Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Valentin Stanev

Abtastratenumsetzung nach dem Direct-Down-Conversion Prinzip als FPGA-IP-Core für nachfolgende Auswertung durch Matlab

Fachhochschule Westküste
Fachbereich Technik

Fachhochschule Westküste - University of Applied Sciences
Faculty of Engineering

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Hamburg University of Applied Sciences
Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Valentin Stanev

Abtastratenumsetzung nach dem Direct-Down-Conversion Prinzip als FPGA-IP-Core für nachfolgende Auswertung durch Matlab

Masterarbeit eingereicht im Rahmen der Masterpüfung
im gemeinsamen Studiengang Mikroelektronische Systeme
am Fachbereich Technik
der Fachhochschule Westküste
und
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Jürgen Reichardt
Zweitgutachter : Prof. Dr.-Ing. Stephan Hußmann

Abgegeben am 02. Februar 2011

Valentin Stanev

Title of the master thesis

Sample rate conversion using the Direct-Down-Conversion principle as FPGA-IP-Core and Matlab evaluation of the results

Keywords

Software-Defined-Radio, Field Programmable Gate Array, Digital Signal Processing , Softcore Microprocessor, System-on-Programmable-Chip, USB, Microblaze

Abstract

The evolution of digital electronics allowed the realtime processing of radio frequencies and the implementation of demodulation circuits directly on a single chip and by thus creating flexible solutions for different digital communication problems. With the refinement of today's signal processing chips, such as Field Programmable Gate Arrays and Digital Signal Processors, topics like Software-Defined-Radio are not any more part of the theoretical discussion but can be directly implemented in digital hardware.

This paper discusses the implementation of the DDC algorithm, a well defined principle with respect to Radio Frequency signal recovery, as an Intellectual Property Core and its integration in a Microblaze softcore microprocessor-based embedded system responsible for the USB data transmission of the incoming data stream.

Valentin Stanev

Thema der Masterarbeit

Abtastratenumsetzung nach dem Direct-Down-Conversion Prinzip als FPGA-IP-Core für nachfolgende Auswertung durch Matlab

Stichworte

Software-Defined-Radio, Field Programmable Gate Array, Digitale Signalverarbeitung , Softcore Mikroprozessor, System-on-Programmable-Chip, USB, Microblaze

Kurzzusammenfassung

Die Entwicklung der digitalen Elektronik erlaubt die Echtzeit-Verarbeitung von Funkfrequenzen und die Umsetzung der Demodulationsschaltungen direkt auf einem einzigen Chip und schafft damit flexible Lösungen für unterschiedliche Probleme aus dem Bereich der digitalen Übertragungstechnik. Mit der Verfeinerung der heutigen Signalverarbeitungs-Chips, wie Field Programmable Gate Arrays und digitalen Signalprozessoren, Themen wie Software-Defined-Radio sind nicht mehr Teil der theoretischen Diskussion, sondern können direkt in digitaler Hardware umgesetzt werden.
Dieser Arbeit behandelt die Implementierung von dem DDC-Algorithmus, ein definiertes Verfahren in Bezug auf die RF Signalverarbeitung, als Intellectual Property Core und seine Integration in einem Microblaze softcore Mikroprozessor-basierte Embedded-Systems die für die USB-Datenübertragung von den eingehenden Datenstrom verantwortlich ist.

Acknowledgement

I wish to express my appreciation to Prof. Dr. Jürgen Reichardt for supervising my master thesis and giving me the opportunity to work on this impressive project. The provided support and motivation have been very rewarding for me and the valuable experience, earned during the development process, allowed me to extend my engineering knowledge. Also, I would like to thank Prof. Dr.Ing. Stephan Hußmanm for his engagement as second examiner and for the provided guidance during this thesis.

Additionally, I would like to express my gratitude to Professor Sauvagerd for providing me with valuable information in the field of signal processing and filter design.

Special thanks go to Dipl.-Ing. J. Neugebauer, Dipl.-Ing. G. Volkmann and Mr. G. Wolff for their contribution and engagement during the test phase of the project.

Last but not least, I would like to thank my parents for the constant support and inspiration, without which I could have never made it. Also, I am very thankful to my girlfriend for motivating me throughout the hard moments I experienced with this thesis. Finally, I thank all my friends for their faith and support.

Hamburg, February 2011 **Valentin Stanev**

Table of Contents

List of Figures

Index of Tables

List of Acronyms

ADC	Analog-to-Digital-Converter	**SNR**	Signal-to-Noise Ratio
DAC	Digital-to-Analog-Converter	**SINAD**	Signal-to-Noise-and-Distortion
DDC	Direct Down Conversion	**SFDR**	Spurious Free Dynamic Range
IQ	In Phase/Quadrature	**FFT**	Fast Fourier Transform
FPGA	Field Programmable Gate Array	**AAF**	Anti-Aliasing Filter
DSP	Digital Signal Processor	**IP**	Intelectual Property
SDR	Software Defined Radio	**CPU**	Centrla Processing Unit
USB	Universal Serial Bus	**ISR**	Interrupt Service Routine
BBB	Bulk Only	**NPI**	Native Port Interface
FIR	Finite Impulse Response	**GPIO**	General Purpose IO
IIR	Infinite Impulse Response	**SIE**	Serial Interface Engine
CIC	Cascaded Integrator Comb	**DMA**	Direct Memory Access
VLIW	Very Long Instruction Word	**DDS**	Direct Digital Synthesizer
MPMC	MultiPort Memory Controller	**NCO**	Numerically Controlled Oscillator
XCL	Xilinx CacheLink	**VCO**	Voltage Controlled Oscillator
FAT	File Allocation Table	**LUT**	Look Up Table
PLB	Processor Local Bus	**GUI**	Graphical User Interface
SPLB	Slave PLB Interface	**MPD**	Microprocessor Peripheral Definition
MPLB	Master PLB Interface	**MHS**	Microprocessor HW Specification
UTMI	Universal Transceiver Macrocell IF	**PAR**	Place And Route
ULPI	UTMI + Low Pin Interface	**USRP**	Universal Software Radio Peripheral
RTL	Register Transfer Level	**MIPS**	Million Instructions Per Second
UCF	User Constraint File	**MSS**	Mass Storage Support
SoPC	System-on-Programmable-Chip		

1 Introduction

During the last decades the world of digital electronics and mobile communications has undergone a massive evolution. Not only is nearly every device of today's everyday life controlled by a simple microprocessor, but even more and more standards have been shifted to the digital domain. One such example is the conversion of the Television Broadcasting system to Digital Video Broadcasting Terrestrial(DVB-T) for the whole European Union by the end of 2012 which will leave Analog Television Broadcasting in history. Digital Broadcasting will eventually ensure better signal quality and bring additional benefits with it. Furthermore, more and more people use the Internet to listen to radio or watch television. One topic which gains popularity very fast is the concept of **S**oftware-**D**efined-**R**adio(**SDR**).

A SDR system is a radio communication system where the processing part is shifted to a personal computer or embedded system. Different hardware components, such as mixers, filters and modulators, are integrated either in software or by means of hardware synthesis. The rapidly evolving capabilities of digital electronics render practically many processes which used to be only theoretically possible[1]. In order to retrieve the signal, different algorithms may be adopted to suite the project specifications. One way of recovering the **R**adio **F**requency(**RF**) signal in baseband is the so called **D**irect-**D**own-**C**onversion(**DDC**) principle. This algorithm performs the three main signal processing operations required to retrieve the signal in baseband, namely down-mixing, filtering and decimation. In particular, the incoming signal will be shifted to baseband so that an optimal sampling rate reduction can be adopted. This is regarded as crucial, because of the fact that today's mobile devices are battery operated and by reducing the sampling frequency improved battery life and longer recharge intervals may be guaranteed. Once the data is recovered , there are numerous ways of processing it. In the case of real-time algorithms immediate refinement is desired and the data must be available in a given time frame. If, on the other side, the data is offloaded to a special storage then the evaluation may be done at any time and the stream can be reproduced without any timing constraints.

1 Reference: http://en.wikipedia.org/wiki/Software-defined_radio [14.11.2011]

1.1 The aim of this project

The aim of this project is to create a system which allows efficient RF signal recovery. It should provide the basic communication link between the analog and digital domain as well as establish a transmission layer between the evaluation platform and analysis software. The hardware implementation of the Direct-Down-Conversion algorithm together with the construction of a softcore microprocessor environment are the two main topics in the specification of this project. Likewise, the software design of the firmware and the analysis application play a vital role in the signal quality and the achieved transmission rates. The main objective will be to setup the hardware configuration and perform initial data transfers using simplified measurement conditions. The failure-free low level communication between the different components as well as the lossless data transfer must be assured. Finally, an off-board Matlab analysis should aid the evaluation of the received data stream and will serve as the main criteria in the measurement quality estimation. This approach reduces the design time because the complete test algorithm must not be developed and tested. Moreover, the fact that the output sequence is permanently stored, allows the execution of extensive evaluation tests.

1.2 Outline of this paper

The practical activity of this master thesis consists of the development and design of a FPGA-based softcore microprocessor embedded system with respect to the outlined requirements and criteria. The following work-flow will serve as a reference and lists the main topics to be covered throughout the work:

- ➔ **Theoretical analysis of the Direct-Down-Conversion method**

  This part will touch the main theory required for the Direct-Down-Conversion algorithm. Also, Chapter 3 focuses on Analog-to-Digital conversion topic.

- ➔ **Hardware platform evaluation and selection**

  Chapter 4.1 brings up a short overview of the used hardware platforms.

➔ **Design concept of the project**

Chapter 2 will present the design concept to be implemented. All decisions done in this part have the purpose to optimize the proposed solution and minimize the hardware consumption.

➔ **Hardware implementation of the softcore microprocessor system**

The main topic in this part will be the design of a **I**ntellectual **P**roperty(**IP**) Core, which encapsulates the complete DDC algorithm and may be used as a standalone system. Also, the complete embedded system, created during this master thesis, is described in Chapter 4.

➔ **Software development of the firmware and analysis application**

The software development process focuses on the USB firmware extension as well as the design of the data acquisition application. Chapter 5 handles also the Matlab analysis script and the synchronization between the FPGA and the desktop PC.

➔ **Evaluation of the results**

As a conclusion, diverse test measurement conditions should prove the functionality of the embedded system and serve as an estimate for the success of the design. The results are listed in Chapter 6.

2 Concept

This chapter will make the reader familiar with the design concept of this master thesis. Additionally, a summary of the current state of the technology, concerning the Software-Defined-Radio topic, will be presented and the relationship between this project and the currently available solutions will be done.

2.1 Software Defined Radio

As digital hardware is keeping up with Moore's Law[2] the term Software-Defined-Radio is drawing more and more attention. By increasing the maximum achievable operating frequencies as well as memory capacities of today's embedded systems, the idea of performing the complete RF signal recovery in the digital domain is adopted much faster that one would expect. The flexibility of digital hardware allows the implementation of functions, which were kept in the analog world for a long time. As chip prices are going down and semiconductor technologies improve, a digital platform would be the cost-efficient choice for the market. Not only allow **F**ield **P**rogrammable **G**ate **A**rrays(**FPGA**s) and **D**igital **S**ignal **P**rocessors(**DSP**s) high data processing throughput, but also **A**nalog-to-**D**igital **C**onverter(**ADC**) chips ensure the fluent transition between the analog and digital domain. Most of the disadvantages of analog components are not present when compared with their digital counterparts, which is another argument for considering this kind of processing. Once an analog signal is converted by the ADC, the designer is free to setup the optimal accuracy for the arithmetic operations and storage elements of the system. Numerous sophisticated tools allow the creation of simulation models which reduce the chance of error propagation in the design phase as well as provide evaluation data for further research topics. This reduces the time-to-market issue which could be a vital factor in the product's success.

2 For more information on the topic please visit: http://en.wikipedia.org/wiki/Moore%27s_law

2.2 DSPs and FPGAs compared

Digital signal processing algorithms typically require a large number of mathematical operations to be performed quickly and repetitively on a set of data. To perform this in real-time, all calculations should be done in a given amount of cycles. That is why all DSP applications have constraints on latency. Due to this issue, the main difference between a general purpose processor and a DSP is the hardware architecture chosen. While a RISC microprocessor, such as the PowerPC 750, achieves 525[3] **M**illion **I**nstructions **P**er **S**econd(**MIPS**) at 233 MHz , a TMS320C6713 DSP from Texas Instruments is capable of doing 1800[4] MIPS at 225 MHz. This optimization effort boosts the calculation capacity of the DSP and allows the implementation of complex mathematical expressions. This approach requires the implementation of the Harvard Architecture[5], which separates code and memory space. This way the CPU can access instruction and data memory simultaneously. The so called **V**ery **L**ong **I**nstruction **W**ord(**VLIW**) instruction set provides the possibility of driving multiple arithmetic cores at the same time, see Figure 2.1[6].

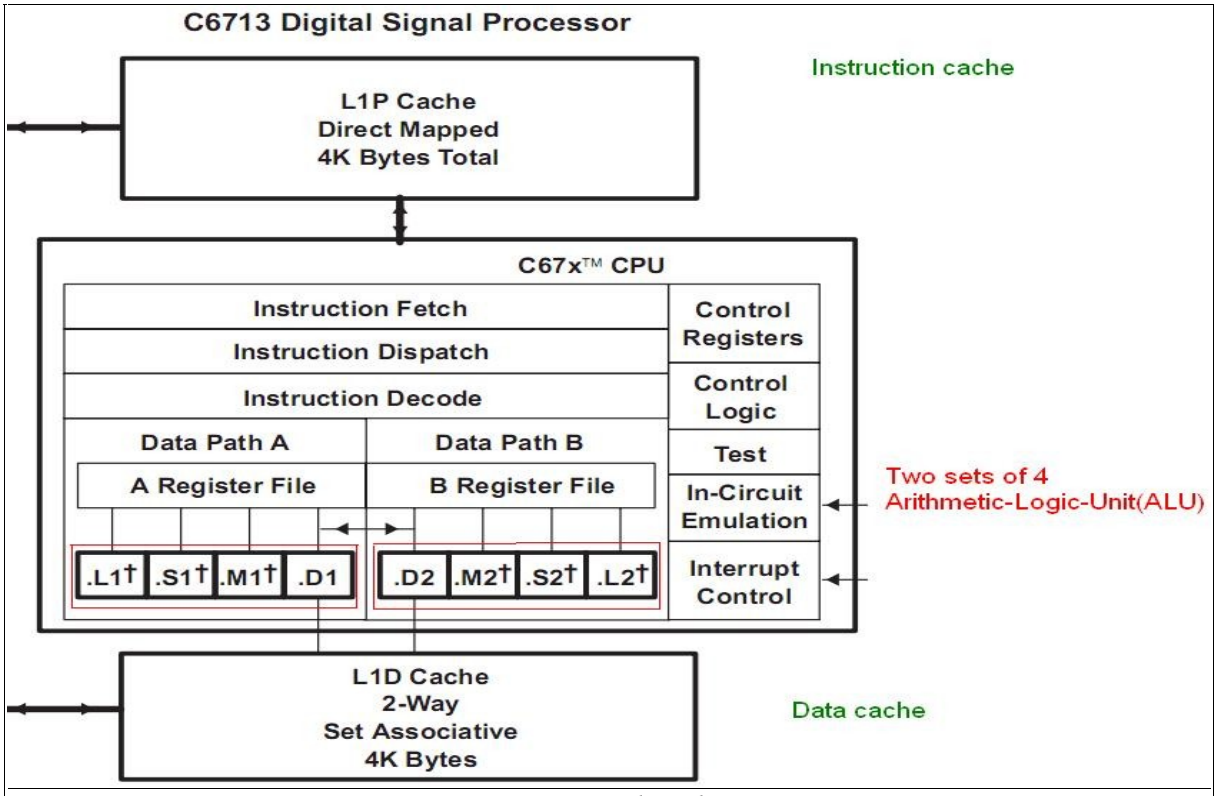


Figure 2.1: C6713 CPU overview, ALU units and cache memories

3 Information source: http://en.wikipedia.org/wiki/Instructions_per_second, [24.01.2011]
4 Information source: http://focus.ti.com/docs/toolsw/folders/print/tmdsdsk6713.html [24.01.2011]
5 Wikipedia, http://en.wikipedia.org/wiki/Harvard_architecture, [20.12.2010]
6 Image reference: [DSK]

This way it is possible to concurrently multiply two pairs of numbers, load new data from memory and evaluate different flags. Addressing modes are also a feature which can bring benefits to the system. By using "Modulo Addressing", it is possible to implement circular buffers without check condition for the wrap around case.

These benefits provide enhanced arithmetic capabilities for the DSP, but it is often up to the compiler support, which is responsible for optimal translation of the source code to the machine level of execution. Often the generated code overhead requires the hand optimization of the generated assembly instructions to provide better timing. This demands that the developer is familiar with the design environment and the DSP architecture used. Writing the machine code by hand would require more time than developing a C code application, but if compiler optimization does not achieve the desired timing results, it is most probably the only way of extracting better results.

Alternatively, there is another type of signal processing chips which compete with DSPs. A Field Programmable Gate Array(FPGA) is an integrated circuit, designed to be configured by modifying the available internal components. The elements of the FPGA are normally inferred using Hardware Description Languages. The description is then translated into logic components such as memories, registers, embedded multipliers and the like. As seen on Figure 2.2[7], FPGAs consists of logic blocks, which provide certain functionality.

