Darmstadt University of Technology

Department of Electrical Engineering and Information Technology
Institute of Microelectronic Systems
Prof. Dr. Dr. h.c. mult. Manfred Glesner

Hardware and Software Implementation of a Radio Frequency High-Speed Data Conversion Unit for Digital Control Systems

Submitted for the degree of Bachelor of Science in Information and Communication Engineering

Author : Mohammad Shahab Sanjari
Advisors : Dr.-Ing. Peter Zipf (TU-Darmstadt), Dipl.-Ing. Martin Kumm (GSI)

Start : May 02, 2006
End : October 02, 2006

Herewith I declare, that I have made the presented paper myself and solely with the aid of the means permitted by the examination regulations of the Darmstadt University of Technology. The literature used is indicated in the bibliography. I have indicated literally or correspondingly assumed contents as such.

Darmstadt, October 2006

Mohammad Shahab Sanjari

In the Name of God, The Compassionate, The Merciful

Abstract

Digital control systems are always preferable where precision and flexibility matter. This thesis describes the construction and implementation of a radio frequency high-speed interface for use in a variety of applications in the heavy ion accelerator facility at Geselschaft für Schwerionenforschung mbH in Darmstadt. In this work, the author tries to shed light on some considerations concerning the design of such an interface.

Hardware design has been detailed along with problems and achievements during the construction phase. Using a hardware description language, the author implements and explains the steps needed to build a communication link between the design and the host system.

The last chapters portray the outcomes and contemplations for improvements and corrections of the design for the future revisions.

Acknowledgments

I would like to dedicate this work to my dear mother and father, to whom my absence during the studies had been most unpleasant.

I wish to express my sincerest gratitude to Dipl.-Ing. Martin Kumm at GSI, who always had a generous answer to my questions and supported me patiently through the work. Many thanks go to Dipl.-Ing. Peter Moritz and Dr. Harald Klingbeil at GSI who with a deep insight provided the most practical advices at the moments where they were most needed.

I would like to thank Dr.-Ing. Peter Zipf, my thesis advisor at Darmstadt University of Technology, for his admirable concerns and practical information during the work.

I also thank Mr. Manoj Badkas, MSc., who helped me with useful advices in VHDL programming.

I like to acknowledge my brother, Mr. Mohammad Ali Sanjari, PhD candidate at Amirkabir University of Technology in Tehran, Iran, for his invaluable supports during my studies in Darmstadt.

I wish to thank Prof. Dr. Dr. Manfred Glesner, for the lecture on Electronics and beyond. Ever since I have never lost my passion for circuit design, and I hope I won't ever.

Contents

List of Tables

List of Figures

1 Thesis Overview

1.1 About GSI

GSI operates a large, in many aspects worldwide unique accelerator facility for heavy-ion beams. Researchers from around the world use the facility for experiments in basic research[1]. GSI is federally funded and is a member of the Helmholz Association with more than 1000 employees. The chief tools in GSI are the Universal Linear Accelerator (UNILAC), the heavy-ion Synchrotron SIS18 and the Experimental Storage Ring (ESR)[2].

Currently researchers are planning the new international accelerator facility, FAIR, with a circumference of about 1.1 km. This new facility has many advantages over the existing one[3].

1.2 About this Thesis

1.2.1 History of the Design

In 2003, Martin Kumm at GSI provided a solution for an interface for analogue to digital conversion with a more specific application in an automatic gain control system using a micro-controller [22]. This solution is still used. Later, more and more use-cases have been found that would profit from a faster and more general board. This led to the idea of using programmable logic devices and accomplish the task in a hardware description language.

In 2006 Johannes Jöst started his Diploma Thesis on designing an interface board with 10 MSPS ADCs and 30 MSPS DACs controlled by a CPLD [20]. His work has been an underlying basis for the following thesis.

[1] Source: www.gsi.de
[2] Source: www.wikipedia.org
[3] Please refer to www.gsi.de for more information on the FAIR project.

1.2.2 Thesis Scope

This thesis continues these efforts and extends the design to a much faster system. The analogue to digital and digital to analogue conversion rates has been enhanced to support up to the state of the art 125 MSPS and 210 MSPS respectively. The dimensions of the board have been changed to allow better temperature characteristics and proper placement of components. Efforts have been made to lessen the inter-channel interference and reduce noise and signal crosstalk specially that of the main clock signal.

During this thesis following tasks have been undertaken.

- Testing the existing ADC/DAC board and writing some routines to get familiar with the design environment.
- Fully redesigning the libraries and the schematics of the next revision.
- Learning new techniques on designing high-speed boards.
- Routing and production of the new PCB revision.
- Writing routines in VHDL for testing the prototype.
- Implementing the first fully functioning version of the CPLD-side code for the communication protocol.
- Writing routines in VHDL for testing the communication link with the host system.

1.2.3 Use Cases

Some of the use cases of the design include but are not limited to:

- Fast replacement of the ADC/DAC board [20].
- Digital Amplitude Control [10].
- Digital Eigenfrequency Control [10].
- Synthesizer for Barrier-Bucket System [25]

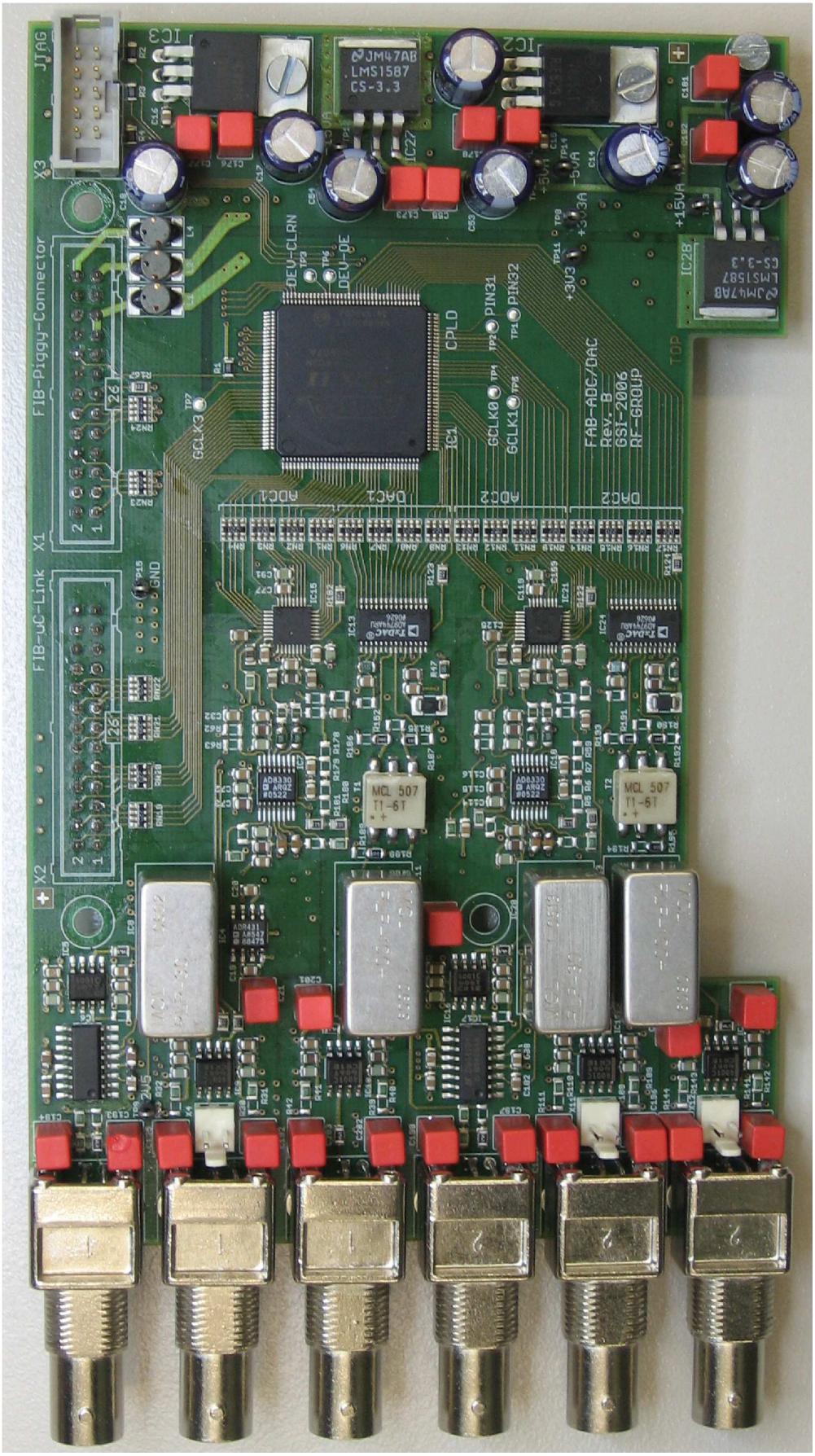


Figure 1.1: Top View of the Final Populated Board

2 Hardware design

2.1 Introduction

The board is designed to meet several requirements for application in different control systems. Signals applied to the board range from some ten millivolts up to 2 volts peak to peak. The design allows DC as well as AC coupled signals. It features two channels each with an ADC input, a monitor output and a DAC output. The design which is called FAB (FIB Application Board) will be used as a sort of daughter board connected via a digital interface to a main board named FIB for FPGA Interface Board. The FIB board in turn implements different buses and interfaces and acts as a central data management and signal processing unit and a bridge to other units in the digital control system such as the DSP unit[1].

Several 0Ω resistors have been used in the circuit to allow the change of configuration at the time of manufacturing simply by using a different pick and place data set for the mounting machine.

2.1.1 Modes of Operation

In the AC operation mode, signals go through variable gain amplifiers (VGA). Due to the relatively high offset of the VGAs in the DC operation mode, the VGAs are deactivated so that the signals are connected directly to the ADCs after passing through the input stages. The VGAs could have a fixed gain, or be controlled by the output of the DACs (for use with future automatic gain control algorithms).

[1] Please refer to [21] for more information on the digital cavity synchronisation project in GSI, or refer to section 1.2.3 for other use cases.

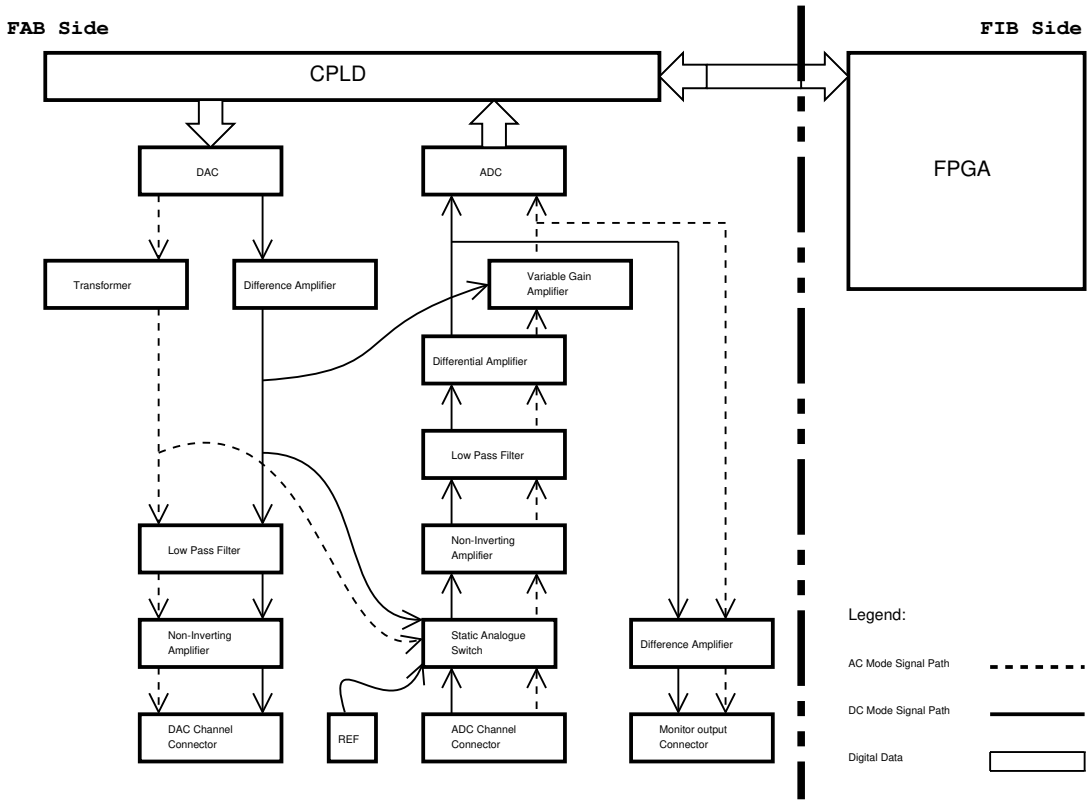


Figure 2.1: System Block Diagram Showing one Channel.

The system block diagram is depicted in figure 2.1. Different elements are shown as interconnected blocks. The difference between a board configured for AC operation and a board configured for DC operation is apparent in this figure.

2.1.2 External Connectivity

The FAB board is connected to the FIB board using two connectors. Clock, an address bus and a 16-bit data bus as well as some control signals are provided on the connectors. BNC connectors have been chosen for the RF side. In applications such as AGC where the signal applied to the board is nearly DC, 2 pole Molex[2] connectors are used. Another connector and some test pins is reserved for programming/debugging purposes.

[2] www.molex.com

2.2 The Power Supply and Reference Voltage

Due to different supply requirements on the board, it was necessary to study the application field. FAB is designed to be used in 19" racks as well as cased stand alone in aluminium housings. The rack system provides regulated supply voltages for the cards.

2.2.1 Power Zones

As it could be seen in figure 2.2, internal power planes have been sliced to allow separation between the channels as well as allowing enough copper to keep the conductor impedance as low as possible.

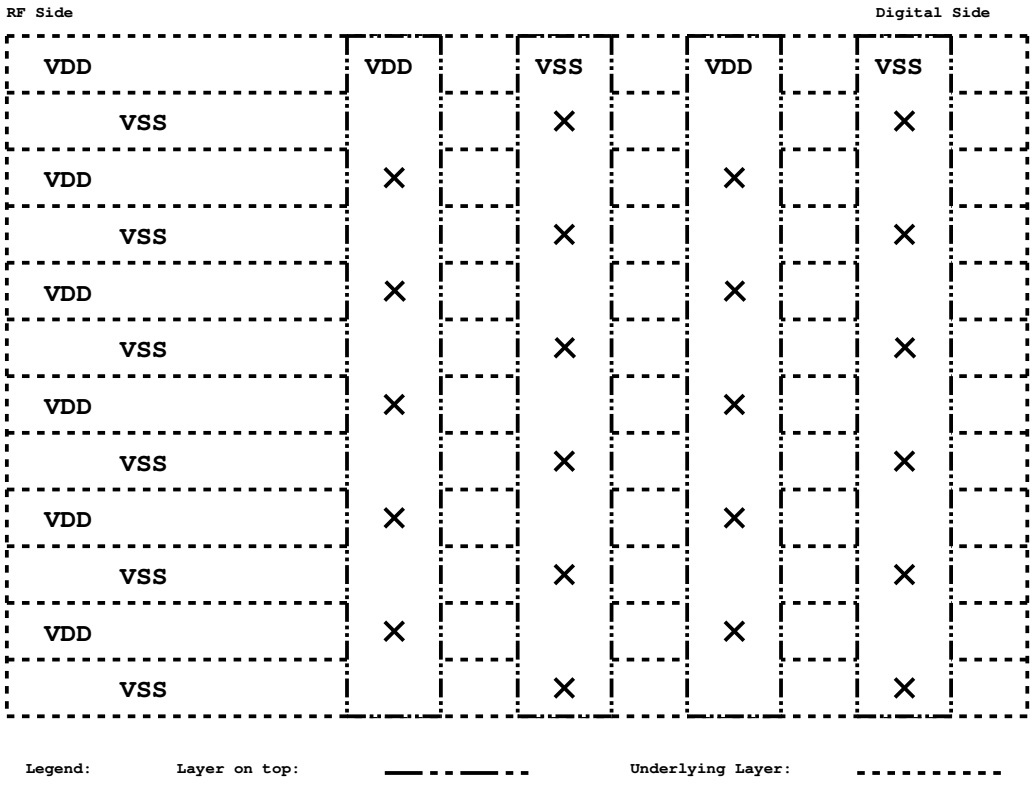


Figure 2.2: Arrangement of Power Polygons on the middle layers of the PCB.

The planes that carry $\pm 15V$ and $\pm 5V$ have this form. The effective connection has been made possible by the vertical stripes which act as bridge connections between the slices.

For the purpose of providing a reference voltage, we have used an ADR431 from Analog Devices[3]. This is a low noise, low temperature coefficient 2.5V voltage reference with a wide operating range. The voltage reference contributes to the converters as an external reference as well as the calibration circuitry of each channel (see section 2.5).

2.3 Grounding Issues

The position of the system star point has always been a concern when designing high speed circuits. After several meetings we decided to slice the ground plane and allow connections only where it is suitable. As always, compromises had to be made, since the front panel serves as an unwanted secondary star point. Making the central connection as low impedance as possible, we could nearly ignore the effect of the front panel (See figure 2.3. Nonetheless, the ground plane island which has been isolated for the CPLD and it's digital interface to the converters had to be treated specially.

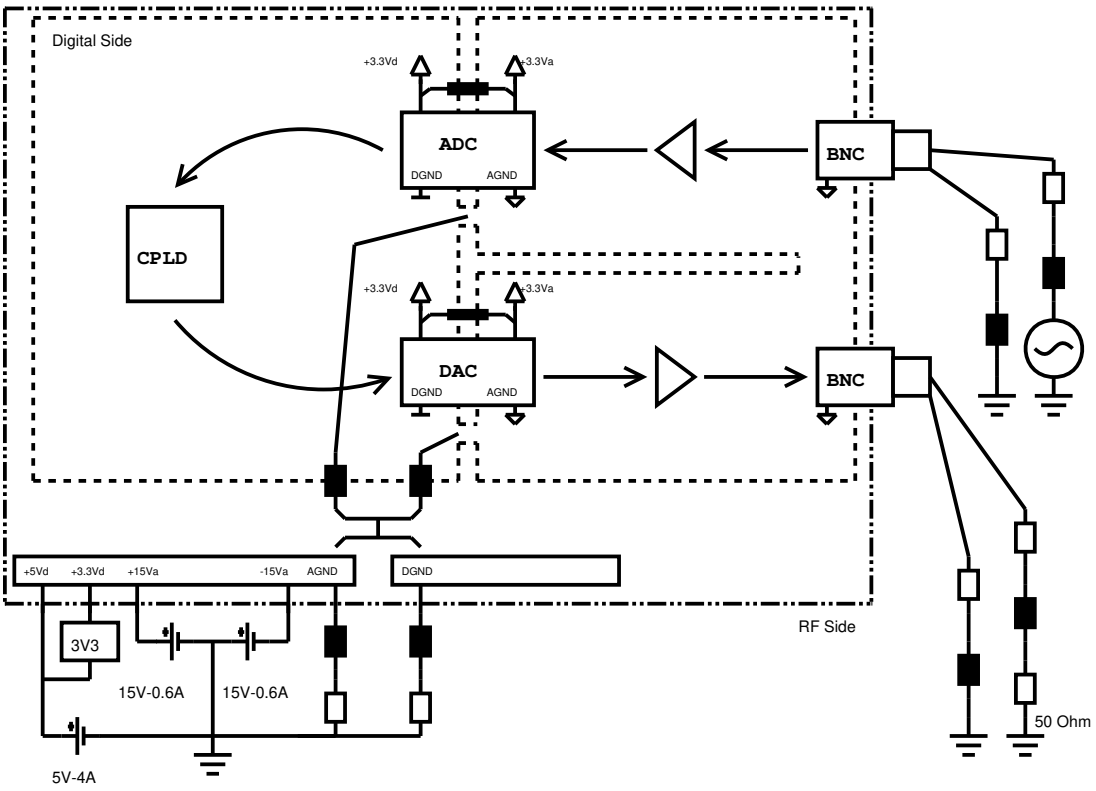


Figure 2.3: Overview of the Board Signals, Backplane and Target Connections.

[3]www.analog.com

A stripe of copper on the bottom layer defines the main star point directly beneath the converters (see figure 2.4). This is very desirable as stated in many data sheets and application notes[4] since the difference between the voltage levels on either side of the converters remains low. Care has been taken to keep a constant distance between channels themselves and the ground plane used for the digital part in order to reduce inductive noise.

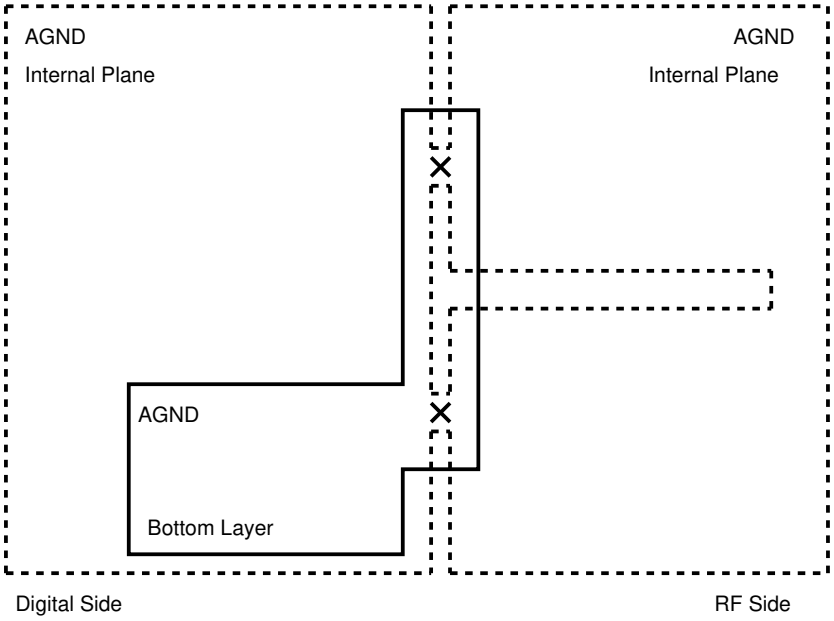


Figure 2.4: Ground Plane Topology.

For the return path of the digital signals connected to the CPLD, there is an extra connection parallel to the main system star point to allow transfer of higher current peaks. These would otherwise appear as fluctuations and would distort the signal passing through the converters.

2.4 The CPLD

The chosen CPLD is from the MAXII ® family of CPLDs from ALTERA[5]. It operates off an almost dedicated 3V voltage regulator. It contains 1270 macro cells and is packaged in a 144-QFP case. The CPLD is clocked directly from the FPGA board. The clock signal is grounded over a $1k\Omega$ resistor so that the rise and fall times remain

[4]Please refer to [6], [5], [29] and [9] for more information on grounding.
[5]www.altera.com

short. Depending on the specific application, CPLDs with different speed grades could be mounted on the board. On the prototype boards, the fastest CPLDs with 3nS propagation delay are used. Important signals are accessible via test pins.

2.4.1 Clock Distribution

Distributing clock signal between the components on the board needed a well planned strategy. Clock traces should have a low impedance and be as short as possible. Special care has been taken so that these signals have a larger distance to other traces, otherwise they would easily influence the quality of signals in the main path. As An Example, According to the data sheet of the AD9744 DAC [14], any noise or jitter in the clock will transfer directly into the DAC output.

2.5 Calibration Circuitry

A series of applications need precise, absolute signals. All analogue components have tolerances so that for obtaining absolute signals, calibration must be performed. Since the era of potentiometers is over, in a digital control system, this task has to be accomplished digitally. For this purpose, each channel is provided with an analogue switch. This switch is in turn controllable by the CPLD. The idea is to perform a linear approximation on the output signal under the influence of existing physical restraints, which could nonetheless be almost considered as linear. Additionally a simple passive network of reactance/resistance has been place on the signal path to compensate for the non linearities of the switch itself.

During a calibration process, ground and the reference voltage are switched sequentially into the ADCs, the values are saved in special registers. Then linear regression will be performed, i.e. the slope and the axis intercepts are calculated. After calibration is done, if the CPLD is asked to deliver calibrated data, it divides the value into the already calculated slope and subtracts the axis intercept before sending the data to the main FPGA board, FIB.

The calibration algorithm is not implemented in the prototyping stage of the project though, until further developments and study of each specific application has been carried out. It should be mentioned that the calibration operation is limited to the DC operation mode. For calibration in the AC operation mode it is possible to feed the DAC output into the switch.

2.6 Signal Flow and Filtering

The scheme used for the overall signal flow follows the 50Ω circuit technique used in many radio frequency designs. Signals have been terminated with precision 49.9Ω resistors where needed. The gain of the amplifiers is trimmed to keep the signal unamplified throughout the signal path so that the gains are solely controlled by the VGA or not at all in case of DC-coupled connection. For the latter, the gain of the input stage might be trimmed to a fixed value. These settings are strongly application dependant.

2.6.1 Connectors and the Origin of the Signals

As stated in the introduction, BNC connectors are used at the RF side. The signals applied to the board could have a frequency of up to 10 MHz and an amplitude of up to 2 V_{PP} in AC and up to 10 V_{PP} in DC operation mode. Each channel has a monitor output. Upon correct choice of configuration resistors, this output could be used to get an analogue signal which has already been amplified by the VGA, in case digital values are not intended in that specific application.

The DAC outputs are capable of both DC and AC coupled operation. The latter is achieved using a wide band RF transformer, T1-6T from Mini-Circuits[6] which contributes to hight-quality signal transmission.

2.6.2 High-speed Op Amps

The operational amplifiers used in input and output stages are THS4001 from TEXAS Instruments[7]. These are wide band voltage-feedback amplifiers with ideal characteristics in both inverting and non-inverting configurations. This amplifier has been chosen because of it's good performance in 50Ω systems. The common mode rejection ratio (CMRR) is also good in the desired operation frequency. Figure 2.5 shows the closed loop gain of the op amp versus operation frequency.

[6] www.minicircuits.com
[7] www.ti.com

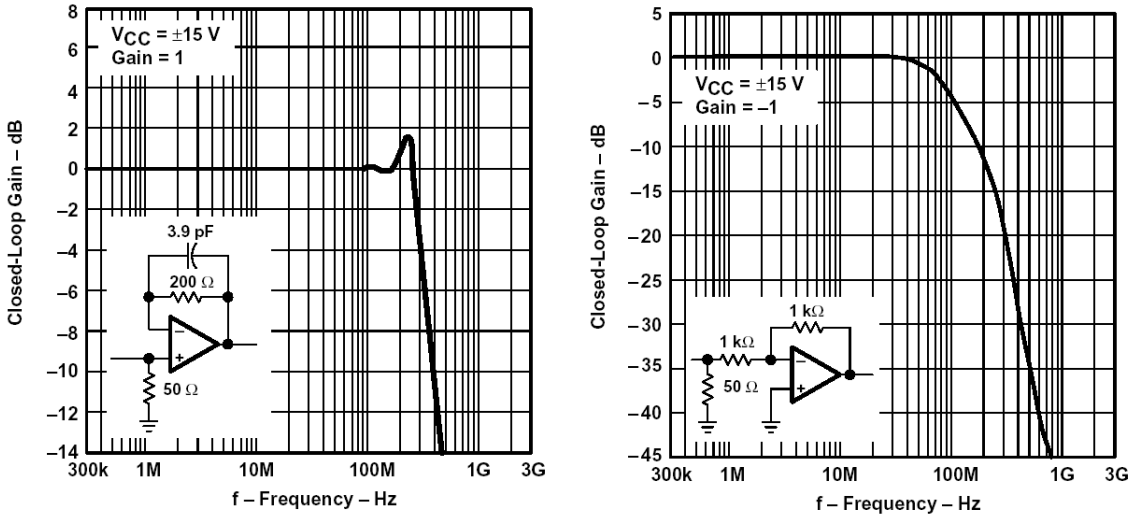


Figure 2.5: THS4001: Closed Loop Gain vs. Frequency [16]

2.6.3 The Differential Amplifier

The quality of performance of modern ADCs is much better if the signal applied to them is differential in nature. Usually the signal has to be levelled up to a certain bias voltage. This is best achieved using the so called differential amplifiers. The differential amplifier used in the circuit is AD8131 from Analog Devices. It is a differential or single-ended input to differential output driver that needs no external components for a fixed gain of 2 [13]. It could be considered as an impedance converter, in many applications a better replacement for transformers (specially in the DC operation mode where use of transformers is impossible), being less susceptible to magnetic interference. It's -3 dB Bandwidth of 400 MHz is more than acceptable for our application.

2.6.4 Noise Elimination and the Filters

Throughout the design it has been tried to use as many decoupling capacitors as needed to bypass glitches at the components' supply pins. Polyester film capacitors have been used abundantly to block unwanted RF frequencies. Digital signals have been provided with a small series resistance to reduce noise due to possible reflections. CPLD signals which control the analogue switch have low pass filters on them.

Keeping an eye on the sampling theorem and the Nyquist frequency, the filters chosen for the channels are each 7th order low pass filters with -3 dB corner frequency

of 100 MHz for the DACs and 50 MHz for the ADCs respectively. The filters are of type PLP-100 and PLP-50 from Mini-Circuits. These settings are also application dependant.

2.7 The Variable Gain Amplifier

Variable gain amplifiers (VGA) play a central role in applications involving automatic gain control. The chosen VGA is from Analog Devices [11]. It has a moderately low distortion from DC to 150 MHz, it could therefore be considered a wide band amplifier. The peak differential input it ±2V which allows sine wave operation at 1 V_{RMS} with enough headroom. These could even be driven from a single ended source, but in this design they are driven differentially to improve signal quality on the board. It's outputs have the same features. The output impedance is 150Ω.

