amplifiers that converts an differential input current to an differential output voltage. This stage form a 1-pole low-pass filter [14]. The following calculations performed on this transimpedance stage refer to figure 4.19a.

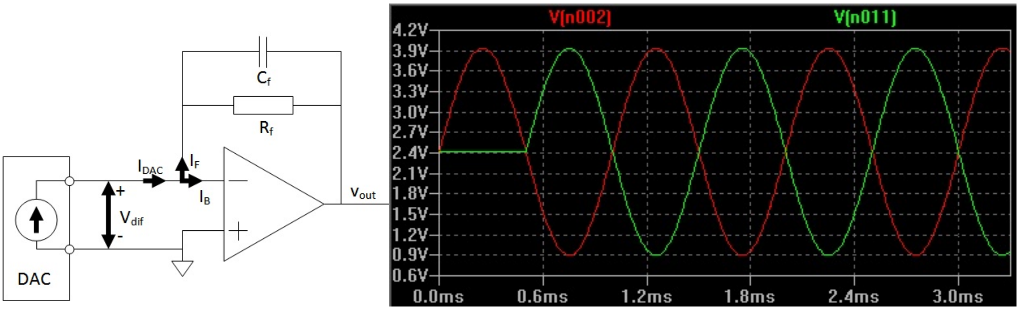
In this calculation the op-amp are considered to be ideal. This implies that the bias current, $I_B = 0$ and the potential difference between the inverting respectively the non-inverting inputs, $V_{diff} = 0$ [11]. These assumptions forces the current from the DAC(I_{DAC}), to flow via Z_f which is $C_f//R_f$. This gives us the following formula for the output voltage(V_{OUT}). This formula is derived using KCL [3].

$$KCL \Rightarrow I_{DAC}+I_F+I_B = 0 = I_{DAC} - \frac{(0 - V_{out})}{Z_f} = 0 \Rightarrow V_{out} = -I_{DAC}*Z_f \quad (4.4)$$

This formula gives us the following voltage output min and max due to the maximum respectively minimum output currents from the DAC chip. The min and max values from the DAC chip are found in table 4.6 on page 29.

- $V_{outmax} = 3.9V$
- $V_{outmin} = 0.9V$

The calculated output levels, V_{outmax} and V_{outmin} are exactly the same result as the LTspice IV simulator gives us. Figure 4.19b plots the outputs from the simulated transimpedance-stage. The red curve represents measure point 7 in figure 4.20 whereas green represents point 8 in the same figure.



(a) Transimpedance stage.

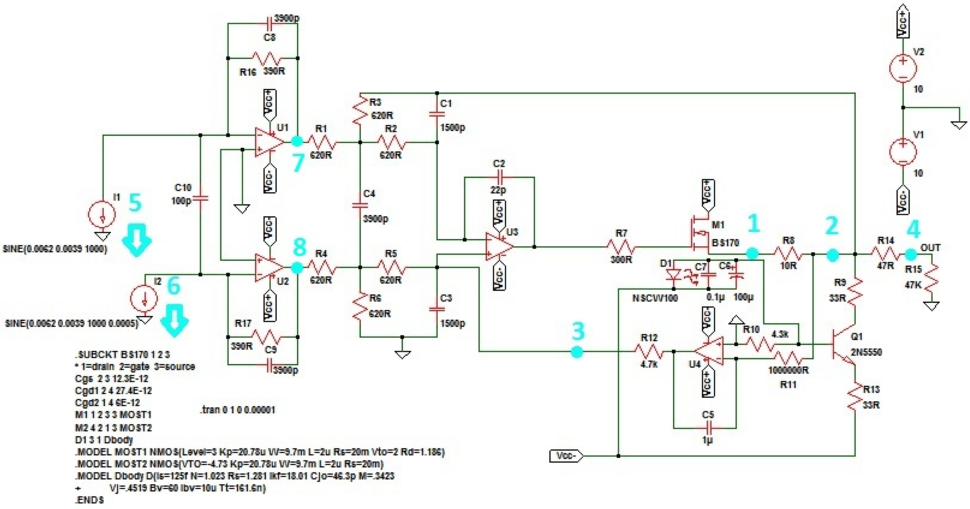
(b) Point 7 and 8 in figure 4.20.

Figure 4.19: Analog stage figures.

The transimpedance stage is then connected to the second box in figure 4.18. This stage converts a differential signal to a single ended signal.

The third box in figure 4.18 are a so called DC-servo circuit[7]. The idea behind this technique is to make up a separate global feedback loop that acts at DC to control amplifier output offsets. This is done by filtering the output signal from the mosfet transistor Q600 with a low-pass filter. This signal are then amplified and feed back into the non-inverting pin on op-amp U601A. To perform the low pass filtering an integrator circuit are inserted [4]. Measure point 1, 2 and 3 in figure 4.20 are displayed in plot 4.21a. In this plot the red signal represents measure point 1, the green, point 2 and the blue point 3. The green and red signal truly shows that the DC-servo does its work by lowering the 0.6V DC-offset from the emitter terminal of the mosfet transistor. The blue curve shows the input to the inverting respectively the non-inverting input terminal at the operational amplifier U3.

An alternative to the DC-servo would be to insert a capacitor in series on the output terminal. This would also remove the DC-level from the output but would add far more non-linearities which would lead to a unwanted distorted audio signal.



```
.SUBCKT BS170 1 2 3
* 1=drain  2=gate  3=source
Cgs  2 3 12.3E-12
Cgd1 2 4 27.4E-12
Cgd2 1 4 6E-12
M1 1 2 3 3 MOST1
M2 4 2 1 3 MOST2
D1 3 1 Dbody
.MODEL MOST1 NMOS(Level=3 Kp=20.78u W=9.7m L=2u Rs=20m Vto=2 Rd=1.186)
.MODEL MOST2 NMOS(VTO=-4.73 Kp=20.78u W=9.7m L=2u Rs=20m)
.MODEL Dbody D(Is=125f N=1.023 Rs=1.281 Ikf=18.01 Cjo=46.3p M=.3423
+         VJ=.4519 Bv=60 Ibv=10u Tt=161.6n)
.ENDS
```

```
.tran 0 1 0 0.00001
```

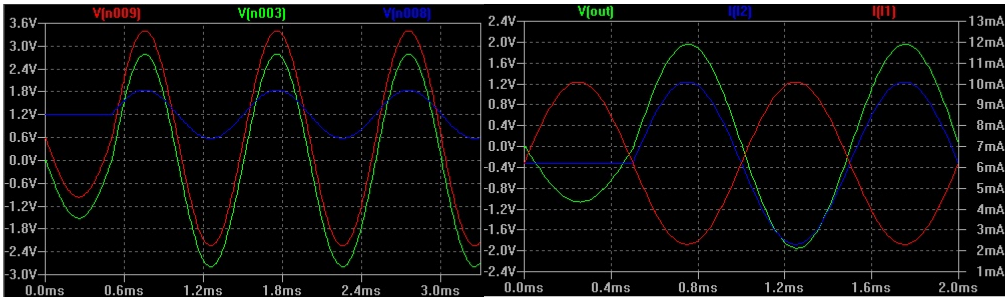
Figure 4.20: This figure shows the simulation setup in LTspice IV.