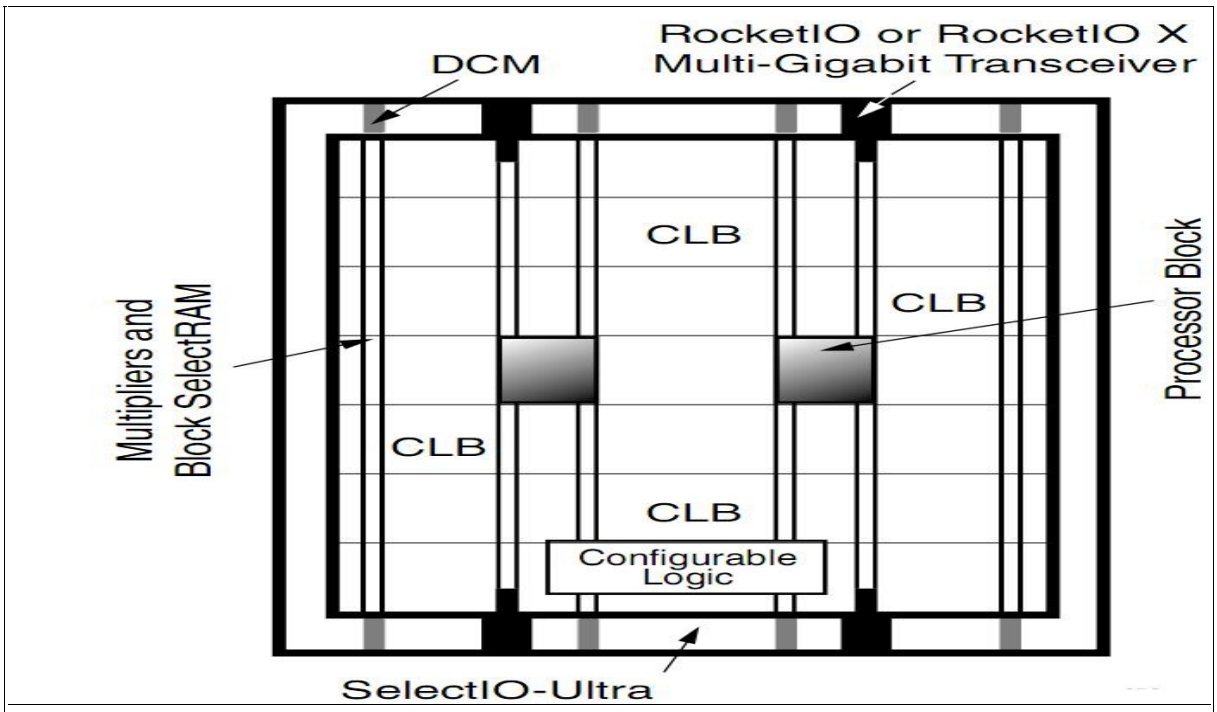


Figure 2.2: Virtex II architecture overview

7 Image reference: [DS083]

By interconnecting the logic cells it is possible to perform complex combinatorial functions and implement a vast majority of algorithms. Furthermore, the evolution of the so called DSP48E components, depicted on Figure 2.3[8], have provided fast arithmetic power and is the main advantage of FPGAs over DSPs in terms of signal processing. Each DSP48E slice contains a 25x18 multiplier as well as an adder and an accumulator.

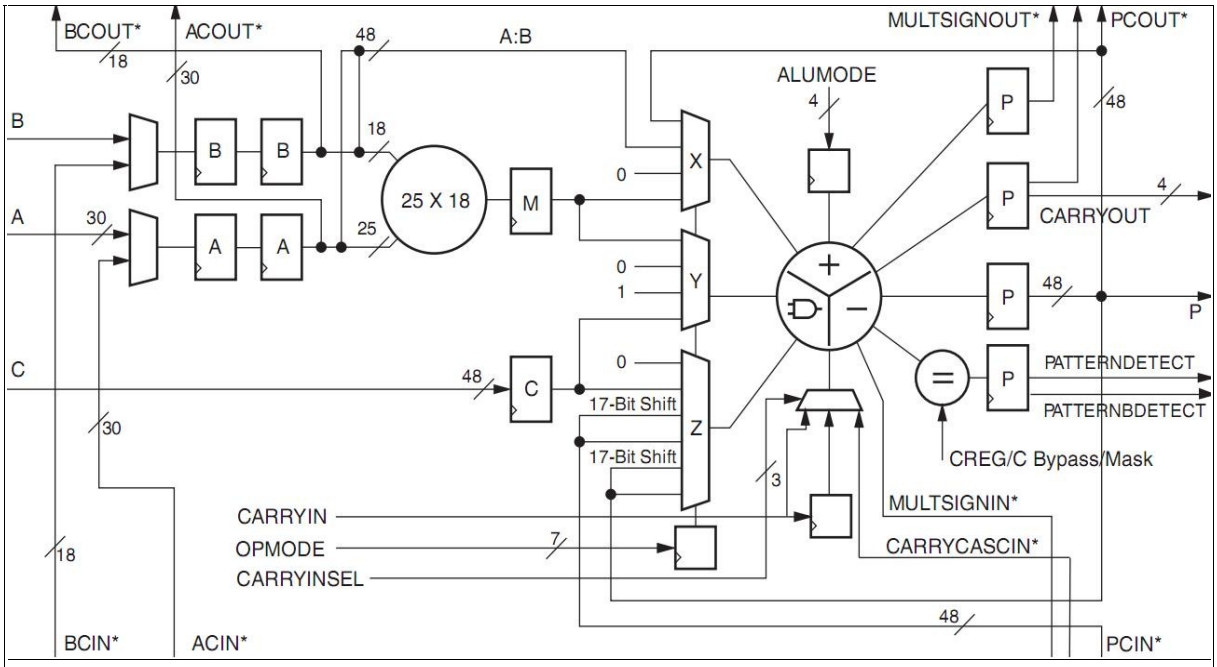


Figure 2.3: DSP48E slice architecture overview

While a DSP can be clocked faster, its structure still has only a limited number of multipliers. Each device of the Virtex5 FPGA family, on the other side, contains at least 24[9] of these block elements which results in at maximum 24 parallel multiplications. This flexibility makes FPGAs suitable for nearly any kind of problems, especially in the field of signal processing. On the other hand, the development process, involving both hardware and software design, is much more complex with respect to other architectures. While a DSP comes with a fixed hardware architecture, in the case of an FPGA design, it is the developer, who defines the structure and evaluates the interfaces to be implemented. An optimal FPGA design would consume only a certain amount of the available resources by leaving the rest for another application. By using techniques such as Resource and Register Sharing it is possible to pipeline a design and this way one could split a number of calculations in several clock

8 Image reference: Chapter 1 of [DS193]
9 A complete list of the hardware resources is available in Table 1 of [DS100]

cycles. That is why a FPGA solution is always a tradeoff between Speed and Area requirements. In order to tackle this problem, development tools, such as Matlab/Simulink, Xilinx ISE and System Generator, have emerged to provide optimal implementation results. The supplied vendor-specific libraries ensure that no hardware resources are wasted which permits the integration of more logic on a single chip.

2.3 The Current State

Although the main purpose of a SDR system is to recover a RF signal, different methodes can be employed to achieve this. For this reason, a number of projects exist, which provide flexible solutions and can serve as a reference during the design phase of this project. It is obvious that the implementation of a complete SDR solution would expect knowledge in the field of hardware and software development as well as layout design and verification. With this in mind, it is clear that proprietary solutions, such as the those of “Software Radio Laboratory LLC” and “GE Intelligent Platforms”[10], would provide remarkable results at the cost of a higher price. The former company offers the “SRL QuickSilver QS1R Receiver”, an advanced direct sampling receiver which features a LTC2208 16-bit ADC as well as an Altera EP3C25 Cyclone III FPGA. Connectivity to the PC is done through a USB 2.0 interface and the spectral range, covered by the board is from 15 KHz up to 62 MHz in its standard configuration. Furthermore, if the so called "undersampling" method is selected then frequencies up to 500 MHz can be processed. Additionally, a SDRMAX Software is supplied for data analysis and visualization[11].

The “GE Intelligent Platforms”, on the other side, provides a dozen of solutions for specific applications which include both FPGA and DSP components, responsible for handling the high data throughput. All boards are designed for multi-channel processing and include a number of Analog-to-Digital converters. The processed data is transmitted using high-speed serial I/O and a sample rate conversion using the Direct-Down-Conversion algorithm is performed[12].

Besides those two products, the “LYRTECH SFF SDR Development board” is also worth

10 Source: http://www.designspark.com/content/software-defined-radio-sdr [21.01.2011]
11 Furhter information on the product: http://www.srl-llc.com/ [21.01.2011]
12 For a list of the "GE Intelligent Platforms" products visit: http://www.ge-ip.com/products/family/software-defined-radio [21.01.2011]

mentioning. This complex device integrates a DSP, FPGA and a general-purpose-processor which makes it versatile solution for nearly any signal processing task. An 10/100 Mbps Ethernet interface is used for the data transmission. Additionally, the SFF SDR evaluation module comes with a stereo codec and various power management capabilities as well as Simulink model-based design environment for fast design and evaluation[13].

In comparison to the proprietary solutions, more and more open-source projects, focusing on the SDR topic, gain popularity. One such approach is the "Cool USB Radio" coming from "Commgenuity Inc"[14]. The design, shown on Figure 2.4[15], makes use of a Altera Cyclone II FPGA, which performs the digital Up- and Down-conversion, whereas a Cypress FX2LP USB 2.0 chip is used to transmit the digital information between the board and a desktop computer. Both 10-bit AD and 12-bit DA converters are provided and run at 105 and 210 MHz respectively. Moreover, both analog inputs are provided with user-configurable 7-pole filters for band selection. The software development kit comes with a Windows XP GUI for data analysis.

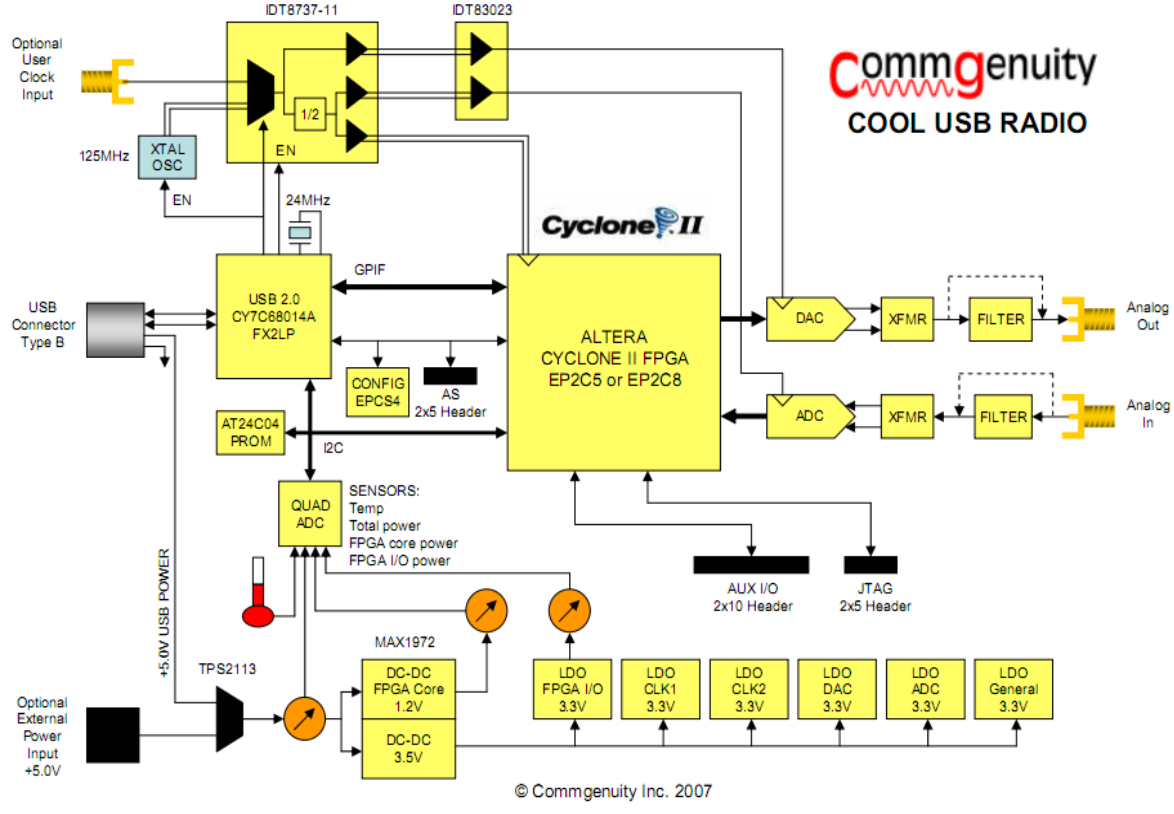


Figure 2.4: Cool USB Radio block diagram

13 For information http://www.ceanet.com.au/Products/Lyrtech/SFFSDRDevelopment/tabid/290/Default.aspx [21.01.2011]
14 Official website: http://www.coolusbradio.com/About_Us.html [21.01.2011]
15 Image source: http://www.coolusbradio.com/uploads/coolusbradio_detailed_block.pdf [24.01.2011]

Another interesting solution is the GNU Radio, an open-source project initiated by Eric Blossom. The reason for this project was to move the complexity of the hardware components to the software domain and relocate the signal processing algorithms as near as possible to the antenna. The project makes use of the **U**niversal **S**oftware **R**adio **P**eripheral, **USRP**, which can host a wide selection of daughter boards, each of which implements a signal processing block found in the GNU Radio software package The original USRP is a low cost software radio device which connects to the host computer through a USB 2.0 interface, and can send up to 16 MHz of RF bandwidth in both directions. It hosts an FPGA which can be reprogrammed, 4 high-speed Analog to Digital Converters (ADCs), 4 high-speed Digital to Analog Converters DACs), and many auxiliary analog and digital I/Os[16].

As last, the "PERSEUS" project from Nico Palermo deserves some comment. This SDR platform, depicted on Figure 2.5[17], uses a LTC2206 14-bit ADC which feeds the signal directly in a XC35250E FPGA board which performs the DDC computations. The covered spectrum range of the input signal lies between 10 KHz and 30 MHz. A low-pass filter attenuates any frequency above this level. An internal **D**irect **D**igital **S**ynthesizer (**DDS**), generates the quadrature components, required for the I/Q signal generation. An **A**nti-**A**liasing-**F**ilter (**AAF**) follows the mixer and prepares the newly constructed signals for decimation.

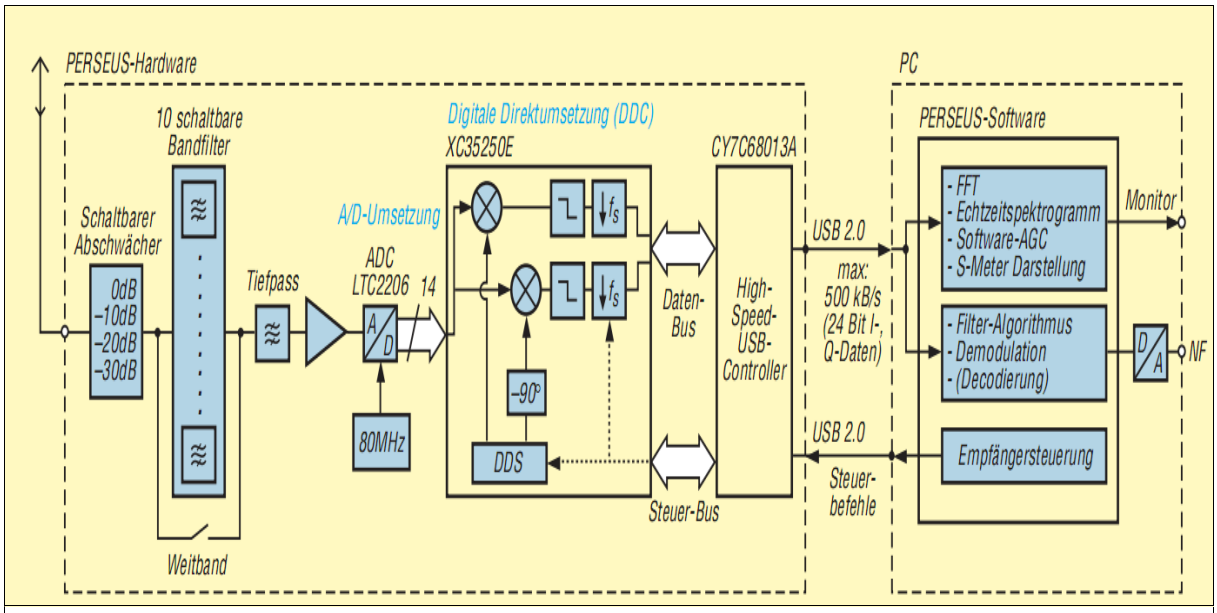


Figure 2.5: "Perseus" project block diagram

16 Product overview: http://dev.emcelettronica.com/gnu-radio-open-source-software-defined-radio [21.01.2011]
17 Image source: http://www.funkempfang.de/funkempfang/8service/pdf/PERSEUS.pdf [24.01.2011]

As a digital interface, a Cypress CY768013A Chip is selected to emulate a USB 2.0 connection to the outer world. The transferred information is visualized with an application-specific software which also allows the configuration of the DDC using different USB control strings.

These example projects are just a small part of the available devices which handle the SDR issue, but due to the fact that the approach used is similar to the one which is adopted in this master thesis. In fact, some conclusions can be drawn based on the introduced products which directly influence the platform selection and concept for this project.

- First of all, it is obvious that due to the high sampling rates, required by the ADC components, a FPGA chip would be much more suitable to handle the high data bandwidth than a DSP. The benefits of embedded multipliers as well as the possibility to select the optimal bit-resolution for every single computation favors this choice. Moreover, performing extensive digital filtering at frequencies of some Megahertz can often lead to bottlenecks if executed by a DSP. Consequently, the output can be directly sent to a desktop PC for evaluation and analysis or further processed on-board.
- Another important feature of the above mentioned devices is the interface, used for data transmission. The majority of the products go for the high-speed USB 2.0 standard, not only because it provides high bandwidth, up to the theoretical value of 60 MB/s, but also allows for flexible integration of the module. Both Cypress Semiconductors as well as Future Technology Devices International Ltd. provide dedicated solutions for USB interfaces. On the other side, The Xilinx Embedded Development Kit, EDK, comes with a hardware IP core which implements the USB standard and, with the help of a dedicated UTMI + **U**ltra **L**ow **P**in **I**nterface(**ULPI**) interface board, responsible for the physical layer of the transmission, can create a complete USB solution.
- Additionally, the vast majority of the products goes for the Linear Technology Analog-to-Digital converters, which can operate at high sampling rates and provide good conversion characteristics.
- Finally, some words about the method, used to recover the signal, should be said. All current proposals adopt the Direct-Down-Conversion(DDC) principle, a simple-to-understand but highly efficient algorithm when high sample rate change is desired.

Because of the fact that the implementation steps, such as mixing and filtering, are done in the digital domain, this method provides noticeable results together with small hardware footprint. A more elaborate discussion on the topic, done in Chapter 3.2, should point out its benefits.

2.4 Block diagram and requirements specification

After discussing the SDR topic together with the available hardware platforms, this chapter will provide an overview of the approach, adopted in this master thesis. A top-down analysis of the system should aid for better understanding of the implementation flow.

First of all, the hardware platform, together with the external devices, must be selected. The main components of the system can be found on Figure 2.6.