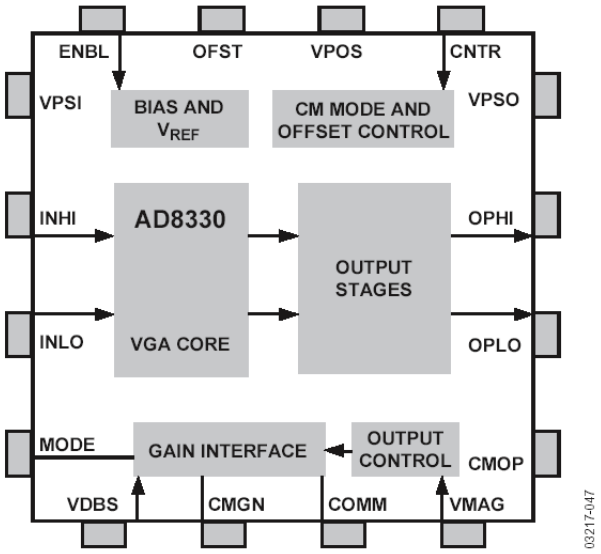


Figure 2.6: Block Schematic of the AD8330 [11]

2.7.1 Transfer Function

The basic gain function is linear-in-dB, controlled by the voltage applied to the VDBS pin. The gain may be changed ranging from 0 dB to 50 dB for control voltages between 0V and 1.5V with a slope of 30mV per dB. Figure 2.7 illustrates this feature. The

voltage on the MODE pin changes the polarity of the slope of the transfer function, so the amplifier could do without an inverting stage in case of need.

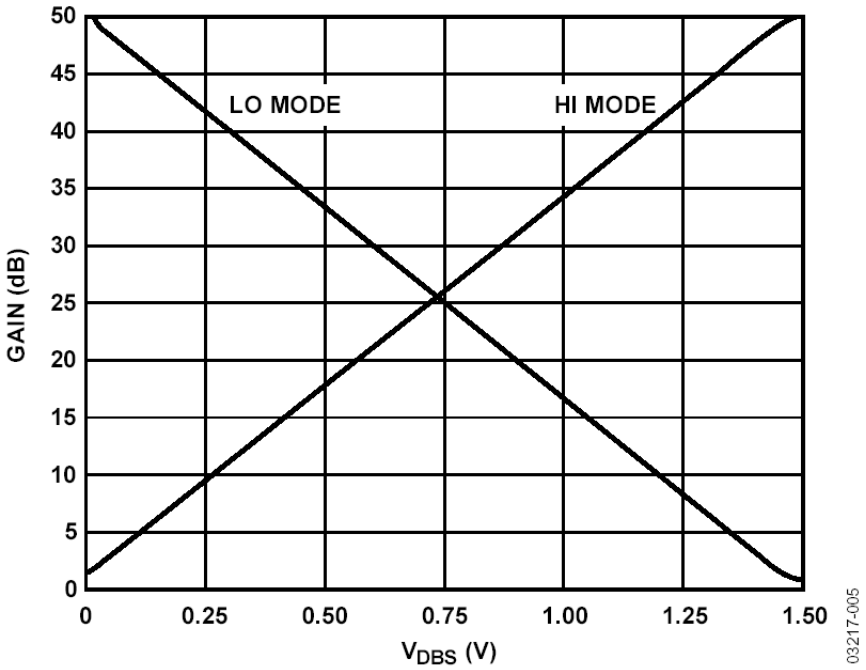


Figure 2.7: AD8330 Gain vs. V_{DBS} [11]

A second gain control port is provided on chip at pin VMAG, which allows the user to vary the numeric gain from 0.03 to 10 as shown in figure 2.8. Using this feature, the basic gain set by the VDBS pin could be repositioned to any value from 20 dB higher to 30 dB lower.

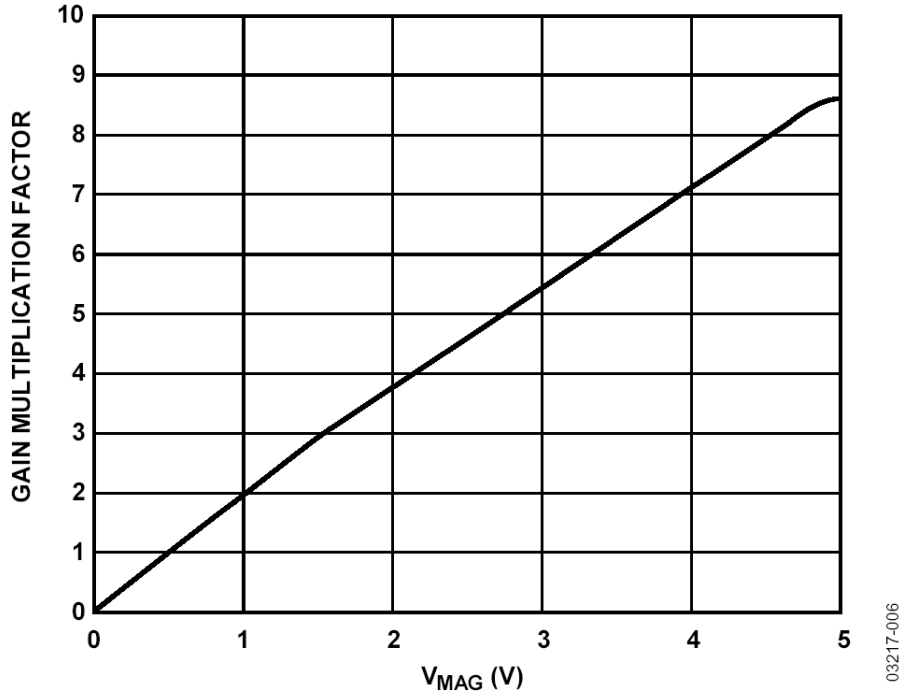


Figure 2.8: AD8330 Gain vs. V_{MAG} [11]

The output voltage may then be approximated as follows:

$$V_{OUT} = 2 \times V_{IN} \times V_{MAG} \times 10^{(\frac{V_{DBS}}{0.6V})}$$

Analog Devices has published an on-line simulator for this transfer function on their web site. By default the output voltage is placed half-way through the power supply. It may also be trimmed using the CNTR pin. This is particularly useful in case following stages such as ADCs require a certain input voltage offset.

Although AD8330 is capable of operation in DC-coupled mode, we have decided to use it on the board only for the the ac-coupled signal path. On the prototype board [20], the chip had not shown a desirable output DC offset.

2.8 Analogue to Digital Conversion

Speed of data conversion had been of primary importance in the design. The chosen converter is from Linear Technology's family of LTC22XX converters[8]. This product

[8] www.linear.com

family features pin compatible devices each having different conversion speeds, resolutions and of course costs. This allows application specific choice of devices without changing the board layout. Table 2.1 lists possible alternatives.

125Msps	LTC2253 (12-Bit)	LTC2255 (14-Bit)
105Msps	LTC2252 (12-Bit)	LTC2254 (14-Bit)
80Msps	LTC2229 (12-Bit)	LTC2249 (14-Bit)
65Msps	LTC2228 (12-Bit)	LTC2248 (14-Bit)
40Msps	LTC2227 (12-Bit)	LTC2247 (14-Bit)
25Msps	LTC2226 (12-Bit)	LTC2246 (14-Bit)
10Msps	LTC2225 (12-Bit)	LTC2245 (14-Bit)

Table 2.1: Pin Compatible family of LTC22XX ADCs.

2.8.1 Structure

For the prototype boards LTC2255 and LTC2249 with 125 MSPS and 80 MSPS respectively were used. Each have a 14-bit data bus. They run off a single 3V power supply. An internal clock stabilisation feature could be turned on in which case the ADC generates a 50% duty cycle pulse on the rising edge of the single ended clock signal. This is desired on boards with sensitive clock distribution, since the accuracy of conversion depends on both rising and falling edges of the clock.

The ADCs are capable of straight binary or 2's complement output. Poor matching can result in higher order harmonics, so it has been tried to keep the output impedance of the previous stage near 100Ω. For the sake of reducing power consumption, the chip's power saving NAP mode have been used. This is faster than it's SLEEP mode in exchange for a couple of milliwatts more power consumption. In this mode, it takes only 100 clock cycles for the ADC to wake up and provide valid data on the outputs, which otherwise have been in the Hi-Z state. A central ground pad on the bottom of the ADC's lead-less package allows an effective heat transfer to the board.

2.9 Digital to Analogue Conversion

According to the desired specifications, we decided to choose from the TxDAC ® family of Analog Devices's DACs. AD9744 has a single supply 14-bit interface also capable of straight binary or 2's complement input. It could be clocked up to 210 MHz. The DAC has been a simple to use component. More details on power connections and noise is provided in the related sections (see section 2.6.4).

2.10 Mechanical Aspects

2.10.1 External Dimensions

As stated before, the board is designed to be placed in 19" racks. The board measures 160mm by 100mm which corresponds to a standard Euro format board. It is provided with M3 holes for connection to the main board. The contour has been milled to free the area above the IDC connectors of the main board.

2.10.2 Board Layout

Care has been taken that the placement of the elements are optimal in the sense of shortest distance to the corresponding power source. As it might be seen from the figure 2.9 polygon planes have been stretched to ensure this feature.

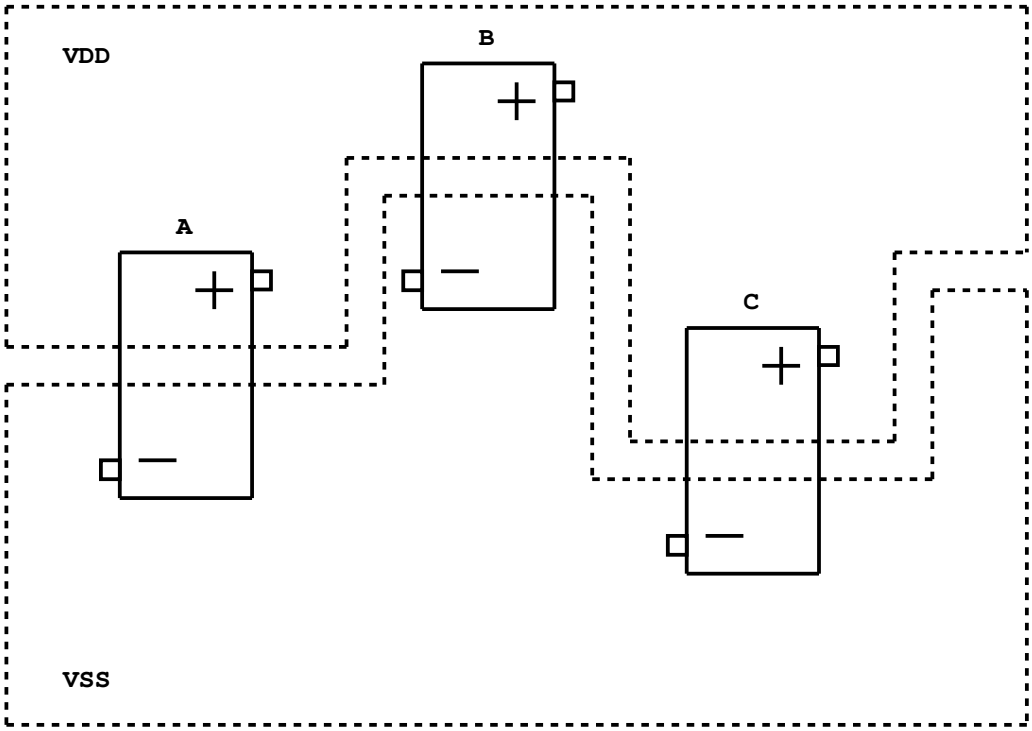


Figure 2.9: Orientation of the Components to the Differential Supply Polygons.

Experience confirms that a good placement is the most important factor affecting the routing phase of a layout job. Apart from that, good placement reduces inter-

channel interference and unwanted signal crosstalk. Also, power components have been placed far from the sensitive elements.

2.11 Production

Production is the most time consuming phase of a design. The choice of used elements had been an iterative process since some of them were not available in the desired quantities, package forms or at all. In the case of the AG8330 VGA, for example, the board had to be reconfigured almost in last days before sending the job to the fabrication house, since the required package was not at deliverable any more through the distribution network in Germany. Many components had long delivery times. Some very expensive parts, such as the ADCs, have been ordered as samples, until after the verification, buying them in larger quantities becomes more plausible.

2.11.1 PCB Fabrication

The PCB job was sent to CONTAG[9] fabrication house. The board consists of 6-Layers, containing no blind or buried vias. Following are the physical specifications of the board:

Layers	6
Material	FR4
Copper	$35\mu m$
Finish	HAL lead free
Track/Distance	$150\mu m$
Drill/Annular Ring	$300\mu m/170\mu m$
Number of Drills	620
Electrical Check	Flying Probe
Solder Mask	Top and Bottom, Green
Silkscreen	Top and Bottom, White

Table 2.2: Physical Specifications of the Board.

[9] www.contag.de

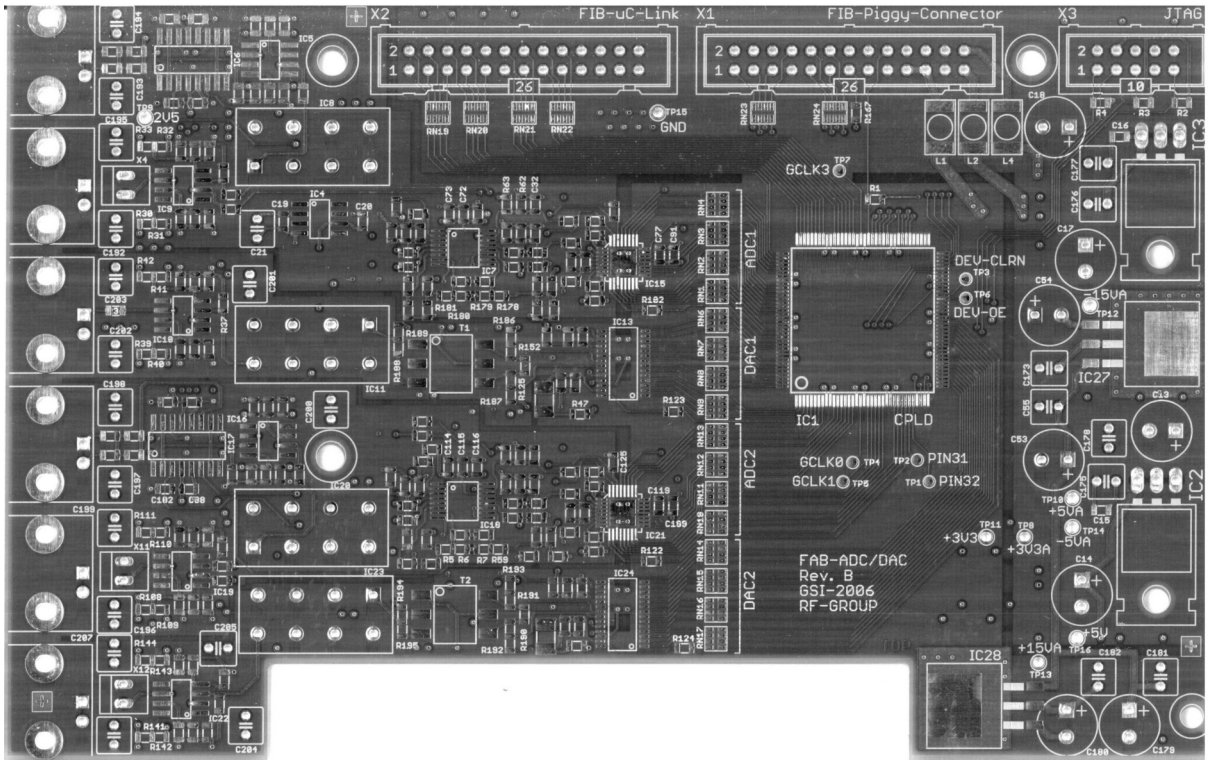


Figure 2.10: Top View of the Unpopulated Board.

The fabrication of the boards took 5 working days. They were sent promptly and the boards had a good quality.

GERBER Data

The CAM job has been exported in the newer RS247-X format. This is sometimes called extended GERBER[10] or X-GERBER. These files contain the coordinates for the photo plotter and other manufacturing information. Unlike the older version where the aperture information are stored separately, in the newer version, these are located in the CAM files.

[10] www.gerberscientific.com

Drill Data

EXCELLON[11] file format has been chosen for the drill data. "In electronics manufacturing, an Excellon file is a text-based file format which is used to control the actions of a CNC drilling machine, commonly used in the drilling of printed circuit boards (PCB). The Excellon file format is a variant of standard RS-274C. It consists of commands to instruct a CNC drilling machine to drill holes of specific diameters at specific locations on a PCB."[12]

2.11.2 Component Mounting

After the PCBs and the last parts arrived the boards were ready to be populated. The paste mask which has been specially ordered according to the needs of GSI's internal automatic mounting facility arrived a couple of days later. The mask was made of $100\mu m$ steel sheet. Both top and bottom sides of the PCB were placed on the same mask mirrored in respect to each other, so that the solder cream could be applied on the PCBs using only one side of the mask, therefore eliminating the need to turn and cleanse the mask after application on each side.

Now the component coordinates and orientation had to be exported into a space-separated and column-oriented text file. This format was suitable for the GSI's internal automatic mounting facility. For the prototyping purposes at GSI, usually the PCB is populated only with the SMD parts. The rest of the elements had to be mounted by the author. As it has always been, this too had been a time consuming task.

2.12 PC Link: The UDL Board

For future expansion and testing purposes, a fast connection to personal computers was needed. Soon it became obvious that USB connection is most appropriate for such an application. The card will be inserted into the DSP-Link connector of the FIB board, so that the FAB/FIB combination could be connected to the computer and tested separately without being in the main control system. This adds to the flexibility and has the advantage of ease in localisation of possible programming errors.

It should be remarked the production of the UDL board was outside the scope of the thesis. At the time of completion of this thesis, the schematics, board outline, connections and functionalities had been agreed upon.

[11] www.excellon.com
[12] Source: www.wikipedia.org

2.12.1 Circuit Description

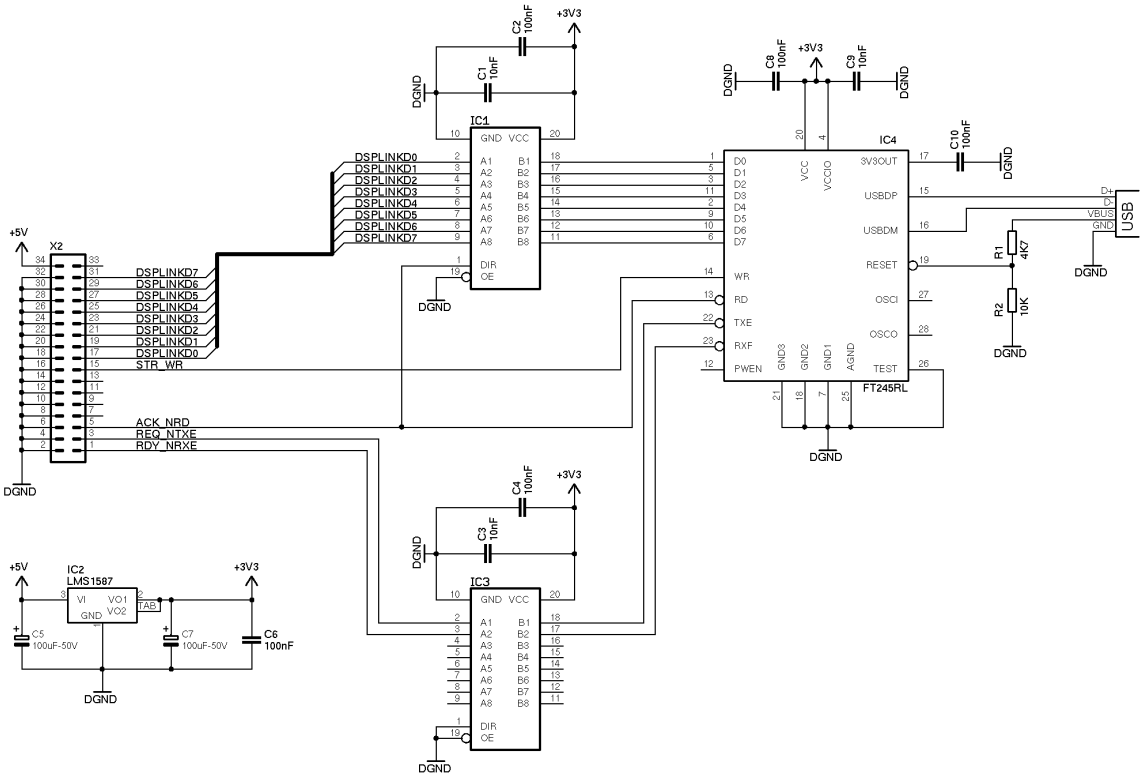


Figure 2.11: The Schematics of the UDL Board.

USB FIFO

As it may be seen from figure 2.11, the USB FIFO chip FTD254RL from FTDI[13] has been used. It provides a parallel FIFO bidirectional data transfer interface with the entire USB protocol being handled on the chip. The interface is easy-to-implement on both computer and circuit side. The chip has been wired according to the data sheet [24] for a self powered configuration. The rest is pretty straight forward. In time, the existing library in GSI for an older version of the USB chip will be adapted for use with the newer chip.

[13] www.ftdichip.com

Power Supply and Bus Drivers

The signals on the DSP Link of FIB have 220Ω pull up and 330Ω pull down resistors each. The choice of the drivers was limited to those who are capable of delivering the current needed to drive these signals. In consequence, the voltage regulator had to provide all the current needed for the drivers and the FIFO chip. SN74LVT245B [15] from TEXAS Instruments fulfilled our needs. It is an octal bus transceiver supporting 3.3V and 5V inputs, and it outputs to a 3.3V system.

LMS1587 from National Semiconductor Corporation[14] in TO-263 package provides the required current [8].

[14] www.national.com

3 Software Design

3.1 Introduction

The notion of software is usually interpreted differently in the literature. Since VHDL is defined as a *hardware* description language, many do not consider the code written in VHDL as software. Nevertheless, throughout this thesis the coding part has been much *softer* than constructing the board itself. Herein, the term *software* is used to represent the routines that were written for the programmable logic devices.

3.2 Behavioural Description

According to the plan, to the host system, FAB should look like a black box with address, data and control bus. The different functionalities should then be accessible via registers that are implemented inside the CPLD. This means for instance that each ADC channel has it's own value register. When the host system needs to read a value from one of the ADCs, it puts the address of the register which corresponds to the actual ADC value. The value is then made ready by the CPLD on the data bus.

R/W	Adr.	Name	Description	Default Val.
W	0x00	CTRL	Control Register	0x4400
R	0x01	STAT	Status Register	Not Set
R	0x02	ADC1-VAL	Value of the ADC on channel 1	Not Set
R	0x03	ADC2-VAL	Value of the ADC on channel 2	Not Set
W	0x04	DAC1-VAL	Value of the DAC on channel 1	0x0000
W	0x05	DAC2-VAL	Value of the DAC on channel 2	0x0000
W	0x06	ADC1-CLKDIV	Clock div. const. on ADC ch. 1	0x0002
W	0x07	ADC2-CLKDIV	Clock div. const. on ADC ch. 2	0x0002

W	0x08	DAC1-CLKDIV	Clock div. const. on DAC ch. 1	0x0001
W	0x09	DAC2-CLKDIV	Clock div. const. on DAC ch. 2	0x0001

Table 3.1: Registers implemented inside CPLD.

There are more such registers programmed into the CPLD. Please refer to table 3.1 for a list of implemented registers. Using such register based organisation, future expansions are easy. As an example of a possible application, the automatic gain control or the automatic calibration could be named. In case of automatic calibration, some more registers are needed to handle the gain and offset values for each channel. Even the control register will be reset to it's default value and hence the global reset bit. So there is no need that the user does this manually. Table 3.1 also shows the default values of the registers used for the prototype boards which had 100MSPS ADCs and 210MSPS DACs.

All registers are 16-bit wide. The two most significant bits of the value registers of ADCs and DACs have been permanently set to zero, since these devices have a 14-bit wide data bus. Some registers are read-only whereas others are not. The value of all registers could nonetheless be read from the host system.

3.2.1 Special Registers

As shown in figure 3.1, the control register contains special purpose bits. Bit 2 is the global reset. If the host system writes a 1 to this bit, FAB will restart, setting all registers to their default values.

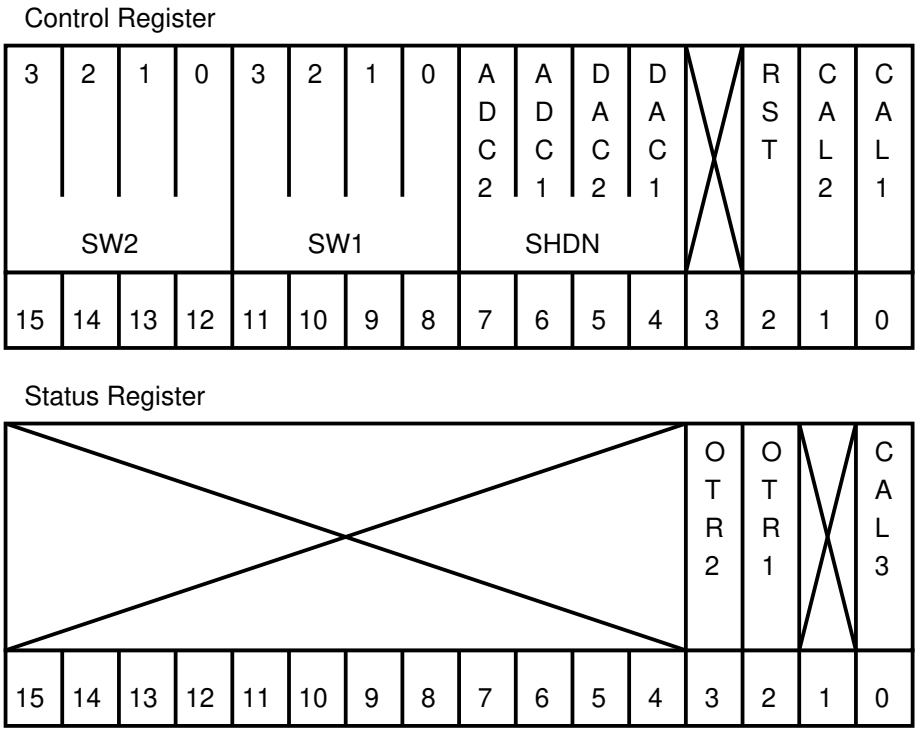


Figure 3.1: Control and Status Registers Implemented Into the CPLD.

The eight most significant bits are assigned to the two analogue switches. Writing to these bits changes the switching state of these ICs. The possible values are listed in table 3.2.

Value	Description
0x01	Connect ADC to 2.5V Reference
0x02	Connect ADC to GND
0x04	Default Signal Connection
0x08	Connect ADC to DAC output of the opposite channel

Table 3.2: Possible Values for the Analogue Switches.

The first two switch values are used to start a calibration operation for each channel. The last value is reserved for future applications such as enhanced AC signal calibration (see section 2.5). The default signal connection is achieved by writing

0x04 to the respective nibble.

Bit 4 to 7 of the control register sets each ADC or DAC of each channel in it's power down mode (NAP Mode for ADC and SLEEP Mode for DAC respectively). In these modes, each device consumes much less power than it's normal operation state.

The status register on the other hand is designed for read only operation, i.e. signalling purposes. Bit 2 and 3 are out of range indicators. CPLD sets these bits when the ADC from the respective channel falls into an overflow state. This happens when the signal applied to the ADC is greater than it's allowed input range. The first bit of this register is reserved for calibration operations.

3.2.2 Reset Generation

As stated before (see section 3.2.1) it is possible to reset FAB by writing a logic high in the first register's bit 3. But since the design is based on finite state machines (FSM), there must be a way to reset the main FSM. This is done using a simple counter. It counts some clock cycles before asserting an active high on the global reset signal which is used throughout the project as the reference reset. In this project reset signals are always active high.

3.2.3 Synchronisation

When two FSMs on both sides need to communicate, all delays throughout the paths including those needed for the communication protocol and those caused by PCB tracks need to be considered.

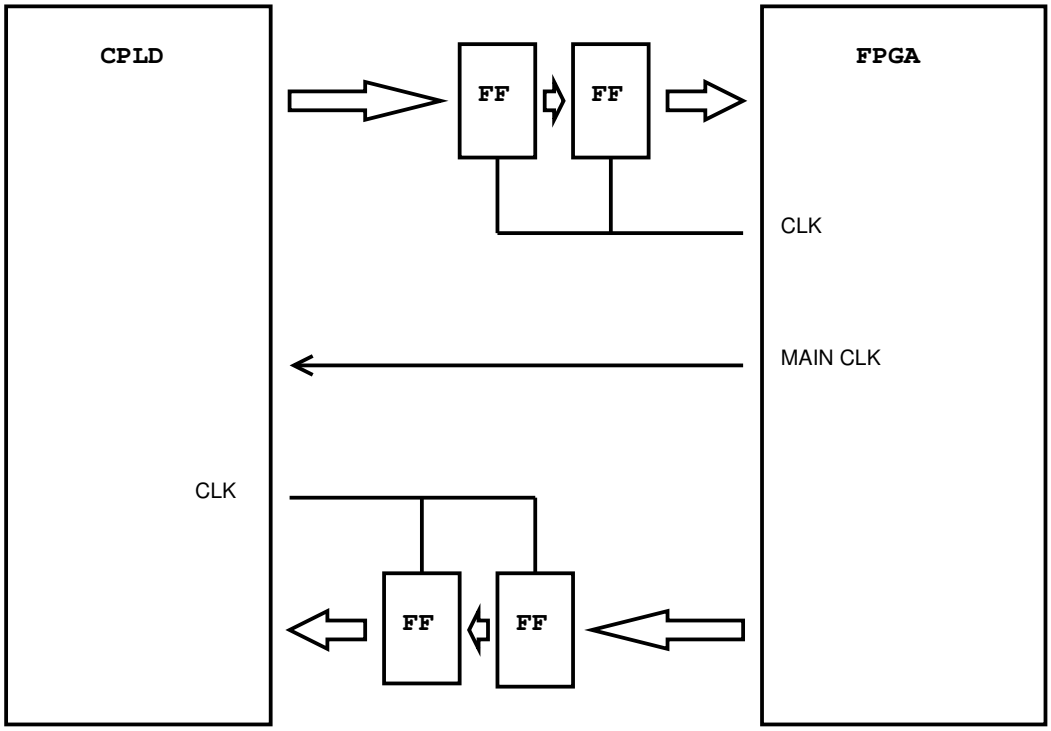


Figure 3.2: Crossing clock domains.