The fourth box includes the output mosfet transistor and a voltage divider lumped circuit net. Due to problems to find the exactly equivalent simulation models for the OP-amps and the mosfet tranistor Q600 some results are not fully accurate with the actual USB 2.0 project. In figure 4.21b the red signal represents measuring point 5, the blue, point 6 and the green the output signal which is represented by point 4. The current sources I1 and I2 has the following parametrization. These parameters are distinguished by the DAC circuit output currents found in table 4.6 on page 29.

- DC offset[A] = 0.0062

- Amplitude[A] = 0.0039

- Freq[Hz]= 1000



(a) Point 1, 2 and 3.

(b) Point 4, 5 and 6.

Figure 4.21: Simulation results.

I2 has a added delay parameter which is 0.0005 s. This delay is half the signal period which is used to simulate the differential input signal. The green curve in figure 4.21 gives us the information that the output voltage from the analog stage will be $\approx \pm 1.7V$. This gives a nominal level of $\frac{1.7V}{\sqrt{2}} \approx +4dBu$ which is a commonly used line level in professional audio equipment.

4.8 Power domains

The USB audio 2.0 design consist of a variety of circuits that requires different power supplies. Figure 4.22 views a flow graph of these power domains and which circuits who uses them. The (A) suffix refers to the analogue domain whereas (D) refers to the digital domain. The power supplies are divided into an analog and a digital domain to separate these from each other. This to not plant the low frequency ripple and high frequency spikes from the DC/DC switched regulators into the analog domain. The analog domain uses only linear voltage regulators. The ”DC/DC Converters” block consist of a couple DC/DC converters that either performs a step up or step down regulation on the input voltage. This to achieve the +10V, +5V, +3.3V, +1.8V and +1.0V domains. In this design lots of different regulators were used in learning purpose.

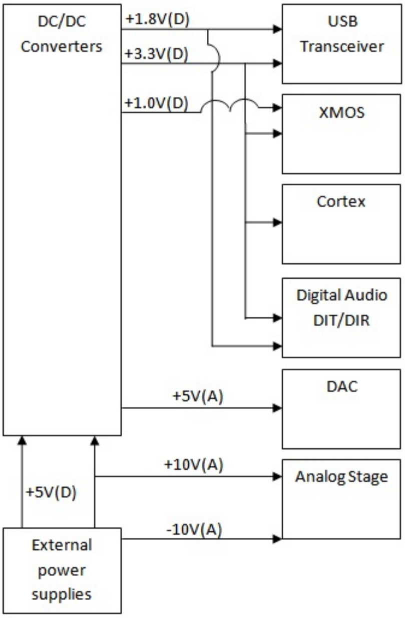


Figure 4.22: Overview of the power domains and which IC:s that uses them.

4.8.1 +10V to +5V(Analog domain)

To achieve the +5V in the analog domain there had to be some kind of conversion from the +10V input. In the design this is performed by the LM78 circuit in figure 4.23. This chip is a linear voltage regulator that accepts an input of +10V and regulates it down to +5V. This circuit can deliver a maximum of 0.5 [A]. This power domain is used by the DAC and therefore the linear voltage regulator was chosen to lower the noise in the analogue domain [1]. D400 and R400 are added to protect the voltage regulator from damage if the +10V input would get shorted. In the case when the +10V terminal would get shorted and C405, C406 are fully charged. This would generate a current which is forced to travel via the LM78 circuit and surely destroy it. When the diode is inserted the current travels via the diode when the forward thresh-hold voltage is passed. This would imply in a destroyed diode but the more expensive regulator is not damaged.

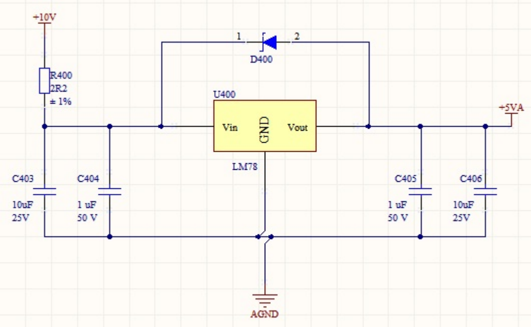


Figure 4.23: Circuit used to perform the +10V(A) to +5V(A) conversion.

4.8.2 +5V to +1.8V(Digital domain)

This step-down regulation uses the same technique as in section 4.8.1 but uses the LM317 circuit instead. This circuit can deliver 1.0 [A] to the output terminal.

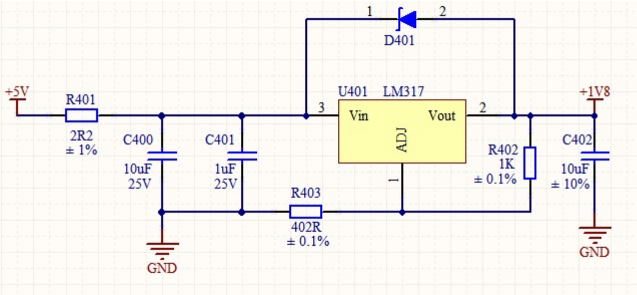


Figure 4.24: Schematic from USB audio 2.0 that perform the +5V to +1.8V conversion.

4.8.3 +5V to +3.3V(Digital domain)

When performing a step down regulation in the digital domain DC/DC converters and linear regulators are used. Figure 4.25 shows the schematic in the USB audio 2.0 project that performs the +5V to +3.3V step-down action. The LM2831 circuit used to create this voltage domain can deliver 1.5 [A] at the output terminal. The component values and characteristics are taken from the data sheet of the LM2831 chip.

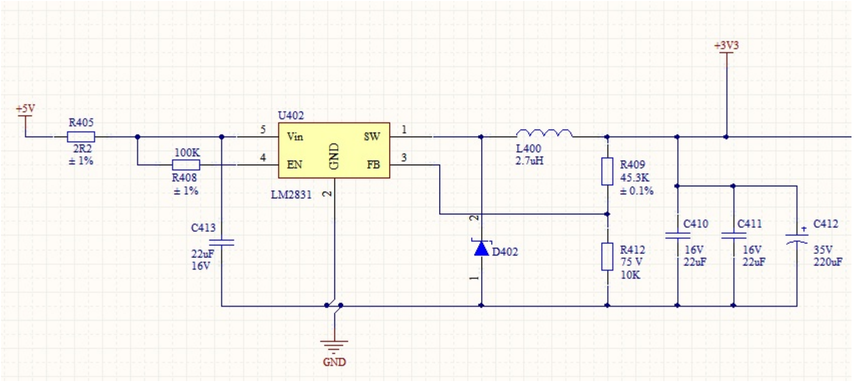


Figure 4.25: The schematic part that performs the +5V to +3.3V conversion.