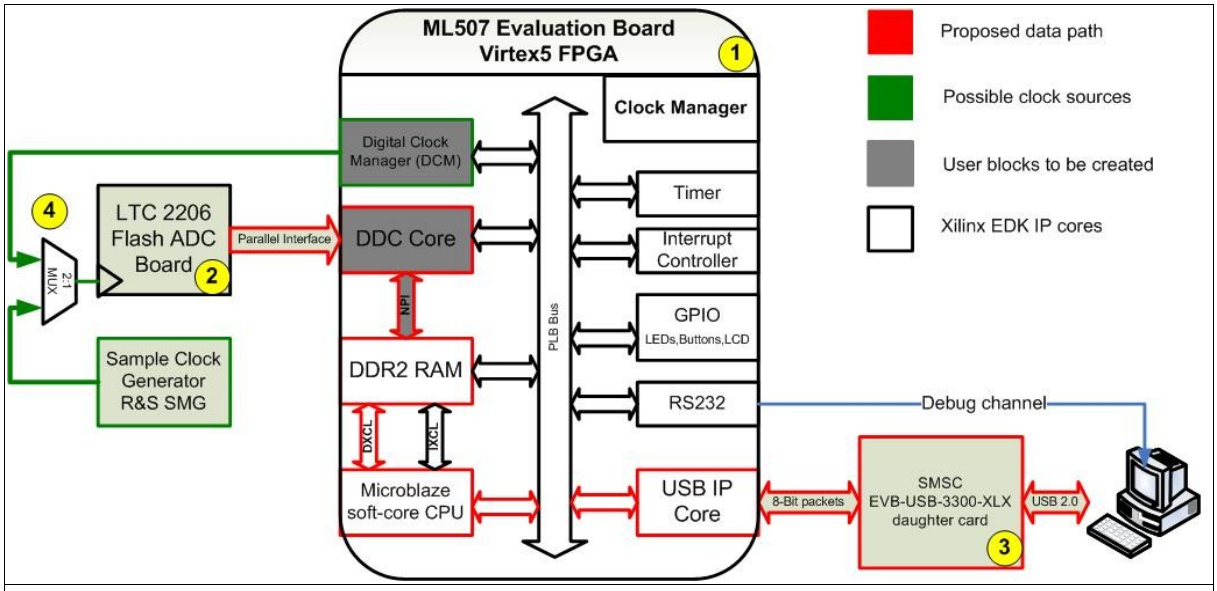


Figure 2.6: Overview of the design concept

1. **Signal processing chip**

   After Chapter 2.2 has pointed out the main differences between the DSP and FPGA devices, and the fact that the majority of the SDR solutions, available nowadays, select an FPGA chip to perform the down conversion, it is clear that the FPGA approach is the beneficial one. For this reason, this master thesis makes use

of a Virtex5 FPGA, integrated in a ML507 evaluation board. An overview of the starter kit can be found in Chapter 4.1.1. Table 1 summarizes the available resources of the device.

Available Slices	BlockRAMs		DSP48E slices	Clock Management Tile, contains two DCMs and one PLL
	18Kb	36Kb		
11200	296	148	128	6

Table 1: Virtex5 XC5VFX70T resource overview

Notable is the fact that the FPGA contains 128 embedded multipliers, which aid the high speed digital signal processing at high frequencies. The available resources permit the integration of a soft-core microprocessor with optional peripherals, which can be used during the development phase. Moreover, the CPU can be used to configure the dynamically reconfigure diverse design parameters.

2. **Analog-to-Digital converter**

   This component will sample the signal and provide the converted values to the FPGA. Most of the presented SDR projects make use of the LTC2206 and LTC 2207 ADC converters, which offer high sampling frequencies, up to 80 MHz and 105 MHz, respectively. For this reason, a DC918C demonstration circuit with an LTC2206 16-bit ADC has been selected as sampling circuit. The chip has a full power bandwidth of 700 MHz and is capable of sampling higher frequencies than the Nyquist theorem allows. The advantage of this feature is explained in Chapter 3.4.

3. **Output interface of the system**

   All the proposed examples employ a digital interface to off-load the data from the processing device to a desktop processor. This allows the extensive evaluation of a large amount of data. Moreover, the DDC design is kept small and storage issues do not play any role during the development phase. Most of the developers go for the high-speed USB 2.0 interface, because it offers a high bandwidth, theoretically limited to 60 MB/s, as well as flexibility by means of the "Plug and Play" feature. Although the there is an on-board Cypress CY7C67300 USB 1.1 full-speed controller, its bandwidth of 1.5MB/s is not sufficient to the specifications, listed in

Table 2. That is why, a Xilinx hardware USB 2.0 IP core will be integrated in the design. The firmware, which controls the USB transactions, will be run on the soft-core microprocessor and a physical interface chip is required to translate the data packets into electrical impulse on the bus. A special version of the SMSC EVB-USB3300 Daughter card, suitable for any designs involving the Virtex4 and Virtex5 devices, will be adopted in this project. Chapter 4.1.3 contains a summary of the board.

4. **Clock management**

A very important feature of the sampling process is the ADC clock source. Any distortions in this signal directly influence the output of the converter. Due to the fact that the LTC2206 clock input can be driven by a sinus as well as rectangular wave, two clock sources have been evaluated in this design. The first possibility is to generate a sine wave using an external signal generator. This has the drawback that the flexibility of the system is reduced. Another option is the usage of a **D**igital **C**lock **M**anager(**DCM**), which generates a rectangular clock signal that can be used internally in the FPGA or be routed to an output pin. The main disadvantage of this approach is that available jitter in the signals as well as the imperfect duty cycle. A comparison between the output of the ADC with both clock sources can be found in Chapter 6.2.2.

Subsequent to the hardware outline, the design specifications, found in Table 2, together with the design tools, summarized in Table 3, are presented.

Parameter	Specified value
ADC resolution	**16 bit**
Initial sampling frequency, F_S	**80 MHz**
Output sampling frequency, F_{OUT}	**1.25 MHz**
Channel Bandwidth, cut-off frequency	**200 KHz**
Stop-band attenuation	**At least 70 dB**
DDS Frequency step (32-bit phas accu)	**0.018626 Hz**
Decimation Ratio, R	**64**
Digital interface data rate	**5 MB/s**

Table 2: Design specifications

Because of the fact that a FM radio channel is located between 88 MHz and 108 MHz and the

maximum sampling frequency is limited to 80 MHz, the undersampling method may be used to sample frequencies above the Nyquist range. How this is done is explained in Chapter 3.4.

It can be seen that the used sampling frequency, F_S is much higher that the required one, $F_{S,optimal}$ = 2 * Bandwidth = 400 KHz. Because oversampling will result in better signal quality, the selected output frequency, 1.25 MHz, is kept above the optimal. Because of the fact that the I/Q approach doubles the data rate, the minimum data throughput of the system equals 1.25MHz * 16 bit * 2 = 40 Mb/s = 5 MB/s.

Design tool	Version	Description
Matlab/Simulink	2009a	Used to create a simulation model for the DDC algorithm and perform output evaluation
Xilinx System Generator	11_04	Used to synthesize the simulation model and create a hardware component out of it
Xilinx Embedded Development Kit (EDK)	11_04	Used to generate the complete embedded system (allows the integration of the Microblaze soft-core microprocessor, the USB IP core and a number of free peripheral cores), hardware synthesis as well as VHDL implementation of the DDC core;
Xilinx Software Development Kit (SDK)	11_04	Used to develop and manage the software application, running on the Microblaze microprocessor
Xilinx Core Generator	11_04	Used to create Xilinx-specific hardware components, such as a dual-sided **F**irst **I**n **F**irst **O**ut (**FIFO**) element, used for clock domain synchronisation
Linear Technology PScope	6	An application software, provided with the DC918C demonstration circuit, used for ADC output evaluation
Microsoft Visual Studio 2005	8	Used to design the acquisition software, used to read out the samples out of the FPGA

Table 3: Summary of the design tools, used in this project

3 Theory

This chapter will provide the theory needed to cope with the problems concerning this project. The techniques, described in this chapter, will be analyzed, documented and justified by examples.

3.1 Realtime Digital Signal Processing

Digital Signal Processing is concerned with the representation of signals by a sequence of numbers and the processing of these numbers. The goal of DSP is usually to measure and/or filter continuous real-world analog signals. The first step is usually to convert the signal from an analog to a digital form, by sampling it using an **A**nalog-to-**D**igital **C**onverter(**ADC**) which turns the analog signal into a stream of numbers. Even though this process is more complex than analog processing and has a discrete value range, the computational power to digital signal processors brings many advantages over analog processing in many applications, such as error detection and correction in transmission[18]. The usual approach of handling signal processing problems is depicted on Figure 3.1.

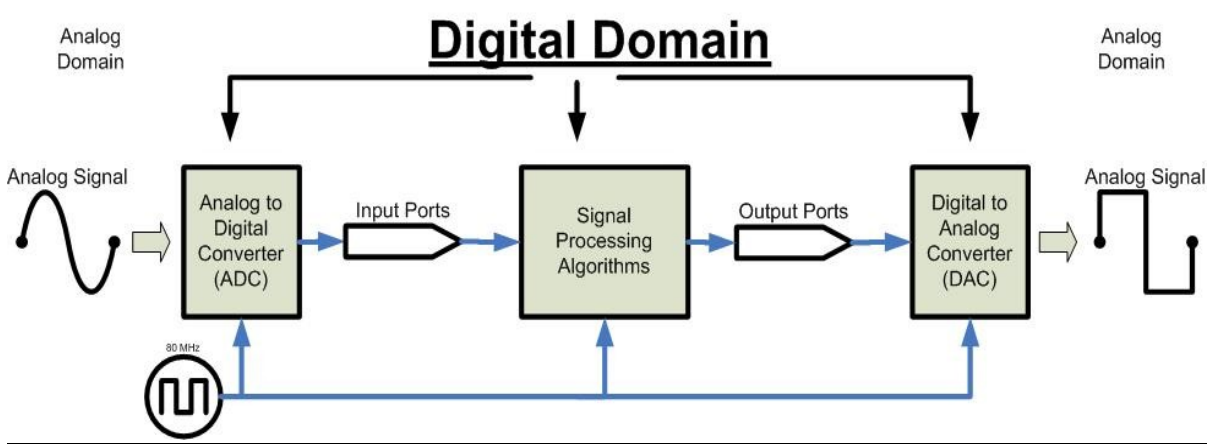


Figure 3.1: Signal Processing overview

As technology evolved, the DSP algorithms have been shifted from standard computers to more powerful systems called Digital Signal Processors which adopt an optimized architecture required for the extensive calculations. As an alternative to DSPs today's market

18 Wikipedia, http://en.wikipedia.org/wiki/Digital_signal_processing , [20.12.2010]

offer Field Programmable Gate Arrays and it is up to the engineer to select the optimal device for a certain project. The main differences between those architectures will be discussed in the next subsections.

3.2 Direct Down Conversion

In terms of signal processing, narrowband systems are generally characterized by the fact that the bandwidth of the signal of interest is significantly less than the sampled bandwidth; that is, a narrow band of frequencies must be selected and filtered out from a much wider spectral window in which the signal might occur. This means that large sample rate changes must be undertaken to efficiently process the signal for either transmission or reception.[XAPP1113]

A **D**igital **D**own **C**onverter(**DDC**) is supposed to translate a given pass-band signal down to base-band which would reduce the processing effort tremendously. This is achieved by mixing the signal, low-pass filtering to prevent aliasing and decimation of the sampling rate. During this process the proper channel selection must be implemented and all filters should be designed with respect to the channel's bandwidth. Generally, the DDC input is sampled at a very high sampling rate while the output operates at a much lower frequency. As depicted on Figure 3.2, the signal is first shifted by a mixer to a certain frequency range and then decimated by a predefined factor.

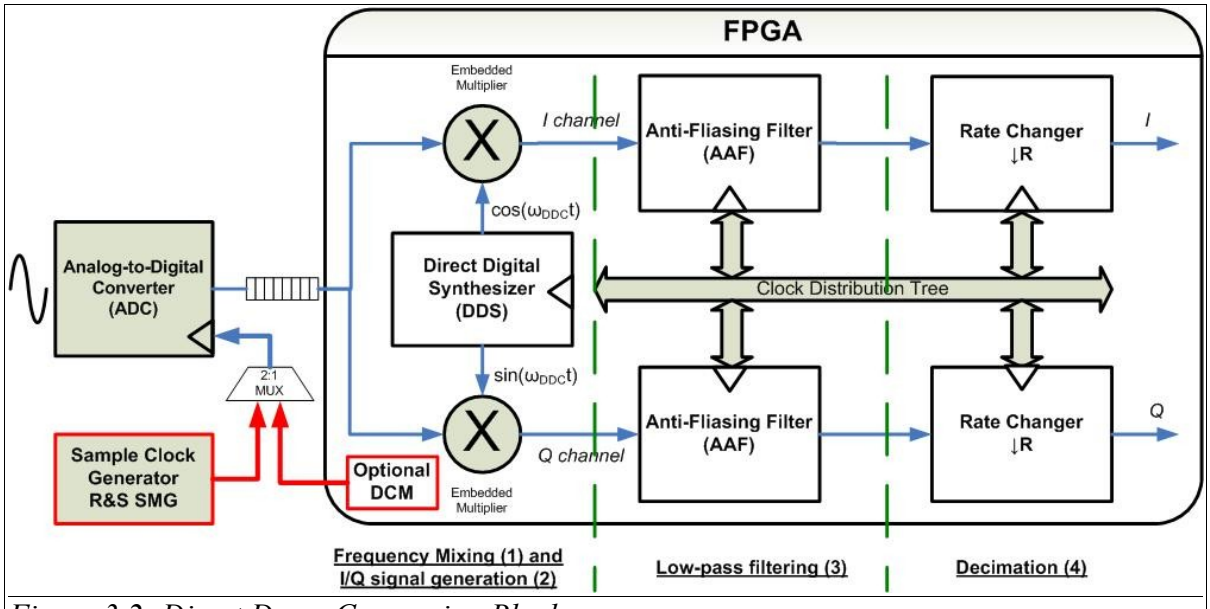


Figure 3.2: Direct Down Conversion Block

Four major steps are undertaken so that the signal is down converted in base-band:

(1) frequency mixing

An easy way to shift the spectrum of a signal is by multiplying it with a sine wave, which then corresponds to convolving the spectra of both signals in frequency domain. The generation of the mixing signals is explained in Chapter 3.2.1.

(2) I/Q signal generation

Quadrature signal recovery is beneficial for the case of down conversion. More on this topic can be found in Chapter 3.3.

(3) low-pass filtering of the mixer output

The two most common structures, namely the **F**inite **I**mpulse **R**esponse(**FIR**) and **C**ascaded **I**ntegrated **C**omb(**CIC**) filter structures will be considered in the following subsections.

(4) sampling rate decimation

This is the process of reducing the sampling frequency. No hardware component is required, but the decimation factor must be carefully considered, otherwise aliasing may occur.

3.2.1 Mixing

The main purpose of the mixing circuit in a DDC system is to shift the spectrum of the input signal to a predefined spectrum region, generally in base-band. Moreover, by varying the mixer's frequency it is possible to select different channels, which would be the desired effect when more then one source is to be recovered. The easiest way to implement the mixer in digital hardware is by creating a **D**irect **D**igital **S**ynthesizer (**DDS**). The generated wave, is a function of the configured phase step, which can be used to modify the output frequency. That is why this block is the digital equivalent of the **V**oltage **C**ontrolled **O**scillator (**VCO**), an analog circuit, used to control the mixing frequency.

The best way to describe the effects of mixing is by reviewing the multiplication of two sine waves in time domain:

$$\cos(a)*\cos(b)=1/2*[\cos(a+b)+\cos(a-b)]$$

In the general case the important component is the difference of the frequencies and the high frequency element is suppressed by filtering the output of the mixer. If the carrier frequency of the signal of interest is known, an easy way to recover the signal into base-band is by setting the local oscillator at this value and low-pass filtering the output sequence. Figure 3.3 represents this simple method of relocating the signal of interest into base-band.

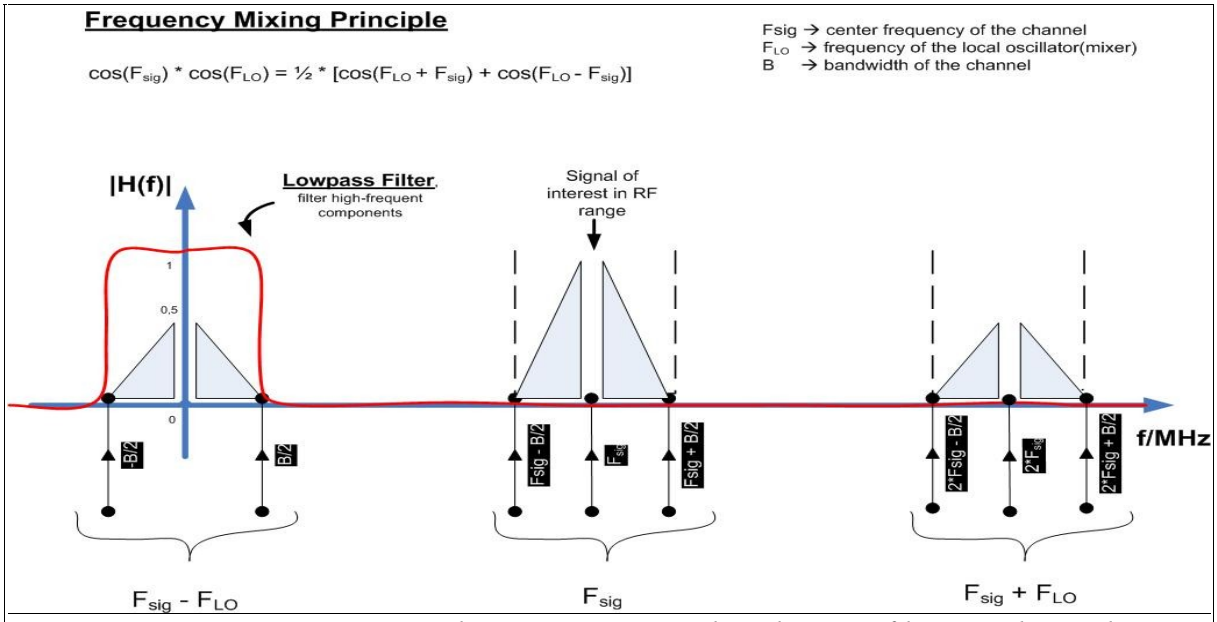


Figure 3.3: Frequency mixing results in two new signals; a lowpass filter may be used to reject the unwanted components

As soon as the incoming signal is sampled it is fed into the DDC block, the conversion process begins with mixing the signal to a suitable frequency. Not only is the DDS responsible for the shift of the input's spectrum, but it also influences the quality of the output signal with the quantization of the sine/cosine pair. To address this issue one should take a closer look in the data sheet of the employed DDS Compiler V4.0 and its architecture, depicted on Figure 3.4.