Crossing clock domains is the magic phrase. Figure 3.2 demonstrates a cure. All signals travelling from FIB reach the CPLD some time later because some of them like the signals of the data bus are routed through an external buffer and some like the signals of the address bus are not. Since the FPGA on the main board provides the clock signal for the CPLD, the CPLD must synchronise the arrived data using it's local clock, which is in turn a delayed version of the original FPGA clock. The reverse is also true. Signals travelling back to FPGA need to be synchronised with the local clock of the FPGA. Usually one flip flop is enough to achieve the desired behaviour. But in practical implementations of the code in the CPLD, two flip flops are used. The reason behind this is that in case the first FF goes into metastability, i.e. set-up and hold times are not met due to fast data transfer rate, the second flip flop avoids the output of the first metastable FF to be carried forward into the circuit and hence avoids the failure of the complete circuit.

3.2.4 The Bidirectional Bus Driver

The bus driver plays a central role in the project. Since one data bus is used as a bidirectional link to connect FPGA and CPLD together, one of such entity is needed

on either side of the bus.

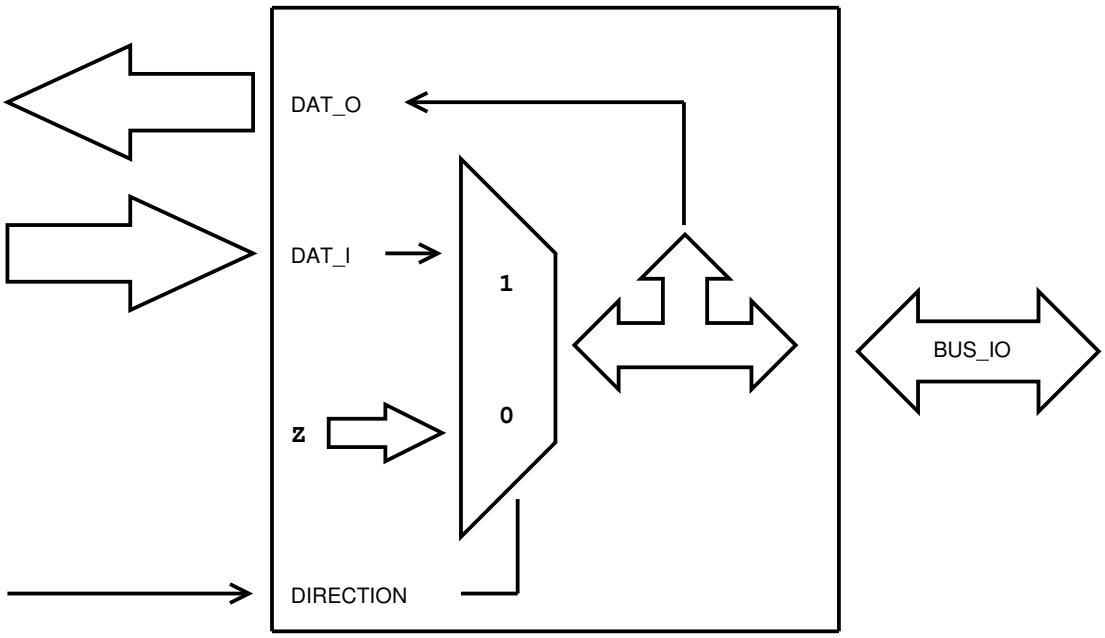


Figure 3.3: The Bus Driver.

The `inout` Data Type

The `inout` data type in VHDL is reserved for bidirectional signals or vectors. In order to make use of such data types one should implement a conditional assignment. This means that according to the state of a third control signal, the signal which is defined as `inout`, is assigned to an `in` or an `out` source or sink respectively[1]. As it may bee seen from figure 3.3 one end of the bus is put to high-Z state when the bus is used in the read direction, otherwise it is driven with logic. The data travelling on the bus, regardless of which direction it has, could be read from the bus. This is much desirable in systems like this with slave configuration being set in an always-listening mode, unless they are requested to send data.

3.3 CPLD Code Elements

Following are some code elements that are implemented inside the CPLD. Most of these appear as a single VHDL entity in a separate file on disk. It has been tried to

[1]Please refer to [18] for more information on the bidirectional data type

encapsulate functionalities in entities for use in future applications.

3.3.1 The Register File

The structure of the registers in CPLD has already been mentioned in section 3.2. Figure 3.4 demonstrates the VHDL entity that contains the register file and provides connectivity to other entities that make use of the registers. In this case, the top level entity instantiates the register file entity.

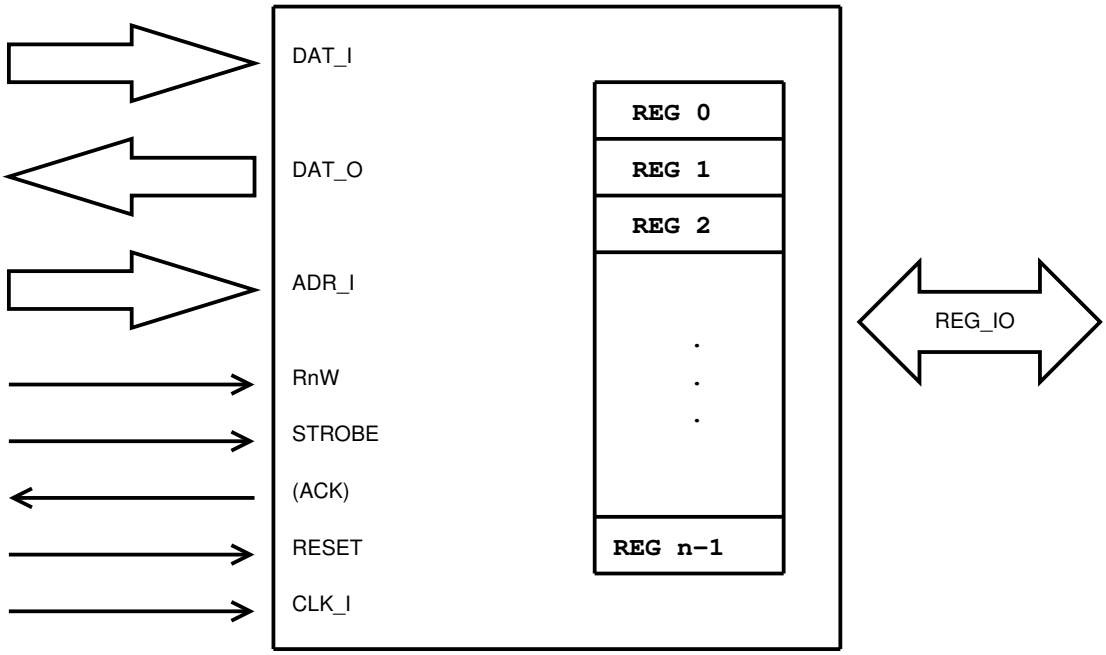


Figure 3.4: Block Diagram of the Register File Entity .

3.3.2 Clock Dividers

Since the sampling clock rate of each ADC or DAC of each channel could be changed upon request from the host system by writing to the respective register, the need for clock division was apparent. The clock divider entity consists of an input, an output and a counter. By setting the division constant, clock divider toggles it's output each time the counter overflows. Possible values are listed in table 3.3.

Value	Description
0	Output Always Set to GND
1	Route the Input directly to the Output
2	Divide Input Clock Freq. by 2
⋮	⋮
n	Divide Input Clock Freq. by n

Table 3.3: Possible Count Constants for the Clock Divider Entity.

3.3.3 Top Level Entity

The CPLD top level entity is shown in block diagrams in figure 3.5. Here the actual connections to the board signals are made. On one side, it has connections to the data and address bus and control signals, and on the other side it associates the relevant registers of the register file entity to ADCs and DACs of each channel.

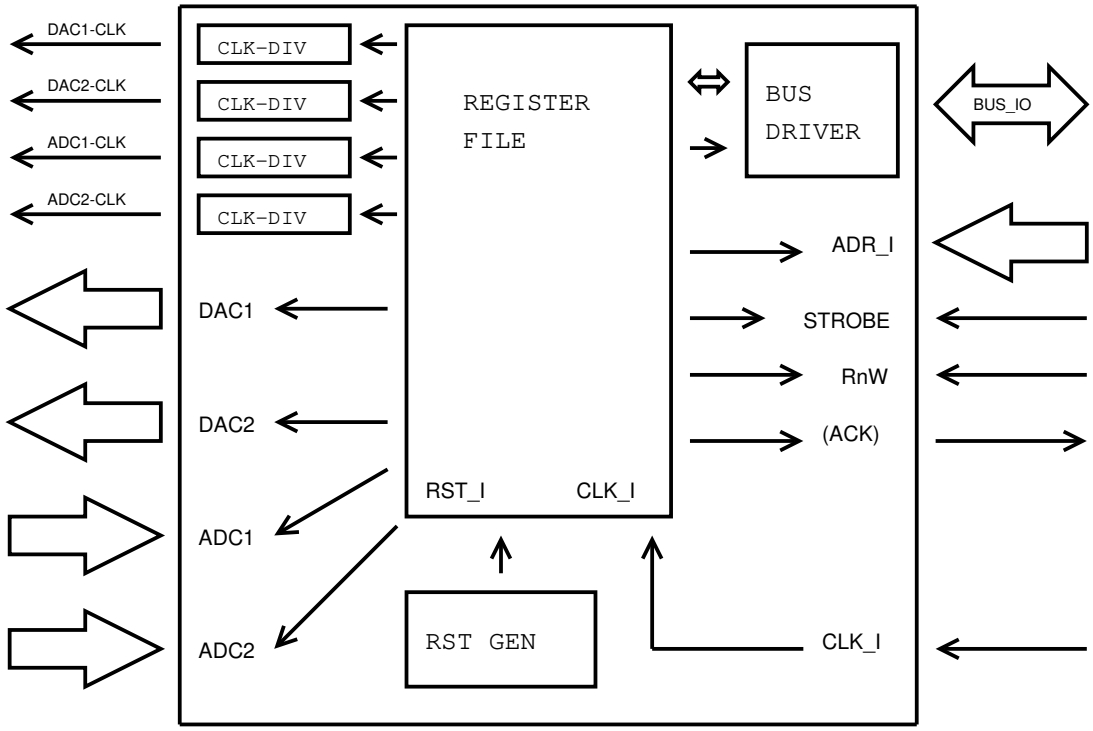


Figure 3.5: Block Diagram of the main Code in CPLD.

Also, the clock dividers, a bus driver and a reset generator are instantiated. It should be noted that the acknowledge signal (ACK) has been reserved for future extensions and is not used for the implemented communication protocol.

3.4 CPLD Test Routines and Simulation

Some routines have been written, to allow evaluation and test of the boards. Here is a list:

3.4.1 CPLD Digital Short

As the name implies, this test implemented a direct connection between ADC and DAC of each channel. Using this routine the signal is digitized by the ADC and then converted back to analogue by the DAC. Clock dividers have been used to feed the ADC with 100MHz. The DACs were fed with a direct connection of the main clock (200MHz).

This routine proved to be very useful for testing the radio frequency characteristics of the board which are presented in chapter 4. A similar routine has been written for the FPGA on the host board, which is thoroughly described in section 3.5.2.

3.4.2 CPLD Sawtooth Generator

DAC channels have been tested directly by counting a variable. After overflow the signal begins counting from -0x2000 up to +0x1FFF again. This produces a sawtooth shape on the output. Note that all numbers are in 2's complement format.

3.4.3 Test Benches

Each entity has been tested using a dedicated test bench. Even the top level entity has been instantiated as a component and has been routed to the so called test bench signals which are needed for the simulation. As an example a simulation clock, a simulation reset signal and data communication signals are often needed. Then the behaviour of the entity could be simulated and the signal transitions and values of vectors could be seen in the simulation window.

3.5 FPGA Code Elements

Code has been written to test the communication protocol on the FIB side i.e. on the FPGA (see figure 3.6). Later this code could be adapted for use as a template for future FIB applications. This is obvious since the FPGA board acts as a central data highway and the link to the CPLD board is only one of it's many interfaces.

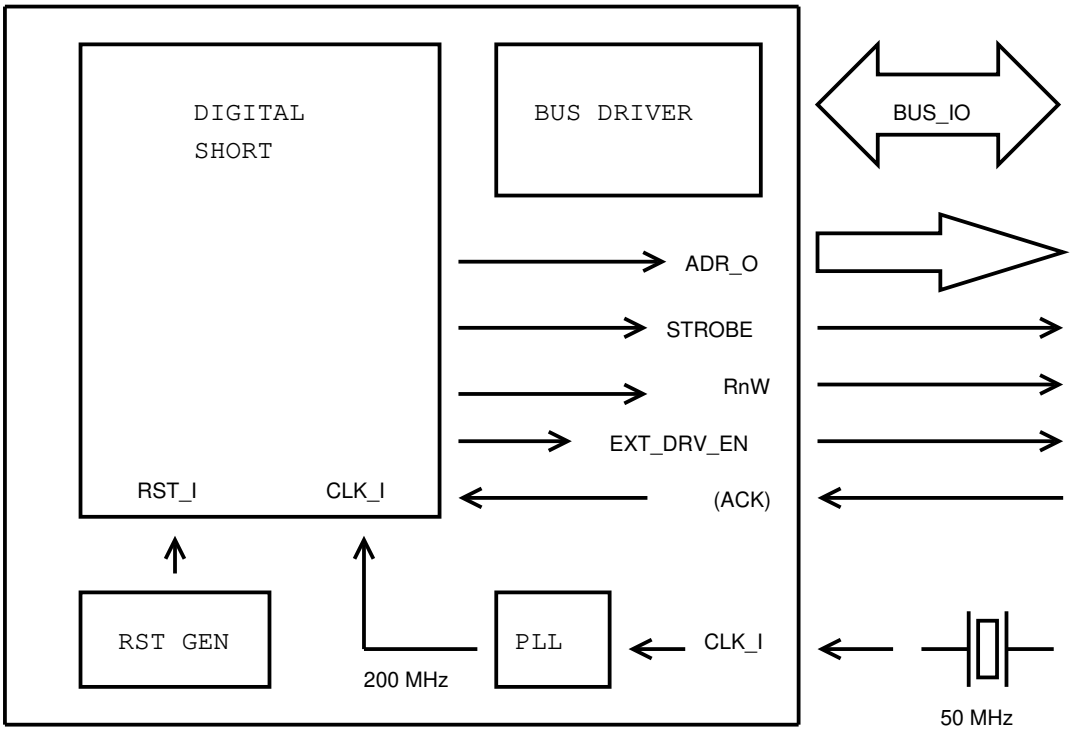


Figure 3.6: Block Diagram of the main Code in FPGA for test purposes.

A similar reset generation scheme has been used for the FPGA (see section 3.2.2).

3.5.1 Internal Phase Locked Loop

The used FPGA on the FIB board is of type EP1C6Q240C8 form the Cyclone family of ALTERA's FPGAs. It features two internal phased lock loops (PLL) that could be programmed to multiply the frequency of the clock source. The activation is usually done using part dependant libraries. Generation of such a PLL entity is made simple using ALTERA's integrated development environment (IDE).

3.5.2 FPGA Digital Short

This routine is similar to the digital short circuit routine implemented in the CPLD as described in section 3.4.1. The difference is that now the CPLD is programmed to perform the full communication protocol, and the FPGA reads data and writes it back to the CPLD board. This routine proved the successful bidirectional operation

of the communication link. The block diagram of the digital short entity in the FPGA is depicted in figure 3.7.

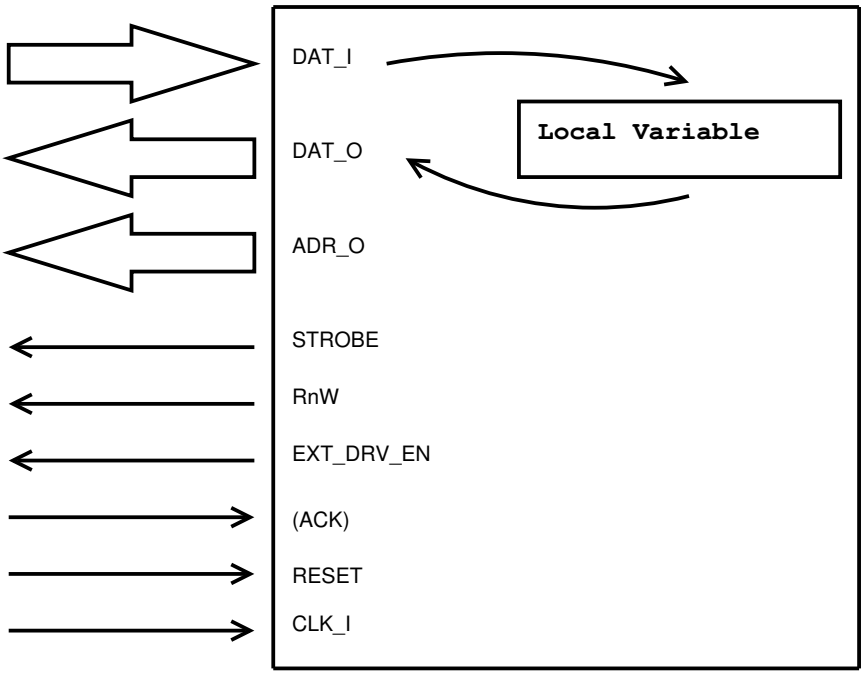


Figure 3.7: Block Diagram of the Digital Short Circuit Entity.

Figure 3.8 shows the state diagram of a complete cycle of reading from and writing to the CPLD board. For a better view, signal transition names have been omitted from the diagram.

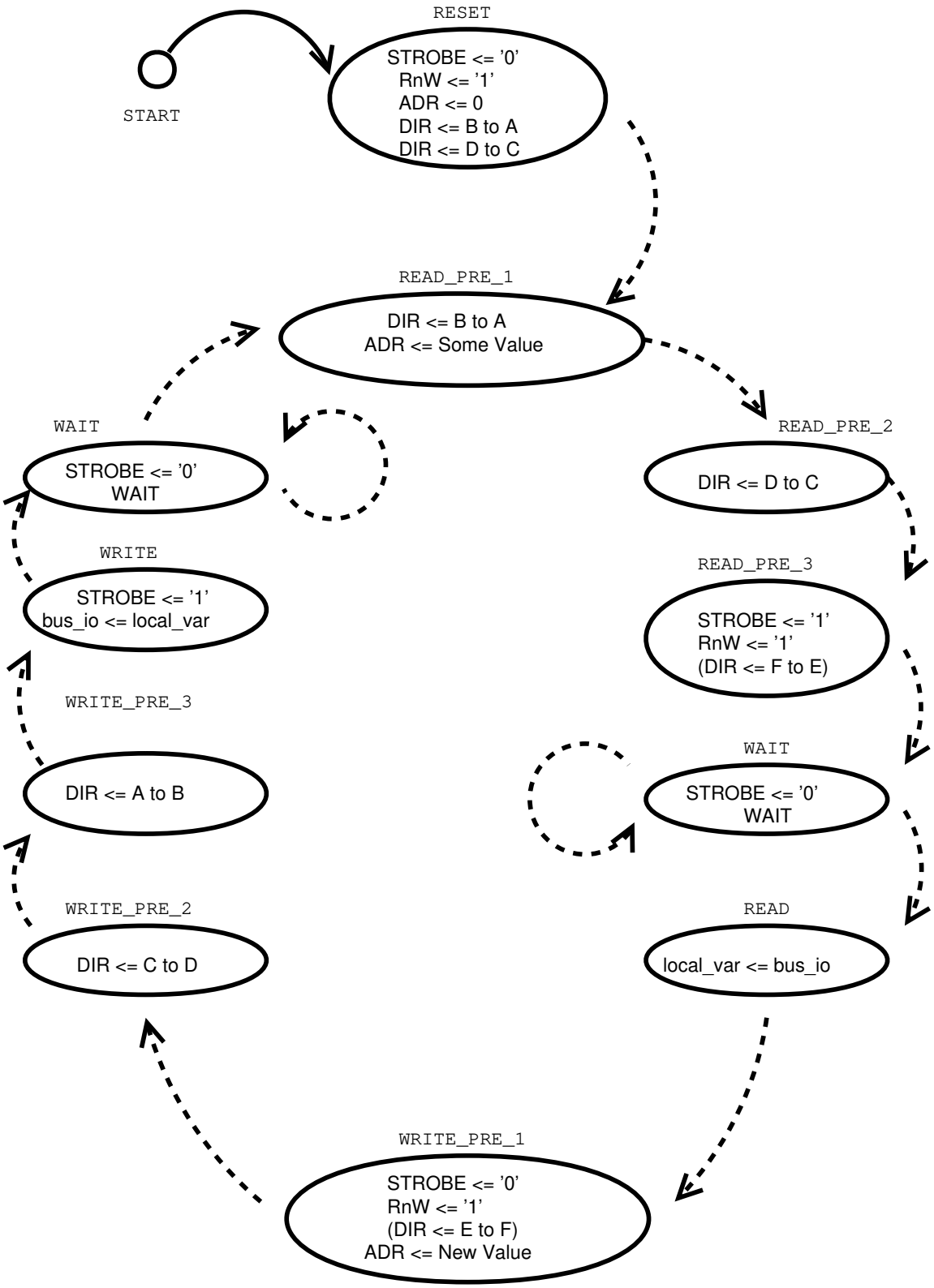


Figure 3.8: State Diagram of the Digital Short Circuit Entity.

The reset state sets the CPLD board to listening mode. It should be remarked that as in figure 3.9 the second bus driver is a real IC residing as a component on the FPGA board. In the forthcoming states, the direction of the three bus drivers is changed sequentially before the data is read in the READ state.

Change of directions should be made in that sequence since otherwise two outputs will be shorted together and the circuit may get warm. By the time of writing data, the same procedure must be followed, only backwards.

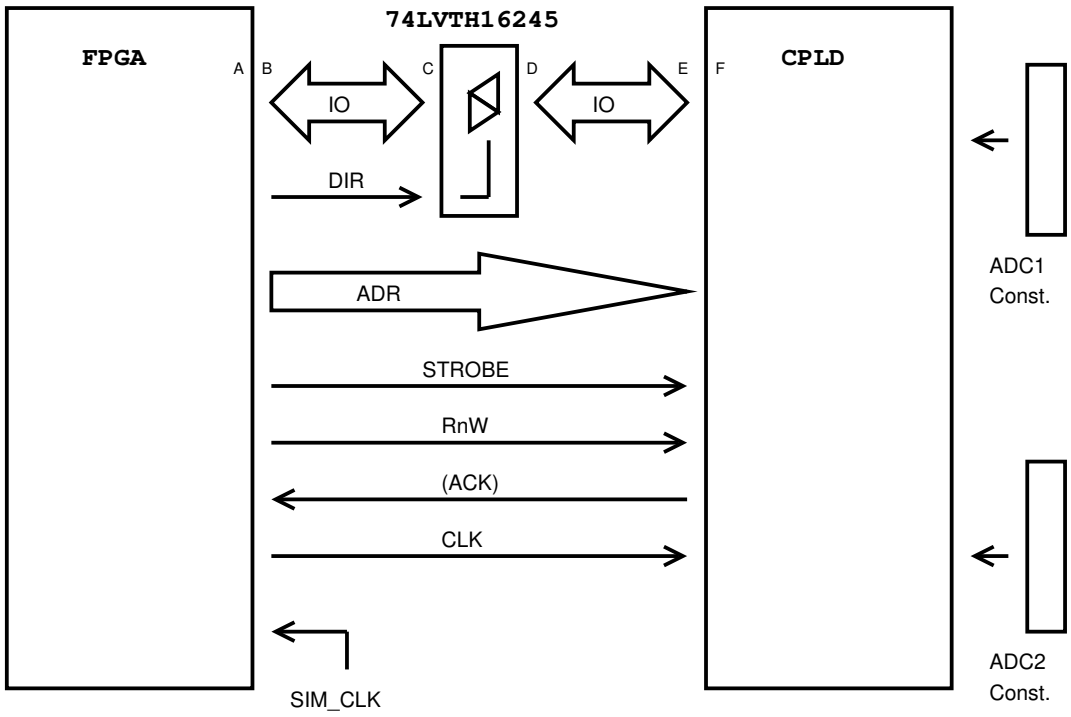


Figure 3.9: Overview of Overall System Components.

The external driver is of type 74LVTH16245 from TEXAS Instruments. The wait states are needed for compensating the delays of this IC driver. In the actual implementation, only one wait state exists. The calling state stores the name of the next state in a variable before calling the WAIT state. The state machine then jumps to that specified sate after the waiting time is over.

Using a 200MHz clock, the maximum data transfer rate possible on the 16-bit wide bus is 3.2 Gigabits per second. In practical applications, this data rate is limited by delays caused by read and write operations i.e. changing address and control signals.

3.5.3 FPGA Sawtooth Generator

Again similar to the sawtooth generator which had been written in the CPLD before, here the same routine has been initiated from within the FGPA, whereas the CPLD was programmed with the full protocol implementation. The test was successful.

3.6 Simulation Test bench

One of the most difficult stages of programming has been the overall simulation of FIB and FAB, i.e. FPGA and CPLD codes inside one single test bench entity. This test bench consisted of the two top level entities one of them being the full protocol implementation of the CPLD and the other the FPGA digital short top level entity. A third entity was also implemented in this test bench. It was the simulation model for the external 74LVTH16245 driver where it's timing behaviour was simulated. This made it possible to set delay parameters or operate the driver as ideal.

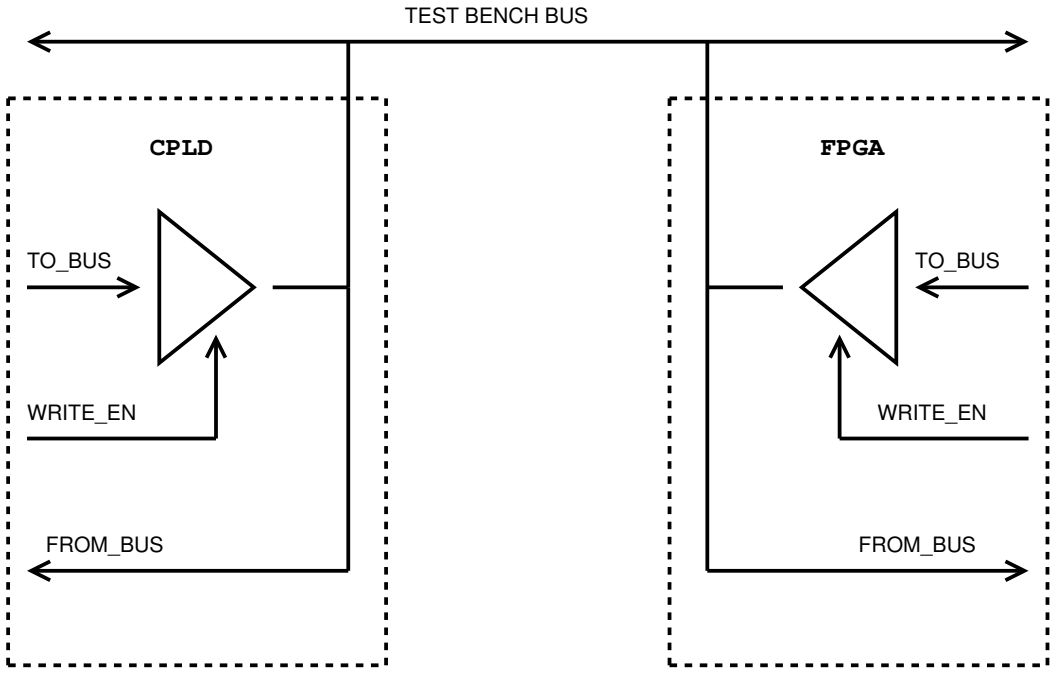


Figure 3.10: IO Bus in the Simulation Test Bench.

The clue for success was assigning two simulation buses to the two sides of the external driver, one being the connection to the FPGA and the other the connection

to the CPLD. Since the entities have used the bi-directional data type `inout` at their interfaces, the simulation buses had to have the same switching characteristic on all their four connection points to the FPGA, CPLD and both of the external driver's ports. After understanding this point, the author learned even more about the structure of the VHDL language. Figure 3.11 shows a screen-shot of the overall simulation.

Without this overall simulation, success of this thesis could not have been broken through; an important argument regarding the importance of the simulation in modern digital designs.