4.8.4 +5V to +1.0V(Digital domain)

To achieve this voltage conversion the same technique as in section 4.8.3 is used. The circuit used perform this voltage drop is the NCP1521B chip. This circuit can deliver 0.6 [A] at the output terminal. Pin number 3 of the NCP1521B in figure 4.26 have a interesting functionality used in this project. This pin is connected with the output from the DC/DC converter in section 4.8.3. This to let the LM2831 chip control the NCP1521B chip. At a start-up sequence V_{en} has to be $\geq 1.2V$ which is the thresh-hold for the enable signal. This to ensure that the +3.3V domain is reached before enabling the +1.0 domain. This action had to be done because of the XMOS processor that requires its peripherals to be powered before the core.

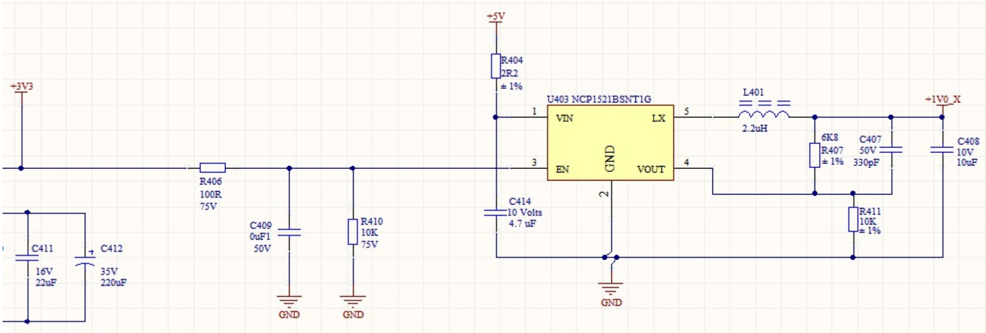


Figure 4.26: The schematic overview if the circuit used to perform the +5V to +1.0V conversion.

The RC-network consisting of R406, C409 and R410 delays the enable signal from passing the thresh-hold.

4.9 PCB

When the schematic sheets are finished the next step in the design process is to export the schematic constraints and connectivity constraints to the PCB editor. This section will focus on the interesting design techniques used when developing the PCB for the USB 2.0 project.

4.9.1 Layers

This section describes the four layers used on the PCB. In figure 4.27 the layer setup and there thickness are displayed.

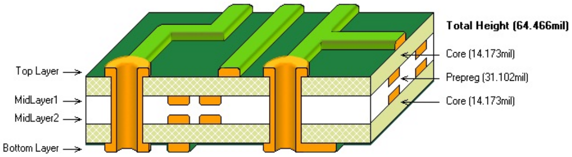


Figure 4.27: Overview of the PCB layers.

4.9.2 Ground layer

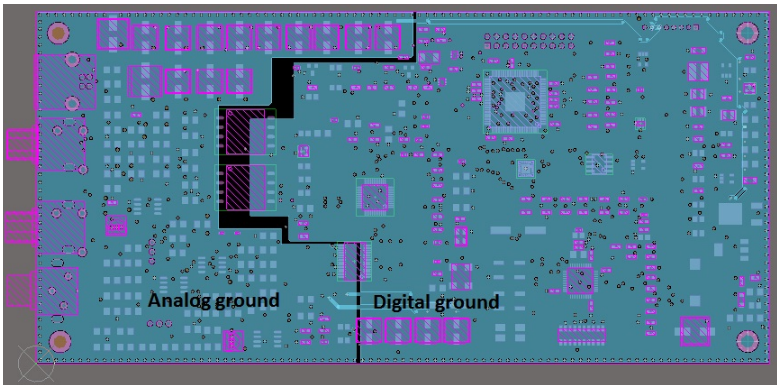
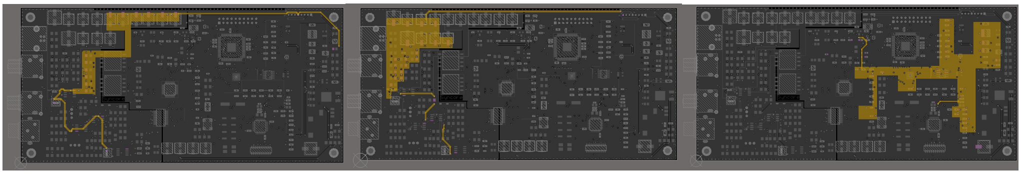


Figure 4.28: Overview of the ground layer.

This section describes the ground layer, referred as MidLayer1 in figure 4.27. Figure 4.28 shows that the layer consists of two separate areas. The leftmost area is referred as the analog ground whereas rightmost is referred as the digital ground. The two names comes from the circuits that are connected to respectively ground polygon. The circuits connected to the analog ground are the part of the power supply that are feeding the analog stage. The DAC and the analog stage that converts the current output from the DAC to a desired output voltage. The rest of the circuits on the PCB, such as the STM32F100C4T6B, XS1-L01A-TQ128-C5 to mention a few are connected to the digital ground. These polygons are connected with a single signal trace via a dual shottky diode. The schottky diode are inserted to prevent accidental DC voltage from developing between the two ground polygons. It also protect against low-frequency voltage spikes[8]. The forward voltage of the diode is 855mV which implies that a current starts to flow when this threshold voltage is passed. The idea behind splitting the digital and analog domain is to prevent high speed digital circuits to interfere with low level analog circuits in this mixed signal project.

4.9.3 Power layer

This section describes the sub-planes in layer three referred as MidLayer2 in figure 4.27. The unshaded part in figure 4.29a represents the polygon created to simplify the signal routing for the +10V domain. This is also done for the -10V and +3.3V domains which polygon shapes are viewed in figure 4.29b respectively 4.29c. The rest of the PCB plane consists of two large ground planes with the same shape as mentioned in 4.9.2. As mentioned in section 4.8 the +10V, -10V are used in the analog stage which components are placed above the analog ground plane. That is why both the +10V and -10V polygon are placed on the left side of the figure 4.28. This also holds true for the +3.3V domain which is used by the digital components.



(a) +10V polygon.

(b) -10V polygon.

(c) +3.3V polygon.

Figure 4.29: Power layer polygons.

4.9.4 Subsections

When placing the components onto the PCB-card the strategy was to divide the design into sub-parts. The blue surrounding line represents these sub-circuits. They are numbered to distinguish the sub-circuits from each other. Table 4.7 describes why they are placed in this manner and what each sub-circuit is. The circuit subsections are divided into analog and digital groups. The analog groups refer to subsections 1 and 2 whereas the other subsections mentioned in table 4.7 refers to digital subsections. The fundamental strategy were to place the anlog respective the digital groups above the analog respectively the digital ground polygons. These planes were mentioned in section 4.9.2. This this is done to minimize the current loops and to separate week analog signals from noisy digital signals[8].