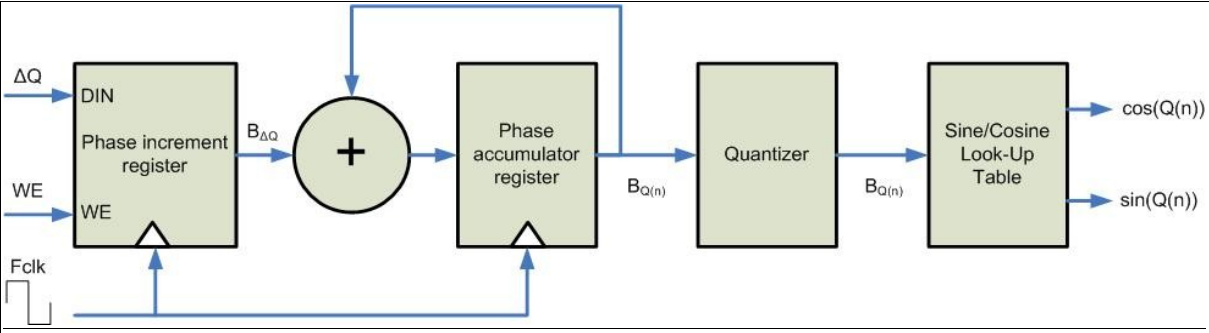


Figure 3.4: DDS architecture overview

The following subjects must be considered so that the DDS output wave can be evaluated:

- **Output frequency derivation and resolution**

  Generally speaking, the output frequency is a function of the system clock, **Fclk,** which triggers the core. Also, the resolution of the phase accumulator, **B**, and the phase increment value, **ΔQ,** are involved in the calculation of the output. The following equation depicts the dependencies between the variables.

$$Fout = \frac{(Fclk * \Delta Q)}{2^B}$$

  In the normal case, all parameters except for the phase increment value are known. Because of the fact that the system clock can be regarded as a constant, it is up to the size of the phase accumulator to calculate the achievable phase resolution, which is just the value of the system clock divided by the maximum value of the phase accumulator. Because of the fact that the soft-core microprocessor operates on 32-bit registers, it has been considered to synthesize one software register which holds the DDS phase increment. For this reason, the resolution of the NCO, calculated with respect to above eqution, is 0.018626 Hz, which does not limit the range of operation of the designed system. The phase accumulator register is then quantized and only a portion of it is used to address the **L**ook-**U**p **T**able (**LUT**), holding the samples. The number of bits, used to address the table, as well as the resolution of the output directly influence its spectral purity.

- **Spectral purity considerations**

  As explained in [DS558], both the phase and amplitude quantization influence the

DDS generated signal. The size of the look-up table, together with the size of the samples, determine the phase angle resolution, which result in a time-base jitter and affect the spectral purity of the output. Two different correction algorithms can be used to enhance the quality, a Taylor Series correction or Phase Dithering.

- Phase dithering

  This approach makes use phase error, introduced by the quantizer, to optimize the **S**purious **F**ree **D**ynamic **R**ange (**SFDR**) of the output. This error, distorts the output spectrum and repeats itself periodically. By breaking this periodicity by using a random sequence, called dither, it is possible to achieve additional 12 dB of SFDR. The noise is added to the phase accumulator before being quantized.

- Taylor series corrections

  The second method takes the discarded phase accumulator bits and calculate a correction which are added to the LUT output. This has the effect of increased SFDR rate compared to the previous discussed principles. The drawback of this method is the usage of DSP48E slices for the computations.

- **Dynamic reconfiguration of the phase step**

Lastly the programming interface of the DDS core is enabled. This allows for the dynamic reconfiguration of the mixer frequency and keeps the system flexible. Figure 3.5[19] shows the programming interface of the DDS core and Table 4 holds the description of the signals.

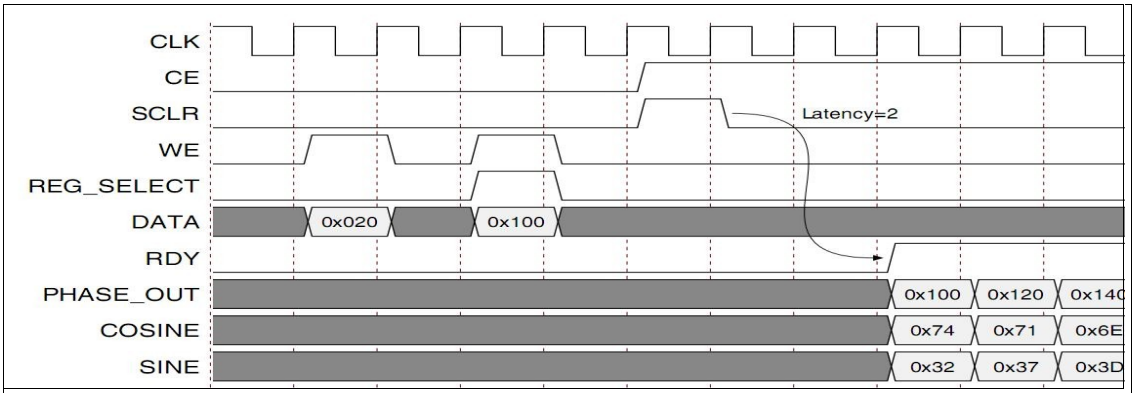


Figure 3.5: DDS "phase increment value" Programming Interface

19 [DS558], figure 18

Signal	Description
CLK	Source clock
CE	Chipe Enable, active high, must be set during normal operation, but is irrelevant during the reconfiguration process
SCLR	Synchronous clear, active high, resets the logic
WE	Write Enable, active high, must be set for at least one CLK cycle
REG_SELECT	If both phase shift and phase increment are programmable, this pin is used to select the register to be written
DATA	Holds the value to be written to the internal registers
RDY	Ready, active high, designates that the DDS output is valid
PHASE_OUT	Optional, outputs the internal phase accumulator
COSINE	Output, cosine wave
SINE	Output, sine wave

Table 4: DDS core signals legend

Optimization level	Slices	FlipFlops	BRAMs	LUTs	Embedded Multipliers
None	33	66	8	92	0
Phase Dithering	49	89	2	132	0
Taylor Series Corrected	23	64	1	52	3

Table 5: DDS block resource estimation with different noise shapping options

Due to the fact that the Phase dithering method consumes no embedded multipliers, see Table 5, and produces the same SNR performance as the Taylor Series correction method, it has been implemented in this project.

3.2.2 Digital Filters

By defining the filter specifications in Chapter 2.4, it is very important to select the proper filter design. A filter structure consists of a number of multipliers and a certain memory space, which holds the delayed samples. This leads to the problem that if a real time implementation is required, the time available for one filter calculation is limited by the sampling frequency. It is up to the available cycle count between two samples which could define the filter structure to be used.

Another important feature of a digital filter is the phase response. This specification describes the filter reaction to different input frequencies. In some signal processing problems, like demodulation, a linear phase is desired for the correct reconstruction of a signal. As next, the two suitable filter structure, the FIR and CIC filters, will be discussed.

- **FIR filter**

A FIR filter can be mathematically described using the following equation:

$$y(k)=\sum_{m=0}^{N-1}(h(k-m)*x(m)) \quad \bullet\!\!-\!\!\circ \quad H(z)=\sum_{k=0}^{N-1}(h(k)*z^{-k})$$

If the impulse response, h(k), is symmetric or antisymmetric then the phase of the filter is linear. Because the length of h(k) can be either even or odd, four different types of symmetry exist. The most preferable one is type 1, a symmetric impulse response with odd number of filter coefficients. This type is suitable for nearly any filter type.

Additionally to the type 1 filter structure, a poly-phase decomposition of an FIR filter can be performed to reduce the computation effort. The following example for a decomposition in 2 branches should show the benefits of this technique:

$$\begin{aligned}H(z)=&h(0)+h(1)*z^{-1}+h(2)*z^{-2}+h(3)*z^{-3}+h(4)*z^{-4}\\&h(5)*z^{-5}+h(6)*z^{-6}+h(7)*z^{-7}+h(8)*z^{-8}\end{aligned}$$

$$\begin{aligned}H(z)=&(h(0)+h(2)*z^{-2}+h(4)*z^{-4}+h(6)*z^{-6}+h(8)*z^{-8})\\&+(h(1)*z^{-1}+h(3)*z^{-3}+h(5)*z^{-5}+h(7)*z^{-7})\end{aligned}$$

$$\begin{aligned}H(z)=&(h(0)+h(2)*z^{-2}+h(4)*z^{-4}+h(6)*z^{-6}+h(8)*z^{-8})\\&+z^{-1}*(h(1)+h(3)*z^{-2}+h(5)*z^{-4}+h(7)*z^{-6})\end{aligned}$$

$$H(z)=E_0(z^2)+z^{-1}*E_1(z^2)$$

and the general description fo a polyphase decomposition in M bands is:

$$H(z)=\sum_{k=0}^{(N-1)/M}(z^{-k}*E_K(z^M))$$

The poly-phase approach has the benefit of reducing the length of the computation sequence and could be used to efficiently implement decimation and interpolation. How exactly this is

done is explained in 3.2.3,Decimation.

The main disadvantage of FIR filters is the fact that their coefficient vector grows rapidly if a sharp transition band of the frequency response at a high sampling rate is assumed. An efficient way of reducing the computational effort is by using the suggested Poly-phase Decomposition method as described in [SP] Chapter 11 and by this distributing the computations in several stages.

- **CIC filter**

As an alternative to FIR filters another type of filter structure, called **C**ascaded **I**ntegrator-**C**omb(**CIC**) or Hogenauer filter, is referred as suitable for the case of DDC. The main advantage of this type is that it does not involve any multiplications in its structure, but is based on cascading Integrator and Comb blocks and by this achieving a low-pass filter response. Because of the fact that no embedded multipliers are required for implementing this filter, this could turn out to be an efficient way of creating a large sampling rate change. If the magnitude response of the integrator and comb elements, depicted on Figure 3.6, is examined, it is easy to see that they are inverse to each other. While the former one introduces an infinite gain at DC, the comb component can be seen as a high-pass filter with an infinite attenuation at DC. For this reason, a cascade of both should result in no amplitude change of the input signal, an the overall frequency response is as follows:

$$H_{tot}(z) = H_{INT}(z) * H_{DIFF}(z) = \left[\frac{1}{(1-z^{-1})}\right] * \left[1-z^{-1}\right] = 1$$

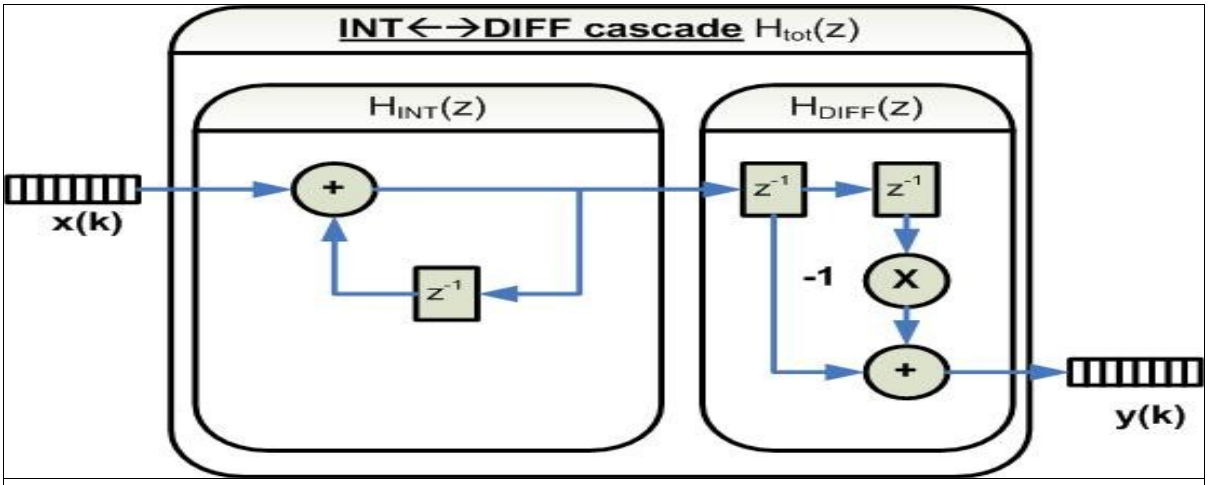


Figure 3.6: Integrator <-> Comb filter cascade

Another filter structure, which must be mentioned before considering the CIC system, is the Moving-Average order. Its structure, depicted on Figure 3.7, results in a low-pass filter characteristic with N-1 spectral zeros form 0 to F_s, where N is the number of filter delays and F_s is the sampling frequency.

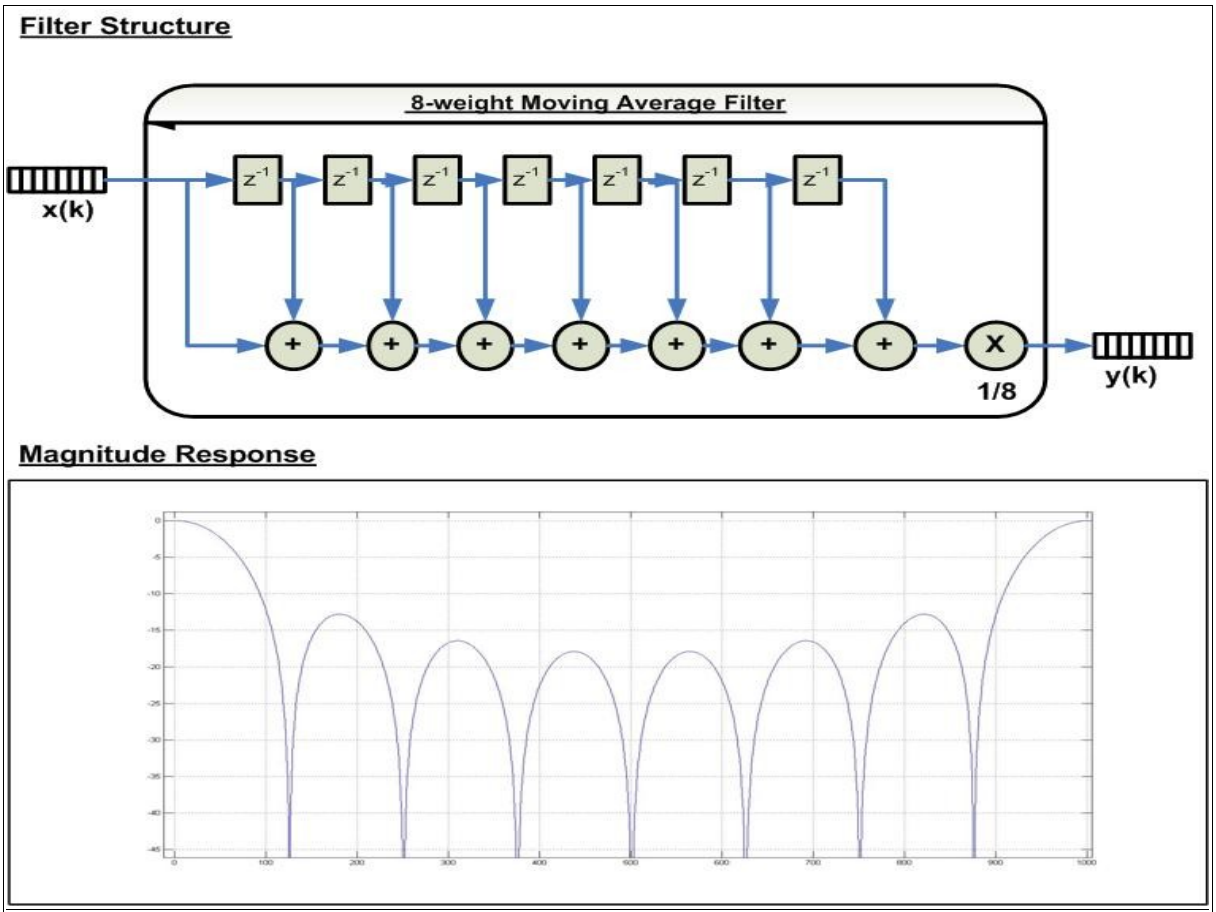


Figure 3.7: 8-weight Moving average Filter, Structure and Frequency Response

After discussing the Integrator, Comb and Moving-Average filters, it is time to take a closer look on the CIC system, which can be found on Figure 3.8. This network differs from the one, shown on Figure 3.6, in that the differentiator is not of 1st order, but of 8th. Because of this, the transfer function of the component is:

$$H_{CIC}(z)=H_{INT}(z)*H_{COMB}(z)=\frac{(1-z^{-8})}{(1-z^{-1})}=1+z^{-1}+z^{-2}+z^{-3}+z^{-4}+z^{-5}+z^{-6}+z^{-7}$$

In fact, the frequency response of the N Comb weight CIC filter is equal to a N-1 weight Moving-Average filter. Furthermore, by cascading CIC filters it is possible to achieve much better attenuation in stop-band. However, this influences the pass-band characteristics of the filter and introduces the so called " **baseband droop**". As it can be seen on Figure 3.9, the pass-band flatness decreases with every additional CIC stage.

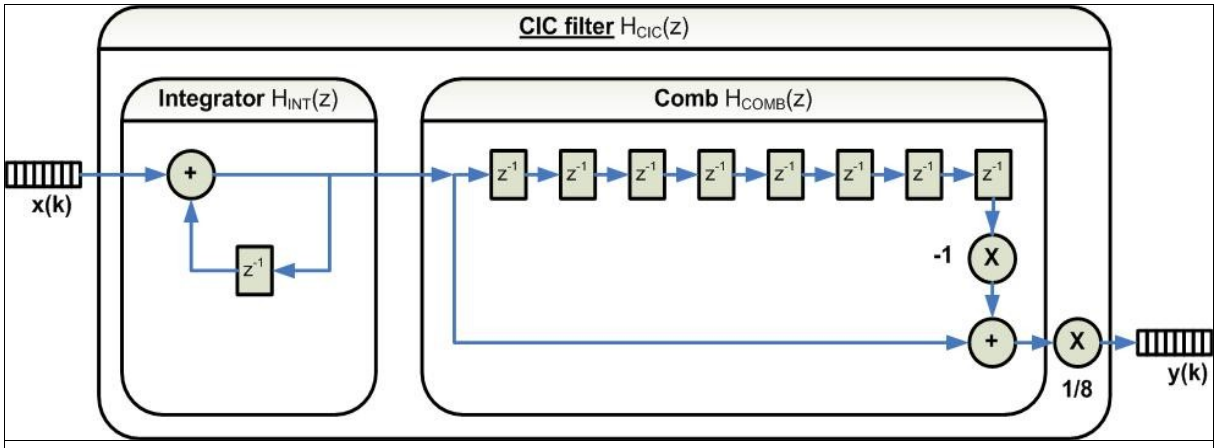


Figure 3.8: CIC filter structure

As a result, a trade-off between attenuation and pass-band droop may reduce the undesired effects. On the other side, another filter may be introduced to equalize this effect and by this recover the flatness of the transfer function. Additionally, this compensation filter can relax the sample rate change specifications of the CIC filter if it is designed as a decimator and by this optimizing the resource utilization.