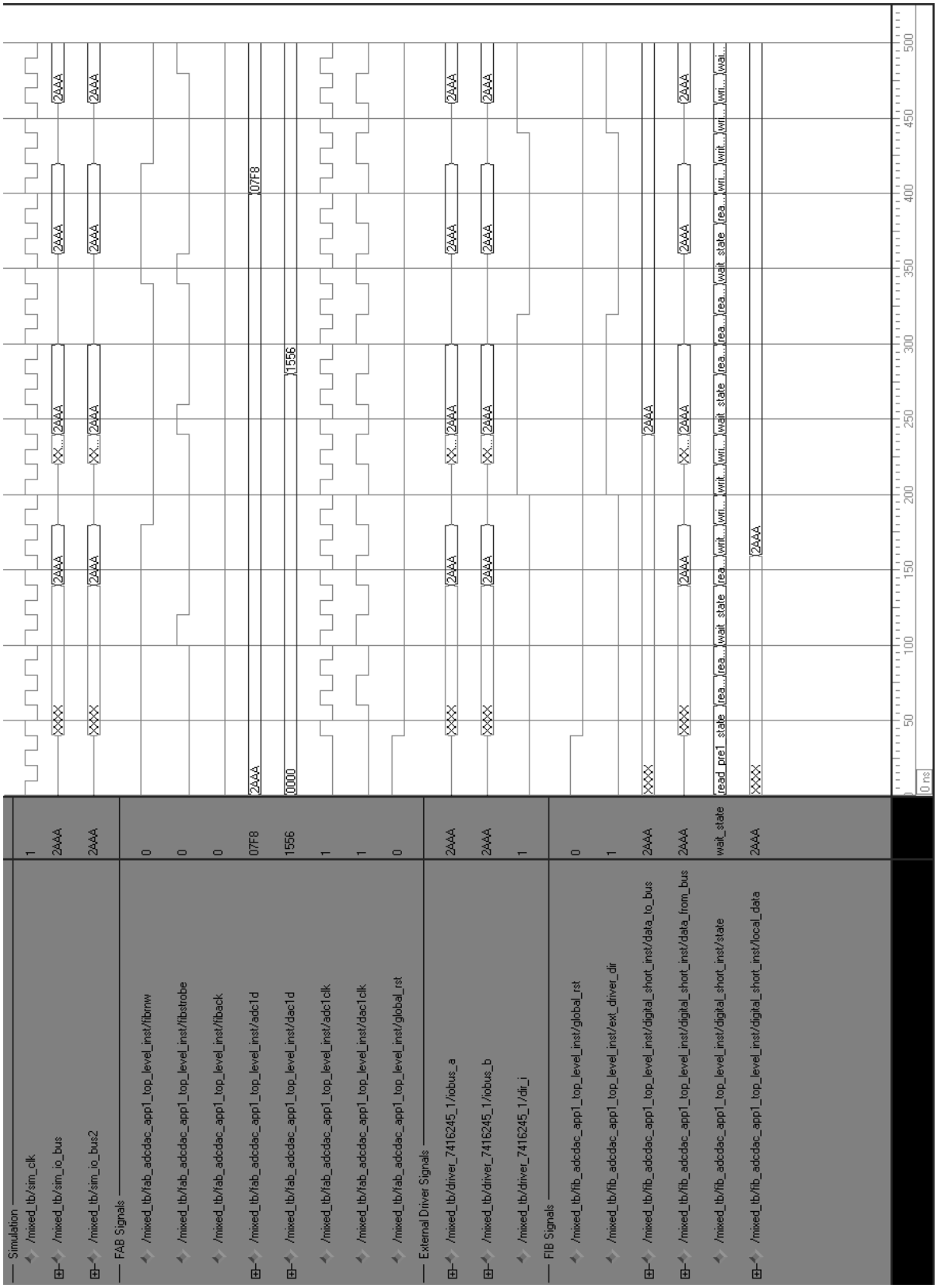


Figure 3.11: Overall System Simulation up to 500ns in Simulator.

4 Measurements and Outcomes

4.1 Introduction

After the production, measurements has to carried out to define the range of working parameters and characteristics of the boards. For the test purposes, the digital short program inside the CPLD has been used, since this together with a program on FPGA which provided a 200MHz clock using its internal PLL, made it possible to drive the components on the board at their highest speed. This routine causes the board to sample the signal applied to its ADC input with 100MHz and return this sampled data to the respective DAC channel, where it is converted back to an analogue signal. Such configuration allowed the author to specify the radio frequency characteristics of the design.

4.2 Thermal Issues

That electronic components get warm, is nothing new. But a sound operation is only guaranteed when the components do not get warmer as is specified in their absolute maximum ratings section of their data sheet. Yet it is better not get near these values either and operate the component at a much less temperature.

The printed circuit board itself acts as a heat sink. This fact is even used intentionally in some IC packages where a large ground pad exists at the bottom side.

Figure 4.1 shows the heat flow diagram of the LM7805 fixed voltage regulator used on the board (IC2 on the schematics) and which is the warmest of all components. It has a TO-220 housing. A maximum junction temperature of 150°C is specified for this package. Also, according to the data sheet [7], the thermal resistance of junction to case of the TO-220 package is typically 4°C/W junction to case and 50°C/W case to ambient. Since the regulator is mounted on the PC-Board, the value of case to ambient is not valid any more. This value was measured using a digital thermometer.

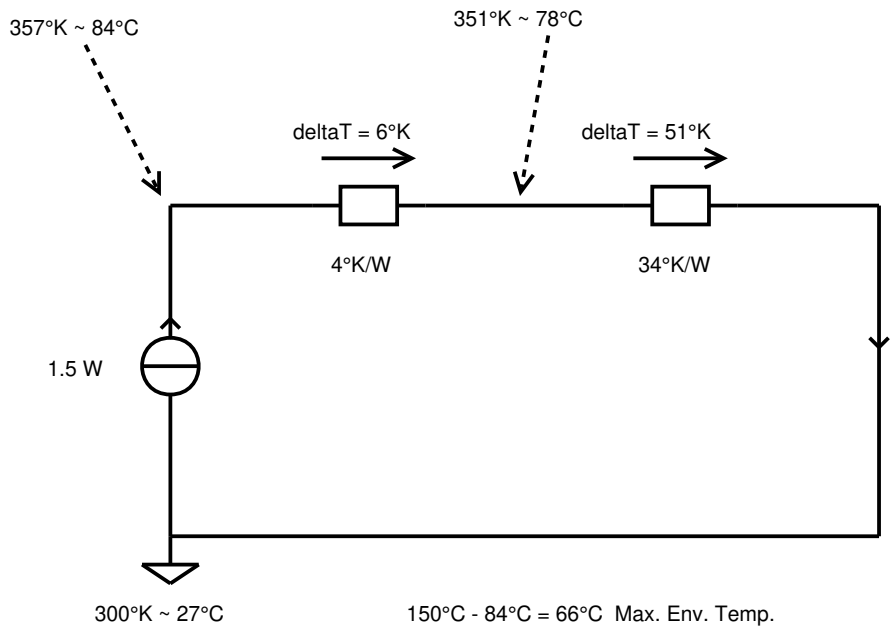


Figure 4.1: Heat Flux Diagram.

On the summer day where the measurement was carried out, the room temperature was 27°C. The digital thermometer showed a maximum value of 78°C on the IC. So as shown in the figure 4.1, the case to ambient temperature resistance when the IC was mounted was therefore 34°C/W. Calculating the results at the same operating current, we have approximately 84°C for the junction temperature which is much less than the allowed 150°C. Using this calculation the maximum environment temperature can also be defined as 66°C (see figure 4.1).

Note that the operating current is important. In this example, the LM7805 is supplied from a +15V source, at 150mA it dissipates $(15V - 5V) \times 0.15A = 1.5W$ which is also shown in figure 4.1. So with a junction to case thermal resistance of 4°C/W, the change in temperature could be calculated as follows:

$$\Delta T = 1.5W \times 4°\text{C/W} = 6°\text{C}$$

The same calculation is done for other voltage regulators.

With:

$T_{J,Max}$	Maximum Allowed Junction Temperature
T_{JC}	Thermal Resistance of Junction to Case
T_C	Temperature of Case

$T_{E,Max}$	Maximum Allowed Environment Temperature
N/A	Not available or unknown from Data Sheets

Similar measurements are summarized in table 4.2.

Designator	Part	$T_{J,Max}$	T_{JC}	T_C	$T_{E,Max}$
IC2	LM7805	150°C	4°C/W	78°C	66°C
IC2	LM7905	150°C	4°C/W	67°C	77°C
IC27	LM1587A	150°C	0.24°C/W	64.9°C	85°C
IC28	LM1587A	150°C	0.24°C/W	55.4°C	94.5°C
IC13	AD9744	150°C	N/A	56.2°C	≈ 80°C
IC15	LTC2255	N/A	N/A	55°C	≈ 80°C
IC1	EPM1270T144	N/A	10.5°C/W	57.4°C	≈ 88°C

Table 4.2: Temperature Values of the Components.

4.3 Component Value Adjustment

The adjustment of the component values that control the signal amplitude throughout the signal path is application dependant. This includes ADC and DAC output values. At the moment these values are set to a default of approximately $1V_{PP}$ for the full output span. More precise values will be set later after each application is studied further.

4.4 Oscillography

As a test of the DAC outputs of the DC-Coupled board, the sawtooth generator routine has been loaded into the CPLD. This implements a 200MHz 14-bit counter. Figure 4.2 shows the sawtooth signal on the oscilloscope with 200mV and $10\mu S$ per division.

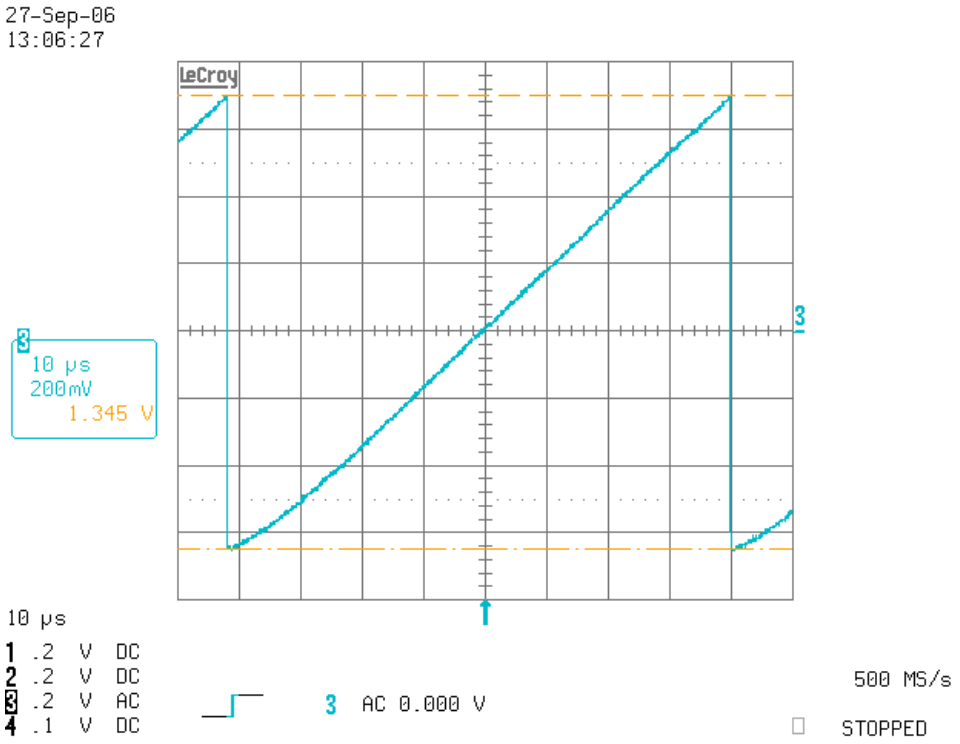


Figure 4.2: DAC1 Output of the DC-Coupled Board as a Sawtooth Generator.

As stated in the introduction section, the digital short routine has been loaded onto the CPLD to test different channels on the DC-Coupled and the AC-Coupled boards.

A single pulse of a 6MHz sine wave has been applied to the ADC connector. As depicted in figure 4.3, an overall delay of about 135nS exists for the signal path. In this diagram the signal from the generator is drawn in red, after passing all components once in the signal path, it is available again as an analogue signal on the DAC connector which is drawn in blue in this diagram on the oscilloscope with 200mV and 50nS per division.

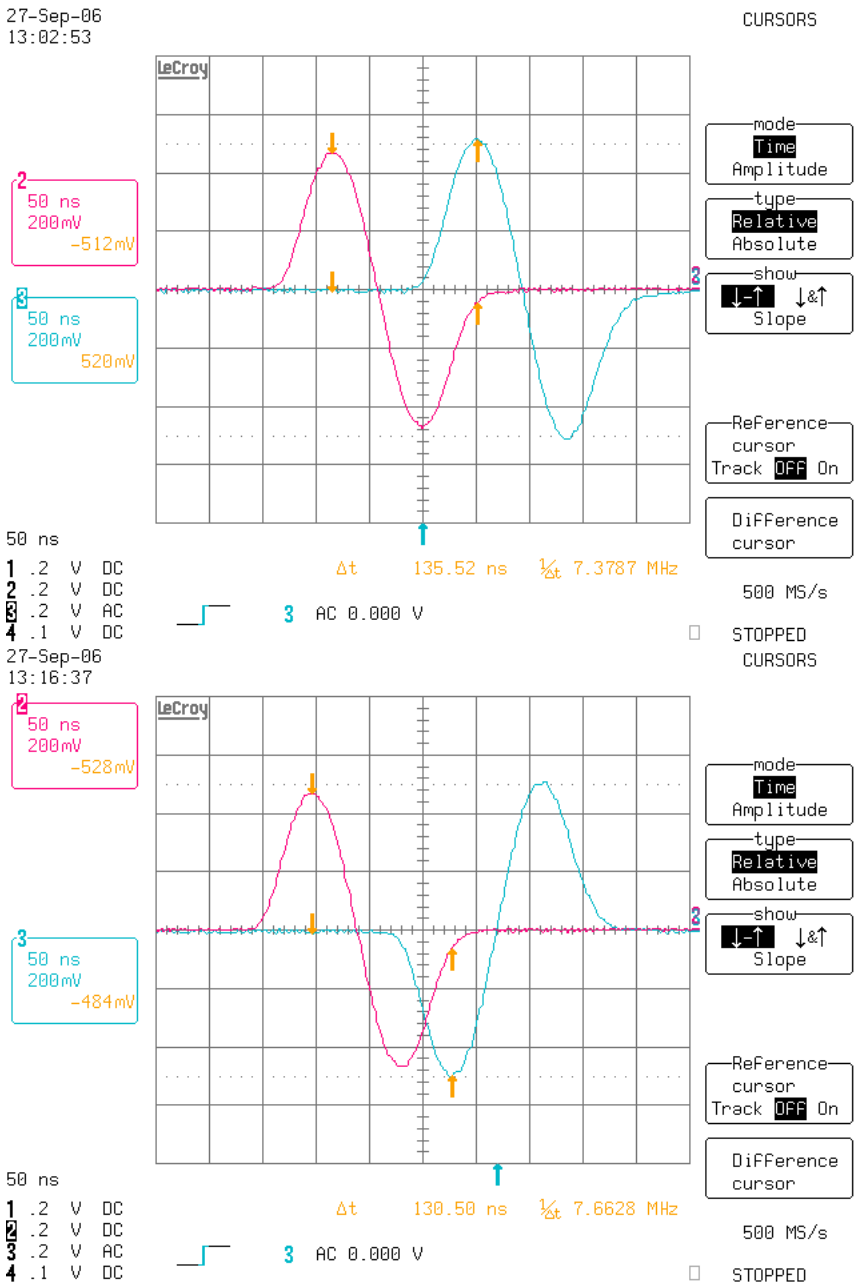


Figure 4.3: DAC1 Output to 0dBm 6MHz Sine Wave Pulse Input at ADC1 of the DC-Coupled Board (Top) and the AC-Coupled Board (Bottom).

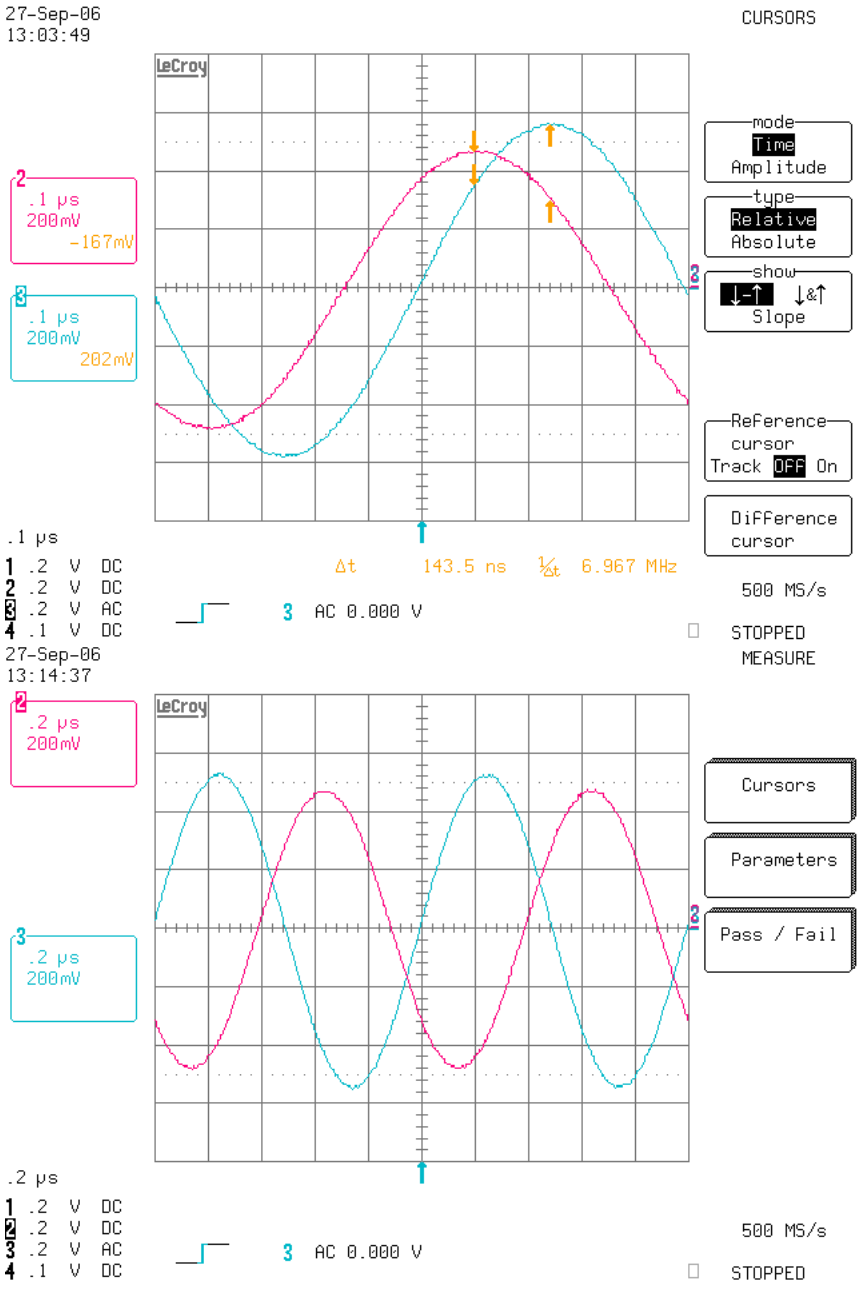


Figure 4.4: DAC1 Output to 0dBm 1MHz Sine Wave Input at ADC1 of the DC-Coupled Board (Top) and the AC-Coupled Board (Bottom).

The same measurement has been carried out using a continuous 1MHz sine wave instead of a single pulse. This is shown in figure 4.4. The oscilloscope is adjusted to 200mV and 100nS per division.

Figure 4.5 shows the same measurement with a 10MHz sine wave as a comparison, with 200mV and 20μS per division.

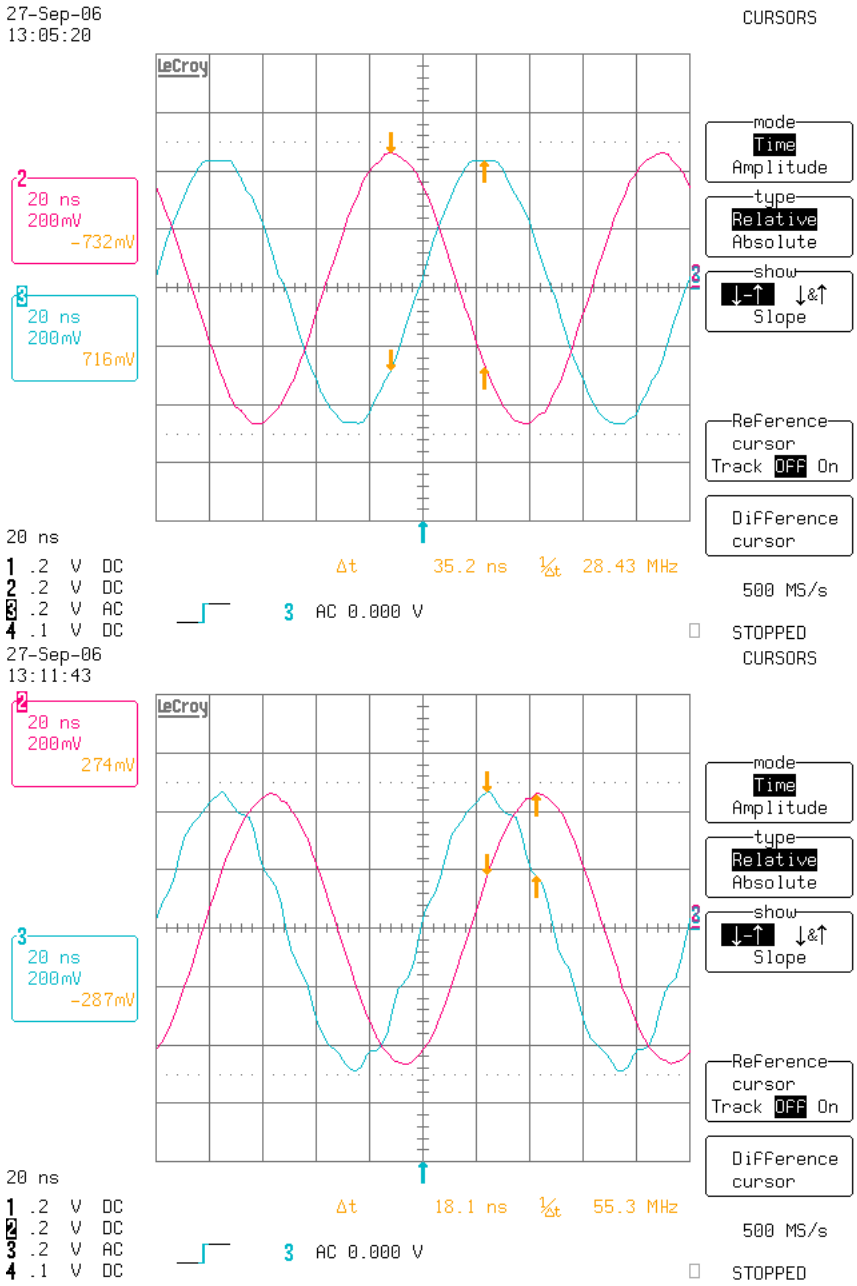


Figure 4.5: DAC1 Output to 0dBm 10MHz Sine Wave Input at ADC1 of the DC-Coupled Board (Top), the AC-Coupled Board (Bottom).

Other channels of other boards have also been tested. They almost had the same quality.

4.5 Spectrography

Spectral characteristics of the boards have been tested using a spectrum analyser. All signals form the generator have been set to 0dBm.

It was important to measure the output noise floor, since this it plays a significant role in the overall system quality later in the digital control application. The output of the DACs have been measured without any signal applied to their respective ADC channel.

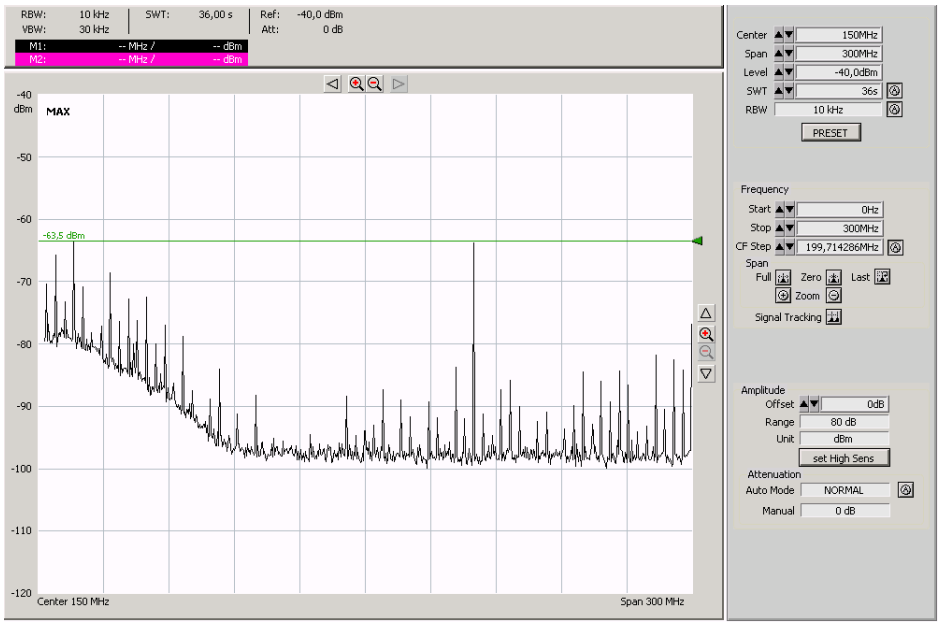


Figure 4.6: DAC1 Output without Signal at ADC1 of the AC-Coupled Board, Indicating the Output Noise Floor.

As it may bee seen from figure 4.6, many signal and noise components exist on the output, specially that of the 200MHz clock, but all noise peaks reside under approximately -63.5dB which is a very satisfactory result.

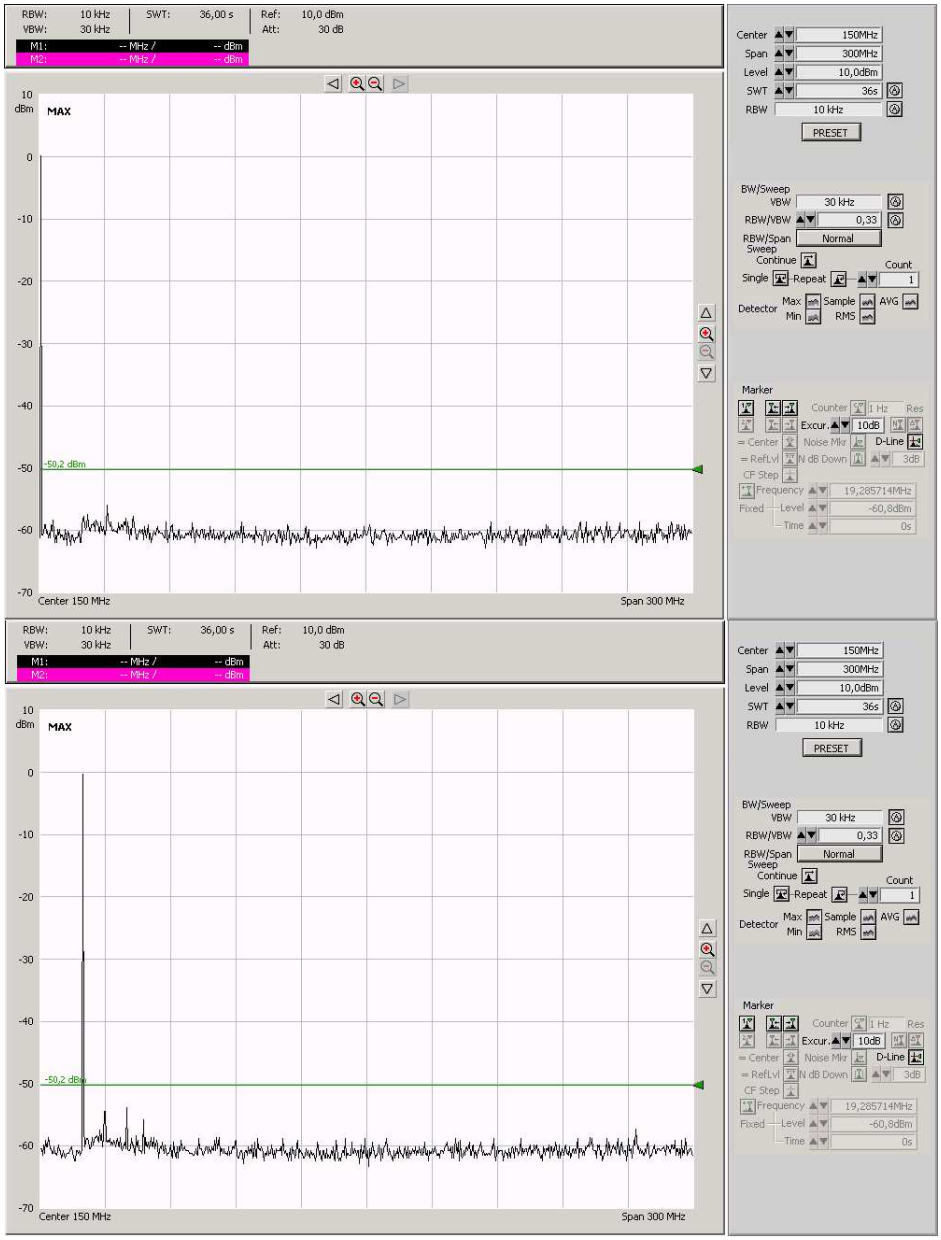


Figure 4.7: Signal Generator Output Alone at 1MHz (Top) and 20MHz (Bottom) 0dBm Sine Wave.

In order for the measurements to be judged correctly, the output of the signal generator has been measured alone so that the effect of the board is easier to figure out. Figure 4.7 shows this.