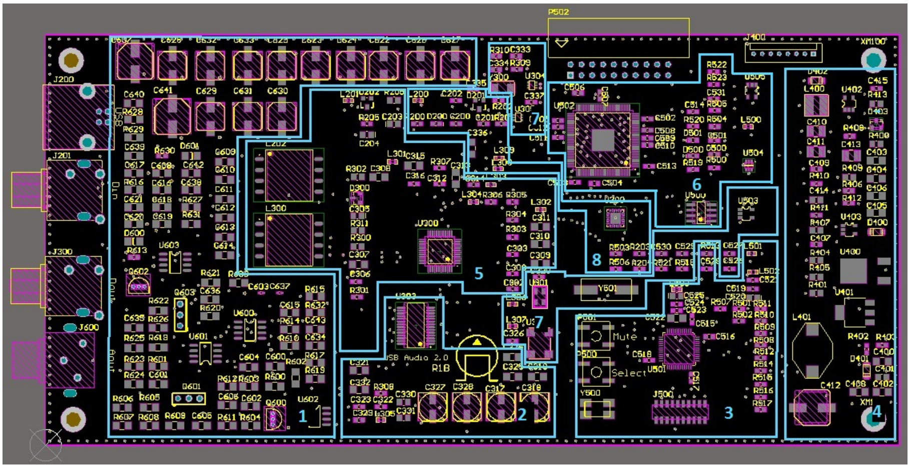


Figure 4.30: Overview of the routed PCB

Nr:	Description
1	This sub-circuit represents the analog output stage. This section is placed near the output RCA-connectors, this to have a short output signal paths.
2	This is the DAC related components that are placed close to the analog stage to shorten the differential current output traces.
3	This section corresponds to all components related to the Cortex processor. This component group are placed on the right side of the PCB. This side represents the digital side of the PCB.
4	This blue lined rectangle encapsulates the power-supply circuits that are placed as far away from the analog domain as possible. This to minimize the switching EMI noise produced by the switching voltage regulators[1].
5	This sub-circuit represents the digital multiplexer(DIX) and its associated components. This block is placed as close to its input RCA-connectors as possible to shorten the signal path.
6	This section encapsulate all components associated with the XMOS chip. This section is also placed on the digital side of the PCB.
7	This sub-circuit consists of two areas in figure 4.30 which represents the clock blocks. They are placed as close as possible to the input pins on the circuits using them.
8	This is the USB 2.0 transceiver chip which is placed close to the XMOS processor. This because of there data exchange. There ware some uncertainties concerning the long receiver path from the USB connector to the transceiver. A risk that had to be taken the analog stage which had the highest priority in this case.

Table 4.7: This table describes each sub-circuit placement strategy.

4.9.5 Routing

This section will discuss some of the strategies used when routing the PCB. When routing the analog left/right channels the strategy was to keep the copper traces of respectively channel, approximately at the same length. This to have the same voltage-drop created by the signal traces internal resistance. By keeping the differential analog signal at approximately the same length it also prevents timing miss matching due to propagation delays. As an example a PCB copper trace based on the IPC-2221 standard with the parameters listed below would generate the result in the item list below the line.

- Current 0.37 [A]
- Thickness 35 [um]
- Ambient temperature 25 °C
- Trace length 150 [mm]

- Required trace width 0.198 [mm]
- Resistance 0.382 Ω
- Voltage drop 0.141 V
- Power loss 0.0523 W

A result that depicts that the trace length play an minor role but small design refinements will add up to a better total result of course. If one of the differential signal traces would have been a ten factor shorter then the other. This would result in a tenth factor lower voltage drop in the trace. This along with the transmission delay aspect, is why the differential analog signal traces are kept as short as possible and are designed to have approximately equal length.

When routing some of the signals which carry high frequency contents the strategy was to keep them well separated from the analog signals. This to avoid unwanted high frequency noise to be transferred into the week analog signals. In this project the S/PDIF, and all clock domains are examples of high frequency signals that had to be routed with caution. In this design the input and output S/PDIF signals are routed on the bottom layer whereas the analog signals are routed on the top layer. This to minimise the S/PDIF signals to interfere with the analog signal. The same methodology is used for the USB transceiver D+ and D- signals.

As the IPC standard depicts in the former example it recommends a minimum trace width of 0.198 mm. This with a current of 0.37 A. In this project a trace width of 0.2 mm is used for information carrying signals whereas power signals have trace width of 1 mm. The width of 0.2 mm was chosen because of the manufacturer of this PCB project had this as a minimum parameter. Using the same calculations as before but with a current of 2.5 A would recommend a trance width of ≈ 1 mm. In this design we have no circuit domain that will consume so much power that more than 2.5 Amps is required. That is why this width was chosen.

Chapter 5

Results

5.1 USB 2.0 streaming device

In this section the result gathering the USB 2.0 audio project will be stated. In figure 5.1 you find a picture of the finished USB audio 2.0 project PCB.

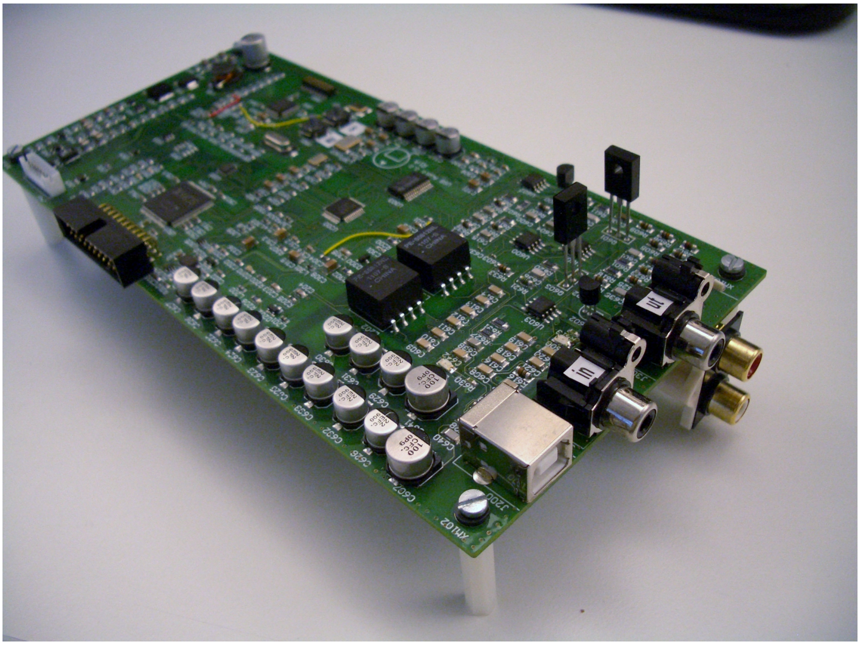


Figure 5.1: Overview of the PCB.

The main goal with this project was to create a audio device based on XMOS USB 2.0 solution. A project which ended up with a fully functioning device which met all requirements set up from the project leaders. These requirements are listed below.

- Be able to support S\PDIF input signal.
- Support USB Audio class 2.0 and 1.0.
- Provide unbalanced, +4dBu L/R line level outputs.
- Provide S\PDIF output signal.