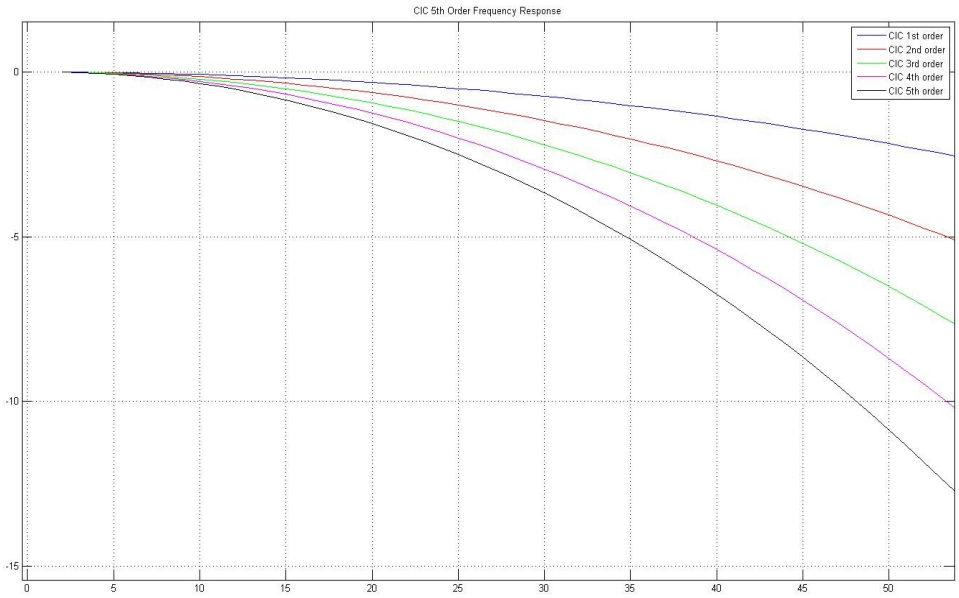


Figure 3.9: CIC Pass-band droop change with respect to the order of the filter

Besides that, the structure of the Cascade CIC filter can be reviewed and the rate changer can be shifted through the Comb elements which results in reduction of the memory requirements. Figure 3.10 represents the required changes in the cascade network.

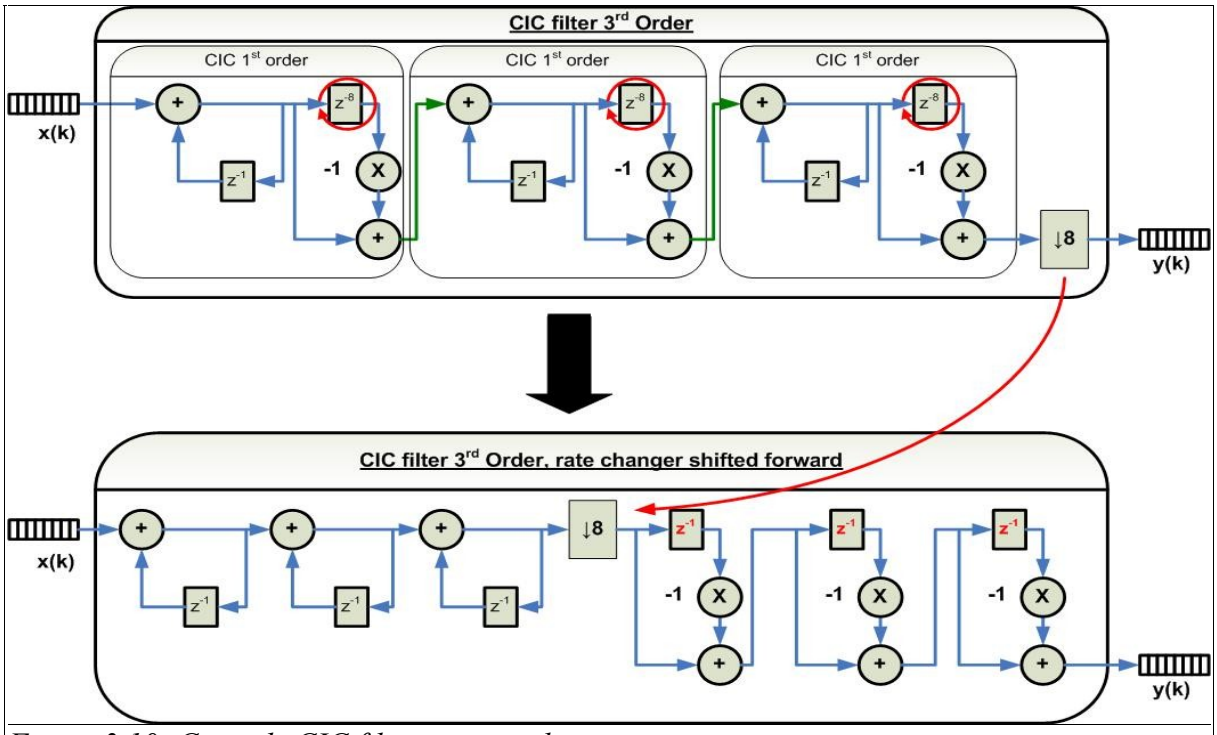


Figure 3.10: Cascade CIC filter, optimizide structure

Finally, the topic of "**Bit Growth**" must be discussed as it concerns both arithmetic accuracy as well as resource consumption. It can be proven that the gain of the CIC filter is $G = R^N$, R being the rate change and N being the number of sections. Hence, the output of the filter, in bits, is calculated as shown in the following equation.

$$B_{OUT} = B_{IN} + [\log_2 G] = B_{IN} + [N * \log_2 R]$$

Additional information on the CIC filter topic can be found in [CIC1].

3.2.3 Decimation

As already mentioned, a system would require a sampling rate which is at least twice the highest fundamental frequency component of the input signal. If the signal is sampled at a higher sampling rate than needed, it is possible to decimate the signal an this way reduce the data throughput of the system. As a result, both memory and computation savings are achieved. In general the DDC principle is applied when high decimation rates are to be designed. In order to perform decimation in the right way, the input sequence should be pre-filtered with an Anti-Aliasing Filter, as shown in Figure 3.11. The AAF ensures that no aliasing will occur after the signal is being decimated.

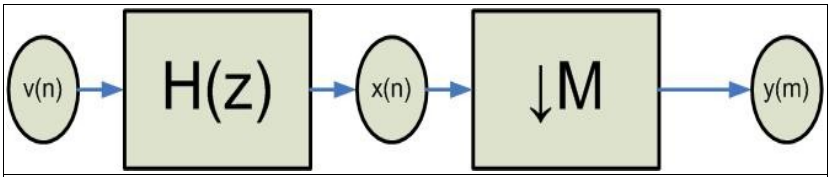


Figure 3.11: Block diagram of a decimator with an AAF filter

The filter specification depends on the decimation factor and system requirements. If a steep-slope filter is required, a cascade of filters could be also considered. This part will show an efficient way of implementing a decimator taking advantage of the poly-phase decomposition, discussed in Chapter 3.2.2.

The following example[20], depicted on Figure 3.12, will be used to explain how poly-phase decomposition can be beneficial when a decimator must be designed.

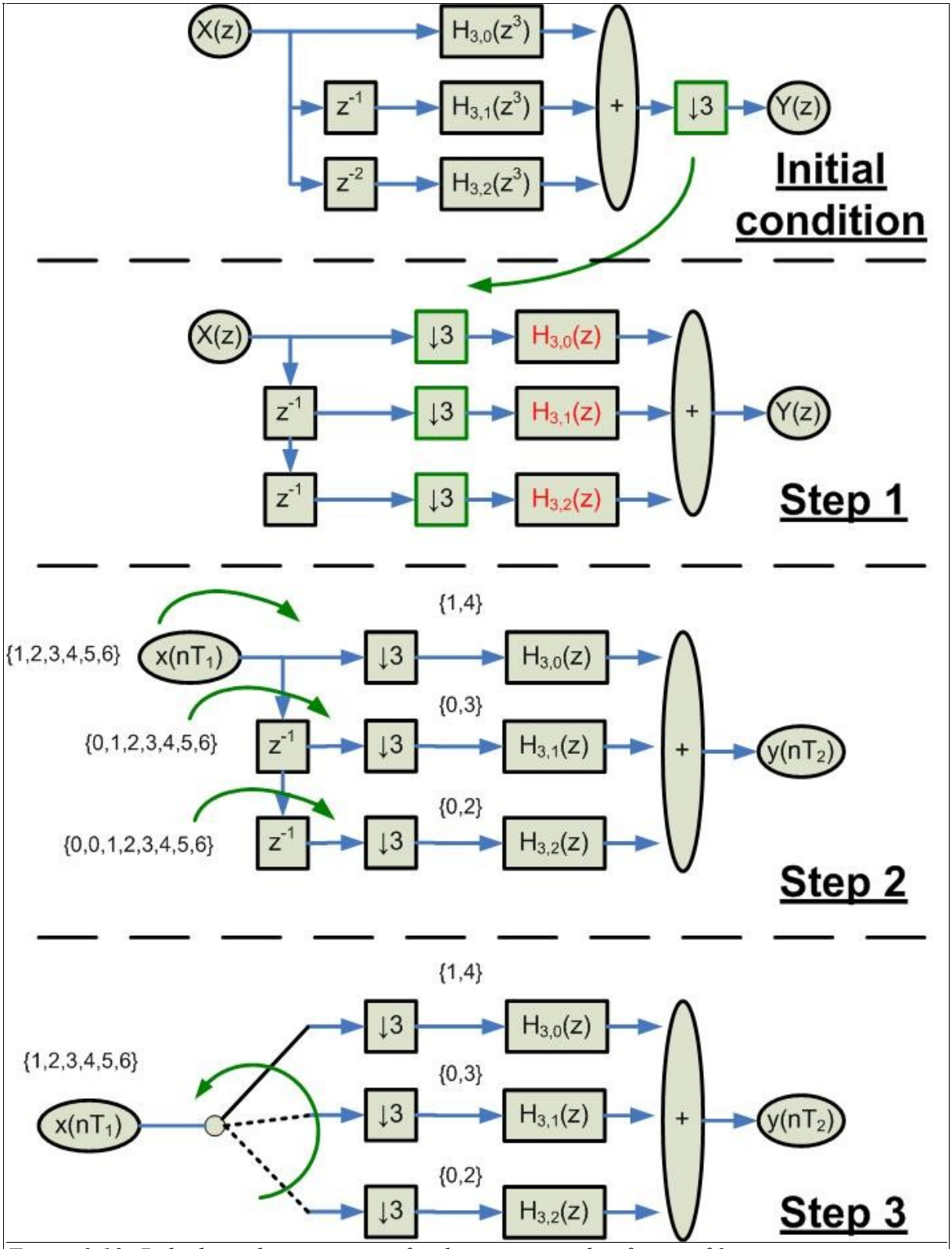


Figure 3.12: Polyphase decomposition for decimation with a factor of 3

20 Reference to the complete example can be found in [SVG].

- **Initial condition**

  The first assignment is to compute the filter and split it to the value of the rate changer, in this case 3, poly-phase components. It is important that the initial FIR filter must be designed at the higher sampling frequency!

- **Step 1**

  Due to the fact that the transfer functions are of 3rd order, it is possible to use the noble identities and shift the rate changer through the filters. This results in the fact that the filters now will be running at the lower sampling frequency.

- **Step 2**

  This simple test using a simple sequence has the purpose to show how the samples are split between the different poly-phase components.

- **Step 3**

  The final step shows how the delay blocks can be translated to a de-interleaver structure, which rotates counter-clock wise. As a result, it can be seen that only one poly-phase component must be computed per cycle. After a total of 3 cycles an output can be produced at the adder output. Additionally, the filters operate at the output sampling frequency.

Compared to a direct structure, this implementation has a higher output latency, but the significant computation and memory savings, it brings, are overwhelm this disadvantage. Still, if the direct FIR filter is too long and the sampling frequency is too high, even this approach may not capable of producing a suitable solution. Recalling the specification of the required filter in Chapter 2.4, a filter running at 80 MHz and having a transition band of less than 0.002525 with respect to the sampling frequency, would result in a coefficient count of a few thousand. A Matlab computation of a sample filter, with less than the required attenuation and a transition band 10 times larger than the desired one, returns a filter length of 2418. Even though that a decimation factor of 64 would reduce the computations dramatically, it is still required to perform ~40 multiplications per cycle at this high frequency. For this reason a FIR solution is not suitable for this project. A cascade of a CIC filter with another FIR filter would require much less resources and achieve better characteristics.

3.3 I/Q signal generation

As stated in [AN1298], using Quadrature signals for demodulation brings important benefits regarding the system design. InPhase/Quadrature(I/Q) signals are known also as orthogonal to each other which means that they have a phase shift of 90 degree. For this reason the scalar product of both is equal to 0. The main advantage of quadrature signals is that they do not interfere with each other. When recombined, they are summed to a composite output signal. There are two independent signals in *I* and *Q* that can be sent and received with simple circuits. This simplifies the design of digital radios. The benefit of *I/Q* modulation is the symmetric ease of combining independent signal components into a single composite signal and later splitting such a composite signal into its independent component parts.

The profit of performing I/Q detection in the digital domain, instead of using analog components, are the absence of the gain-balancing problem as well as the impedance matching issue. Also, sampling the signal before being shifted into baseband results in no DC gain errors. Because of the fact that both the InPhase and Quadrature branches are generated by the same component in the digital domain, the Direct Digital Synthesizer, these error sources does not exist.

As it can be seen on Figure 3.13, soon after the input is being sampled, it is multiplied with a sine/cosine pair and two branches processing branches are established. Due to the fact that the I/Q pair is generated by the same components both signals are in 90 degree out of phase and power loss occurs. Afterwards, a low-pass filter rejects the unwanted high-frequency components outside the band of interest as well as prevents aliasing. At last, the sampling frequency is reduced and the decimated output sequence is stored for later processing. At this point, different demodulation techniques may be adopted to recover the information with respect to the I/Q pair of signals. For the reason that the initial signal analysis will be done off-board, the I/Q outputs will be transmitted on a desktop computer and any processing will be done in software.

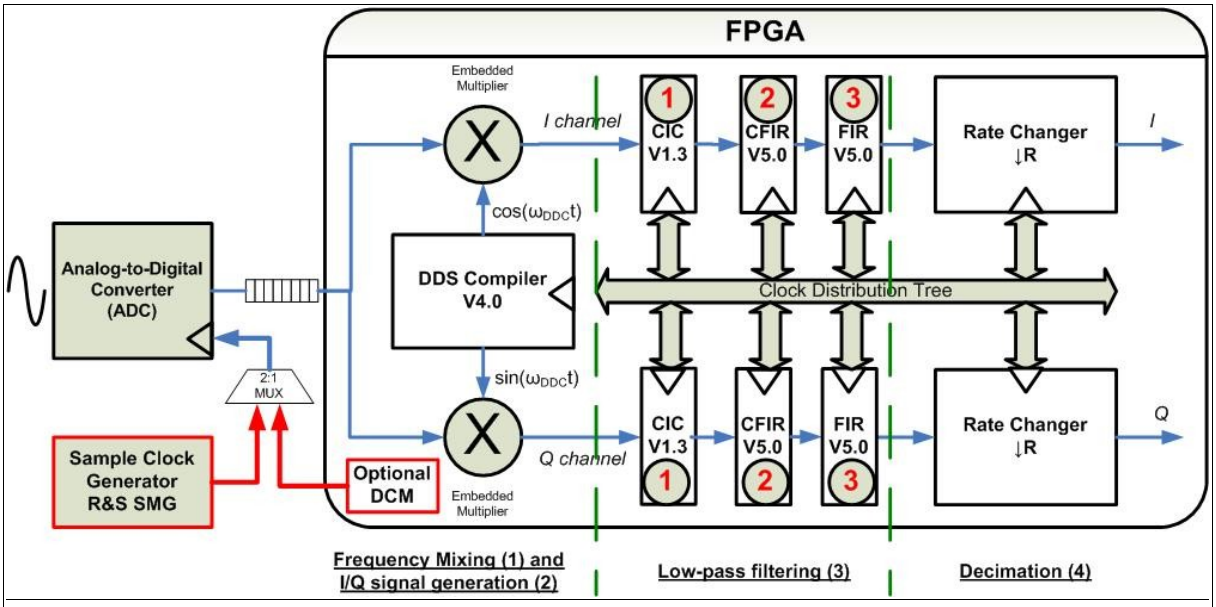


Figure 3.13: I/Q signal generation, filter cascade and sampling rate change

3.4 Sampling, Undersampling

Based on the Nyquist Sampling Theorem, in order to get a unique representation of a frequency content of a signal, the signal should be sampled at a rate at least twice the value of the highest frequency component of the signal. If the signal of interest is considered to be in baseband then the required sampling frequency would be also the lowest one achievable for sampling this signal. A problem arises if a given signal is modulated at a higher frequency. This would mean that the required sampling frequency will be higher than the optimal value. As a drawback, the data rate to be processed increases and the memory, required to store the samples, would also grow. Because all the unnecessary information, sampled together with the signal of interest, must be discarded, a solution to tackle this kind of problem is required. An efficient way of sampling signals at a frequency, lower than the Nyquist value, is the so called **Undersampling** or **Bandpass Sampling** method.

Undersampling takes advantage of the main problem which arises if the sampling rate does not satisfy the Nyquist theorem – Aliasing. As it can be seen on Figure 3.14, a high frequency component, sampled at a lower frequency than required, would appear as low frequency signal. In the worst case this will result in aliasing, which would mean that it is not possible to recover the signal. But if the signal is mapped to a frequency range, which is free of spectral components, then it can be completely recovered.

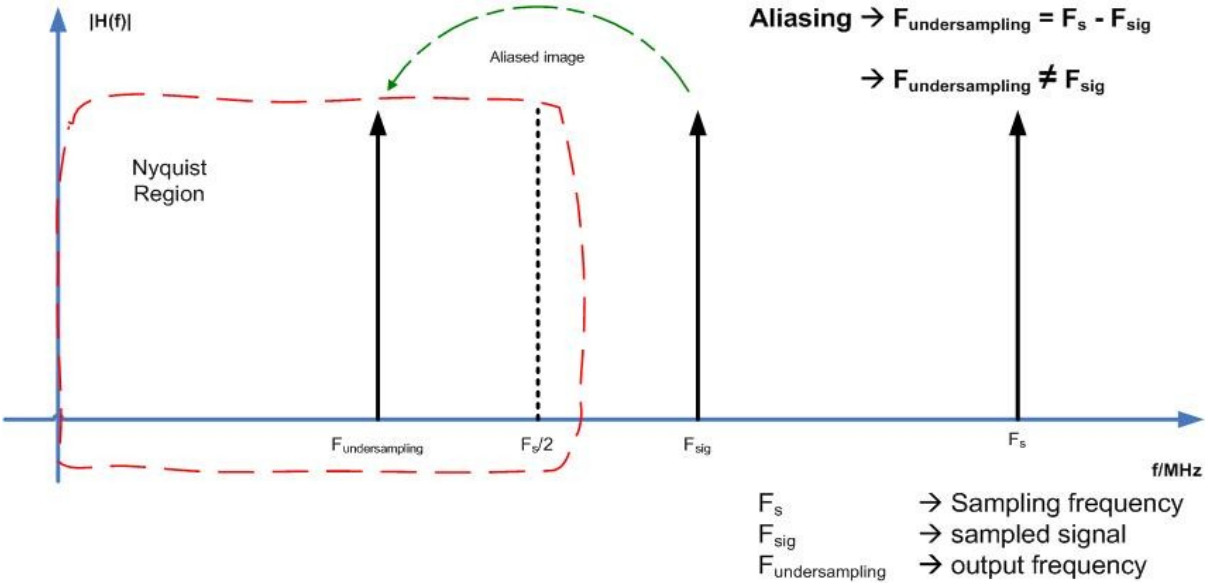


Figure 3.14: The effect of aliasing if the Nyquist theorem is not satisfied

The parameters required for calculating a suitable sampling frequency, so that the sampled signal is moved to an free intermediate frequency range, are the lower and upper frequencies of the signal of interest. As stated in [DAC], „If a (real) bandpass waveform has a nonzero spectrum only over the frequency interval $f_1 < |f| < f_2$, where the transmission bandwidth B_T is taken to be the absolute bandwidth $B_T = f_2 - f_1$, then the waveform may be reproduced from sample values if the sampling rate is $\mathbf{f_s \geq 2*B_T}$“. The condition of the waveform being only non-zero in the given interval can be achieved by introducing a band-pass filter with a suitable specification.