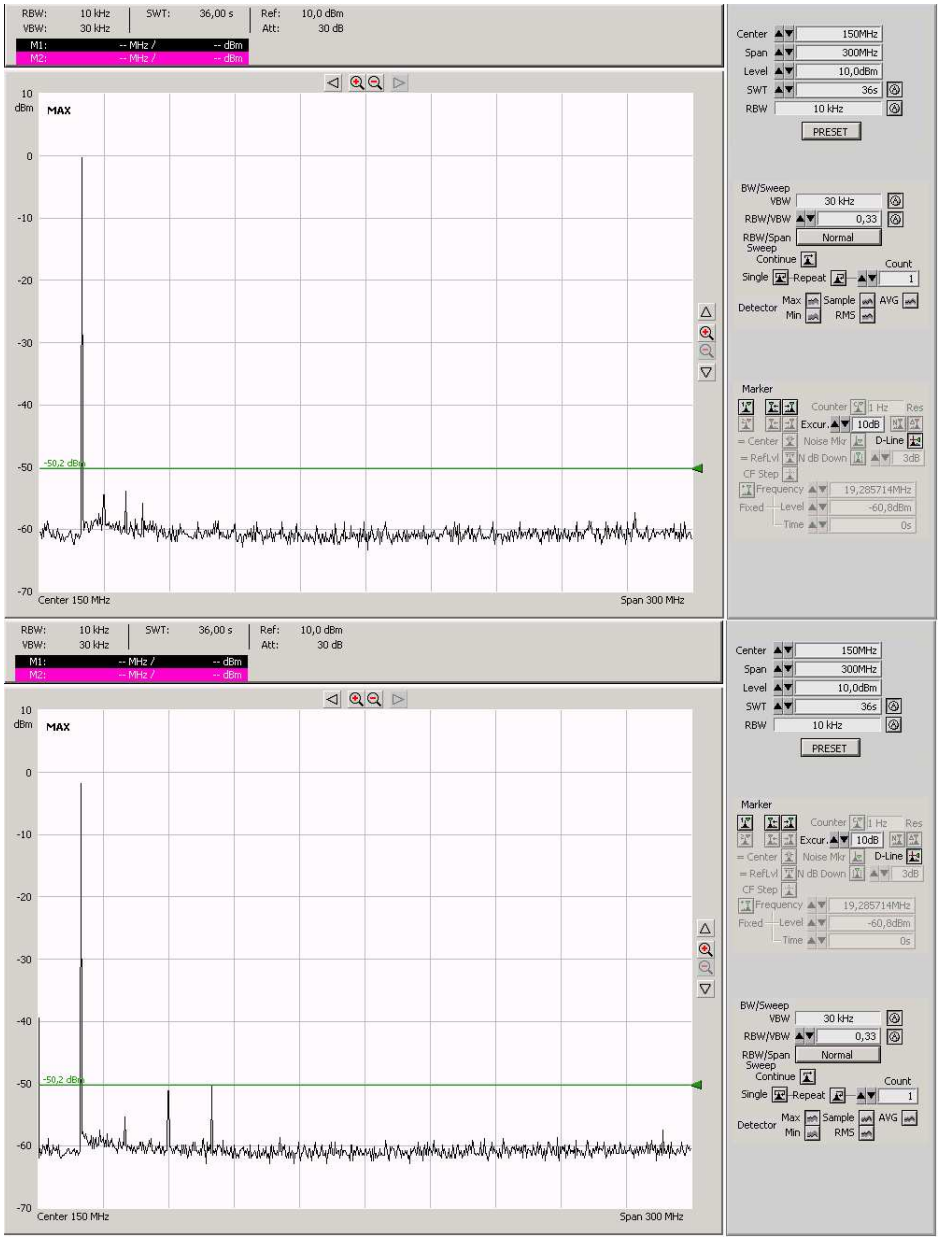


Figure 4.8: DAC1 Output to 0dBm 1MHz (Top) and 20MHz (Bottom) Sine Wave Input at ADC1 of the AC-Coupled Board.

Comparing this to figure 4.8 shows that at 20MHz, some noise is present, more remarkable is the generator noise which is fed through the output. But all the levels are still below -50.2dB. In more practical signal frequencies, like the figure on top, the noise level could almost be ignored.

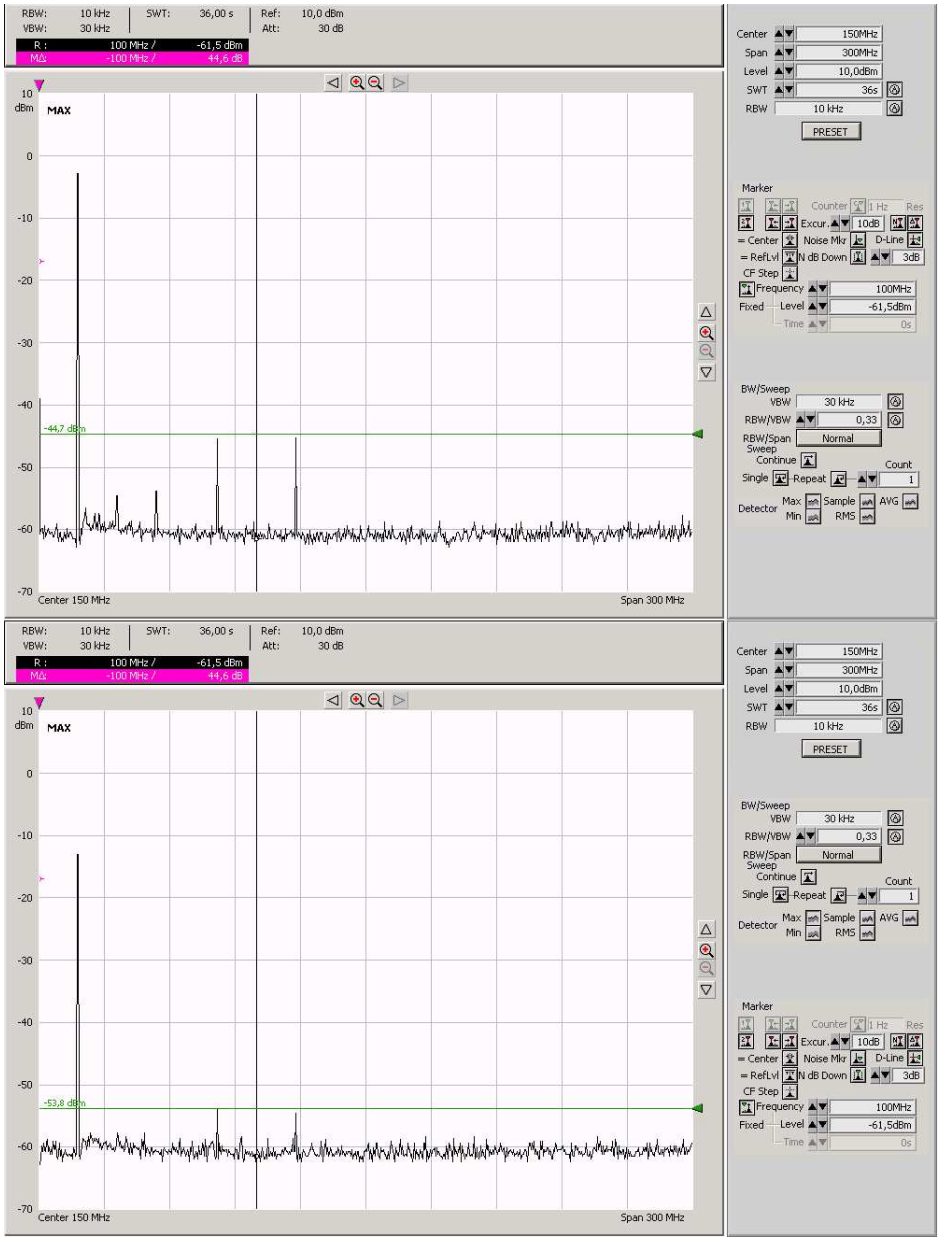


Figure 4.9: MON2 Output to 0dBm (Top) and -10dBm (Bottom) 18MHz Sine Wave Input at ADC2 of the DC-Coupled Board.

As an example of the quality of the monitor output of the board, MON connector of the second channel on the DC-coupled board has been connected to the spectrum analyser. The 0dBm signal at 18MHz could be seen on 18, 36, 54, 82 and 118 MHz as small peaks. A marker has been set on 100MHz for ease of measurement. For

a signal with a power level of 0dBm the peaks lie under 44.7dB. For more practical signal power like -10dB, the peaks reside below -53.8dB. This is illustrated in figure 4.9. Other channels of other boards behave almost the same.

4.6 Signal analysis using a Vector Network Analyser

The vector network analyser (VNA) has been used to perform a transmission test on the channels. The VNA has been set to sweep slower than the delay of the circuit, so that the measurement is carried out correctly. The following graphs show magnitudes versus frequency.

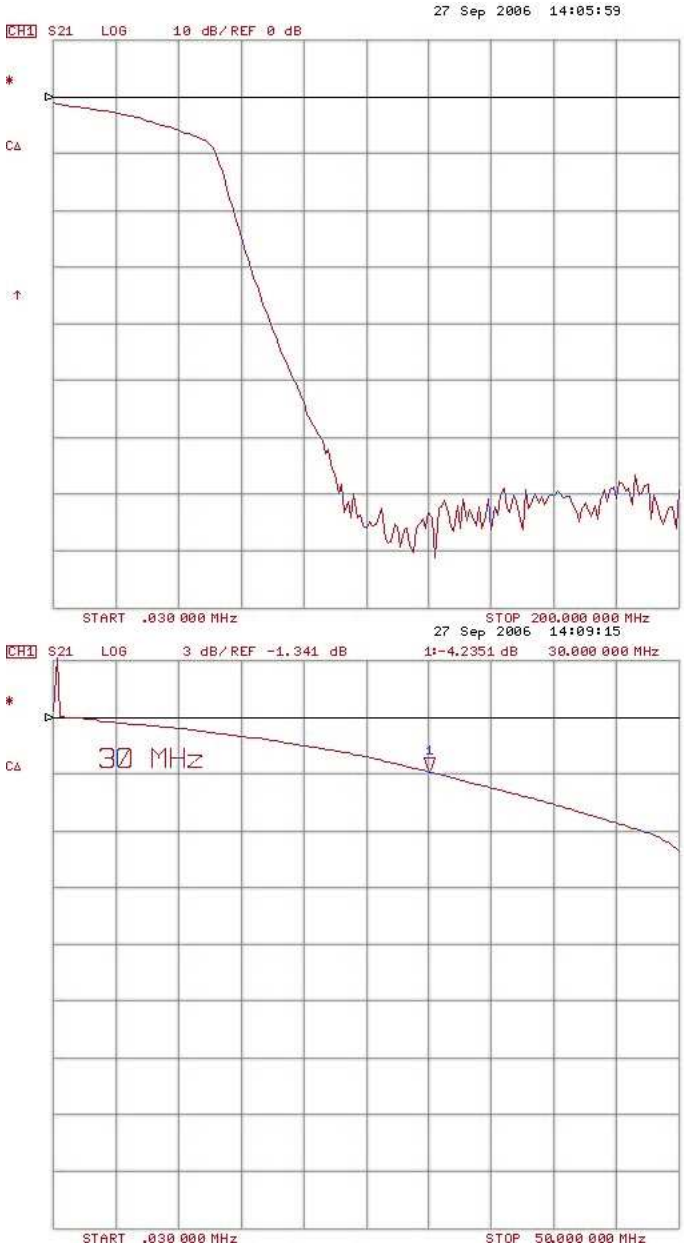


Figure 4.10: MON1 Output to ADC1 Input up to 200MHz, 10dB/Div. (Top) and up to 50 MHz, 3dB/Div. (Bottom) of the DC-Coupled Board.

Figure 4.10 shows the MON output of the first channel on the DC-coupled board. The whole channel shows low-pass characteristics. In the range of the signals of interest, only about 1dB attenuation is visible.

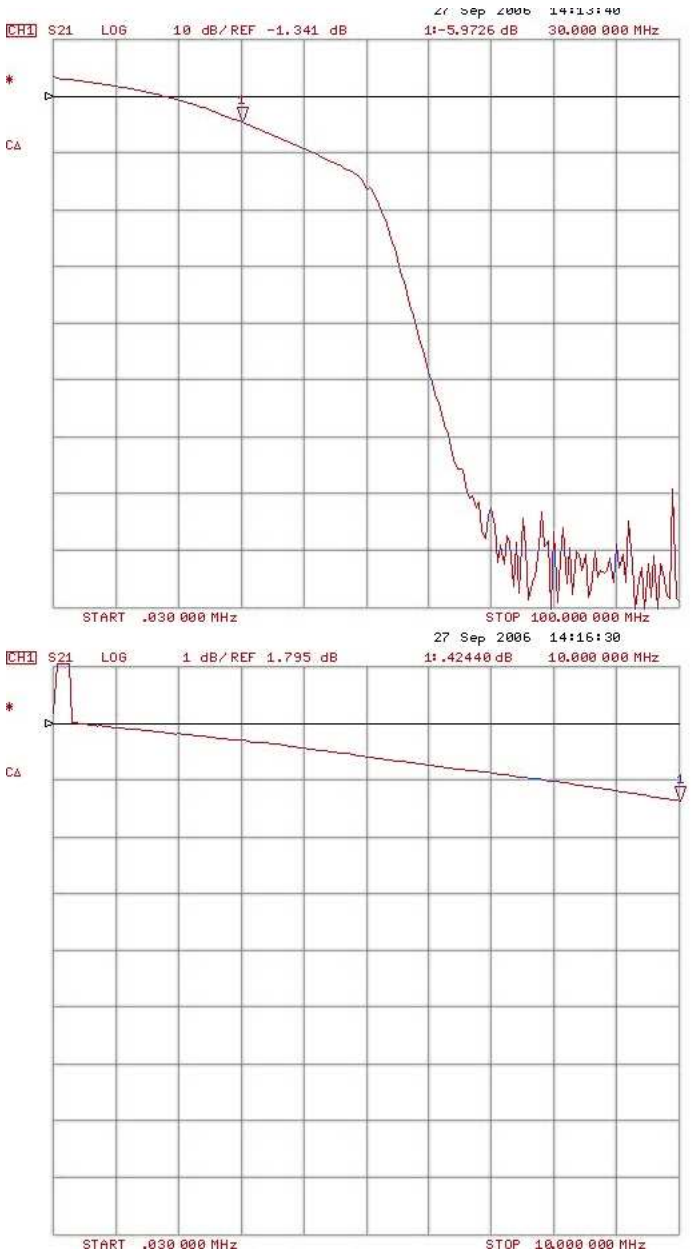


Figure 4.11: DAC1 Output to ADC1 Input up to 100MHz, 10dB/Div. (Top) and up to 10 MHz, 3dB/Div. (Bottom) of the DC-Coupled Board.

Figure 4.11 shows a repeated experiment, this time using the DAC output of the channel.

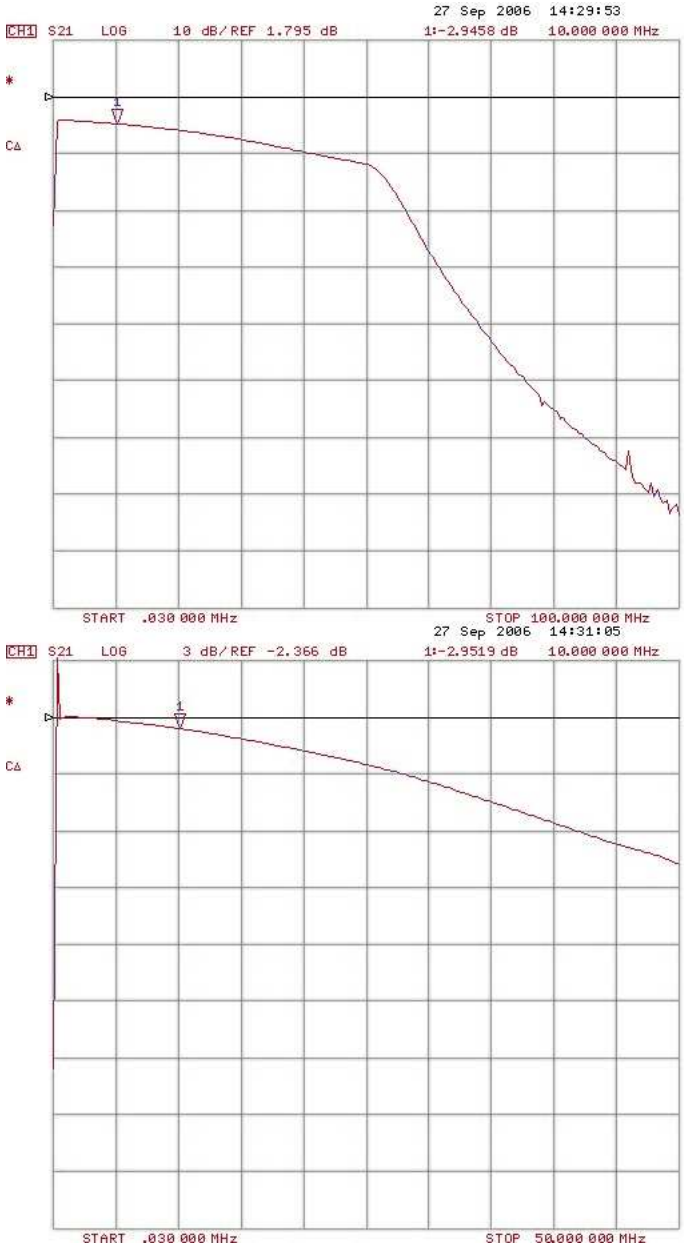


Figure 4.12: MON1 Output to ADC1 Input up to 100MHz, 10dB/Div. (Top) and up to 50 MHz, 3dB/Div. (Bottom) of the AC-Coupled Board.

The same experiment is also done with the MON output of the first channel on the AC-coupled board which is shown in figure 4.12.

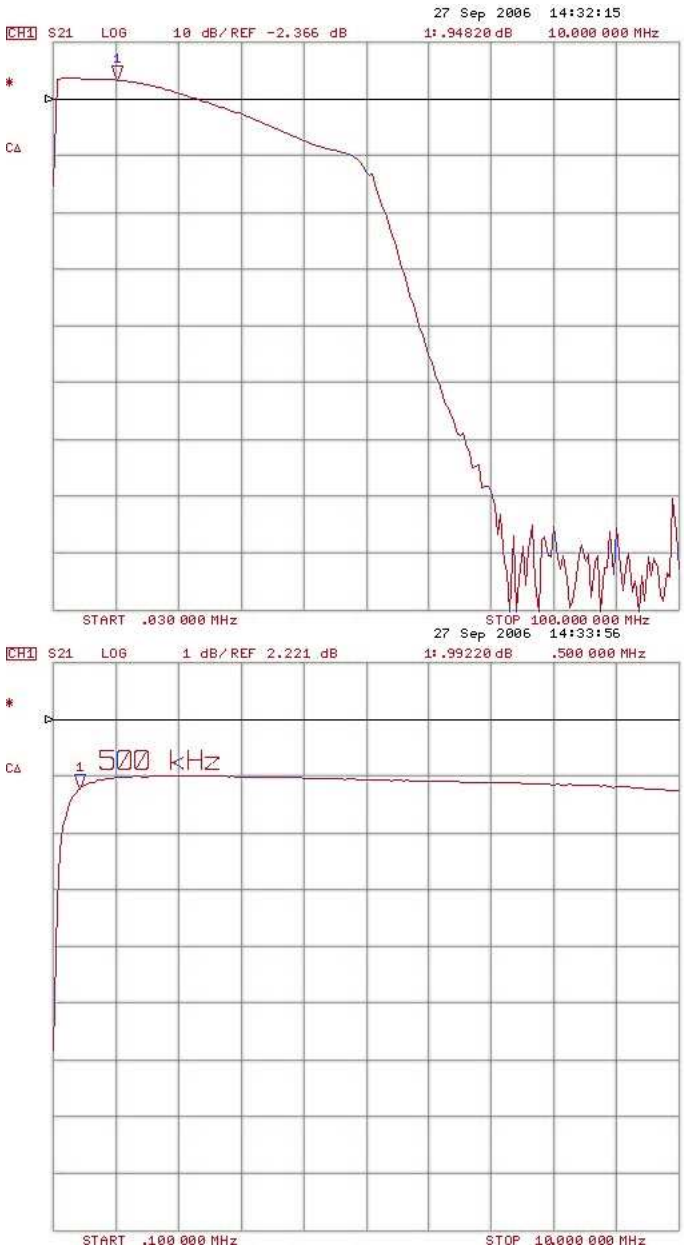


Figure 4.13: DAC1 Output to ADC1 Input up to 100MHz, 10dB/Div. (Top) and up to 10 MHz, 3dB/Div. (Bottom) of the AC-Coupled Board.

Since the AC-coupled board has almost band-pass characteristics, the last experiment defined a lower corner frequency for the channel. Figure 4.13 shows that signals above approximately 500kHz could be used with this board.

4.7 Technical Specifications of the Design

After the measurements, the following features could be summarized in table 4.3.

Parameter	Condition	Min.	Typ.	Max.	Unit
τ_{CH}	6MHz Single Pulse, DC-Coupled		135		nS
τ_{CH}	6MHz Single Pulse, AC-Coupled		130		nS
η_{DAC}	DC-Coupled		-57.1		dBm
η_{DAC}	AC-Coupled		-63.5		dBm
ΔG_{DAC}	$0 \rightarrow$ 10MHz, DC-Coupled		≈ 1.5		dB
ΔG_{DAC}	$0 \rightarrow$ 10MHz, AC-Coupled		≈ 0.5		dB
ΔG_{MON}	$0 \rightarrow$ 10MHz, DC-Coupled		≈ 1		dB
ΔG_{MON}	$0 \rightarrow$ 10MHz, AC-Coupled		≈ 1		dB
$SFDR_{DAC}$	20MHz Input Sine Signal		50		dB
$SFDR_{MON}$	18MHz Input Sine Signal		45		dB
f_C	AC-Coupled	≈ 0.5		≈ 25	MHz
$I_{S,+15V}$	DAC at 200MHz, ADC at 100MHz	140	150	160	mA
$I_{S,-15V}$	DAC at 200MHz, ADC at 100MHz	90	100	110	mA
$I_{S,+5V}$	DAC at 200MHz, ADC at 100MHz	505	510	515	mA
θ_E				65	°C

Table 4.3: Technical Specifications of the Design.

With:

τ_{CH}	End to End Channel Delay: ADC $\rightarrow$ DAC
η_{DAC}	DAC Output Noise Floor
ΔG_{DAC}	Gain Variation: ADC $\rightarrow$ DAC
ΔG_{MON}	Gain Variation: ADC $\rightarrow$ MON
$SFDR_{DAC}$	DAC Output Spurious Free Dynamic Range
$SFDR_{MON}$	MON Output Spurious Free Dynamic Range
f_C	Corner Frequency of Band Pass Characteristics

I_S	Supply Current
θ_E	Allowed Environment Temperature

5 Conclusion

In this thesis, the author tried to cover the most important steps towards designing a radio frequency high-speed data conversion interface for use in digital control systems. During this work, the author was introduced to numerous new concepts and methods in high-speed circuit design.

The existing ADC/DAC board was tested using some routines in VHDL. The author became familiar with the design environment and gained programming skills.

The libraries needed for a full redesign of the board were generated and checked. Components' data sheets have been studied thoroughly. The Schematics of the new design have then been drawn iteratively to ensure best element configuration for a well-thought circuit.

The Layout of the board has then been arranged according to the plans for grounding and power supply schemes. Different layers have then been assigned to power signals and sliced to form different power regions. The elements have been placed on their final positions and the board has then been routed manually.

The author wrote some testing routines for the board itself and the communication link to the host system. Also a first fully functioning version of the CPLD-side code for the communication protocol was implemented. Using these routines the parameters of the board could be measured. During these tests the characteristics of the board have been figured out.

Following improvements might be needed:

- Design Aspects
  - Placing the connectors for the AGC application on ADC inputs.
  - Preparing the board for series production.
- Mechanical Aspects
  - Increasing annular rings for some footprints.
  - Better adjustment of the drill sizes of some footprints.
  - More distance between some polygons.

The author will take part in the future applications of this design in a continued cooperation with GSI.

Bibliography

[1] *AN-280, Mixed Signal Circuit Techniques.* Analog Devices Application Notes, One Technology Way, P.O Box 9106, Norwood, MA 02062-9106.

[2] *AN-282, Fundamentals of Sampled Data Systems.* Analog Devices Application Notes, One Technology Way, P.O Box 9106, Norwood, MA 02062-9106.

[3] *AN-75, High-Speed Board Designs.* ALTERA Corporation, Application Note, 101, Innovation Drive, San Jose, CA 95134, NOV 2001.

[4] J. Ardizzoni. *A Practical Guide to High-Speed Printed-Circuit-Board Layout.* Analog Devices, Analog Dialogue, One Technology Way, P.O Box 9106, Norwood, MA 02062-9106, SEP 2005.

[5] P. Brokaw. *AN-202, An IC Amplifier User's Guide to Decoupling, Grounding and Making Things Go Right for a Change.* Analog Devices Application Notes, One Technology Way, P.O Box 9106, Norwood, MA 02062-9106.

[6] P. Brokaw. *AN-342, Analog Signal-Handling for High Speed and Accuracy.* Analog Devices Application Notes, One Technology Way, P.O Box 9106, Norwood, MA 02062-9106, NOV 1977.

[7] National Semiconductor Corporation. LM78XX Datasheet. MAY 2000.

[8] National Semiconductor Corporation. LMS1585A Datasheet. JUN 2005.

[9] N. Gray. *The problem of ADC and mixed-signal grounding and layout for dynamic performance while minimizing RFI/EMI.* National Semiconductor Corporation, Analog Edge Monthly, 2900 Semiconductor Dr., P.O. Box 58090, Santa Clara, CA 95052-8090, NOV 2004.

[10] M. Kumm H. Klingbeil. *SIS18 Regelsystemtopologie.* Geselschaft für Schwerionenforschung mbH, Darmstadt, FEB 2006.

[11] Analog Devices Inc. AD8330 Datasheet. 2004.

[12] Analog Devices Inc. ADR431 Datasheet. 2004.

[13] Analog Devices Inc. AD8131 Datasheet. 2005.

[14] Analog Devices Inc. AD9744 Datasheet. 2005.

[15] Texas Instruments Incorporated. SN74LVTH245B Datasheet. JAN 1995.

[16] Texas Instruments Incorporated. THS4001 Datasheet. MAR 1999.

[17] Texas Instruments Incorporated. SN74LVTH16245A Datasheet. OCT 2005.

[18] B. Schwartz J. Reichardt. *VHDL-Systhese, Entwurf digitaler Schaltungen und Systeme*. Oldenbourg Wissenschaftsverlag GmbH, München, 2003.

[19] Dr. H. Johnson. *Multiple ADC Grounding*. Signal Consulting Inc., http://www.sigcon.com/, 2006.

[20] J. Jöst. *Entwicklung eines ADC-DAC Boards für digitale Regelungssysteme in einer DSP-Umgebung*. Diplomarbeit, Fachhochschule Bielefeld, Bielefeld, MAY 2006.

[21] Dr. H. Klingbeil. *Präsentation zu "Digitale Kavitätensynchronisation" IAP UNI Frankfurt*. Geselschaft für Schwerionenforschung mbH, Darmstadt, NOV 2004.

[22] M. Kumm. *Hardware- und Software-Entwicklung einer Mikrocontroller-Leiterplatte für eine automatische Verstärkungsregelung*. Diplomarbeit, Fachhochschule Fulda, Fulda, JUN 2003.

[23] K. Kundert. *Power Supply Noise Reduction*. The Designer's Guide Community, ken@designers-guide.org, MAR 2005.

[24] Future Technology Devices International Ltd. FT245R Datasheet. JAN 2006.

[25] G. Schreiber. *Technical Concept Barrier Buckets*. Geselschaft für Schwerionenforschung mbH, Darmstadt, SEP 2004.

[26] A. Sherry. *AN-611, 50 Hz/60 Hz Rejection on Sigma-Delta ADCs*. Analog Devices Application Notes, One Technology Way, P.O Box 9106, Norwood, MA 02062-9106.

[27] Linear Technology. LTC2249 Datasheet. 2004.

[28] Inc. Vishay Siliconix, Vishay Intertechnology. DG641 Datasheet. SEP 2004.

[29] J. Bryant W. Kester and M. Byrne. *MT-031: Grounding Data Converters and Solving the Mystery of "AGND" and "DGND"*. Analog Devices Technical Articles, One Technology Way, P.O Box 9106, Norwood, MA 02062-9106, FEB 2006.

A Used Abbreviations

ADC	**A**nalogue to **D**igital **C**onverter
AGC	**A**utomatic **G**ain **C**ontrol
BNC	**B**ayonet **N**eill **C**oncelman
CAD	**C**omputer **A**ided **D**esign
CAM	**C**omputer **A**ided **M**anufacturing
CNC	**C**omputer **N**umerical **C**ontrol
CMRR	**C**ommon-**M**ode **R**ejection **R**atio
CPLD	**C**onfigurable **P**rogrammable **L**ogic **D**evice
CSS	**C**ascading **S**tyle **S**heets
DAC	**D**igital to **A**nalog **C**onverter
DSP	**D**igital **S**ignal **P**rocessor
EDA	**E**lectronic **D**esign **A**utomation
EPS	**E**ncapsulated **P**ost **S**cript
FAB	**F**IB **A**dapter **B**oard
FAIR	**F**acility for **A**ntiproton and **I**on **R**esearch
FF	**F**lip **F**lop
FIB	**F**PGA **I**nterface **B**oard
FIFO	**F**irst **I**n **F**irst **O**ut
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
FR4	**F**lame **R**esistant 4
FSM	**F**inite **S**tate **M**achine
GSI	**G**eselschaft für **S**chwer**i**onenforschung mbH

GNU	**G**NU's **N**ot **U**nix (recursive acronym!)
HAL	**H**ot **A**ir **L**eveling
HPGL	**H**ewlett-**P**ackard **G**raphics **L**anguage
IDC	**I**nsulation-**D**isplacement **C**onnector
IDE	**I**ntegrated **D**evelopment **E**nvironment
IC	**I**ntegrated **C**ircuit
MSPS	**M**ega **S**amples **p**er **S**econd
OTR	**O**ut of **R**ange
PCB	**P**rinted **C**ircuit **B**oard
PLL	**P**hased **L**ock**L**oop
PP	**P**eak to **P**eak
RF	**R**adio **F**requency
RMS	**R**oot **M**ean **S**quare
RS-274C	**R**ecommended **S**tandard -274C
SMD	**S**urface-**M**ount **D**evices
SME	**S**mall and **M**edium-sized **E**nterprises
SFDR	**S**purious **F**ree **D**ynamic **R**ange
SVG	**S**calable **V**ector **G**raphics
TO-220	**T**ransistor **O**utline -220
TO-263	**T**ransistor **O**utline -263
TQFP	**T**hin **Q**uad **F**lat **P**ackage
UDL	**U**SB **D**SP **L**ink
USB	**U**niversal **S**erial **B**us
VHDL	**V**HSIC **H**ardware **D**escription **L**anguage
VHSIC	**V**ery-**H**igh-**S**peed **I**ntegrated **C**ircuit
VGA	**V**ariable **G**ain **A**mplifier
VNA	**V**ector **N**etwork **A**nalyser
XML	**E**xtensible **M**arkup **L**anguage

B Used PC Software

Following is a list of PC software which has been used to prepare this thesis. It has been tried to use free/open source software wherever possible. This appendix provides a small introduction with the purpose of promoting the use free software in educational and industrial environments.

B.1 EDA

This section introduces the software used for preparation of the board, simulation or programming.