In this part of the document there would have been of great interest to see some measuring result commonly used to state the performance of audio devices. This to give the reader a stronger belief of the result. With lack of time and good measuring equipment this was left out and the reader has to rely on our opinions. The requirements that this project had are fully met and verified via oscilloscope measurements and sound tests. To test the performance of the analog stage and the digital audio signals we created a 24-bit, 192 kHz silent audio file. This file were played with high volume on the amplifier to which our device were connected to. This to be able to hear the noise performance from the anlog stage and to recognise occurring errors on the digital signals. When the analog stage was tested we connected our USB 2.0 project to a amplifier by Primare named I32. In the analog stage we recognised a slight noise at high levels, which is hard to not get in the analog parts. We think that we should have kept the analog signal differential through the whole analog chain to lower the noise floor. But to be mentioned that this noise was recognised at a high level of the amplifier. The digital S\PDIF signals were verified with Primares SPA22 unit. When verifying the digital signal we could not recognise any hazards when listening.

We think that this audio device performed over our expectations and we are happy with the result. With a great interest of hi-fidelity equipment we know that this device actually performs well and sounds good.

5.2 Componen databases

In this section we will discuss our opinions gathering the functionality of the component databases and the tools provided by Altium to simplify the database usage.

Tools

The tools provided by Altium works very well and really improves the time that has to be spent to administrate the databases. We have not faced any remarkable problems during our project related to these tools. With this said we really think that Altium corporation has succeeded to create smart tools to simplify the work surrounding component databases.

Usage

This subsection will depict the interesting aspects that we found using the component databases. Under the project we have used the database created in chapter 3.

As mentioned in that chapter, Altium provides the opportunity to let an arbitrary number of persons being able to administrate the component databases. Leaving the role decisions to the companies/arbitrary users. The list below enumerates some of the most important advantages of using component databases.

- A good way to deal with reversions(if .*SVNDBLib is used).
- Simplifying BOM(bill of materials) creations.
- Keep up to date component parameters(linkage to suppliers).
- Simplifying reuse of earlier created schematic symbols and footprints.
- Free of charge if having a Altium designer license.

We discovered a big disadvantage when using the subversion treatment of a repository. To be able to use the .*SVNDBLib all component- and pcb-libraries has to be split into its constituent parts. With this we mean that a pcb-library containing resistors has to split all containing footprints into separate pcb-libraries. This to be able to treat them under subversion. We think that this really destroys a good way of making hierarchical structure. We have seen on Altiums web-page that they want the administrator to structure the repository under subversion treatment with folders instead. This is possible but we think they have destroyed a good container structure and introduced a component explosion in the repository. This could lead to a messy repository and complicate the administration.

On the big perspective we think that the whole package surrounding component databases are good and could easily simplify the job for developers. An important aspect to be considered today is to shorten the release of a product. Using these tools, will give the developers the capability of reusing footprints and schematic symbols that has already been created in earlier projects. A factor that surely would decrease the developing time.

Although this is a very good solution we think that at least one person(depending on company size) should exclusively work with the administration of the databases. This because we have discovered that the administration is quite time-consuming. A burden that should not be added to the developers workload.

To be mentioned is that the Altium corporation has stopped there development of this specific way of handle components. Today they work on a new system where the component databases are stored in the ”cloud”. A trend we see in many applications. This service costs extra money apart from the Altium designer license. A service that are not evaluated in this document but jet to be mentioned.

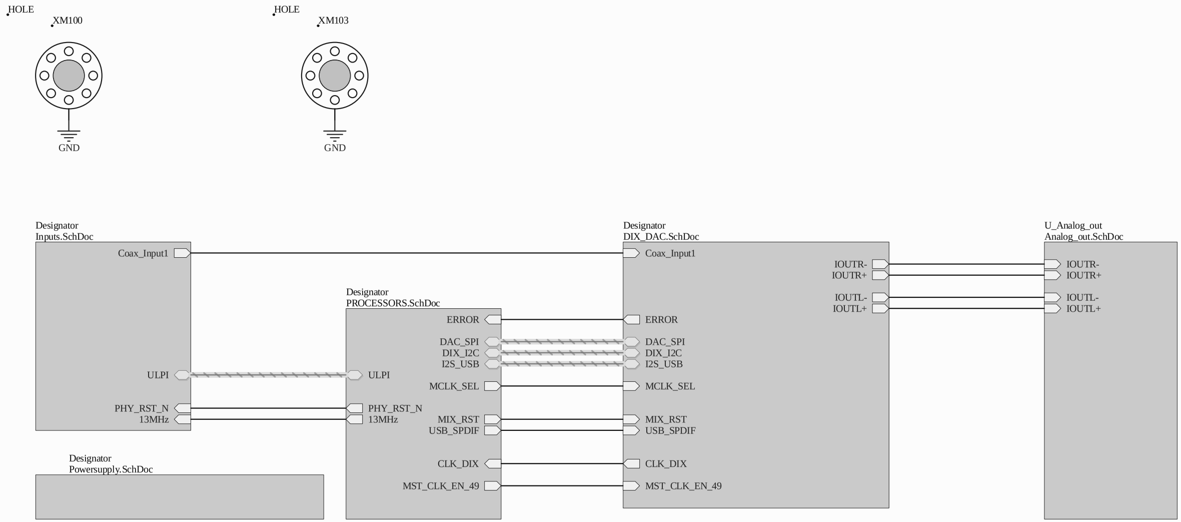
With this restriction the conclusion is that although the component databases package is a powerful tool it is not suitable for smaller companies. This because it may not be affordable to dedicate on person to exclusive work with the databases. We leave the question whether to use the database package or not open. This because there are to many aspects to be taken into consideration and it is so application specific.