To meet this condition, the sampling frequency should satisfy the following equality:

$$\left(\frac{(2*f2)}{n}\right) \leqslant fs \leqslant \left(\frac{(2*f1)}{(n-1)}\right)$$, where

$$1 \leqslant n \leqslant \left[\frac{f2}{(f2-f1)}\right]$$

If an adequate value is found then the ADC can be supplied with this frequency and the signal

can be reconstructed with a lower sampling rate than required by the Nyquist theorem.

Figure 3.15 illustrates how the DDC principle takes advantage of the undersampling method, together with frequency mixing and low-pass filtering, to retrieve a RF signal in baseband.

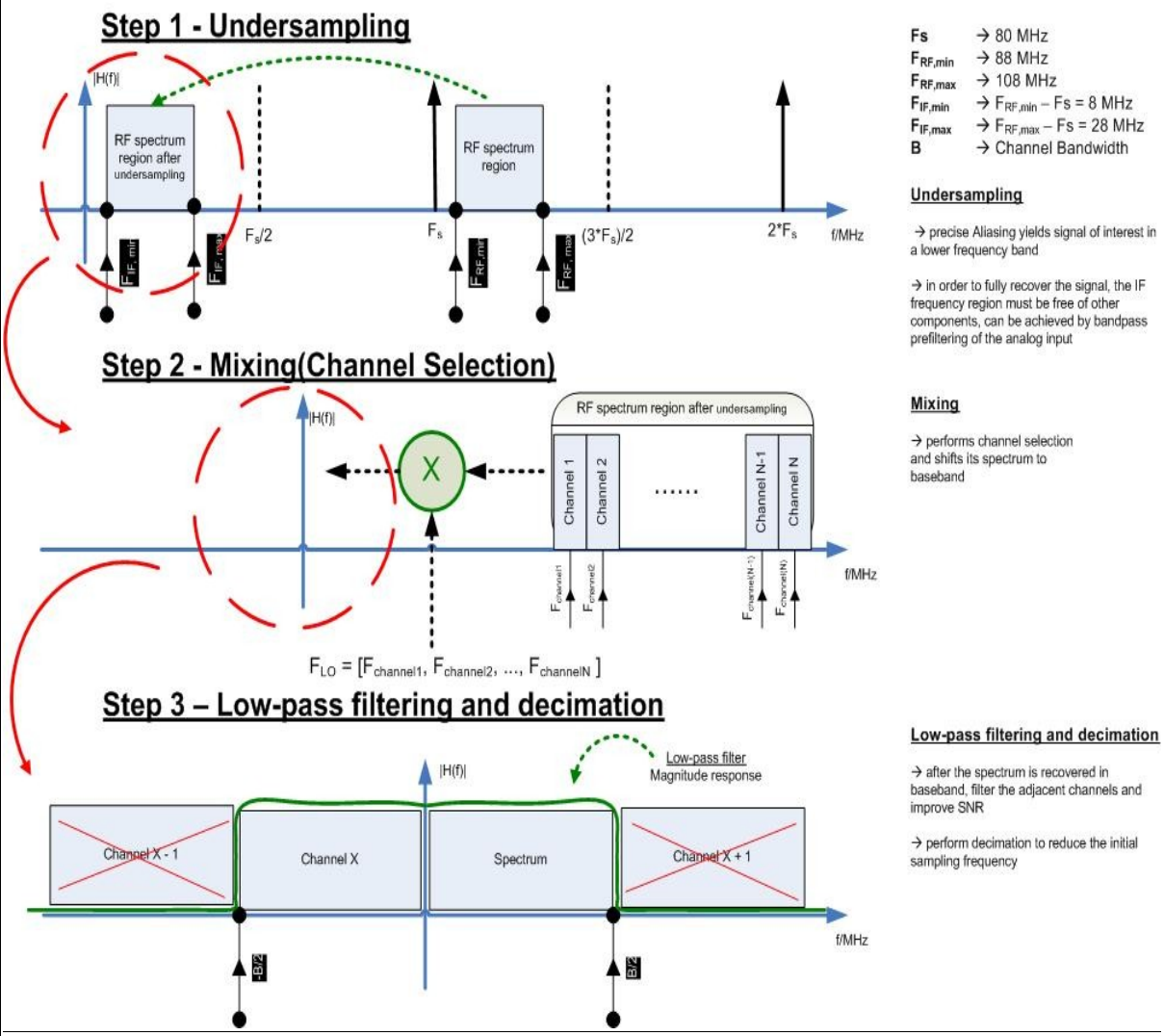


Figure 3.15: DDC Principle, processing sequence and spectral modifications

3.5 Analog to Digital Conversion

In the case of digital signal processing, the first step performed is to digitize the signal. By doing this, the signal is converted to a digital bit string, which is then processed by the algorithm implemented on the DSP or FPGA. There are different ways of converting and

representing the data. Generally the ADC selection process is driven by the cost and timing requirements of the end product. If a high frequency signal is to be sampled then a flash ADC may be the right choice. If, on the other side, a slowly changing signal is to be converted, it would be possible to use an ADC with a simpler structure. In the case of DDC and SDR applications, the signals of interest are located in the HF range, between 3 and 30 MHz, as well as VHF range, between 30 and 300 MHz, which would require fast conversion timings. For this reason, a flash ADC could be the right choice. The main advantage of this ADC type is that the analog value is converted to a binary string and all bits are evaluated in parallel. This technique provides the fastest conversion time but also has the drawback that the hardware requirements grow exponentially with an increase in the resolution. The following subsections elaborate on three of the most important quality measurement values, describing an ADC chip.

- **Spurious Free Dynamic Range(SFDR)**

  It is not only the bit resolution and conversion time which specify how good one ADC is, but there are also important parameters which describe the signal quality and must be taken into consideration when selecting the proper chip. Because of the fact that the converter has a finite precision, it is obvious that a certain quantization error is introduced during the conversion process. Another important characteristic is the SFDR, which is the ratio of the rms value of the signal to the rms value of the worst spurious signal regardless of where it falls in the frequency spectrum. The worst spur may or may not be a harmonic of the original signal. SFDR is an important specification in communications systems because it represents the smallest value of signal that can be distinguished from a large interfering signal (blocker)[21].

- **Signal-to-Noise-and-Distrortion(SINAD)**

  Another important parameter of a ADC is the **SINAD** value. This is the ratio of the rms signal amplitude to the mean value of the root-sum-square (rss) of all other spectral components, including harmonics, but excluding DC. SINAD is a good indication of the overall dynamic performance of an ADC because it includes all components which make up noise and distortion[22].

21 [AD_ADC] SFDR
22 [AD_ADC] SINAD

- **ADC clock jitter**

  Last but not least, the topic of ADC clock jitter will be reviewed. This measure represents the time variation of the clock time period. If the clock edges are not stable then a poor **S**ignal-to-**N**oise-**R**atio (**SNR**) will be the result. According to [JIT], two types of jitter exist:

  - **Deterministic jitter**

    this is the effect of cross-coupling different signals, as well as overshoots of the signal

  - **Random jitter**

    Sources of this type of jitter are thermal noise and electron flow

The phase noise, introduced by an unstable ADC clock signal, does not keep the frequency stable and it varies around the fundamental value. This spectral distortion will be added to the spectrum of the sampled signal and will reduce the SNR. Also, the phase shift of the clock signal results in sampling the signal at different points, thus acquiring different amplitudes. In the case of a SDR application, as well as other communication applications, the importance of jitter-free sample clock will allow the correct digitizing of low-amplitude signals. If, on the other hand, jitter is present, then the spectrum of the sampled signal is distorted and information is lost.

The effect of ADC clock jitter can be seen in Chapter 6.2.2, where two different clock sources have been compared:

- **Rohde & Schwarz signal generator of type SMG**

  Generates a sine wave with 50% duty cycle.

- **Virtex5 FPGA internally generated clock signal**

  Generates a rectangular wave using a DCM component. The duty cycle of the signal varies between 45% and 50% which, as stated in the datasheet of the LTC2206 ADC, may result in conversion distortions. Moreover, the Xilinx CoreGenerator report estimates a period jitter of 0.174 nanoseconds. These constraints result in poor signal quality, compared to the results, measured with the generator's clock signal.

4 Hardware Implementation

This chapter summarizes the most important issue concerning the implementation process. Table 6 is an overview of the implementation flow for this project. Summarizes the implementation steps performed.

Implementation step	Description
Matlab/Simulink model of the DDC algorithm	The first step is to create a simulation model for the DDC component. Using the Xilinx blockset for Simulink, it is possible to evaluate the hardware performance of the components in the Simulink environment. The DDS block, together with the low-pass filter cascade, are configured.
Synthesis of the model using the Xilinx System Generator tool	After an adequate model is generated, it can be synthesized to create a hardware description template, which will be later included in the user IP core.
Base system builder design of the Microblaze system	A Microblaze system is created using the Xilinx EDK environment. This includes the soft-core microprocessor itself as well as all relevant peripherals, such as the USB 2.0 IP Core, the MPMC memory controller, Interrupt controller and debug interfaces.
Design of dedicated IP cores using the Xilinx Core Generator	The clock domain synchronization between the ADC clock and the FPGA internal reference clock is done using a dedicated FIFO, generated by this tool. Also, the DCM, which generates the evaluation clock signal for the ADC, is configured with this environment.
Development of the DDC IP Core	All of the above mentioned components are integrated in the Microblaze system as a user-defined IP core, labeled DDC IP Core. The System Generator component and the synchronization FIFO are inferred in this module and their behavior can be modified using memory-mapped software registers, accessible by the microprocessor. A hardware interrupt is generated by this core and must be handled in software. The DCM block is modeled as a separate IP Core because it is only used for evaluation.
Synthesis of the complete design	Finally, the complete design is synthesized and can be loaded in the FPGA.

Table 6: Overview of the hardware implementation steps

4.1 Hardware Configuration

In order to get familiar with the design, this section will provide an overview of the main components of the DDC project. A short summary of the selected FPGA as well as the ADC board and the ULPI physical interface chip, required for the USB packet transmission, should serve the reader as the basis for understanding the concept of the system.

4.1.1 ML507 Evaluation Board

The Xilinx ML507 FPGA board, depicted on Figure 4.1[23], is a highly versatile development board supplied with a number of interfaces. It can be used for high speed transmission designs as well as perform intense arithmetical computations. In the case of Direct Down Conversion, a system with fast multipliers as well as great memory capabilities is required.

Figure 4.1: ML507 Board

The ML507 board contains expansion headers for easy connection of external components. Different standards are supported by these and must be configured over the provided jumpers. Because of hte fact that both the DC918C demonstration circuit and the SMSC daughter card

23 Source http://www.impulseaccelerated.com/Tutorials/Xilinx/ML507/images/ML507.jpg , [29.12.2010]

must be connected to those, additional effort must be spent on selecting the proper IOSTANDARD for the signals. A closer look on the data sheets of the DC918C board and the SMSC 3300 XLX board shows that they operate on different CMOS Voltage levels. For the reason that the layout of the SMSC Daughter card is already configured with respect to the J5 connection of the ML507 board, the only suitable location for the DC 918C board would be on the connector J4. The configured pin locations are listed in Table 7.

J4 Connector pin number	FPGA pin	ADC pin
2	XXX	Not connected
4	L34	ADC clock out
6	XXX	Not connected
8	K33	ADC output bit 15 (MSB)
10	N32	ADC output bit 14
12	P32	ADC output bit 13
14	R34	ADC output bit 12
16	T33	ADC output bit 11
18	R32	ADC output bit 10
20	R33	ADC output bit 9
22	T34	ADC output bit 8
24	U33	ADC output bit 7
26	U31	ADC output bit 6
28	U32	ADC output bit 5
30	V33	ADC output bit 4
32	V32	ADC output bit 3
34	V34	ADC output bit 2
36	W34	ADC output bit 1
38	AA33	ADC output bit 0 (LSB)
40	Y33	Active high, enable the ADC, controlled by software

Table 7: Pin layout of the FPGA <-> ADC connection

The voltage level of all expansion headers is controlled using the J20 jumper and can be either 2.5V or 3.3V. Based on the fact that the ADC is the more sensitive component a decision to set the voltage level to 2.5V, which matches its output, has been done. Due to the high

operating frequency of the ADC board any uncertainties in the voltage levels may degrade the output signal. On the other side, the SMSC Daughter card does not show any malfunctioning using the 2.5V level instead of the specified 3.3V on its pins. Moreover, as it can be seen on Table 8, the supply voltage pins of the SMSC daughter card match the layout of the J5 connector and are not influenced by the overall pin configuration.

FPGA J5 connector pin number	SMSC pin	Voltage level
1	1	+5V
2	4	+5V
3	7	+5V
4	10	+5V
5	13	Not connected
6	16	+3.3V
7	19	+3.3V
8	22	+3.3V
9	25	+3.3V

Table 8: Pin layout voltage supply FPGA <-> SMSC daughter card

Lastly, the DCM output clock is routed to the differential pin pair H14, H15. The H15 is tied to ground and the single-ended output is forward to H14.

4.1.2 DC918C demonstration circuit

The DC918C board, as shown on Figure 4.2, comes with SMA connectors for both the ANALOG and ENC clock inputs. Both are matched with 50Ω so that no reflections occur. The converted value is forwarded to the 2.5V CMOS digital output pins which are connected to the ML507 J4 connector as listed in Table 7. The LTC2206 generates a digital clock out of the analog ENC clock input, which is then output on Pin 3 of the demonstration circuit. As a result, the clock signal can be fed in the FPGA and part of the design can be synchronized with respect to this signal. This way, the incoming data processing stream may be separated from the internal clock domain of the FPGA. If the FPGA design has more than one clock

domain, as is the case for this project, then clock domain synchronization must be performed to prevent metastability as well as preserve data consistency. For full specification of the DC918C board, please consider [DC918].

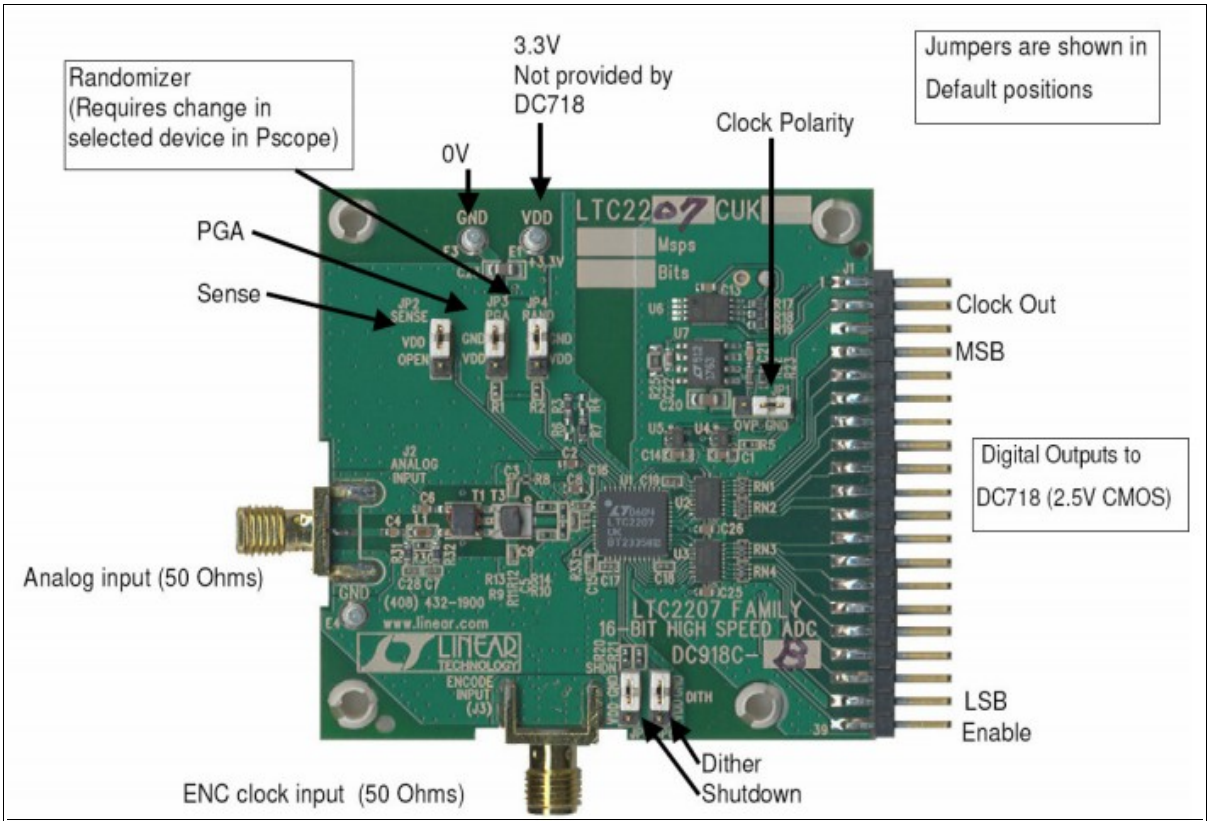


Figure 4.2: DC918C Board, source: [DC918]

4.1.3 SMSC EVB_USB_3300_XLX Transceiver

In order to obey the USB electrical standard, a physical interface is required to translate the messages to be transferred in electrical impulses as defined in [USB20]. In general, there are a number of ways to organize the data packets. One way is to use a dedicated microprocessor which handles the complete USB requests together with the frame organization. This technique is used by most of the example projects, presented in Chapter 2.3. Most of the projects make use of a Cypress controller, which handles the USB traffic. Another choice, as selected for this project, is to use the USB 2.0 Device Hardware IP Core provided with the Xilinx Embedded Development Kit, which handles the USB communication by generating special interrupts to the embedded microprocessor whenever a certain event on the USB bus has occurred. If the later approach is selected, a **SMSC EVB USB 3300** daughter card or a

similar device must be used as physical interface. The USB IP Core, described in Chapter 4.2.2.2, supports the **U**TMI(Universal Transceiver Macrocell Interface) + **L**ow **P**in **I**nterface(**ULPI**) interface [XPS_USB]. It communicates with the daughter card using the ULPI interface, which generates the bus protocol signals. The IP Core generates interrupts upon certain conditions, determined from the control and status signals of the communication link. The daughter card is an effective way of minimizing the development effort, because it carries out all low level protocol issues. The board is powered by the FPGA and voltage connections are listed in Table 8. Furthermore, the data and control signals, toured to the J6 connector of the FPGA, are summarized in .