B.1.1 Cadsoft EAGLE

According to the website of Cadsoft, EAGLE is "an easy to use, powerful and affordable schematic capture and printed circuit board design package that gets the job done"[1]. It includes solutions for PCB design, including Schematic Capture, Board Layout, and an Autorouter.

EAGLE has a free light version for personal non-profit use limited to boards with dimensions less than 100mm × 80mm. At GSI, the author used a full licensed version without limitations. The author recommends EAGLE for small and medium-sized enterprises (SME) and for research institutions. This software works also under Unix-like operating systems.

B.1.2 GraphiCode GC-Prevue

Graphicode provides a free viewer for the production data. "GC-Prevue is the industry standard software for viewing and printing electronic manufacturing data. GC-Prevue reads all of the common CAD generated electronic manufacturing out-

[1] Source: www.cadsoft.de

puts, including Gerber-X (RS-274X), Gerber-D (RS-274D), DPF (Barco), Excellon, Sieb & Meyer, HPGL, and HPGL2."[2]

B.1.3 ALTERA Quartus II

Quartus II Web Edition Software is the free version of the Quartus II family of design software provided by ALTERA for use with programmable logic made by this manufacturer. It features different analysis modes, VHDL and Verilog editor, compiler, simulator, fitter, optimizer and programmer[3].

B.1.4 ModelSIM Xilinx Edition

Instead of using the internal simulator of Quartus II Web Edition software, the author used ModelSIM Xilinx Edition which is a free version of the ModelSim family of products from Mentor Graphics[4]. The free version is usually enough for small and middle-sized projects. The only limitation is the lack of possibility to simulate part specific libraries that are provided by the IC manufacturer. As an example, the internal PLL of the FPGA could not be simulated using the free version. But in this case this was really not needed. For simulation of more complex designs where the user implements the specific libraries to activate special functions (also known as mega-functions in ALTERA's products) such as internal multipliers or flash memories, the free version of ModelSim is not sufficient.

B.1.5 GNU Emacs

Almost all the code has been written in GNU Emacs[5]. When set to its `vhdl-mode`, this editor is the ultimate solution for writing VHDL code. This editor is available under today's popular operating systems.

B.2 Text Processing: LaTeX

The typesetting of this document has been done with LaTeX, to be more precise the TeTeX[6] distribution which is usually freely available under Linux and other Unix-like

[2]Source: www.graphicode.com
[3]More information on ALTERA's products can be found at www.altera.com
[4]www.model.com
[5]www.gnu.org/software/emacs/
[6]www.tug.org/tetex/

operating systems or under CygWIN[7].

B.3 Graphic Software

The preparation of diagrams and other graphics throughout this document wouldn't be possible without the correct tools. Following are a set of free software that has been used. All of them are available under popular operating systems.

B.3.1 InkScape

"Inkscape is an Open Source vector graphics editor. The main goal is to create a powerful and convenient drawing tool fully compliant with XML, SVG, and CSS standards. It also aims to maintain a thriving user and developer community by using open, community-oriented development."[8]

B.3.2 Dia

"Dia can be used to draw many different kinds of diagrams."[9] It is easy to use and exports directly to `EPS` and `SVG`format which is desired for embedding in a LaTeX document or processing further in InkScape (See section B.3.1).

B.3.3 The GIMP

"GIMP is the GNU Image Manipulation Program. It is a freely distributed piece of software for such tasks as photo retouching, image composition and image authoring. It works on many operating systems, in many languages."[10]

[7] www.cygwin.com
[8] Source: www.inkscape.org
[9] Source: www.gnome.org/projects/dia/
[10] Source: www.gimp.org

C CPLD Code

The following is the code written for the CPLD. Please note that some entities have been used also for the FPGA. Test benches to the respective code has been omitted here.

```
------------------------------------------------------------

--
-- 16-bit Bus driver with Tri State outputs
-- 20.07.2006/sh
--

library ieee;
use ieee.std_logic_1164.all;

entity busdriver is

  port (
    en_write_to_bus : in    std_logic;  -- enable the buffer
    data_bus        : inout std_logic_vector (15 downto 0);  -- Bus connection
    data_to_bus     : in    std_logic_vector (15 downto 0);  -- data written into the bus
    data_from_bus   : out   std_logic_vector (15 downto 0)  -- data read from the bus
    );

end busdriver;

architecture busdriver_arch of busdriver is

begin  -- busdriver_arch

  data_bus      <= data_to_bus when en_write_to_bus = '1' else (others => 'Z');
  data_from_bus <= data_bus;

end busdriver_arch;

------------------------------------------------------------

--
-- clock divider
-- 21.07.2006/sh
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity clk_divider is
```

```
  generic (
    clk_divider_width : integer := 16);  -- Bit Width of the clock divider

  port (
    clk_div : in  std_logic_vector (clk_divider_width - 1 downto 0);  \
    -- clock division constant
    rst_i   : in  std_logic;            -- async reset  in
    clk_i   : in  std_logic;            -- clk input
    clk_o   : out std_logic);           -- clk out

end clk_divider;

architecture clk_divider_arch of clk_divider is

  signal clk_cnt     : integer range 0 to 2**clk_divider_width - 1;  \
  -- clk counter variable
  signal clk_o_local : std_logic;       -- local clock for the operations

begin  -- clk_divider_arch

  clk_o <= clk_o_local;                 -- always connect these two

  -- purpose: divides clock input
  -- type   : sequential
  -- inputs : clk_i
  -- outputs: clk_o

  p_clock : process (clk_i, rst_i, clk_div, clk_cnt)
  begin  -- process p_clock
    if rst_i = '1' then                 -- asynchronous reset (active high)

      clk_cnt     <= conv_integer (clk_div);  -- initialize with the constant
      clk_o_local <= '0';               -- initialize the output clock to zero

    elsif clk_cnt = 0 then
      clk_o_local <= '0';

    elsif clk_cnt = 1 then
      clk_o_local <= clk_i;

    elsif clk_i'event and clk_i = '1' then  -- rising clock edge

      if clk_cnt = 2 then
        clk_cnt     <= conv_integer (clk_div);
        clk_o_local <= not clk_o_local;

      else
        clk_cnt <= clk_cnt - 1;
      end if;

    end if;
  end process p_clock;

end clk_divider_arch;

---------------------------------------------------------------

-- FAB ADC/DAC
-- Register File
-- Start 26.07.2006/sh

library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity register_file is

  port (
    clk_i    : in std_logic;            -- clock input
    rst_i    : in std_logic;            -- reset input
    rnw_i    : in std_logic;            -- read/write signal
    strobe_i : in std_logic;            -- Strobe Signal

    adr_i         : in  std_logic_vector (5 downto 0);  -- input adress bus
    data_from_bus : in  std_logic_vector (15 downto 0);
    data_to_bus   : out std_logic_vector (15 downto 0);

    register_00 : out std_logic_vector (15 downto 0);
    register_01 : in  std_logic_vector (15 downto 0);
    register_02 : in  std_logic_vector (15 downto 0);
    register_03 : in  std_logic_vector (15 downto 0);
    register_04 : out std_logic_vector (15 downto 0);
    register_05 : out std_logic_vector (15 downto 0);
    register_06 : out std_logic_vector (15 downto 0);
    register_07 : out std_logic_vector (15 downto 0);
    register_08 : out std_logic_vector (15 downto 0);
    register_09 : out std_logic_vector (15 downto 0)
    );
end register_file;

architecture register_file_arch of register_file is

  -- local_registers

  signal local_register_00 : std_logic_vector (15 downto 0);  -- Register
  signal local_register_01 : std_logic_vector (15 downto 0);  -- Register
  signal local_register_02 : std_logic_vector (15 downto 0);  -- Register
  signal local_register_03 : std_logic_vector (15 downto 0);  -- Register
  signal local_register_04 : std_logic_vector (15 downto 0);  -- Register
  signal local_register_05 : std_logic_vector (15 downto 0);  -- Register
  signal local_register_06 : std_logic_vector (15 downto 0);  -- Register
  signal local_register_07 : std_logic_vector (15 downto 0);  -- Register
  signal local_register_08 : std_logic_vector (15 downto 0);  -- Register
  signal local_register_09 : std_logic_vector (15 downto 0);  -- Register

begin  -- register_file_arch

  -- inteconnections
  --  register_04 <= local_register_04 when strobe_i = '1' and rnw_i = '0';

  -- processes

--  p_write_enable : process (local_register_04, strobe_i, rnw_i)
--  begin  -- process test
--    if strobe_i = '1' and rnw_i = '0' then
--      register_04 <= local_register_04;
--    end if;
--  end process p_write_enable;

--  p_write_enable : process (local_register_00, local_register_04, \
local_register_05, local_register_06, local_register_07, local_register_08, \
local_register_09, strobe_i, rnw_i)
--  begin  -- process test
--    if not local_register_04 = "ZZZZZZZZZZZZZZZZ" then
--    if strobe_i = '1' and rnw_i = '0' then
```

```
      register_00 <= local_register_00;
      register_04 <= local_register_04;
      register_05 <= local_register_05;
      register_06 <= local_register_06;
      register_07 <= local_register_07;
      register_08 <= local_register_08;
      register_09 <= local_register_09;
--    end if;
--  end process p_write_enable;

  p_read_enable : process (register_01, register_02, register_03, strobe_i, \
  rnw_i)
  begin  -- process p_read_enable
    if strobe_i = '1' and rnw_i = '1' then

      local_register_01 <= register_01;
      local_register_02 <= register_02;
      local_register_03 <= register_03;
    end if;
  end process p_read_enable;

  p_write_local_registers : process (clk_i, rst_i, strobe_i, rnw_i)
  begin  -- process p_write_local_registers
    if rst_i = '1' then                 -- asynchronous reset (active high)

      -- reset all local_registers to the default values

      local_register_00 <= "0100010000000000";
--      local_register_01 <= (others => 'Z');
--      local_register_02 <= (others => 'Z');
--      local_register_03 <= (others => 'Z');
      local_register_04 <= x"0000";
      local_register_05 <= x"0000";
      local_register_06 <= x"0002";
      local_register_07 <= x"0002";
      local_register_08 <= x"0001";
      local_register_09 <= x"0001";

    elsif clk_i'event and clk_i = '1' then  -- rising clock edge

      if strobe_i = '1' then
        if rnw_i = '0' then             -- fib wants to write

          -- the read-only local_registers 0x01, 0x02 and 0x03 shouldn't \
          be overwritten

          case conv_integer (adr_i (5 downto 0)) is
            when 0      => local_register_00 <= data_from_bus;
            when 4      => local_register_04 <= data_from_bus;
            when 5      => local_register_05 <= data_from_bus;
            when 6      => local_register_06 <= data_from_bus;
            when 7      => local_register_07 <= data_from_bus;
            when 8      => local_register_08 <= data_from_bus;
            when 9      => local_register_09 <= data_from_bus;
            when others => null;
          end case;

        end if;
      end if;
    end if;
  end process p_write_local_registers;

  p_read_local_registers : process (clk_i, strobe_i, rnw_i)
```

```
  begin  -- process p_read_local_registers
    if clk_i'event and clk_i = '1' then  -- rising clock edge
      if strobe_i = '1' then
        if rnw_i = '1' then             -- fib wants to read

          -- The value of all local_registers could be read by fib

          case conv_integer (adr_i (5 downto 0)) is
            when 0      => data_to_bus <= local_register_00;
            when 1      => data_to_bus <= local_register_01;
            when 2      => data_to_bus <= local_register_02;
            when 3      => data_to_bus <= local_register_03;
            when 4      => data_to_bus <= local_register_04;
            when 5      => data_to_bus <= local_register_05;
            when 6      => data_to_bus <= local_register_06;
            when 7      => data_to_bus <= local_register_07;
            when 8      => data_to_bus <= local_register_08;
            when 9      => data_to_bus <= local_register_09;
            when others => null;
          end case;

        end if;
      end if;
    end if;
  end process p_read_local_registers;
end register_file_arch;

-------------------------------------------------------------

--
-- Saw tooth generator by counting
-- 25.07.2006/sh
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity sawtooth is

  generic (
    counter_width : integer := 14);

  port (
    dat_o : out std_logic_vector (counter_width - 1 downto 0);  -- data output
    rst_i : in  std_logic;                                      -- reset input
    clk_i : in  std_logic);                                     -- input clock

end sawtooth;

architecture sawtooth_arch of sawtooth is

  signal counter : integer range 0 to 2**counter_width -1;  -- counter variable

begin  -- sawtooth_arch

  p_saw_tooth : process (clk_i, rst_i, counter)

  begin  -- process p_dac1_test

    if rst_i = '1' then                 -- reset active high
```

```
      counter <= 2**counter_width - 1;

    elsif clk_i'event and clk_i = '1' then  -- rising clock edge

      if counter = 0 then
        counter <= 2**counter_width - 1;

      else
        counter <= counter - 1;
      end if;

      dat_o <= conv_std_logic_vector (counter, counter_width);

    end if;

  end process p_saw_tooth;

end sawtooth_arch;

------------------------------------------------------------

-- FAB ADC/DAC
-- Top Level Entity for ALTERA MAXII
-- Start 17.07.2006/sh

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity fab_adcdac_app1_top_level is

  generic (
    clk_divider_width_toplevel : integer := 16;  -- width of the \
    divider in bits
    reset_clks_toplevel        : integer := 2);  -- Tells how many \
    clocks the POR takes

  port (

    -- common signals

    fibclk : in std_logic;              -- main clock in

    -- fib signals

    fibd : inout std_logic_vector (15 downto 0);  -- fib data bus
    fiba : in    std_logic_vector (5 downto 0);   -- fib address bus

    fibrnw    : in  std_logic;          -- read/write signal from fib:
                                        -- 1 = read, 0 = write
    fibstrobe : in  std_logic;          -- strobe signal from fib
    fiback    : out std_logic;          -- ack output to fib

    -- board signals

    adc1d : in  std_logic_vector (13 downto 0);  -- ADC1 data input
    dac1d : out std_logic_vector (13 downto 0);  -- DAC1 data output
    adc2d : in  std_logic_vector (13 downto 0);  -- ADC2 data input
```

```
    dac2d : out std_logic_vector (13 downto 0);  -- DAC2 data output

    adc1sw : out std_logic_vector (3 downto 0);  -- calibration switch for ADC1
    adc2sw : out std_logic_vector (3 downto 0);  -- calibration switch for ADC2

    -- clock signals

    adc1clk : out std_logic;            -- clock for ADC1
    adc2clk : out std_logic;            -- clock for ADC2
    dac1clk : out std_logic;            -- clock for DAC1
    dac2clk : out std_logic;            -- clock for DAC2

    -- static config signals

    adc1of   : in  std_logic;           -- overflow from ADC1
    adc2of   : in  std_logic;           -- overflow from ADC2
    adc1shdn : out std_logic;           -- shut down ADC1
    adc2shdn : out std_logic;           -- shut down ADC2
    dac1slp  : out std_logic;           -- shut down DAC1
    dac2slp  : out std_logic;           -- shut down DAC2

    -- test pins

    tp1_tio1     : out std_logic;       -- test pin 1
    tp2_tio1     : out std_logic;       -- test pin 2
    tp3_dev_clrn : in  std_logic;       -- test pin 3
    tp4_gclk0    : in  std_logic;       -- test pin 4
    tp5_gclk1    : in  std_logic;       -- test pin 5
    tp6_dev_oe   : in  std_logic;       -- test pin 6
    tp7_gclk3    : in  std_logic        -- test pin 7

    );

end fab_adcdac_app1_top_level;

architecture fab_adcdac_app1_top_level_arch of fab_adcdac_app1_top_level is

  -- components declaration

  component reset_gen
    generic(
      reset_clks : integer := 2
      );
    port
      (
        clk_i : in  std_logic;
        rst_o : out std_logic
        );
  end component;

  component clk_divider

    generic (
      clk_divider_width : integer);     -- Bit Width of the clock divider

    port (
      clk_div : in  std_logic_vector (clk_divider_width - 1 downto 0);  \
      -- clock division constant
      rst_i   : in  std_logic;          -- async reset  in
      clk_i   : in  std_logic;          -- clk input
      clk_o   : out std_logic);         -- clk out

  end component;
```

```
  component register_file
    port (
      clk_i         : in  std_logic;
      rst_i         : in  std_logic;
      rnw_i         : in  std_logic;
      strobe_i      : in  std_logic;
      adr_i         : in  std_logic_vector (5 downto 0);
      data_from_bus : in  std_logic_vector (15 downto 0);
      data_to_bus   : out std_logic_vector (15 downto 0);
      register_00   : out std_logic_vector (15 downto 0);
      register_01   : in  std_logic_vector (15 downto 0);
      register_02   : in  std_logic_vector (15 downto 0);
      register_03   : in  std_logic_vector (15 downto 0);
      register_04   : out std_logic_vector (15 downto 0);
      register_05   : out std_logic_vector (15 downto 0);
      register_06   : out std_logic_vector (15 downto 0);
      register_07   : out std_logic_vector (15 downto 0);
      register_08   : out std_logic_vector (15 downto 0);
      register_09   : out std_logic_vector (15 downto 0));
  end component;

  component busdriver

    port (
      en_write_to_bus : in    std_logic;  -- enable the buffer
      data_bus        : inout std_logic_vector (15 downto 0);  -- Bus \
      connection
      data_to_bus     : in    std_logic_vector (15 downto 0);  -- data \
      written into the bus
      data_from_bus   : out   std_logic_vector (15 downto 0)  -- data \
      read from the bus
      );

  end component;

-- internal registers

  signal register_00 : std_logic_vector (15 downto 0);
  signal register_01 : std_logic_vector (15 downto 0);
  signal register_02 : std_logic_vector (15 downto 0);
  signal register_03 : std_logic_vector (15 downto 0);
  signal register_04 : std_logic_vector (15 downto 0);
  signal register_05 : std_logic_vector (15 downto 0);
  signal register_06 : std_logic_vector (15 downto 0);
  signal register_07 : std_logic_vector (15 downto 0);
  signal register_08 : std_logic_vector (15 downto 0);
  signal register_09 : std_logic_vector (15 downto 0);

  -- test pins

  signal testpins_vector_o : std_logic_vector (1 downto 0);  -- Vector \
  for the test pins
  signal testpins_vector_i : std_logic_vector (4 downto 0);  -- Vector \
  for the test pins

  --internal variables

  signal global_rst : std_logic;        -- internal global reset signal

  signal por_rst : std_logic;           -- signal from the POR generator

  signal fiba_pre_synched      : std_logic_vector (5 downto 0);  -- synched \
```

```
  signal connected to FIBA
  signal fibrnw_pre_synched    : std_logic;  -- synched signal \
  connected to FIBRNW
  signal fibstrobe_pre_synched : std_logic;  -- synched signal \
  connected to FIBSTROBE

  signal fiba_synched      : std_logic_vector (5 downto 0);  \
  -- synched signal connected to FIBA
  signal fibrnw_synched    : std_logic;  -- synched signal \
  connected to FIBRNW
  signal fibstrobe_synched : std_logic;  -- synched signal \
  connected to FIBSTROBE

  -- Bus Driver Signals

  signal en_write_to_bus : std_logic;  -- internal enable signal \
  to write to bus
  signal data_to_bus     : std_logic_vector (15 downto 0);  \
  -- internal vector interface to the bus
  signal data_from_bus   : std_logic_vector (15 downto 0);  \
  -- internal vector interface to the bus

begin  -- fab_adcdac_app1_top_level_arch

  -- component instances

  reset_gen_inst : reset_gen
    generic map (
      reset_clks => reset_clks_toplevel)

    port map (
      clk_i => fibclk,
      rst_o => por_rst);

  adc1_clk_divider_inst : clk_divider

    generic map (
      clk_divider_width => clk_divider_width_toplevel)

    port map (
      clk_div => register_06,
      rst_i   => global_rst,
      clk_i   => fibclk,
      clk_o   => adc1clk);

  adc2_clk_divider_inst : clk_divider

    generic map (
      clk_divider_width => clk_divider_width_toplevel)

    port map (
      clk_div => register_07,
      rst_i   => global_rst,
      clk_i   => fibclk,
      clk_o   => adc2clk);

  dac1_clk_divider_inst : clk_divider

    generic map (
      clk_divider_width => clk_divider_width_toplevel)

    port map (
      clk_div => register_08,
```

```
    rst_i   => global_rst,
    clk_i   => fibclk,
    clk_o   => dac1clk);

dac2_clk_divider_inst : clk_divider

  generic map (
    clk_divider_width => clk_divider_width_toplevel)

  port map (
    clk_div => register_09,
    rst_i   => global_rst,
    clk_i   => fibclk,
    clk_o   => dac2clk);

blinker1 : clk_divider
  generic map (
    clk_divider_width => 4)
  port map (
    clk_div => "1010",
    rst_i   => global_rst,
    clk_i   => fibclk,
    clk_o   => testpins_vector_o(0));

register_file_1 : register_file
  port map (
    clk_i         => fibclk,
    rst_i         => global_rst,
    rnw_i         => fibrnw_synched,
    strobe_i      => fibstrobe_synched,
    adr_i         => fiba_synched,
    data_from_bus => data_from_bus,
    data_to_bus   => data_to_bus,
    register_00   => register_00,
    register_01   => register_01,
    register_02   => register_02,
    register_03   => register_03,
    register_04   => register_04,
    register_05   => register_05,
    register_06   => register_06,
    register_07   => register_07,
    register_08   => register_08,
    register_09   => register_09);

busdriver_inst : busdriver
  port map (
    en_write_to_bus => en_write_to_bus,
    data_bus        => fibd,
    data_to_bus     => data_to_bus,
    data_from_bus   => data_from_bus);

-- register assignments

global_rst <= por_rst or register_00(3);  -- either of the reset sources

adc2sw <= register_00 (15 downto 12);
adc1sw <= register_00 (11 downto 8);

adc2shdn <= register_00 (7);
adc1shdn <= register_00 (6);
dac2slp  <= register_00 (5);
dac1slp  <= register_00 (4);
```

```
  register_01 (2) <= adc1of;
  register_01 (3) <= adc2of;

  register_02 (13 downto 0)  <= adc1d;
  register_02 (15 downto 14) <= (others => '0');

  register_03 (13 downto 0)  <= adc2d;
  register_03 (15 downto 14) <= (others => '0');

  dac1d                      <= not(register_04 (13 downto 0)) \
  + conv_std_logic_vector (1, 14);
  --  dac1d         <= not(register_04 (13 downto 0);
  register_04 (15 downto 14) <= (others => '0');

  dac2d <= not(register_05(13 downto 0)) + conv_std_logic_vector (1, 14);
  -- dac2d         <= not(register_05(13 downto 0);

  register_05 (15 downto 14) <= (others => '0');

  -- testpins_vector
  tp1_tio1 <= testpins_vector_o(0);
  tp2_tio1 <= testpins_vector_o(1);

  --  testpins_vector_o(1) <= fibclk;       -- test clock out \
  on the first test pin2

  fiback <= '0';

  -- enable only when fib wants to read.
  en_write_to_bus <= fibrnw_synched;

  -- processes

  process (fibclk, fiba, fibrnw, fibstrobe, fibstrobe_pre_synched, \
  fibrnw_pre_synched, fiba_pre_synched)
  begin
    if fibclk'event and fibclk = '1' then  -- rising clock edge

      -- input signal mapping on each rising clock edge

      fiba_pre_synched      <= fiba;
      fibrnw_pre_synched    <= fibrnw;
      fibstrobe_pre_synched <= fibstrobe;

      fiba_synched      <= fiba_pre_synched;
      fibrnw_synched    <= fibrnw_pre_synched;
      fibstrobe_synched <= fibstrobe_pre_synched;

      testpins_vector_i <= (
        0 => tp3_dev_clrn,
        1 => tp4_gclk0,
        2 => tp5_gclk1,
        3 => tp6_dev_oe,
        4 => tp7_gclk3
        );

    end if;
  end process;

end fab_adcdac_app1_top_level_arch;

------------------------------------------------------------
```