Appendix A

Schematics

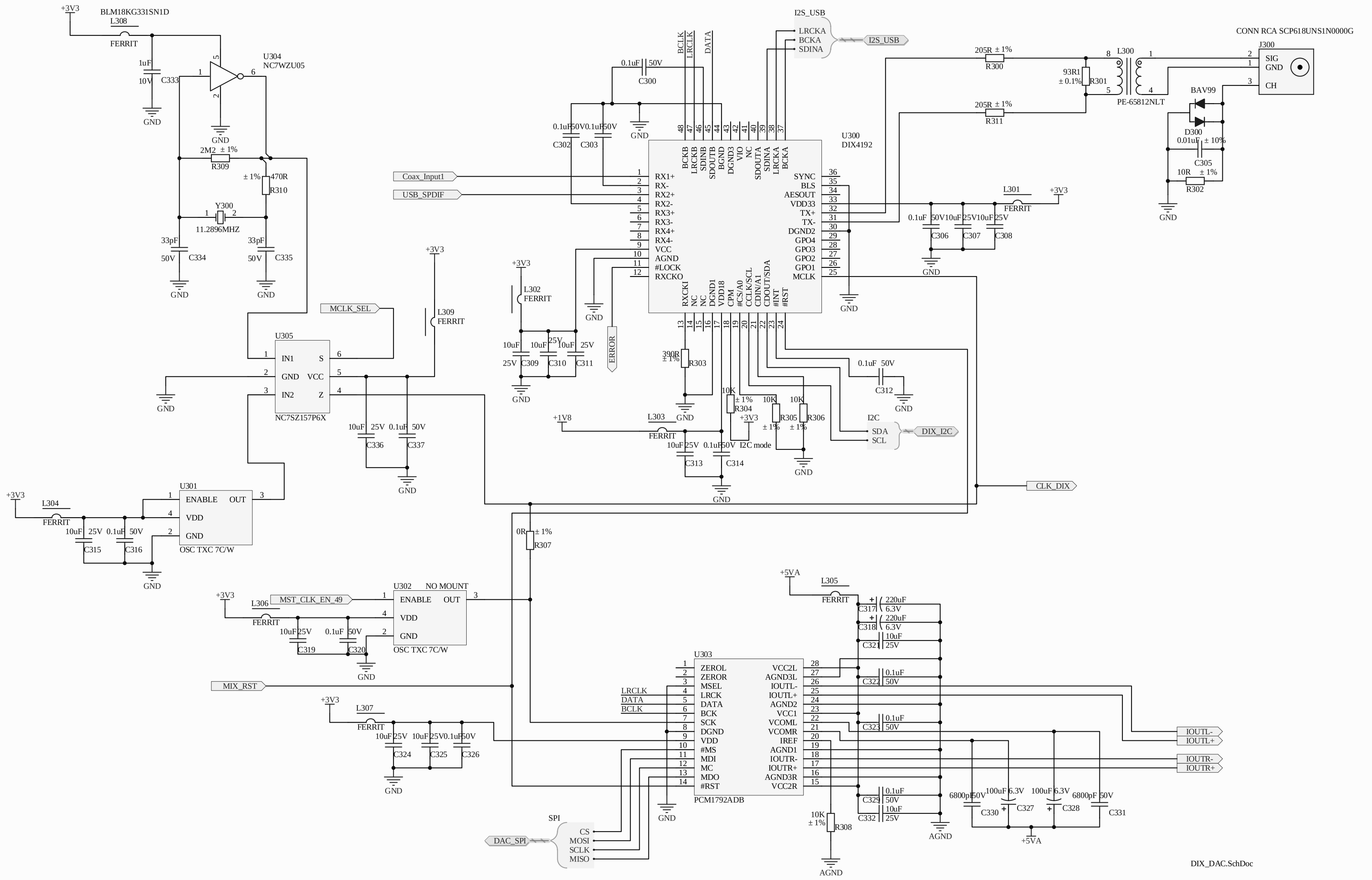
In this appendix the schematic files used in the USB 2.0 project are attached. The ordering of the document files are listed below.

- 1) Main.SchDoc
- 2) DIX_DAC.SchDoc
- 3) Processors.SchDoc
- 4) Inputs.SchDoc
- 5) Power_supplies.SchDoc
- 6) Analog_stage.SchDoc



Title	* main.SchDoc		
Size A3	Number	Version Control Revision Unknown revision	Revision Rev
Date:	2011-12-19	Sheet * of *	0.5
File:	main.SchDoc	Drawn By: *	