J6 connector pin number	FPGA pin	SMSC J1 pin	Description
2	H33	3	Reset
4	F34	6	NXT
6		9	CLKOUT
8	G33	12	DIR
10	G32	15	STP
12	H32	18	DATA7
14	J32	21	DATA6
16	J34	24	DATA5
18	L33	27	DATA4
20	M32	30	DATA3
22	P34	33	DATA2
24	N34	36	DATA1
26	AA34	39	DATA0

Table 9: Pin layout FPGA <-> SMSC daughter card

4.2 Hardware synthesis

This chapter will provide an overview of the hardware synthesis process, which includes the design of the Matlab/Simulink model of the DDC as well as its integration in the Microblaze embedded system.

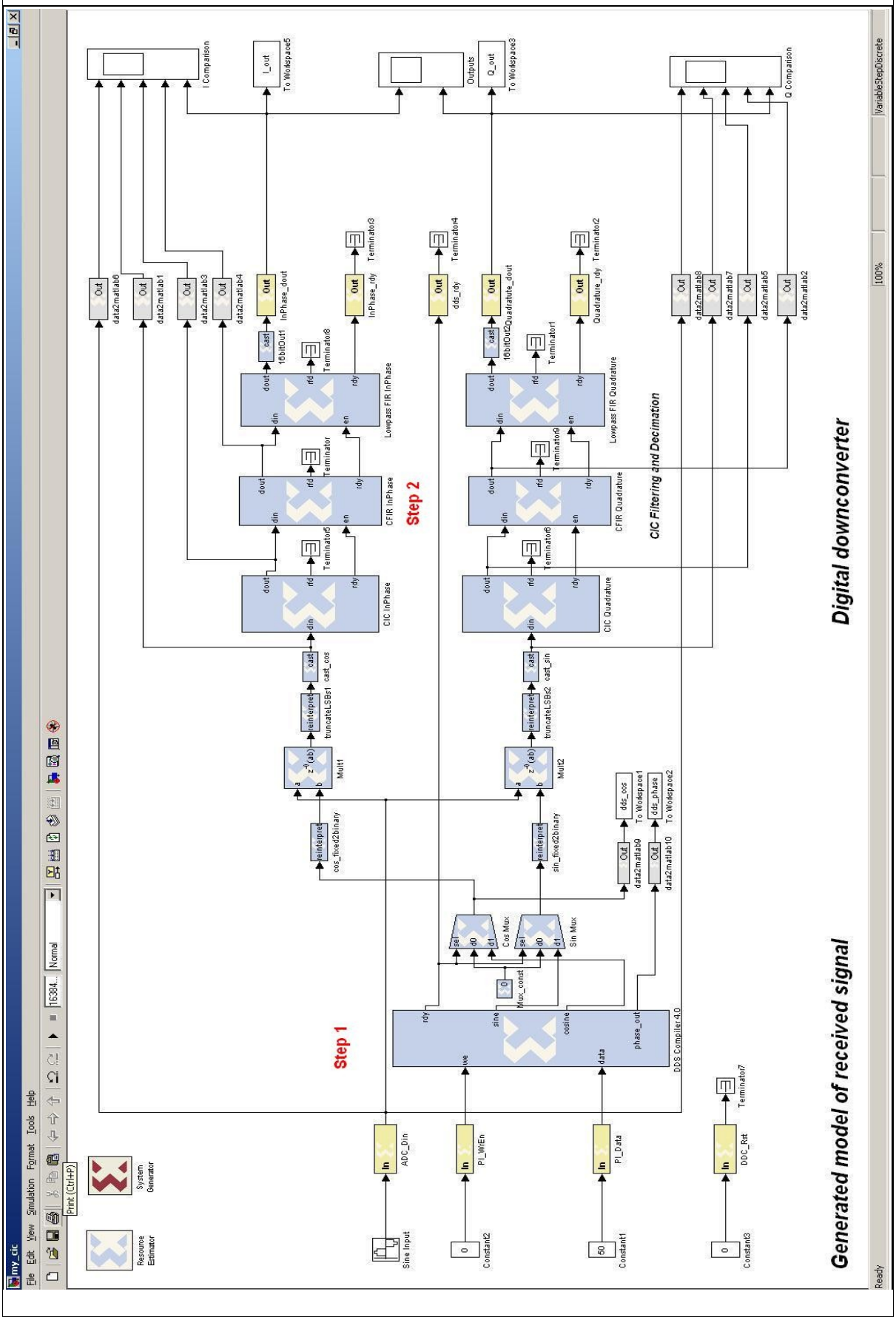
4.2.1 System Generator model of the DDC algorithm

An evaluation of the overall structure of the DDC model, shown in Figure 4.3, should point out the specific components. As already discussed in Chapter 3.2, three operations must be performed prior to storing the samples:

- I/Q signal generation and down-mixing, marked as Step 1
- low-pass filtering for channel selection and alias term rejection, marked as Step 2
- decimation

The I/Q signal generation is achieved by mixing the input sample with a sine/cosine pair.

Figure 4.3: Complete Model of the DDC Algorithm



4.2.1.1 CIC Filter

By substituting the desired rate change and attenuation level, listed in the project specifications from Chapter 2.4, R = 32, N = 5 and M = 2, the first spectral zero, computed with respect to the magnitude response description in Chapter 3.2.2, is expected at $f = \frac{1}{RM} = \frac{1}{64}$ of the sampling frequency F_S, equal to 1.25MHz. The number of stages, N, is set to the maximum value, available in the CIC Compiler V1.3 model. This, together with a differential delay M = 2, provides the highest possible attenuation. The magnitude response on Figure 4.4 clearly shows, that all spectral components, located to the right of this point are attenuated with more than 100 dB. In comparison, the CIC filter design, which has the differential delay configured to 1, provides a lower attenuation and wider main lobe.

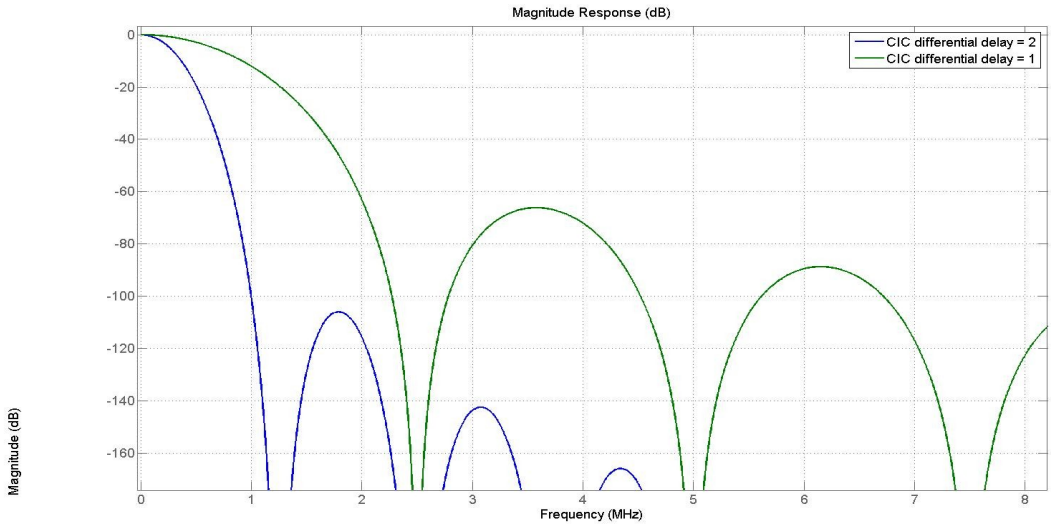


Figure 4.4: Estimation of the differential delay on the overall CIC magnitude response

4.2.1.2 Droop compensation and channel selection

The "*fdesign*" method of the Matlab Filter Design Toolbox offers a vast range of filter structures among which a "*ciccomp*", a CIC compensation filter design, which computes an equalizer with respect to the R, N and M parameters of a the filter. CIC filters present a

(sin$x$/x) profile in the passband and relatively wide transitions. To compensate for this fall off in the passband, and to try to reduce the width of the transition region, a CIC compensator filter can be used that demonstrates an (x/sin$x$) profile in the passband. By taking the number of sections, passband, and differential delay from the previously designed CIC filter and using them in the definition of the CIC compensator, the resulting compensator filter effectively corrects for the passband droop of the CIC filter, and narrows the transition region[24]. The compensation filter can also be designed as a decimator and by this introduce additional sampling rate reduction. Because of the fact that the CIC filter design is limited to a maximum of 5 stages, a decimation factor of two is integrated in the compensation filter.

The final component of the filter cascade, shown on Figure 4.3, is used to implement the channel selection and the final level of attenuation. This way the final stage operates at the lowest sampling frequency. This will reduce the filter length, guarantee shorter transition band and optimize resource utilization. The proposed solution with 3-stage filter cascade reduces the transition band of the design by 42,3% at the expense of one embedded multiplier and 40 additional Slices. As seen on Figure 4.5, the transition band shrinks from 39 KHz to 22.5 KHz. Table 10 summarizes the resource utilization of both cascade component.

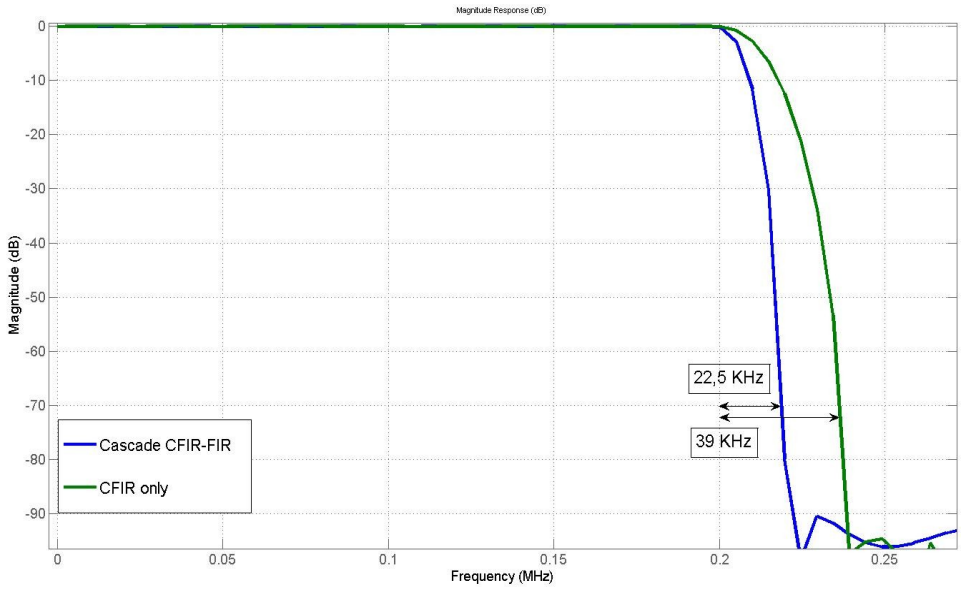


Figure 4.5: Transition band reduction

24 Functional description provided by the Filter Design Toolbox

Component	Slices	FlipFlops	BRAMs	LUTs	Embedded Multipliers
Single CFIR	175	293	0	312	3
CFIR-FIR	218	459	0	442	4

Table 10: Resource Estimation for the Filter Cascade excluding the CIC Filter

4.2.1.3 DDC Model Summary

As shown on Figure 4.6 the CIC↔CFIR↔FIR filter cascade fulfills the desired design specifications listed in Chapter 2.4. The overall magnitude response, marked as "DDC Cascade", provides the required level of attenuation and transition band size. The passband ripple of the filter chain is depicted on Figure 4.7 and is less then the specified 0.1 dB. Last but not least, the desired linear phase response, shown on Figure 4.8, ensures the error-free demodulation of the signal.

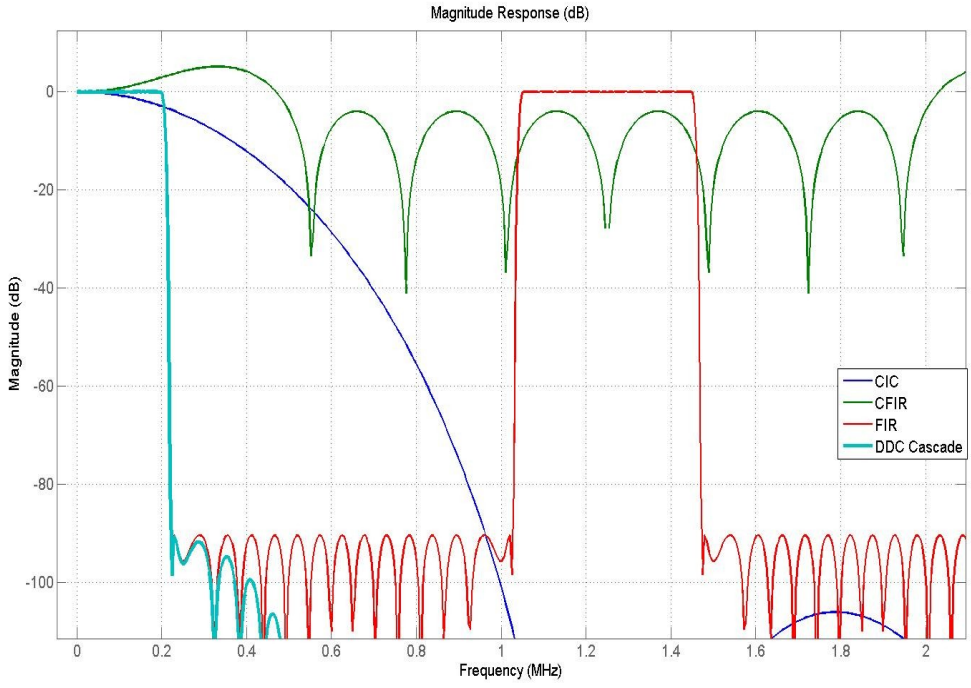


Figure 4.6: Overall Frequency Response of the filter cascade

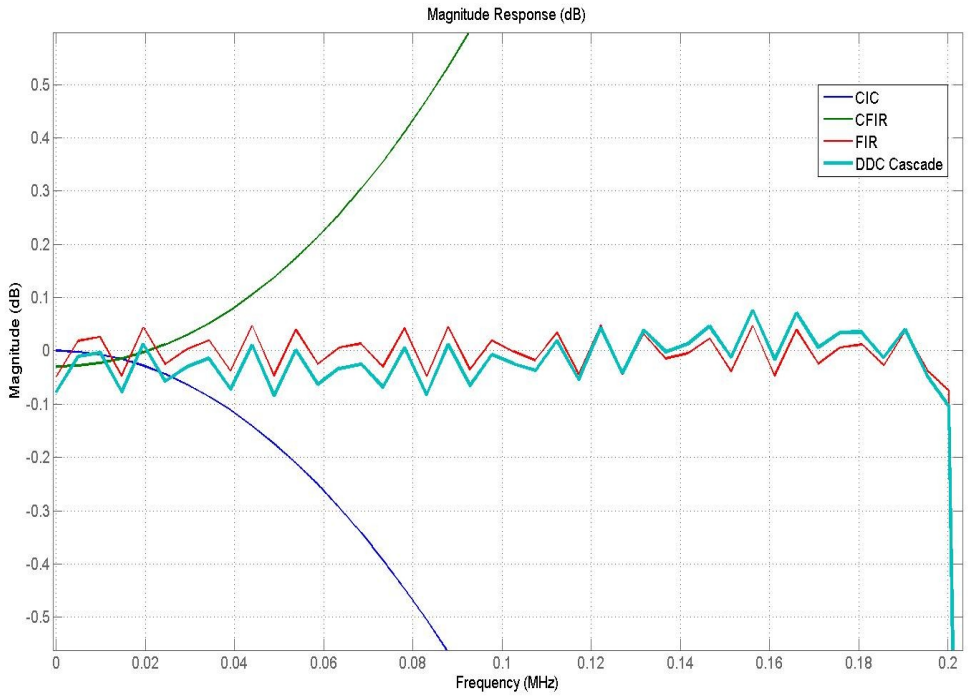


Figure 4.7: Passband ripple of the filter cascade

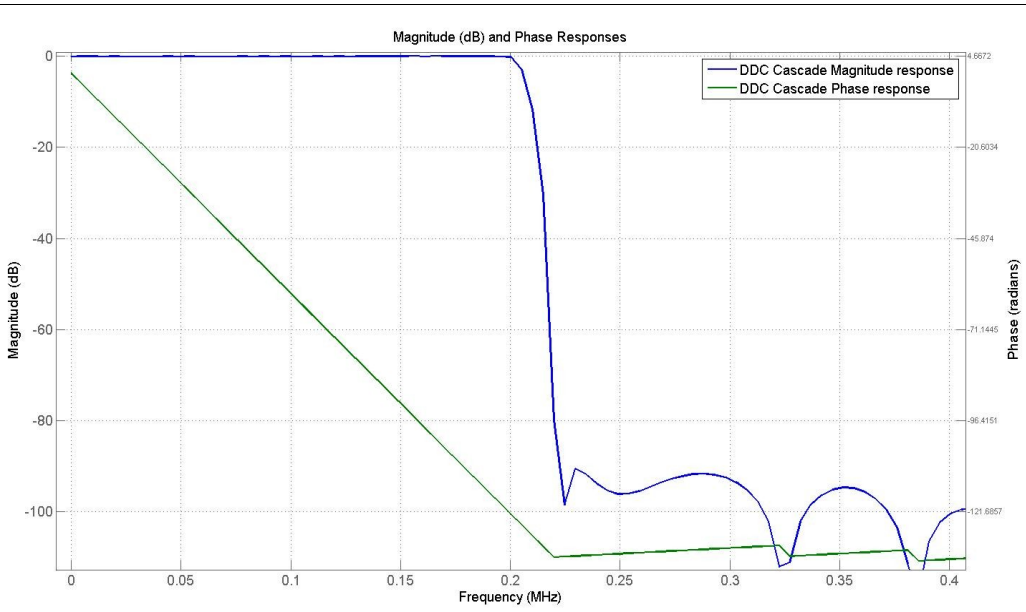


Figure 4.8: DDC filter cascade Magnitude and Phase response

4.2.2 Base system builder and User-IP core integration

Figure 4.9 depicts the main components of the DDC embedded system. The project consists of several Xilinx-specific cores as well as the self-developed DDC IP core and DCM DRP module. The marked data path outlines that the memory-write process is independent of the Microblaze. For this reason, the microprocessor executes the USB firmware and moves the samples from the memory controller to the internal dual-ported, DP, BRAM component of the USB IP core. Furthermore, the Microblaze controls the DDC core using software registers, which hold different parameters. Table 15, Table 16 and Table 17 contain an extensive description of them.