D FPGA Code

```
------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.math_real.all;

entity digital_short is
  generic (
    clk_freq_in_hz : real := 200000000.0;  --system clock frequency
    delay_in_ns    : real := 25.0);        -- 25.0

  port (
    rst_i : in std_logic;               -- reset in
    clk_i : in std_logic;               -- clock in

    rnw_o    : out std_logic;
    strobe_o : out std_logic;
    ack_i    : in  std_logic;

    ext_driver_dir : out std_logic;

    adr_o : out std_logic_vector (5 downto 0);

    en_write_to_bus : out std_logic;

    data_to_bus   : out std_logic_vector (15 downto 0);
    data_from_bus : in  std_logic_vector (15 downto 0)
    );

end digital_short;

architecture digital_short_arch of digital_short is

  --limits the integer to a minimal value of one (for timing counters)
  function limit_to_minimal_value(x : integer; min : integer) return \
  integer is
  begin
    if x > min then
      return x;
    else
      return 1;
    end if;
  end limit_to_minimal_value;

  signal delay_in_ticks : integer := limit_to_minimal_value(integer\
  (clk_freq_in_hz * delay_in_ns / 1000000000.0) - 2, 0);
-- 3 clock cycles are needed for the state machine

  constant FAB_ADC_ADR : std_logic_vector (5 downto 0) := "000010"; \
```

```
  -- Address of the ADC1 channel
  constant FAB_DAC_ADR : std_logic_vector (5 downto 0) := "000100"; \
  -- Address of the DAC1 channel

  signal delay_cnt : integer range 0 to delay_in_ticks;

  type state_type is (READ_PRE1_STATE, READ_PRE2_STATE, READ_PRE3_STATE, \
!\!9  READ_STATE, WRITE_PRE1_STATE, WRITE_PRE2_STATE, WRITE_PRE3_STATE, \
WRITE_STATE, WAIT_STATE);  -- States

  signal state           : state_type;
  signal return_to_state : state_type;

  signal local_data : std_logic_vector (15 downto 0);  -- data read from \
  or written to fab

begin  -- digital_short_arch


  -- purpose: Read from ADC on FAB and write it back in FAB's DAC

  process (clk_i, rst_i, ack_i, state)

  begin  -- process p_shorting

    if rst_i = '1' then                 -- asynchronous reset (active high)

      -- in reset case, all drivers as input
      strobe_o        <= '0';
      rnw_o           <= '1';
      en_write_to_bus <= '0';
      ext_driver_dir  <= '0';
      state           <= READ_PRE1_STATE;
      return_to_state <= READ_PRE1_STATE;
      delay_cnt       <= delay_in_ticks;

    elsif clk_i'event and clk_i = '1' then  -- rising clock edge

      case state is

        when READ_PRE1_STATE =>
          strobe_o        <= '0';
          en_write_to_bus <= '0';
          adr_o           <= FAB_ADC_ADR;
          state           <= READ_PRE2_STATE;

        when READ_PRE2_STATE =>
          ext_driver_dir <= '0';
          state          <= READ_PRE3_STATE;

        when READ_PRE3_STATE =>
          strobe_o        <= '1';
          rnw_o           <= '1';
          state           <= WAIT_STATE;
          return_to_state <= READ_STATE;

        when READ_STATE =>
          local_data <= data_from_bus;
          state      <= WRITE_PRE1_STATE;

        when WRITE_PRE1_STATE =>
          strobe_o <= '0';
          rnw_o    <= '0';
```

```
          adr_o    <= FAB_DAC_ADR;
          state    <= WRITE_PRE2_STATE;

        when WRITE_PRE2_STATE =>
          ext_driver_dir <= '1';
          state          <= WRITE_PRE3_STATE;

        when WRITE_PRE3_STATE =>
          en_write_to_bus <= '1';
--          strobe_o        <= '1';
          state           <= WRITE_STATE;

        when WRITE_STATE =>
          strobe_o        <= '1';
          data_to_bus     <= local_data;
          state           <= WAIT_STATE;
          return_to_state <= READ_PRE1_STATE;

        when WAIT_STATE =>
          strobe_o        <= '0';
          if delay_cnt = 0 then
            state     <= return_to_state;
            delay_cnt <= delay_in_ticks;
          else
            delay_cnt <= delay_cnt - 1;
          end if;

        when others => null;
      end case;

    end if;
  end process;

end digital_short_arch;

-------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity fib_adcdac_app1_top_level is
  generic(
    clk_freq_in_hz   : integer := 50000000;  --100 MHz system clock frequency
    firmware_id      : integer := 1;  --ID of the firmware (is displayed first)
    firmware_version : integer := 3  --Version of the firmware (is displayed after)
    );

  port (
    --common signals
    trig1_in  : in  std_logic;          --rst
    trig2_out : out std_logic;
    clk0      : in  std_logic;
    hf_in     : in  std_logic;

    --FAB signals
    uC_Link_D   : inout std_logic_vector(7 downto 0);  -- FAB Lower Byte
    uC_Link_A   : inout std_logic_vector(7 downto 0);  -- FAB Upper Byte
    Piggy_Clk1  : out   std_logic;                     -- FAB Clock
    Piggy_RnW1  : out   std_logic;                     --dds_wr
    --    Piggy_RnW2  : in  std_logic;                     --dds_vout_comp
    --    Piggy_Strb2 : out std_logic;                     --dds_rst
```

```
Piggy_Strb1 : out   std_logic;                    --dds_update_o
Piggy_Ack1  : in    std_logic;                    --dds_fsk
--    Piggy_Ack2  : out std_logic;                    --dds_sh_key

--static dds-buffer signals
uC_Link_DIR_D, uC_Link_DIR_A : out std_logic;
nuC_Link_EN_CTRL_A           : out std_logic;
uC_Link_EN_DA                : out std_logic;

--backplane signals
A2nSW8      : in  std_logic;
A3nSW9      : in  std_logic;
AOnSW10     : in  std_logic;
A1nSW11     : in  std_logic;
Sub_A6nSW12 : in  std_logic;
Sub_A7nSW13 : in  std_logic;
Sub_A4nSW14 : in  std_logic;
Sub_A5nSW15 : in  std_logic;
nResetnSW0  : in  std_logic;
SW1         : in  std_logic;
nDSnSW2     : in  std_logic;
BClocknSW3  : in  std_logic;
RnWnSW4     : in  std_logic;
SW5         : in  std_logic;
A4nSW6      : in  std_logic;
SW7         : in  std_logic;
NEWDATA     : in  std_logic;
FC_Str      : in  std_logic;
FC0         : in  std_logic;
FC1         : in  std_logic;
FC2         : in  std_logic;
FC3         : in  std_logic;
FC4         : in  std_logic;
FC5         : in  std_logic;
VG_A3nFC6   : in  std_logic;
FC7         : in  std_logic;
SD          : in  std_logic;
nDRQ2       : out std_logic;

--static backplane-buffer signals
BBA_DIR : out std_logic;
BBB_DIR : out std_logic;
BBC_DIR : out std_logic;
BBD_DIR : out std_logic;
BBE_DIR : out std_logic;
BBG_DIR : out std_logic;
BBH_DIR : out std_logic;
nBB_EN  : out std_logic;

--static backplane open-collector outputs
DRDY     : out std_logic;
SRQ3     : out std_logic;
DRQ      : out std_logic;
INTERL   : out std_logic;
DTACK    : out std_logic;
nDRDY2   : out std_logic;
SEND_EN  : out std_logic;
SEND_STR : out std_logic;

--dsp-link signals (read)
DSP_CRDY_W : out std_logic;
DSP_CREQ_W : out std_logic;
DSP_CACK_R : in  std_logic;
```

```
    DSP_CSTR_R : in  std_logic;

    DSP_D_R0 : in std_logic;
    DSP_D_R1 : in std_logic;
    DSP_D_R2 : in std_logic;
    DSP_D_R3 : in std_logic;
    DSP_D_R4 : in std_logic;
    DSP_D_R5 : in std_logic;
    DSP_D_R6 : in std_logic;
    DSP_D_R7 : in std_logic;

    --dsp-link signals (write)
    DSP_CRDY_R : in  std_logic;
    DSP_CREQ_R : in  std_logic;
    DSP_CACK_W : out std_logic;
    DSP_CSTR_W : out std_logic;

    DSP_D_W0 : out std_logic;
    DSP_D_W1 : out std_logic;
    DSP_D_W2 : out std_logic;
    DSP_D_W3 : out std_logic;
    DSP_D_W4 : out std_logic;
    DSP_D_W5 : out std_logic;
    DSP_D_W6 : out std_logic;
    DSP_D_W7 : out std_logic;

    -- leds
    led1 : out std_logic;
    led2 : out std_logic;
    led3 : out std_logic;
    led4 : out std_logic;

    -- only for debug
    piggy_io : out std_logic_vector(7 downto 0);

    --adressing pins via FC
    VG_A4 : in std_logic;               --FC(0)
    VG_A1 : in std_logic;               --FC(1)

    -- dsp-link buffer enable signals

    DSP_DIR_D      : out std_logic;
    DSP_DIR_STRACK : out std_logic;
    DSP_DIR_REQRDY : out std_logic

    );
end entity fib_adcdac_app1_top_level;

architecture fib_adcdac_app1_top_level_arch of fib_adcdac_app1_top_level is

  -- components
  component reset_gen is
    generic (reset_clks : integer := 2);
    port (
      clk_i : in  std_logic;
      rst_o : out std_logic
      );
  end component;

  component clk_divider
    generic (
      clk_divider_width : integer);
    port (
```

```
      clk_div : in  std_logic_vector (clk_divider_width - 1 downto 0);
      rst_i   : in  std_logic;
      clk_i   : in  std_logic;
      clk_o   : out std_logic);
  end component;

--  component internal_pll
--    port (
--      areset : in  std_logic;
--      inclk0 : in  std_logic;
--      c0     : out std_logic;
--      e0     : out std_logic;
--      locked : out std_logic);
--  end component;

  component busdriver
    port (
      en_write_to_bus : in    std_logic;  -- enable the buffer
      data_bus        : inout std_logic_vector (15 downto 0);  \
      -- Bus connection
      data_to_bus     : in    std_logic_vector (15 downto 0);  \
      -- data written into the bus
      data_from_bus   : out   std_logic_vector (15 downto 0)  \
      -- data read from the bus
      );
  end component;

  component digital_short
    generic (
      clk_freq_in_hz : real := 200000000.0;  --system clock frequency
      delay_in_ns    : real := 15.0);        -- delay of the wait state

    port (
      rst_i : in std_logic;             -- reset in
      clk_i : in std_logic;             -- clock in

      rnw_o    : out std_logic;
      strobe_o : out std_logic;
      ack_i    : in  std_logic;

      ext_driver_dir : out std_logic;

      adr_o : out std_logic_vector (5 downto 0);

      en_write_to_bus : out std_logic;

      data_to_bus   : out std_logic_vector (15 downto 0);
      data_from_bus : in  std_logic_vector (15 downto 0)
      );

  end component;

  -- common signals
  signal global_rst : std_logic;

  -- FAB related signals

  signal data_bus : std_logic_vector (15 downto 0);  -- FAB Data bus \
  mapped to uCLinuk Address and Databus

  signal adr_o          : std_logic_vector (5 downto 0);  -- Address \
  bus for FAB
  signal ext_driver_dir : std_logic;
```

```
  signal areset : std_logic;
  signal inclk0 : std_logic;
  signal c0     : std_logic;
  signal e0     : std_logic;
  signal locked : std_logic;

  signal en_write_to_bus : std_logic;
  signal data_to_bus     : std_logic_vector (15 downto 0);
  signal data_from_bus   : std_logic_vector (15 downto 0);

begin

  reset_gen_inst : reset_gen
    port map (
      clk_i => clk0,
      rst_o => global_rst
      );

  blinker1 : clk_divider
    generic map (
      clk_divider_width => 24)
    port map (
      clk_div => x"EEFFFF",
      rst_i   => global_rst,
      clk_i   => clk0,
      clk_o   => led2);

--  internal_pll_inst : internal_pll
--    port map (
--      areset => areset,
--      inclk0 => inclk0,
--      c0     => c0,
--      e0     => e0,
--      locked => locked);

  busdriver_inst : busdriver
    port map (
      en_write_to_bus => en_write_to_bus,
      data_bus        => data_bus,
      data_to_bus     => data_to_bus,
      data_from_bus   => data_from_bus);

  digital_short_inst : digital_short
    port map (
      rst_i           => global_rst,
      clk_i           => clk0,
      rnw_o           => Piggy_RnW1,
      strobe_o        => Piggy_Strb1,
      ack_i           => Piggy_Ack1,
      ext_driver_dir  => ext_driver_dir,
      adr_o           => adr_o,
      en_write_to_bus => en_write_to_bus,
      data_to_bus     => data_to_bus,
      data_from_bus   => data_from_bus
      );

  --static backplane buffer settings
  BBA_DIR <= '0';
  BBB_DIR <= '0';
  BBC_DIR <= '0';
  BBD_DIR <= '0';
  BBE_DIR <= '0';
```

```
  BBG_DIR <= '0';
  BBH_DIR <= '0';
  nBB_EN  <= '0';

  --static backplane open-collector output settings
  DRDY     <= '0';
  SRQ3     <= '0';
  DRQ      <= '0';
  INTERL   <= '0';
  DTACK    <= '0';
  nDRDY2   <= '0';
  SEND_EN  <= '0';
  SEND_STR <= '0';

  --static uC-Link buffer settings

  nuC_Link_EN_CTRL_A <= '1';
  uC_Link_EN_DA      <= '0';

  -- unused buffers get warm!

  DSP_DIR_D      <= '1';
  DSP_DIR_REQRDY <= '1';
  DSP_DIR_STRACK <= '1';

  -- Actual signal interconnection

  uC_Link_D <= data_bus (7 downto 0)  when en_write_to_bus = '1' \
  else (others => 'Z');
  uC_Link_A <= data_bus (15 downto 8) when en_write_to_bus = '1' \
  else (others => 'Z');

  data_bus (7 downto 0)  <= uC_Link_D when en_write_to_bus = '0' \
  else (others => 'Z');
  data_bus (15 downto 8) <= uC_Link_A when en_write_to_bus = '0' \
  else (others => 'Z');

--  data_bus <= test;

  piggy_io (5 downto 0) <= adr_o;

  uC_Link_DIR_A <= ext_driver_dir;
  uC_Link_DIR_D <= ext_driver_dir;

-- PLL signals

--  areset <= '0';
--  inclk0     <= clk0;

  Piggy_Clk1 <= clk0;

--  process (clk0)
--  begin  -- process
--    if clk0'event and clk0 = '1' then  -- rising clock edge

--      data_bus (7 downto 0) <= uC_Link_D;

--    end if;
--  end process;

end architecture fib_adcdac_app1_top_level_arch;

-------------------------------------------------------------
```

E Simulation Code

```
------------------------------------------------------------

--
-- 16-bit Bus driver with Tri State outputs
-- Simulation code
-- 20.07.2006/sh
--

library ieee;
use ieee.std_logic_1164.all;

entity driver_7416245 is
  port (
    iobus_a : inout std_logic_vector (15 downto 0);
    iobus_b : inout std_logic_vector (15 downto 0);
    dir_i   : in    std_logic);
end driver_7416245;

architecture driver_7416245_arch of driver_7416245 is

begin  -- driver_7416245_arch

  iobus_a <= iobus_b when dir_i = '0' else (others => 'Z');
  iobus_b <= iobus_a when dir_i = '1' else (others => 'Z');

--  process
--  begin  -- process
--    loop

--      wait on dir_i;
--      if dir_i = '0' then
--        wait for 5 ns;
--        iobus_a <= iobus_b;
--      else
--        iobus_a <= (others => 'Z');
--      end if;

--      if dir_i = '1' then
--        wait for 5 ns;
--        iobus_b <= iobus_a;
--      else
--        iobus_b <= (others => 'Z');
--      end if;
--    end loop;
--  end process;

end driver_7416245_arch;

------------------------------------------------------------

library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity mixed_tb is

end mixed_tb;

architecture mixed_tb_arch of mixed_tb is

  component fab_adcdac_app1_top_level
    generic (
      clk_divider_width_toplevel : integer;
      reset_clks_toplevel        : integer);
    port (
      fibclk       : in    std_logic;
      fibd         : inout std_logic_vector (15 downto 0);
      fiba         : in    std_logic_vector (5 downto 0);
      fibrnw       : in    std_logic;
      fibstrobe    : in    std_logic;
      fiback       : out   std_logic;
      adc1d        : in    std_logic_vector (13 downto 0);
      dac1d        : out   std_logic_vector (13 downto 0);
      adc2d        : in    std_logic_vector (13 downto 0);
      dac2d        : out   std_logic_vector (13 downto 0);
      adc1sw       : out   std_logic_vector (3 downto 0);
      adc2sw       : out   std_logic_vector (3 downto 0);
      adc1clk      : out   std_logic;
      adc2clk      : out   std_logic;
      dac1clk      : out   std_logic;
      dac2clk      : out   std_logic;
      adc1of       : in    std_logic;
      adc2of       : in    std_logic;
      adc1shdn     : out   std_logic;
      adc2shdn     : out   std_logic;
      dac1slp      : out   std_logic;
      dac2slp      : out   std_logic;
      tp1_tio1     : out   std_logic;
      tp2_tio1     : out   std_logic;
      tp3_dev_clrn : in    std_logic;
      tp4_gclk0    : in    std_logic;
      tp5_gclk1    : in    std_logic;
      tp6_dev_oe   : in    std_logic;
      tp7_gclk3    : in    std_logic);
  end component;

  component fib_adcdac_app1_top_level
    generic (
      clk_freq_in_hz   : integer;
      firmware_id      : integer;
      firmware_version : integer);
    port (
      trig1_in                     : in    std_logic;
      trig2_out                    : out   std_logic;
      clk0                         : in    std_logic;
      hf_in                        : in    std_logic;
      uC_Link_D                    : inout std_logic_vector(7 downto 0);
      uC_Link_A                    : inout std_logic_vector(7 downto 0);
      Piggy_Clk1                   : out   std_logic;
      Piggy_RnW1                   : out   std_logic;
      Piggy_Strb1                  : out   std_logic;
      Piggy_Ack1                   : in    std_logic;
      uC_Link_DIR_D, uC_Link_DIR_A : out   std_logic;
```

```
nuC_Link_EN_CTRL_A          : out   std_logic;
uC_Link_EN_DA               : out   std_logic;
A2nSW8                      : in    std_logic;
A3nSW9                      : in    std_logic;
A0nSW10                     : in    std_logic;
A1nSW11                     : in    std_logic;
Sub_A6nSW12                 : in    std_logic;
Sub_A7nSW13                 : in    std_logic;
Sub_A4nSW14                 : in    std_logic;
Sub_A5nSW15                 : in    std_logic;
nResetnSW0                  : in    std_logic;
SW1                         : in    std_logic;
nDSnSW2                     : in    std_logic;
BClocknSW3                  : in    std_logic;
RnWnSW4                     : in    std_logic;
SW5                         : in    std_logic;
A4nSW6                      : in    std_logic;
SW7                         : in    std_logic;
NEWDATA                     : in    std_logic;
FC_Str                      : in    std_logic;
FC0                         : in    std_logic;
FC1                         : in    std_logic;
FC2                         : in    std_logic;
FC3                         : in    std_logic;
FC4                         : in    std_logic;
FC5                         : in    std_logic;
VG_A3nFC6                   : in    std_logic;
FC7                         : in    std_logic;
SD                          : in    std_logic;
nDRQ2                       : out   std_logic;
BBA_DIR                     : out   std_logic;
BBB_DIR                     : out   std_logic;
BBC_DIR                     : out   std_logic;
BBD_DIR                     : out   std_logic;
BBE_DIR                     : out   std_logic;
BBG_DIR                     : out   std_logic;
BBH_DIR                     : out   std_logic;
nBB_EN                      : out   std_logic;
DRDY                        : out   std_logic;
SRQ3                        : out   std_logic;
DRQ                         : out   std_logic;
INTERL                      : out   std_logic;
DTACK                       : out   std_logic;
nDRDY2                      : out   std_logic;
SEND_EN                     : out   std_logic;
SEND_STR                    : out   std_logic;
DSP_CRDY_W                  : out   std_logic;
DSP_CREQ_W                  : out   std_logic;
DSP_CACK_R                  : in    std_logic;
DSP_CSTR_R                  : in    std_logic;
DSP_D_R0                    : in    std_logic;
DSP_D_R1                    : in    std_logic;
DSP_D_R2                    : in    std_logic;
DSP_D_R3                    : in    std_logic;
DSP_D_R4                    : in    std_logic;
DSP_D_R5                    : in    std_logic;
DSP_D_R6                    : in    std_logic;
DSP_D_R7                    : in    std_logic;
DSP_CRDY_R                  : in    std_logic;
DSP_CREQ_R                  : in    std_logic;
DSP_CACK_W                  : out   std_logic;
DSP_CSTR_W                  : out   std_logic;
DSP_D_W0                    : out   std_logic;
```

```
      DSP_D_W1                    : out   std_logic;
      DSP_D_W2                    : out   std_logic;
      DSP_D_W3                    : out   std_logic;
      DSP_D_W4                    : out   std_logic;
      DSP_D_W5                    : out   std_logic;
      DSP_D_W6                    : out   std_logic;
      DSP_D_W7                    : out   std_logic;
      led1                        : out   std_logic;
      led2                        : out   std_logic;
      led3                        : out   std_logic;
      led4                        : out   std_logic;
      piggy_io                    : out   std_logic_vector(7 downto 0);
      VG_A4                       : in    std_logic;
      VG_A1                       : in    std_logic;
      DSP_DIR_D                   : out   std_logic;
      DSP_DIR_STRACK              : out   std_logic;
      DSP_DIR_REQRDY              : out   std_logic);
  end component;

  component driver_7416245
    port (
      iobus_a : inout std_logic_vector (15 downto 0);
      iobus_b : inout std_logic_vector (15 downto 0);
      dir_i   : in    std_logic);
  end component;

-- fab signals

  constant clk_divider_width_toplevel : integer := 16;
  constant reset_clks_toplevel        : integer := 2;

  signal fibclk_tb       : std_logic;
  signal fibd_tb         : std_logic_vector (15 downto 0);
  signal fiba_tb         : std_logic_vector (5 downto 0);
  signal fibrnw_tb       : std_logic;
  signal fibstrobe_tb    : std_logic;
  signal fiback_tb       : std_logic;
  signal adc1d_tb        : std_logic_vector (13 downto 0);
  signal dac1d_tb        : std_logic_vector (13 downto 0);
  signal adc2d_tb        : std_logic_vector (13 downto 0);
  signal dac2d_tb        : std_logic_vector (13 downto 0);
  signal adc1sw_tb       : std_logic_vector (3 downto 0);
  signal adc2sw_tb       : std_logic_vector (3 downto 0);
  signal adc1clk_tb      : std_logic;
  signal adc2clk_tb      : std_logic;
  signal dac1clk_tb      : std_logic;
  signal dac2clk_tb      : std_logic;
  signal adc1of_tb       : std_logic;
  signal adc2of_tb       : std_logic;
  signal adc1shdn_tb     : std_logic;
  signal adc2shdn_tb     : std_logic;
  signal dac1slp_tb      : std_logic;
  signal dac2slp_tb      : std_logic;
  signal tp1_tio1_tb     : std_logic;
  signal tp2_tio1_tb     : std_logic;
  signal tp3_dev_clrn_tb : std_logic;
  signal tp4_gclk0_tb    : std_logic;
  signal tp5_gclk1_tb    : std_logic;
  signal tp6_dev_oe_tb   : std_logic;
  signal tp7_gclk3_tb    : std_logic;

  -- fib signals
```

```
constant clk_freq_in_hz   : integer := 50000000;
constant firmware_id      : integer := 1;
constant firmware_version : integer := 3;

signal trig1_in                     : std_logic;
signal trig2_out                    : std_logic;
signal clk0                         : std_logic;
signal hf_in                        : std_logic;
signal uC_Link_D_tb                 : std_logic_vector(7 downto 0);
signal uC_Link_A_tb                 : std_logic_vector(7 downto 0);
signal Piggy_Clk1_tb                : std_logic;
signal Piggy_RnW1_tb                : std_logic;
signal Piggy_Strb1_tb               : std_logic;
signal Piggy_Ack1_tb                : std_logic;
signal uC_Link_DIR_D, uC_Link_DIR_A : std_logic;
signal nuC_Link_EN_CTRL_A           : std_logic;
signal uC_Link_EN_DA                : std_logic;
signal A2nSW8                       : std_logic;
signal A3nSW9                       : std_logic;
signal A0nSW10                      : std_logic;
signal A1nSW11                      : std_logic;
signal Sub_A6nSW12                  : std_logic;
signal Sub_A7nSW13                  : std_logic;
signal Sub_A4nSW14                  : std_logic;
signal Sub_A5nSW15                  : std_logic;
signal nResetnSW0                   : std_logic;
signal SW1                          : std_logic;
signal nDSnSW2                      : std_logic;
signal BClocknSW3                   : std_logic;
signal RnWnSW4                      : std_logic;
signal SW5                          : std_logic;
signal A4nSW6                       : std_logic;
signal SW7                          : std_logic;
signal NEWDATA                      : std_logic;
signal FC_Str                       : std_logic;
signal FC0                          : std_logic;
signal FC1                          : std_logic;
signal FC2                          : std_logic;
signal FC3                          : std_logic;
signal FC4                          : std_logic;
signal FC5                          : std_logic;
signal VG_A3nFC6                    : std_logic;
signal FC7                          : std_logic;
signal SD                           : std_logic;
signal nDRQ2                        : std_logic;
signal BBA_DIR                      : std_logic;
signal BBB_DIR                      : std_logic;
signal BBC_DIR                      : std_logic;
signal BBD_DIR                      : std_logic;
signal BBE_DIR                      : std_logic;
signal BBG_DIR                      : std_logic;
signal BBH_DIR                      : std_logic;
signal nBB_EN                       : std_logic;
signal DRDY                         : std_logic;
signal SRQ3                         : std_logic;
signal DRQ                          : std_logic;
signal INTERL                       : std_logic;
signal DTACK                        : std_logic;
signal nDRDY2                       : std_logic;
signal SEND_EN                      : std_logic;
signal SEND_STR                     : std_logic;
signal DSP_CRDY_W                   : std_logic;
signal DSP_CREQ_W                   : std_logic;
```

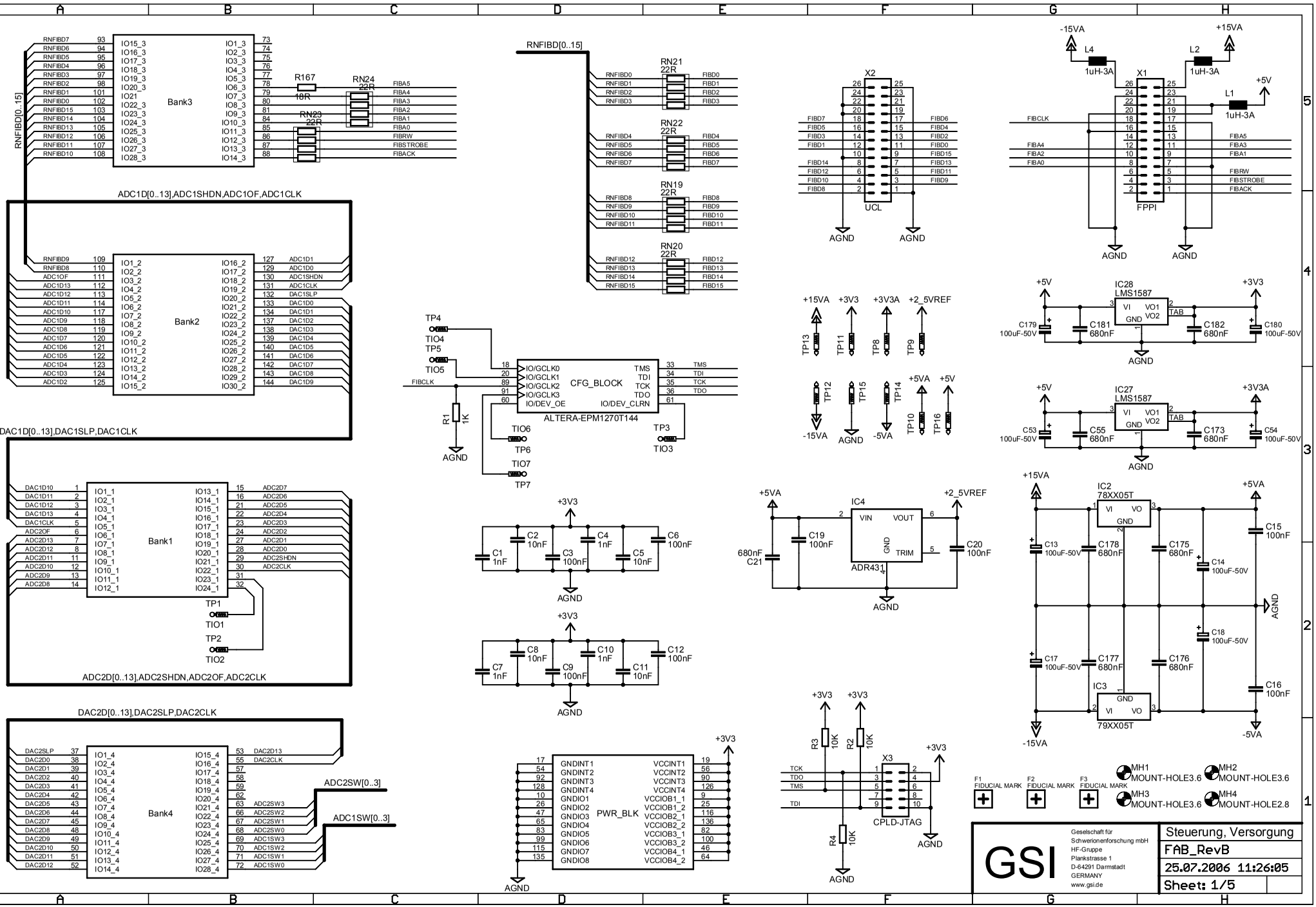
```
  signal DSP_CACK_R                  : std_logic;
  signal DSP_CSTR_R                  : std_logic;
  signal DSP_D_R0                    : std_logic;
  signal DSP_D_R1                    : std_logic;
  signal DSP_D_R2                    : std_logic;
  signal DSP_D_R3                    : std_logic;
  signal DSP_D_R4                    : std_logic;
  signal DSP_D_R5                    : std_logic;
  signal DSP_D_R6                    : std_logic;
  signal DSP_D_R7                    : std_logic;
  signal DSP_CRDY_R                  : std_logic;
  signal DSP_CREQ_R                  : std_logic;
  signal DSP_CACK_W                  : std_logic;
  signal DSP_CSTR_W                  : std_logic;
  signal DSP_D_W0                    : std_logic;
  signal DSP_D_W1                    : std_logic;
  signal DSP_D_W2                    : std_logic;
  signal DSP_D_W3                    : std_logic;
  signal DSP_D_W4                    : std_logic;
  signal DSP_D_W5                    : std_logic;
  signal DSP_D_W6                    : std_logic;
  signal DSP_D_W7                    : std_logic;
  signal led1                        : std_logic;
  signal led2                        : std_logic;
  signal led3                        : std_logic;
  signal led4                        : std_logic;
  signal piggy_io                    : std_logic_vector(7 downto 0);
  signal VG_A4                       : std_logic;
  signal VG_A1                       : std_logic;
  signal DSP_DIR_D                   : std_logic;
  signal DSP_DIR_STRACK              : std_logic;
  signal DSP_DIR_REQRDY              : std_logic;

  -- main clock
  signal sim_clk     : std_logic := '1';
  signal sim_io_bus  : std_logic_vector (15 downto 0);
  signal sim_io_bus2 : std_logic_vector (15 downto 0);



begin  -- mixed_tb_arch

  fab_adcdac_app1_top_level_inst : fab_adcdac_app1_top_level
    generic map (
      clk_divider_width_toplevel => clk_divider_width_toplevel,
      reset_clks_toplevel        => reset_clks_toplevel)
    port map (
      fibclk       => fibclk_tb,
      fibd         => fibd_tb,
      fiba         => fiba_tb,
      fibrnw       => fibrnw_tb,
      fibstrobe    => fibstrobe_tb,
      fiback       => fiback_tb,
      adc1d        => adc1d_tb,
      dac1d        => dac1d_tb,
      adc2d        => adc2d_tb,
      dac2d        => dac2d_tb,
      adc1sw       => adc1sw_tb,
      adc2sw       => adc2sw_tb,
      adc1clk      => adc1clk_tb,
      adc2clk      => adc2clk_tb,
      dac1clk      => dac1clk_tb,
      dac2clk      => dac2clk_tb,
```

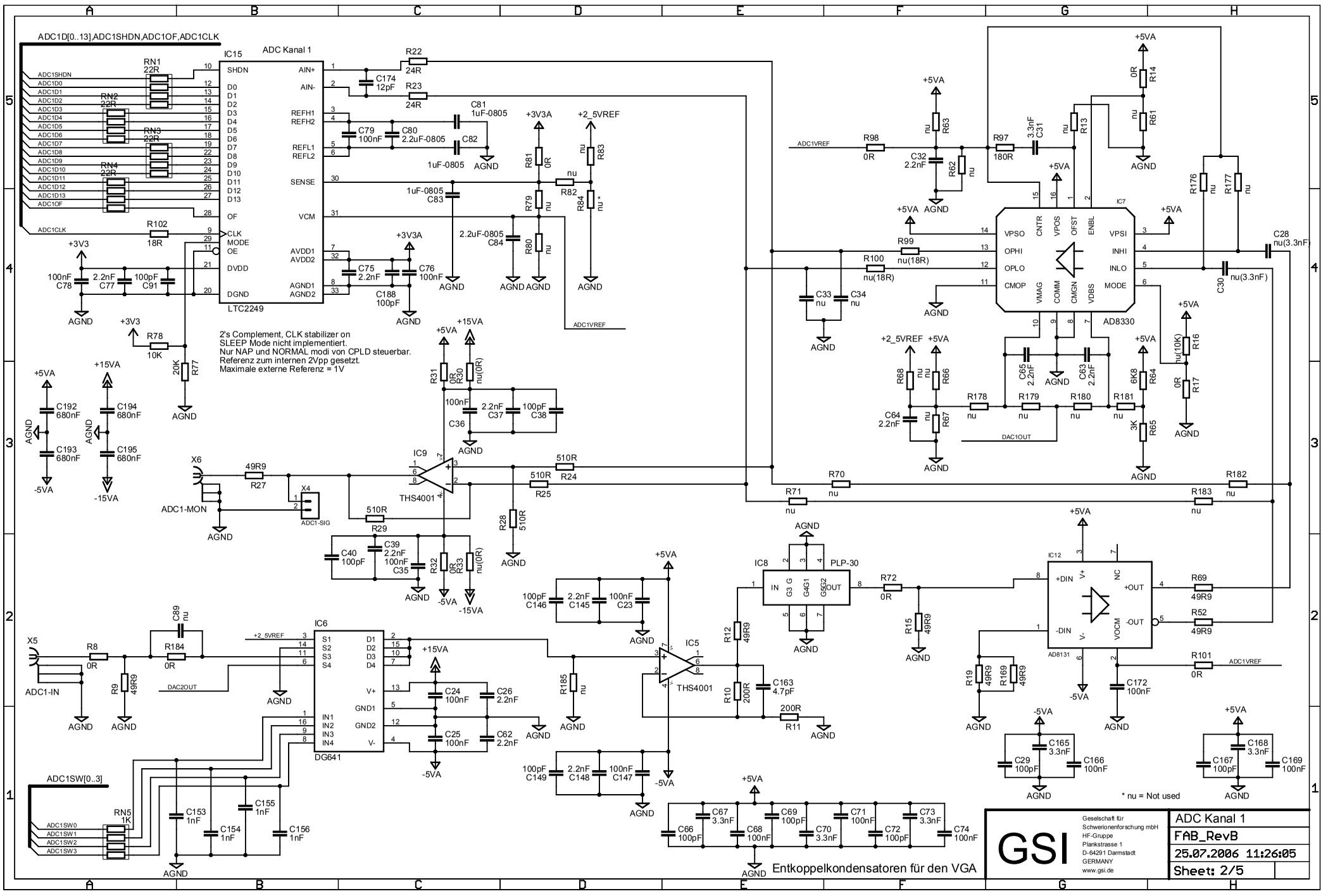
```
    adc1of       => adc1of_tb,
    adc2of       => adc2of_tb,
    adc1shdn     => adc1shdn_tb,
    adc2shdn     => adc2shdn_tb,
    dac1slp      => dac1slp_tb,
    dac2slp      => dac2slp_tb,
    tp1_tio1     => tp1_tio1_tb,
    tp2_tio1     => tp2_tio1_tb,
    tp3_dev_clrn => tp3_dev_clrn_tb,
    tp4_gclk0    => tp4_gclk0_tb,
    tp5_gclk1    => tp5_gclk1_tb,
    tp6_dev_oe   => tp6_dev_oe_tb,
    tp7_gclk3    => tp7_gclk3_tb);

fib_adcdac_app1_top_level_inst : fib_adcdac_app1_top_level
  generic map (
    clk_freq_in_hz   => clk_freq_in_hz,
    firmware_id      => firmware_id,
    firmware_version => firmware_version)
  port map (
    trig1_in           => trig1_in,
    trig2_out          => trig2_out,
    clk0               => clk0,
    hf_in              => hf_in,
    uC_Link_D          => uC_Link_D_tb,
    uC_Link_A          => uC_Link_A_tb,
    Piggy_Clk1         => Piggy_Clk1_tb,
    Piggy_RnW1         => Piggy_RnW1_tb,
    Piggy_Strb1        => Piggy_Strb1_tb,
    Piggy_Ack1         => Piggy_Ack1_tb,
    uC_Link_DIR_D      => uC_Link_DIR_D,
    uC_Link_DIR_A      => uC_Link_DIR_A,
    nuC_Link_EN_CTRL_A => nuC_Link_EN_CTRL_A,
    uC_Link_EN_DA      => uC_Link_EN_DA,
    A2nSW8             => A2nSW8,
    A3nSW9             => A3nSW9,
    A0nSW10            => A0nSW10,
    A1nSW11            => A1nSW11,
    Sub_A6nSW12        => Sub_A6nSW12,
    Sub_A7nSW13        => Sub_A7nSW13,
    Sub_A4nSW14        => Sub_A4nSW14,
    Sub_A5nSW15        => Sub_A5nSW15,
    nResetnSW0         => nResetnSW0,
    SW1                => SW1,
    nDSnSW2            => nDSnSW2,
    BClocknSW3         => BClocknSW3,
    RnWnSW4            => RnWnSW4,
    SW5                => SW5,
    A4nSW6             => A4nSW6,
    SW7                => SW7,
    NEWDATA            => NEWDATA,
    FC_Str             => FC_Str,
    FC0                => FC0,
    FC1                => FC1,
    FC2                => FC2,
    FC3                => FC3,
    FC4                => FC4,
    FC5                => FC5,
    VG_A3nFC6          => VG_A3nFC6,
    FC7                => FC7,
    SD                 => SD,
    nDRQ2              => nDRQ2,
    BBA_DIR            => BBA_DIR,
```