syncore technologies



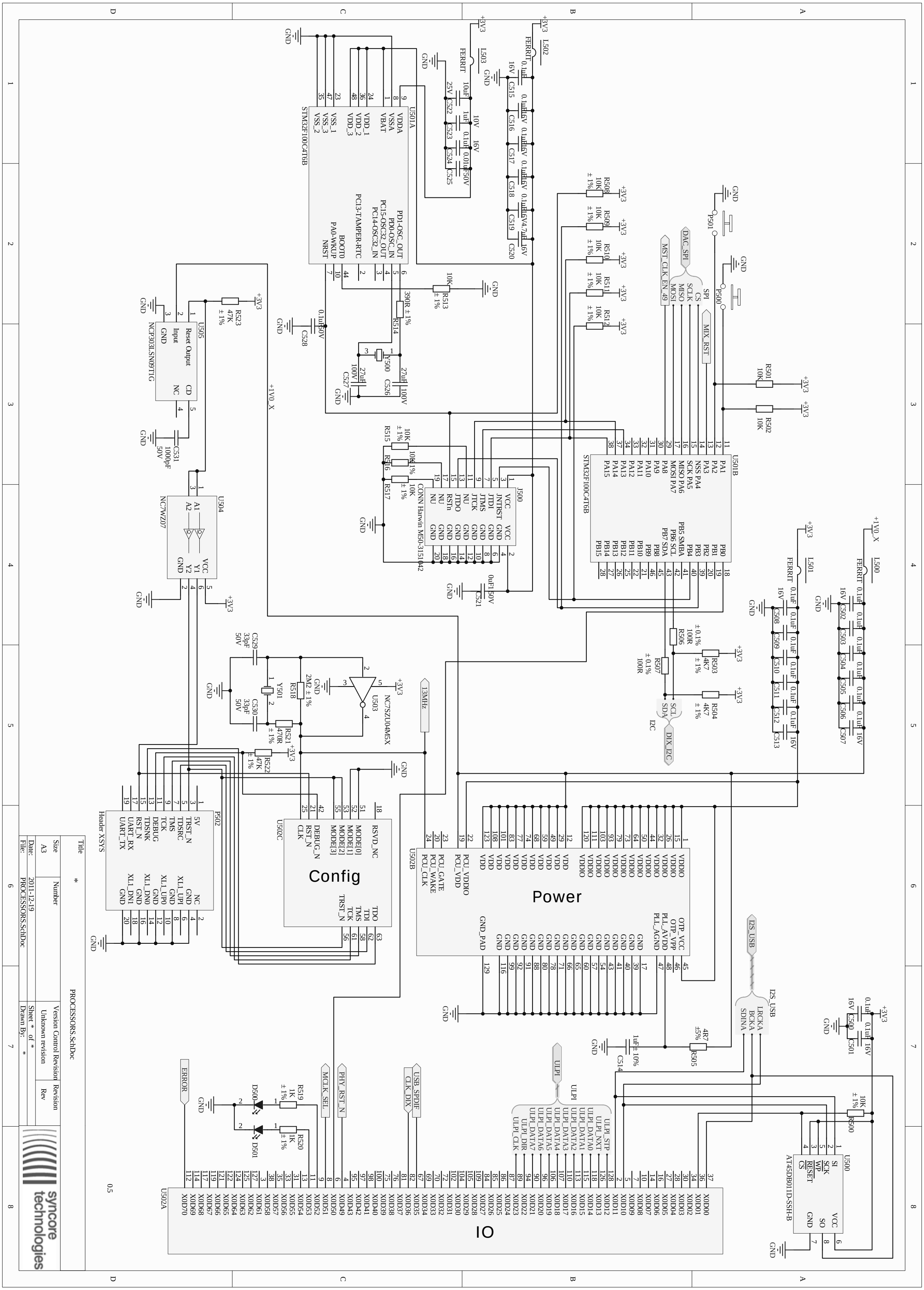
Title *

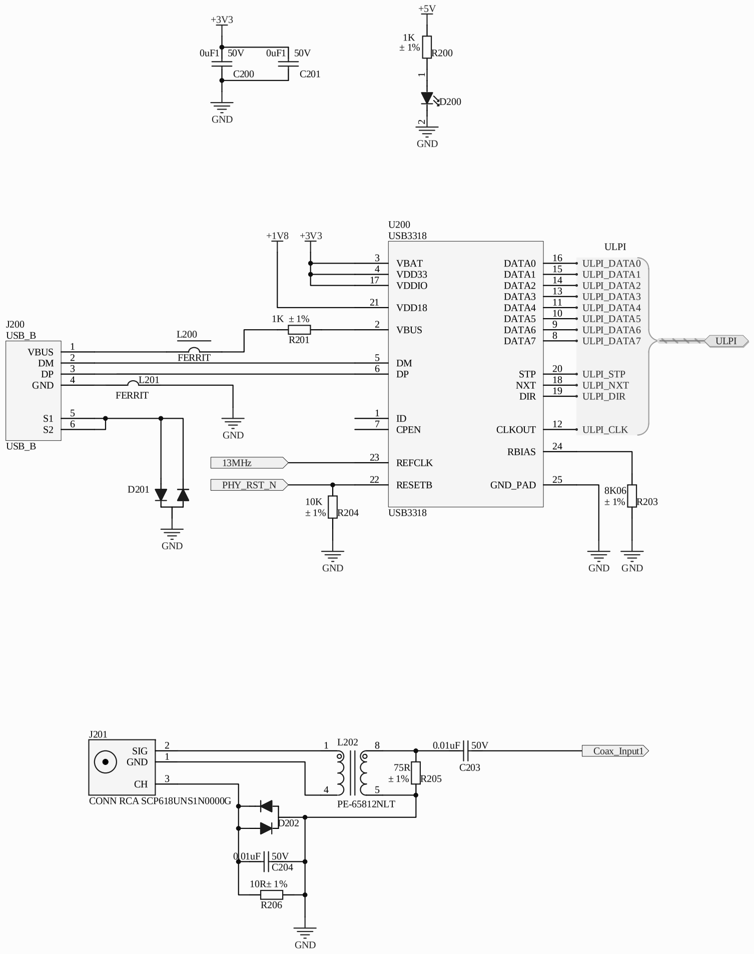
Size A3 | Number | Version Control Revision Unknown revision | Revision Rev

Date: 2011-12-19 | Sheet * of *

File: DIX_DAC.SchDoc | Drawn By: *

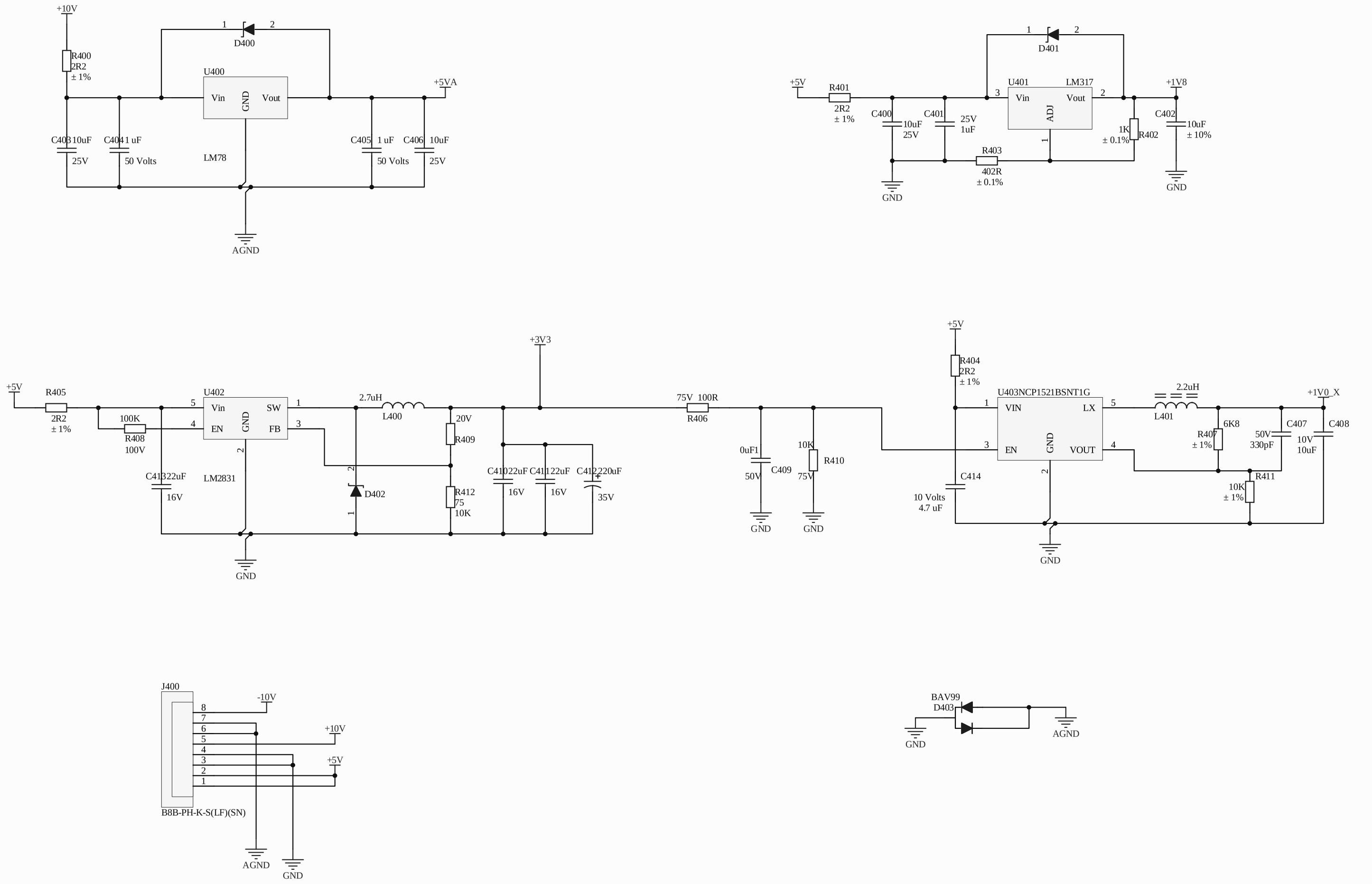
syncore technologies


STM32F100C4T6B
U501A
U501B
NCP303LSN09T1G
U505
NC7WZ07
U504
NC7SZU04M5X
U503
CONN Harwin M50-3151042
J500
Header XSYS
P502
U502C
Config
U502B
Power
U502A
IO
AT45DB011D-SSH-B
U500
ULPI
I2S_USB
I2C
SPI
DAC_SPI
MST_CLK_EN_49
MIX_RST
DIX_I2C
13MHz
ERROR
MCLK_SEL
PHY_RST_N
USB_SPDIF
CLK_DIX
+3V3
+1V0_X
GND
Title
*
PROCESSORS.SchDoc
Size
A3
Number
Version Control Revision
Unknown revision
Revision
Rev
Date:
2011-12-19
Sheet * of *
File:
PROCESSORS.SchDoc
Drawn By:
*
0.5
syncore technologies