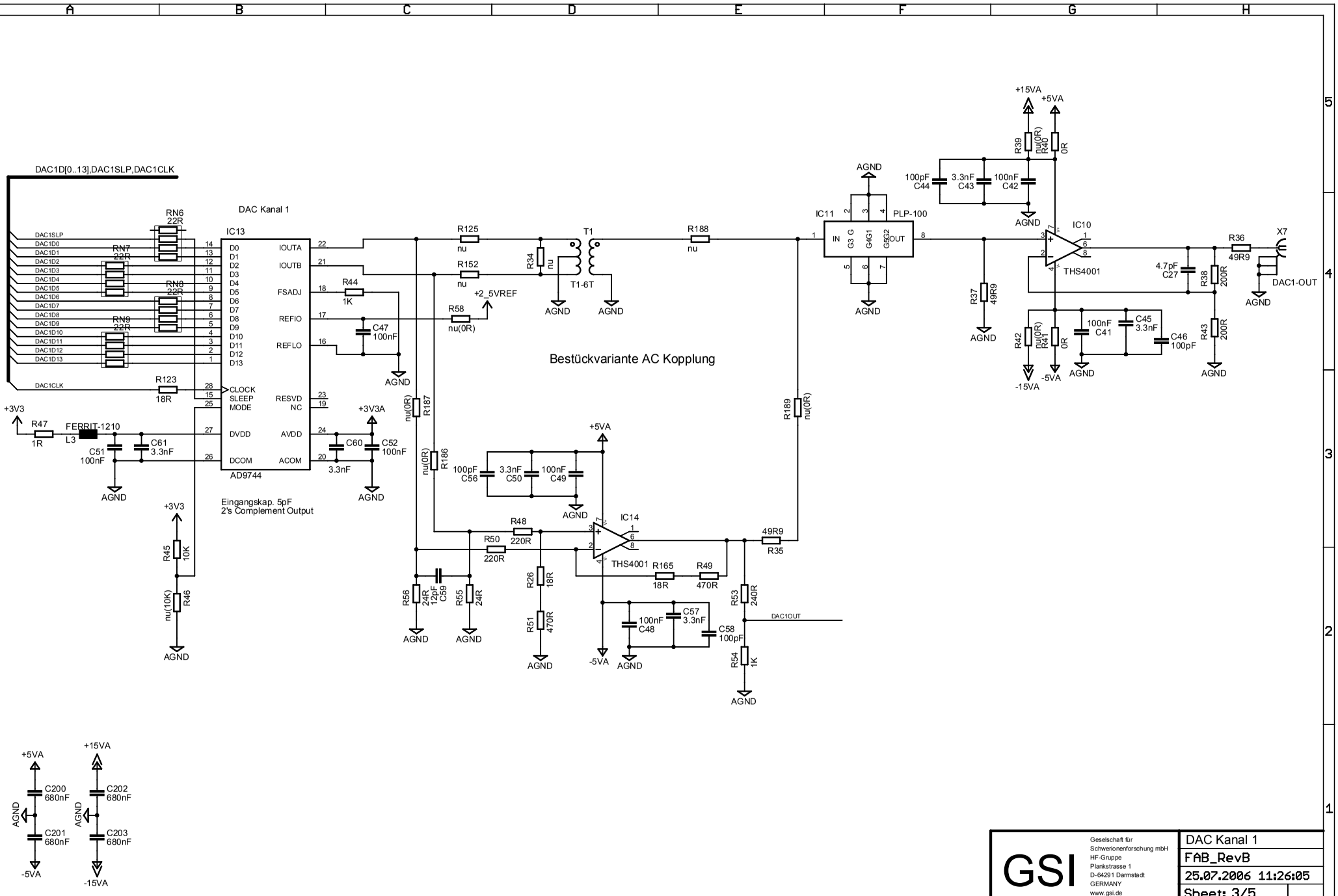
```
      BBB_DIR            => BBB_DIR,
      BBC_DIR            => BBC_DIR,
      BBD_DIR            => BBD_DIR,
      BBE_DIR            => BBE_DIR,
      BBG_DIR            => BBG_DIR,
      BBH_DIR            => BBH_DIR,
      nBB_EN             => nBB_EN,
      DRDY               => DRDY,
      SRQ3               => SRQ3,
      DRQ                => DRQ,
      INTERL             => INTERL,
      DTACK              => DTACK,
      nDRDY2             => nDRDY2,
      SEND_EN            => SEND_EN,
      SEND_STR           => SEND_STR,
      DSP_CRDY_W         => DSP_CRDY_W,
      DSP_CREQ_W         => DSP_CREQ_W,
      DSP_CACK_R         => DSP_CACK_R,
      DSP_CSTR_R         => DSP_CSTR_R,
      DSP_D_R0           => DSP_D_R0,
      DSP_D_R1           => DSP_D_R1,
      DSP_D_R2           => DSP_D_R2,
      DSP_D_R3           => DSP_D_R3,
      DSP_D_R4           => DSP_D_R4,
      DSP_D_R5           => DSP_D_R5,
      DSP_D_R6           => DSP_D_R6,
      DSP_D_R7           => DSP_D_R7,
      DSP_CRDY_R         => DSP_CRDY_R,
      DSP_CREQ_R         => DSP_CREQ_R,
      DSP_CACK_W         => DSP_CACK_W,
      DSP_CSTR_W         => DSP_CSTR_W,
      DSP_D_W0           => DSP_D_W0,
      DSP_D_W1           => DSP_D_W1,
      DSP_D_W2           => DSP_D_W2,
      DSP_D_W3           => DSP_D_W3,
      DSP_D_W4           => DSP_D_W4,
      DSP_D_W5           => DSP_D_W5,
      DSP_D_W6           => DSP_D_W6,
      DSP_D_W7           => DSP_D_W7,
      led1               => led1,
      led2               => led2,
      led3               => led3,
      led4               => led4,
      piggy_io           => piggy_io,
      VG_A4              => VG_A4,
      VG_A1              => VG_A1,
      DSP_DIR_D          => DSP_DIR_D,
      DSP_DIR_STRACK     => DSP_DIR_STRACK,
      DSP_DIR_REQRDY     => DSP_DIR_REQRDY);

  driver_7416245_1 : driver_7416245
    port map (
      iobus_a => sim_io_bus,
      iobus_b => sim_io_bus2,
      dir_i   => uC_Link_DIR_A);

  -- clock generation

  sim_clk <= not sim_clk after 10 ns;   -- 50 MHz simulation clock
  clk0    <= sim_clk;

--  component interconnections
```

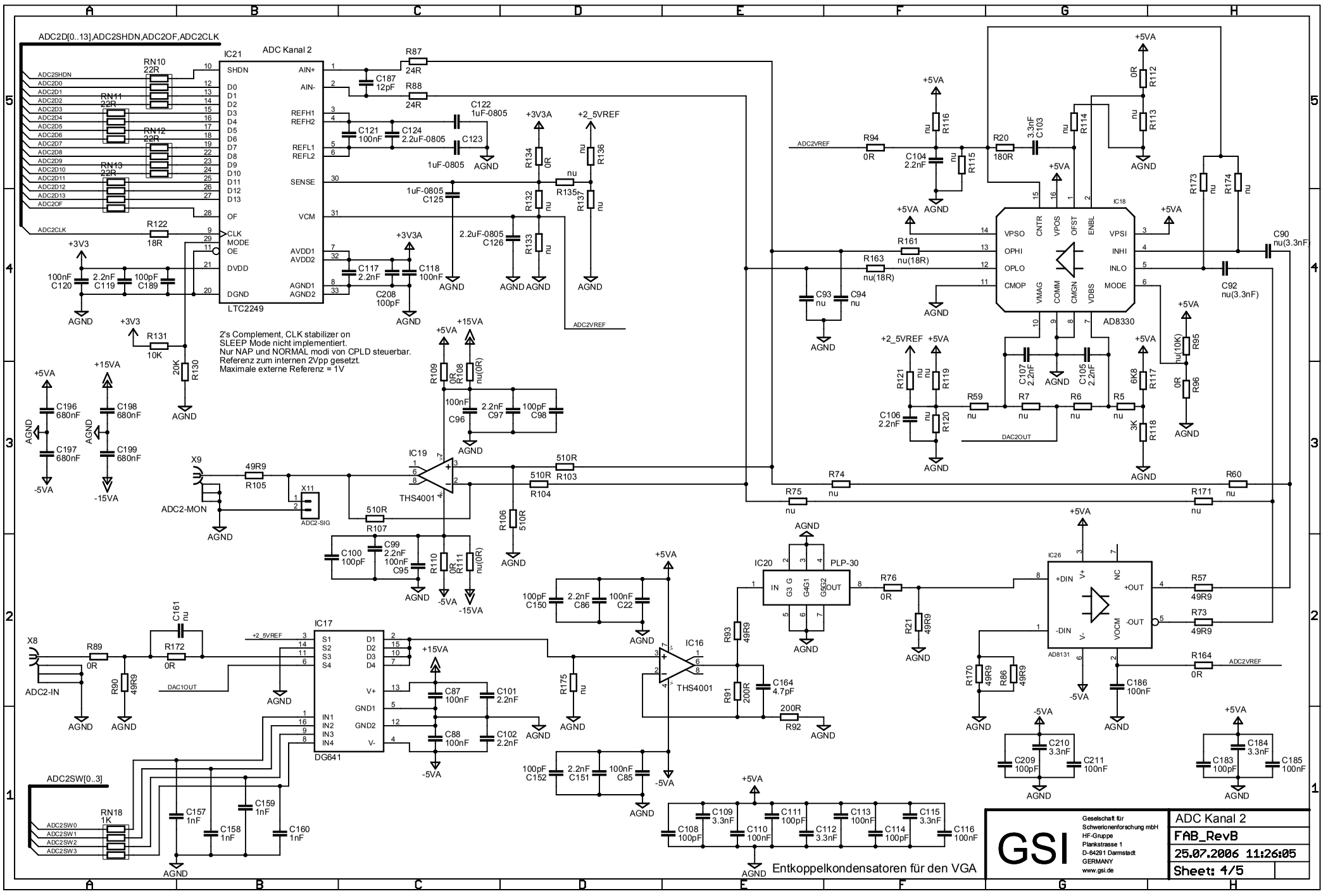
```
  fibclk_tb <= Piggy_Clk1_tb;

  Piggy_Ack1_tb <= fiback_tb;
  fibstrobe_tb  <= Piggy_Strb1_tb;
  fibrnw_tb     <= Piggy_RnW1_tb;

  fiba_tb <= piggy_io (5 downto 0);

  adc1d_tb <= "10101010101010", "00011111111000" after 400 ns, \
  "11111111111111" after 690 ns;

--  uC_Link_D_tb <= sim_io_bus (7 downto 0)  when Piggy_RnW1_tb = \
'1' else (others => 'Z');
--  uC_Link_A_tb <= sim_io_bus (15 downto 8) when Piggy_RnW1_tb = \
'1' else (others => 'Z');

--  sim_io_bus (7 downto 0)  <= uC_Link_D_tb when Piggy_RnW1_tb = \
'0' else (others => 'Z');
--  sim_io_bus (15 downto 8) <= uC_Link_A_tb when Piggy_RnW1_tb = \
'0' else (others => 'Z');


--  sim_io_bus <= fibd_tb    when Piggy_RnW1_tb = '1' else (others \
=> 'Z');
--  fibd_tb    <= sim_io_bus when Piggy_RnW1_tb = '0' else (others \
=> 'Z');

  uC_Link_D_tb <= sim_io_bus (7 downto 0)  when Piggy_RnW1_tb = '1' \
  else (others => 'Z');
  uC_Link_A_tb <= sim_io_bus (15 downto 8) when Piggy_RnW1_tb = '1' \
  else (others => 'Z');

  sim_io_bus (7 downto 0)  <= uC_Link_D_tb when Piggy_RnW1_tb = '0' \
  else (others => 'Z');
  sim_io_bus (15 downto 8) <= uC_Link_A_tb when Piggy_RnW1_tb = '0' \
  else (others => 'Z');


  sim_io_bus2 <= fibd_tb     when Piggy_RnW1_tb = '1' else (others \
  => 'Z');
  fibd_tb     <= sim_io_bus2 when Piggy_RnW1_tb = '0' else (others \
  => 'Z');


end mixed_tb_arch;

-------------------------------------------------------------
```

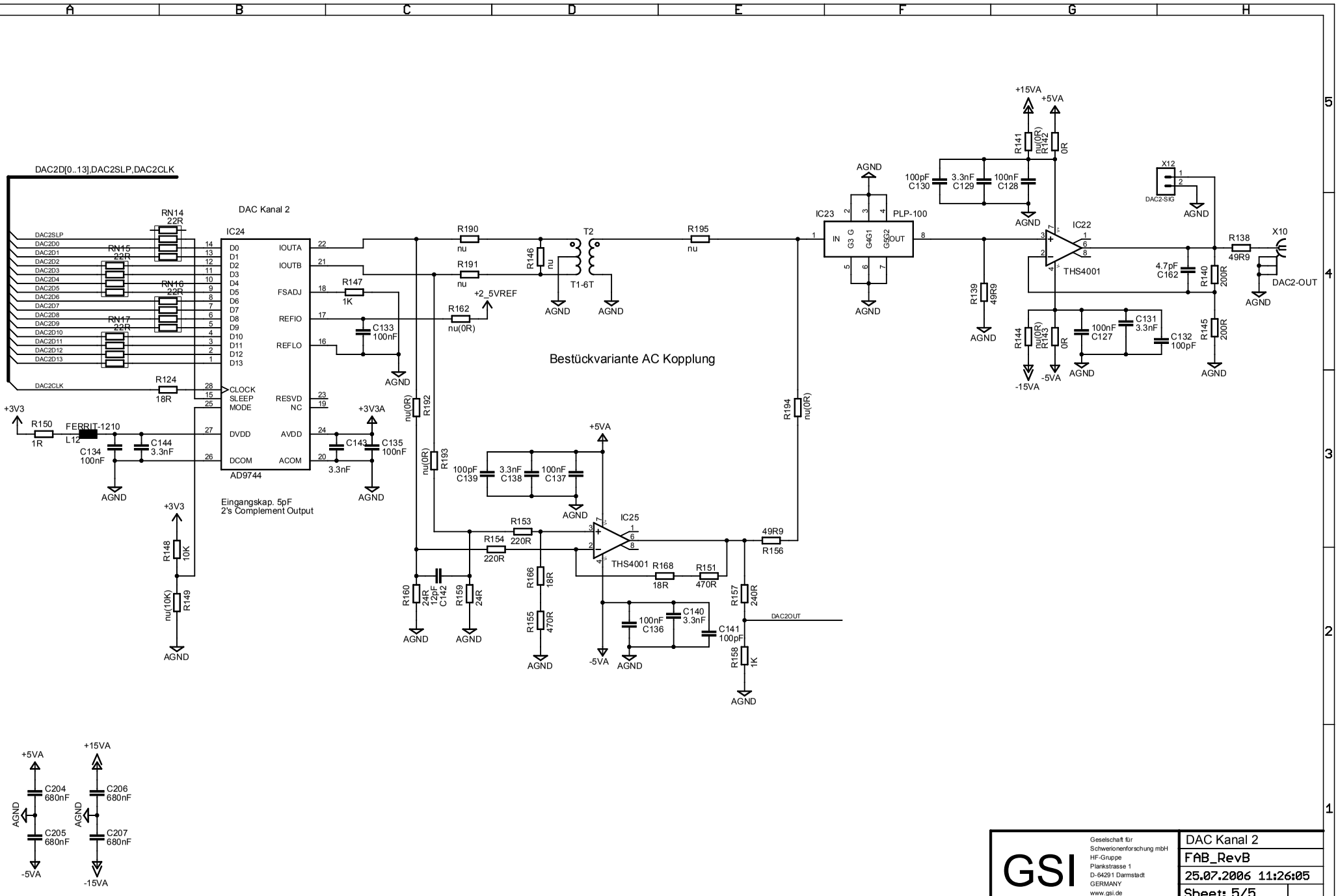
F Schemtatic Diagrams

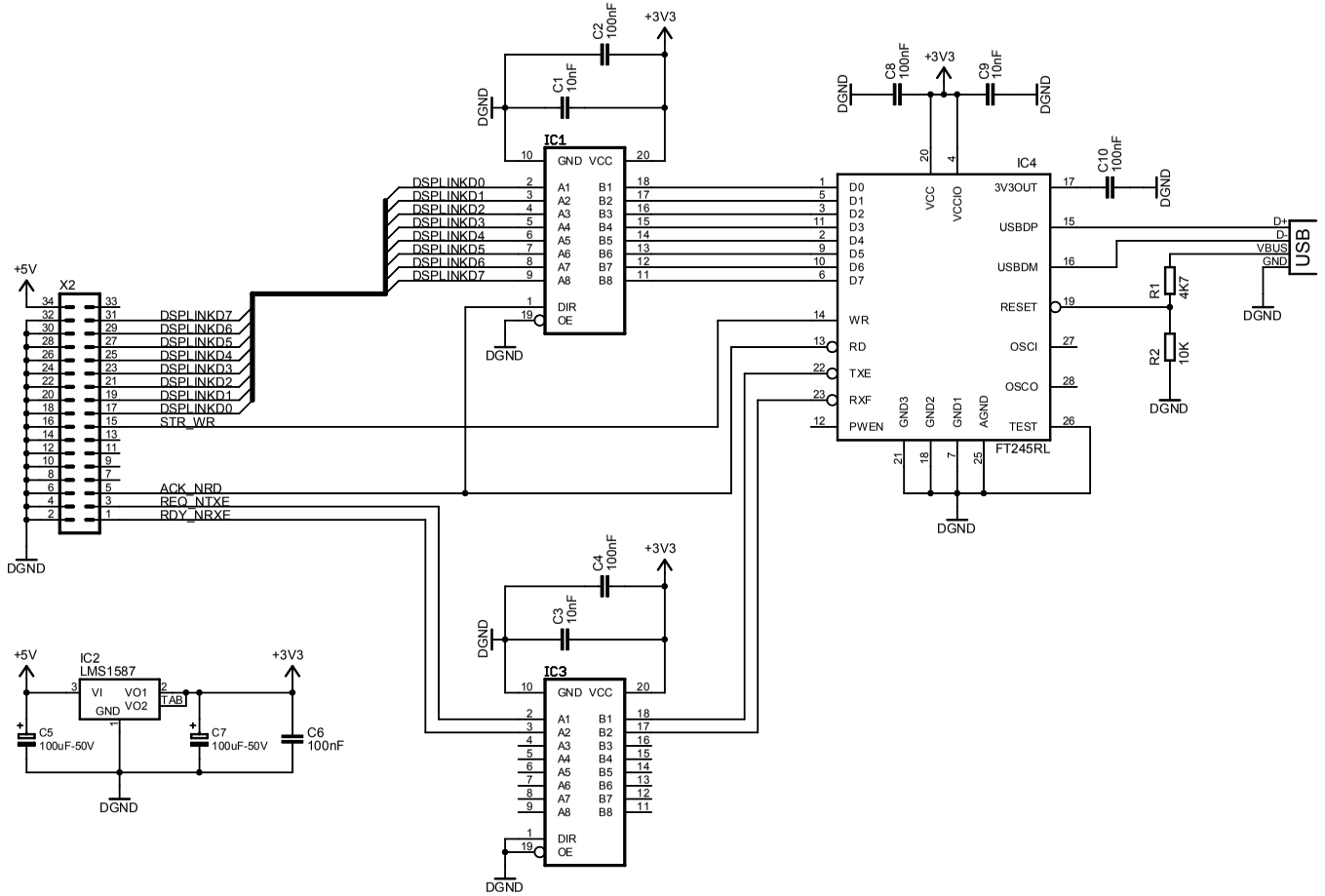

RNFIBD[0..15]
Bank3
R167
18R
RN24
22R
RN23
22R
FIBA5
FIBA4
FIBA3
FIBA2
FIBA1
FIBA0
FIBRW
FIBSTROBE
FIBACK
ADC1D[0..13],ADC1SHDN,ADC1OF,ADC1CLK
Bank2
ADC1D1
ADC1D0
ADC1SHDN
ADC1CLK
DAC1SLP
DAC1D0
DAC1D1
DAC1D2
DAC1D3
DAC1D4
DAC1D5
DAC1D6
DAC1D7
DAC1D8
DAC1D9
DAC1D[0..13],DAC1SLP,DAC1CLK
Bank1
ADC2D7
ADC2D6
ADC2D5
ADC2D4
ADC2D3
ADC2D2
ADC2D1
ADC2D0
ADC2SHDN
ADC2CLK
TP1
TIO1
TP2
TIO2
ADC2D[0..13],ADC2SHDN,ADC2OF,ADC2CLK
DAC2D[0..13],DAC2SLP,DAC2CLK
Bank4
DAC2D13
DAC2CLK
ADC2SW[0..3]
ADC1SW[0..3]
RN21
22R
RN22
22R
RN19
22R
RN20
22R
TP4
TIO4
TP5
TIO5
FIBCLK
R1
1K
CFG_BLOCK
ALTERA-EPM1270T144
TMS
TDI
TCK
TDO
TIO6
TP6
TIO7
TP7
TP3
TIO3
X2
UCL
X1
FPPI
L4
1uH-3A
L2
1uH-3A
L1
1uH-3A
-15VA
+15VA
+5V
AGND
TP13
TP11
TP8
TP9
TP12
TP15
TP14
TP10
TP16
+3V3
+3V3A
+2_5VREF
+5VA
-5VA
IC28
LMS1587
IC27
LMS1587
VI
VO1
VO2
GND
TAB
C179
100uF-50V
C181
680nF
C182
680nF
C180
100uF-50V
C53
C55
C173
C54
IC4
ADR431
VIN
VOUT
TRIM
C19
100nF
C20
100nF
C21
680nF
IC2
78XX05T
IC3
79XX05T
C15
100nF
C16
100nF
C13
100uF-50V
C14
100uF-50V
C17
C18
C175
680nF
C176
680nF
C177
680nF
C178
680nF
C1
1nF
C2
10nF
C3
100nF
C4
1nF
C5
10nF
C6
100nF
C7
1nF
C8
10nF
C9
100nF
C10
1nF
C11
10nF
C12
100nF
PWR_BLK
GNDINT1
GNDINT2
GNDINT3
GNDINT4
GNDIO1
GNDIO2
GNDIO3
GNDIO4
GNDIO5
GNDIO6
GNDIO7
GNDIO8
VCCINT1
VCCINT2
VCCINT3
VCCINT4
VCCIOB1_1
VCCIOB1_2
VCCIOB2_1
VCCIOB2_2
VCCIOB3_1
VCCIOB3_2
VCCIOB4_1
VCCIOB4_2
R3
10K
R2
10K
R4
10K
X3
CPLD-JTAG
TCK
TDO
TMS
TDI
MH1
MOUNT-HOLE3.6
MH2
MOUNT-HOLE3.6
MH3
MOUNT-HOLE3.6
MH4
MOUNT-HOLE2.8
F1
FIDUCIAL MARK
F2
FIDUCIAL MARK
F3
FIDUCIAL MARK
GSI
Gesellschaft für
Schwerionenforschung mbH
HF-Gruppe
Plankstrasse 1
D-64291 Darmstadt
GERMANY
www.gsi.de
Steuerung, Versorgung
FAB_RevB
25.07.2006 11:26:05
Sheet: 1/5


ADC1D[0..13],ADC1SHDN,ADC1OF,ADC1CLK
IC15
ADC Kanal 1
LTC2249
2's Complement, CLK stabilizer on
SLEEP Mode nicht implementiert.
Nur NAP und NORMAL modi von CPLD steuerbar.
Referenz zum internen 2Vpp gesetzt.
Maximale externe Referenz = 1V
IC7
AD8330
IC9
THS4001
IC8
PLP-30
IC12
AD8131
IC6
DG641
IC5
THS4001
ADC1-MON
ADC1-SIG
ADC1-IN
DAC1OUT
DAC2OUT
ADC1VREF
ADC1SW[0..3]
Entkoppelkondensatoren für den VGA
* nu = Not used
GSI
Gesellschaft für Schwerionenforschung mbH
HF-Gruppe
Plankstrasse 1
D-64291 Darmstadt
GERMANY
www.gsi.de
ADC Kanal 1
FAB_RevB
25.07.2006 11:26:05
Sheet: 2/5


DAC1D[0..13],DAC1SLP,DAC1CLK
DAC Kanal 1
IC13
AD9744
Eingangskap. 5pF
2's Complement Output
Bestückvariante AC Kopplung
IC11
PLP-100
IC10
THS4001
IC14
THS4001
T1
T1-6T
DAC1-OUT
DAC1OUT
+15VA
+5VA
-5VA
-15VA
+3V3
+3V3A
+2_5VREF
AGND
GSI
Gesellschaft für Schwerionenforschung mbH
HF-Gruppe
Plankstrasse 1
D-64291 Darmstadt
GERMANY
www.gsi.de
DAC Kanal 1
FAB_RevB
25.07.2006 11:26:05
Sheet: 3/5


ADC2D[0..13],ADC2SHDN,ADC2OF,ADC2CLK
IC21
ADC Kanal 2
LTC2249
2's Complement, CLK stabilizer on
SLEEP Mode nicht implementiert.
Nur NAP und NORMAL modi von CPLD steuerbar.
Referenz zum internen 2Vpp gesetzt.
Maximale externe Referenz = 1V
IC18
AD8330
IC19
IC20
PLP-30
IC16
THS4001
IC17
DG641
IC26
AD8131
ADC2-MON
ADC2-SIG
ADC2-IN
ADC2SW[0..3]
DAC1OUT
DAC2OUT
ADC2VREF
Entkoppelkondensatoren für den VGA
GSI
Gesellschaft für Schwerionenforschung mbH
HF-Gruppe
Plankstrasse 1
D-64291 Darmstadt
GERMANY
www.gsi.de
ADC Kanal 2
FAB_RevB
25.07.2006 11:26:05
Sheet: 4/5


DAC2D[0..13],DAC2SLP,DAC2CLK
DAC Kanal 2
IC24
AD9744
Eingangskap. 5pF
2's Complement Output
Bestückvariante AC Kopplung
IC23
PLP-100
IC22
THS4001
IC25
THS4001
T2
T1-6T
+2_5VREF
DAC2-SIG
DAC2-OUT
DAC2OUT
AGND
+3V3
+3V3A
+5VA
-5VA
+15VA
-15VA
GSI
Gesellschaft für Schwerionenforschung mbH
HF-Gruppe
Plankstrasse 1
D-64291 Darmstadt
GERMANY
www.gsi.de
DAC Kanal 2
FAB_RevB
25.07.2006 11:26:05
Sheet: 5/5



MH1 MOUNT-HOLE3.0
MH2 MOUNT-HOLE3.0
MH3 MOUNT-HOLE3.0
MH4 MOUNT-HOLE3.0

F1 FIDUCIAL MARK
F2 FIDUCIAL MARK

GSI
Gesellschaft für Schwerionenforschung mbH
HF-Gruppe
Plankstrasse 1
D-64291 Darmstadt
GERMANY
www.gsi.de

Systemübersicht	
UDL_RevA	
11.08.2006 17:16:37	
Sheet: 1/1	

G GERBER Plots