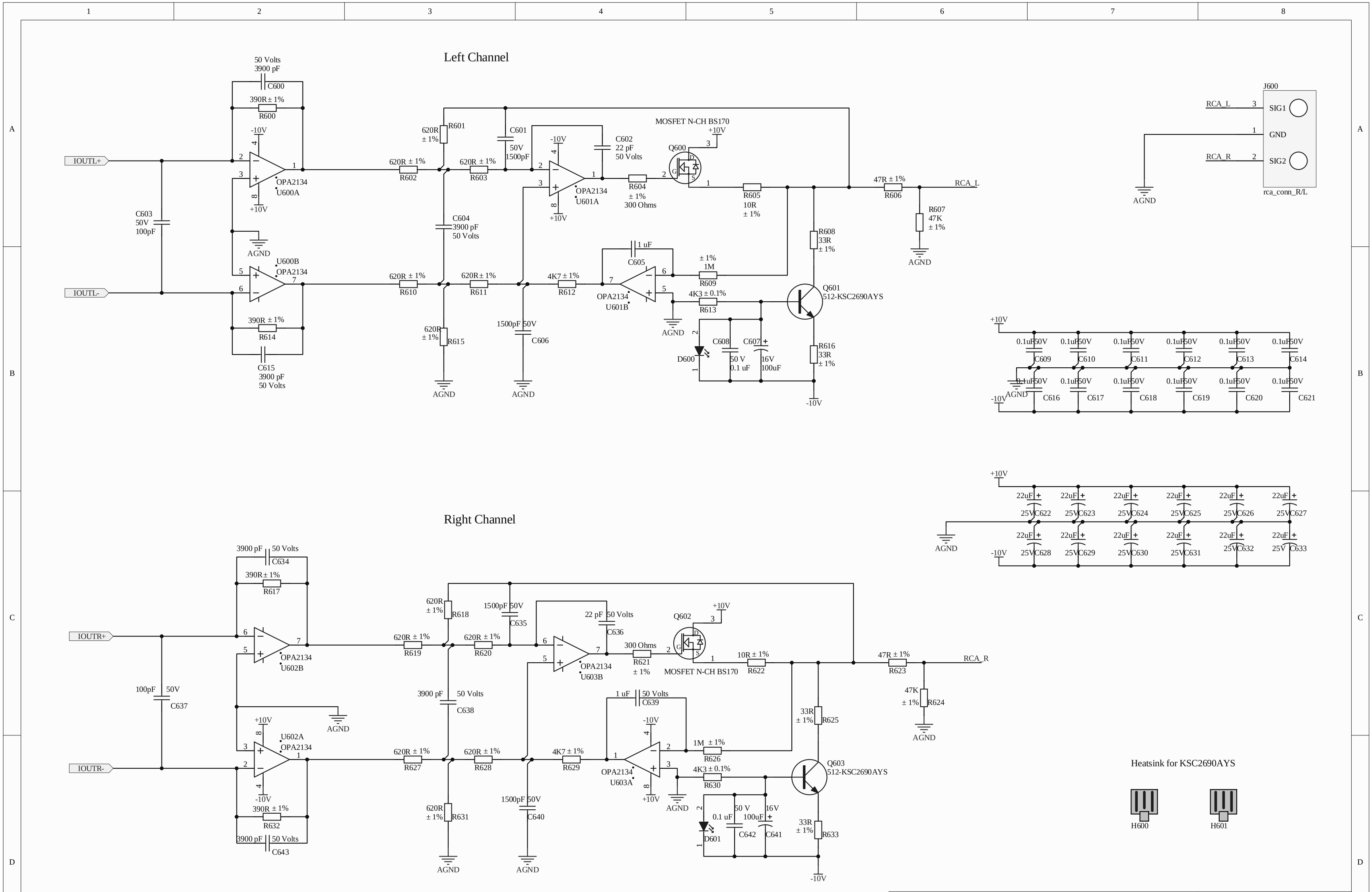
Title			
*			
USB.SchDoc			
Size	Number	Version Control Revision	Revision
A3		Unknown revision	Rev
Date:	2011-12-19	Sheet * of *	
File:	Inputs.SchDoc	Drawn By: *	

syncore technologies



Title	*		
	Powersupply.SchDoc		
Size	Number	Version Control Revision	Revision
A3		Unknown revision	Rev
Date:	2011-12-19	Sheet * of *	
File:	Powersupply.SchDoc	Drawn By: *	

syncore technologies



Title	*		
	Analog_out.SchDoc		
Size	Number	Version Control Revision	Revision
A3		Unknown revision	Rev
Date:	2011-12-19	Sheet * of *	
File:	Analog_out.SchDoc	Drawn By: *	

syncore technologies

Bibliography

[1] Taylor Morey Abraham I. Pressman, Keith Billings. *Switching power supply design*. McGraw-Hill, 2009. ISBN 978-0-07-159432-5.

[2] Temporary account for Schanghai Trainers. Working with version-controlled database libraries, 2010.

[3] Jeffrey H. Lang Anant Agarwal. *Founations of analog and digital electronic circuits*. Elsevier, 2005. ISBN 1-55860-735-8.

[4] Cardell Bob. *Audio power amplifiers*. McGraw-Hill, 2010. ISBN 978-0-07-164025-1.

[5] MX com. Crystal oscillator circuit design, 1997.

[6] Josie Di Costanzo. Component, 2009.

[7] Self Douglas. *Audio power amplifier design handbook*. Elsevier, 2009. ISBN 978-0-240-52162-6.

[8] Fred Eady Lewin Edwards David J. Katz Rick Gentile Ken Arnold Kamal Hyder Bob Perrin Creed Huddleston Jack Ganssle, Tammy Noergaard. *Embedded Hardware*. Elsevier, 2007. ISBN 978-0-7506-8584-9.

[9] Ben Jordan. Linking a simulation model to a schematic component, 2011.

[10] Phil Loughhead. Using components directly from your company database, 2011.

[11] Bengt Molin. *Analog elektronik*. Studentlitteratur, 2001. ISBN 978-91-44-05367-7.

[12] David Parker. Linking existing components to your company database, 2008.

[13] Matthys J. Robert. *Crystal oscillator circuit*. Krieger, 1983. ISBN O-89464-552-8.

[14] Jung Walt. *Op amp applications handbook*. Elsevier, 2004. ISBN 0-7506-7844-5.