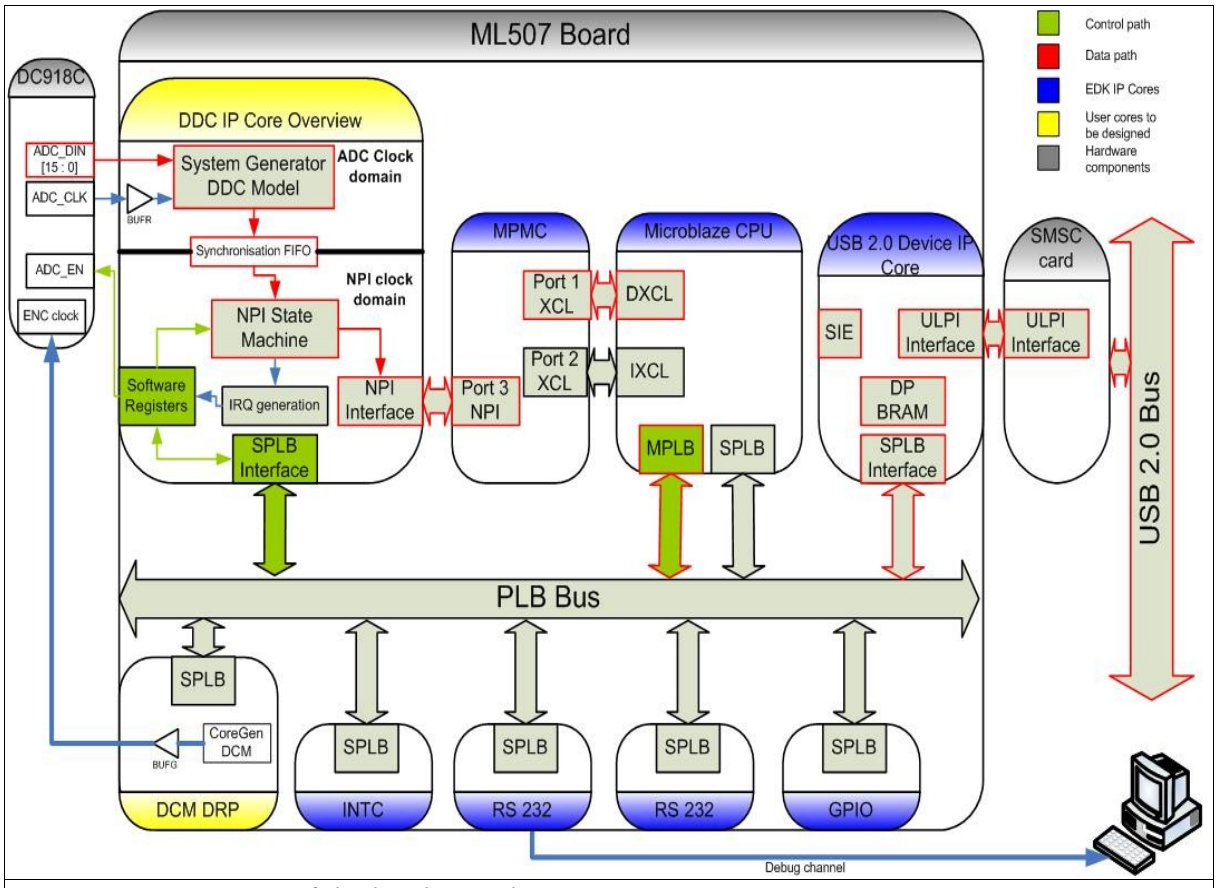


Figure 4.9: Overview of the hardware design

The functionality of each IP core is listed in Table 11. Besides this, the MPMC, USB IP core and DDC IP core will be extensively analyzed in following sections.

IP Core	Description
Microblaze soft-core CPU	The microprocessor executes the USB firmware and handles the read requests, done by the USB IP core
MPMC	The Multi Port Memory Controller (MPMC) provides access to the 256 MB DDR2 memory, available on the ML507 board. 3 out of the 6 ports are configured to allow simultaneous access of multiple devices. Chapter 4.2.2.1 provides more information about the structure of this core.
USB 2.0 Device IP core	This IP core translates the received ULPI packets from the SMSC daughter card and handles them. Also, it requests the Microblaze to move the next block of samples, which is to be sent over the USB. For more information consider Chapter 4.2.2.2
DDC IP core	This core is generated using the "Create or Import peripheral" wizard of EDK. The DDC model, as well as the later discussed NPI core, are instantiated here. Also, the clock domain synchronization is performed in this module. Refer to Chapter 4.2.2.3 for more elaborate overview.
DCM DRP	The CoreGenerator DCM core is inferred in this IP and can be configured for different output frequencies. It outputs the test ADC clock.
INTC	The interrupt controller handles multiple IRQ requests and prioritize them. Due to the fact that the Microblaze has only one interrupt port, this core queues the pending IRQs until the microprocessor is ready to handle them.
RS232	This interface has been used for debug purposes during the software development.
Timer	This interface has been used for debug purposes during the software development.
GPIO	A number of GPIO modules, such as LEDs, buttons and an LCD display have been instantiated for debug purposes

Table 11: IP cores, instantiated in this project

4.2.2.1 Multi Port Memory Controller

The the **M**ulti**P**ort **M**emory **C**ontroller(**MPMC**) IP core interfaces the 256 MB DDR2 memory, available on the ML507 board. This IP Core supports up to 6 different subscribers, requesting data independently of each other. Moreover, different Bus interfaces are offered for design flexibility. The following two types enhance the memory managemnet of the project:

- **Xilinx CacheLink (XCL)**

  If the Microblaze makes use of data and instruction caches for accessing an external memory chip then the dedicated **X**ilinx **C**ache**L**ink(**XCL**) interface may be synthesized and by this separating the data flow from the PLB Bus, which connect all peripherals to the processor, reference Figure 4.9. Since this project requires large memory blocks to be manipulated by the CPU, it may be wise to spare resources on cache interfaces. The cache feature should be used with caution, because it consumes a number of BlockRAM elements. Sometimes it is reasonable to evaluate the resource requirements because in particular cases it may be easier and cheaper in terms of BlockRAMs to extend the internal memory than synthesizing caches.

- **MPMC Native Port interface (NPI)**

  Besides the XCL interface, the MPMC offers the **N**ative **P**ort **I**nterface(**NPI**). This is an easy to implement interface which provides low latency read/write memory requests. Since the incoming data stream will be at rate of some megabits per second, the user IP core may implement it and by means of NPI transactions write the samples without the need of the Microblaze. If, on the other side, Microblaze is involved, then an interrupt must be generated out of the user core to signal new data. The interrupt response has a specific latency which may lead to data loss, if not enough processing time for the **I**nterrupt **S**ervice **R**outine (**ISR**) is available. Furthermore, the interrupt rate will be considerably high and this is not desirable. The user IP core will use the NPI interface to perform similar behavior as a **D**irect **M**emory **A**ccess (**DMA**) device.

Before discussing the implementation of the NPI interface, there are several points which need to be mentioned regarding it:

- **Ratio between the NPI core clock and the MPMC clock**

  As described in [MPMC], an important design restriction is the allowed **clock ratio** between the MPMC memory clock and the NPI interface clock. In comparison to the PLB bus interface of the MPMC, the core, using a NPI interface to the MPMC, must run at a 1:1 clock ratio with respect to the clock specified at the **MPMC_Clk0** port of the MPMC core. For this reason, the user IP core must provide a dedicated port for feeding the clock source and synchronizing the NPI interface implementation to it.

- **Data width of the NPI interface**

  Another point to discuss would be the NPI data width used throughout the design. The MPMC Version 5.04.a allows two different values for this parameter, namely a 32- or 64-bit implementation. If the interface is configured for 64-bit operation then the implementation would consume more FPGA resources then the 32-bit version. As the width of an ADC sample is 16 bit and the InPhase and Quadrature components both result in two 16-bit values, it may be beneficial to implement the NPI size as 32-bit version. This will eventually allow to combine the two outputs to a four byte value and use it as one data portion of the NPI write cycle. With this in mind, the state machine, which will implement the NPI interface, can be designed without wait states for data collection. If, on the other side, a 64-bit data width is selected, a state for buffering the samples before placing them on the data line of the NPI bus must be inserted.

- **NPI transcation type**

  Finally, the type of NPI transaction must be selected. The MPMC core allows variable data packet sizes among which a Byte, Half-Word, Word as well as Burst transfers. Furthermore, 8-Word Cacheline Write transaction is allowed. This approach pushes 8 data packets in the Write FIFO of the corresponding MPMC port and generates an Address Request with the last push. A sample 8-Word Cacheline Transaction is depicted on Figure 4.10[25].

25 Image source: [MPMC]

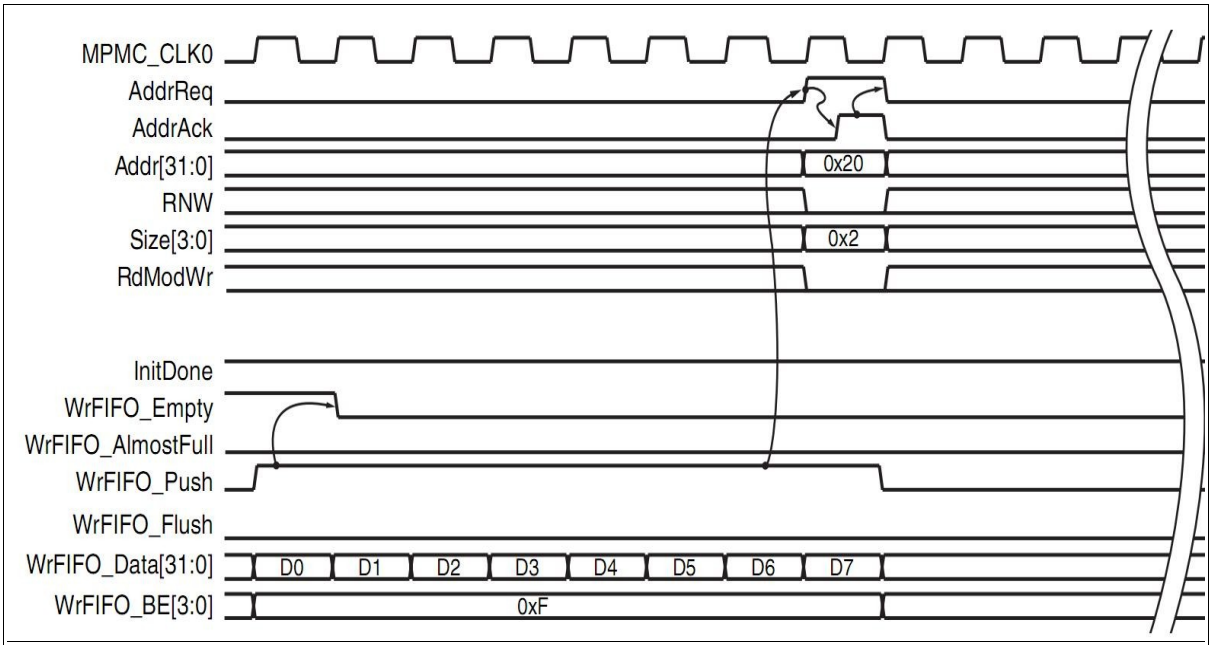


Figure 4.10: 8-Word Write Cacheline Transaction

In order to implement the NPI interface in VHDL, an example project, such as the [AR24912] has been used as reference. A simple NPI Read/Write state machine implements the signal timings, shown on Figure 4.10. For the reason that the DDC core only writes in the memory there is no need for a Read support, only the signals, relevant to the Write process, are synthesized and connected to the MPMC. The state machine automatically processes the incoming data. Among the NPI state machine, which will be explained in detail, this core handles the following tasks:

Task	Description
Memory address management	Because of the fact that the NPI interface is not supposed to use the complete DDR2 memory, a region limit must be selected. This is done using the software registers of the DDC IP core. The start address, as well as the size of the total memory available, are provided to the core. The address is incremented with 32 bytes, 8 values times 4 bytes each, after every successful NPI 8-word cacheline transaction. This project uses 20 MB of the DDR2 memory which is divided into 4 blocks of 5 MB packets. Each packet corresponds to a buffer time of nearly 1 second. The memory is organized as a ring buffer and as soon as the last possible address, is written the write pointer wraps around and starts at the beginning of the NPI memory block. This way the desktop application must be able to read each block in less than 3 seconds. Every increase in the output sampling frequency of the DDC model must be carefully examined because it will shorten the write time of each IRQ block and by this reducing the overall readout time available for the desktop application.
Interrupt generation	The NPI block is responsible for the interrupt generation of the DDC model. This is due to the fact that the complete address computation is managed in this component. The purpose of the generated interrupt is to announce to the Microblaze that a certain amount of data is already written in memory. To reduce the interrupt rate, a request is set when a 32-KB block has been written. The IRQ is generated directly from the internal address counter of the core. Each address line is used as an input to a number of LUTs which logically determine if the address has reached a 32-KB boundary. As a result, an IRQ is flagged every $(\frac{1}{(80e6/64)})*(\frac{32768}{(4\,Bytes)})=6.5536\,ms$. The Microblaze waits until a total of 10240 blocks (5MB / 10240 sectors = 512 bytes per sector) have been written and then issues a USB transfer. This way the protocol overhead, generated during the initialization phase of the transfer, is performed only once.
Link to the synchronization FIFO	The NPI state machine generates a READ_ENABLE signal, which requests data from the synchronization FIFO. The fifo_valid_flag Figure 4.13, is used to test if the current FIFO output should be processed.

Table 12: Summary of the NPI core

The structure of the VHDL component, responsible for the NPI timings, can be found on Figure 4.11. As it can be seen, a state machine is used to control the sub components such as the IRQ generator, address generator and the sample counter.

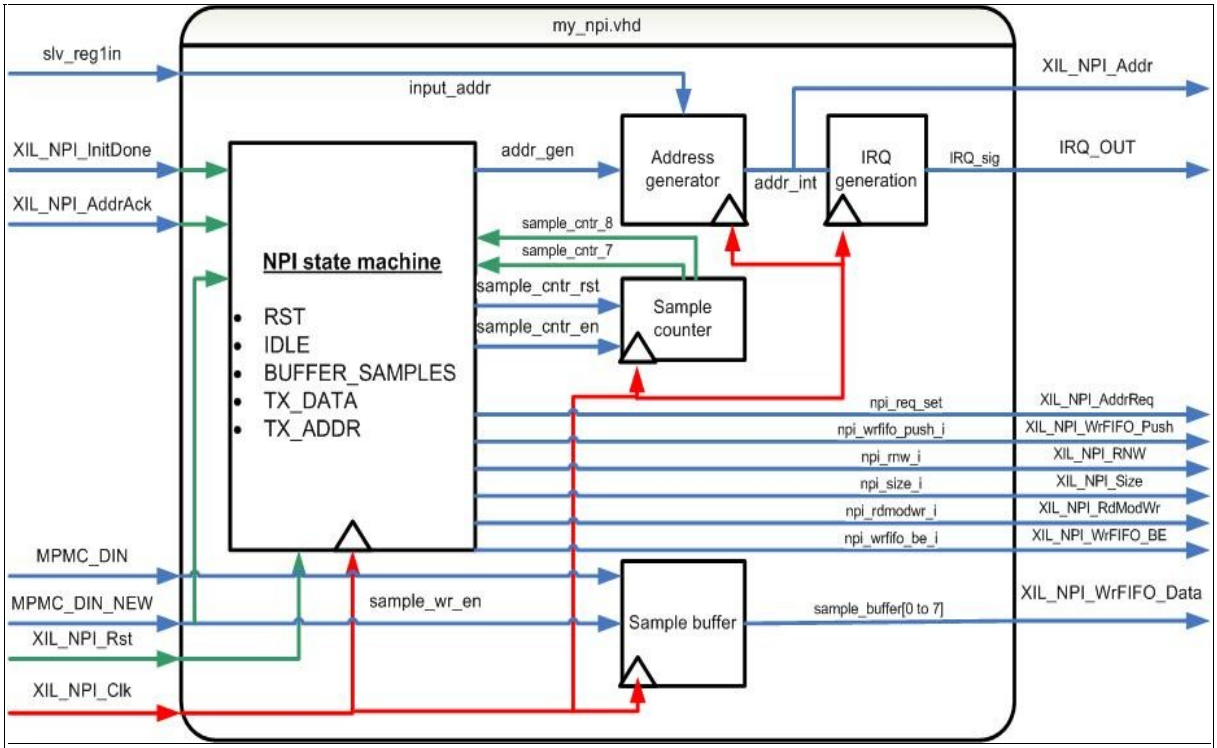


Figure 4.11: NPI core diagram

Table 13 summarizes the tasks of each block.

Component	Description
address generator	This process generates the next valid NPI address after a successful transaction has been made. It takes the offset value, stored in „slv_reg1in“, which is the beginning of the data storage region.
IRQ generator	This process observes the address vector and an IRQ is generated whenever a 32KB boundary is reached.
sample counter	This element is used during the „BUFFER_SAMPLES“ and „TX_DATA“ states. It counts the received samples as well as how much data is pushed in the MPMC FIFO.
sample buffer	This array buffers the samples, coming from the synchronization FIFO until a total number of 8 is received. If this condition is met, the NPI state machine starts a transaction.

Table 13: Component description

Furthermore, Table 14 contains the description of the NPI transition states.

State	Description
RST	Reset state, the state machine enters it if the DDC core is disabled or the MPMC is not ready.
IDLE	Wait state for the case that the system is initialized but no information is found in the synchronization FIFO.
BUFFER_SAMPLES	This state requests data from the synchronization FIFO until 8 samples have been received. If this is the case then a NPI transaction is initiated
TX_DATA	During this state, the data to be written in memory is pushed in the MPMC FIFOs. At the end, an „address request“ is initiated.
TX_ADDR	The system awaits an address acknowledge and if successful, returns in the IDLE state.

Table 14: NPI state description

4.2.2.2 XPS Universal Serial Bus 2.0 IP Core

Another component of interest is the XPS Universal Serial Bus 2.0 Device v2.00a IP Core. This core is fully compliant with the USB Specification and supports High Speed as well as Full Speed USB. As already discussed, an ULPI interface creates a link to an ULPI compliant USB 2.0 PHY device, such as the SMSC daughter card, and enables the translation of the data packets to the correct USB electrical characteristics. A total of eight endpoints are supported, whereas endpoint 0 is preserved as Control Endpoint. Double buffering is included for endpoints 1 to 7. A **D**irect **M**emory **A**ccess(**DMA**) may be also configured.

Figure 4.12[26] represents the internal organization of the USB IP Core. A dual ported BlockRAM, marked as DPRAM, contains all frames which are currently processed. The Microblaze can access the memory over the PLB Bus by using **Port B**, whereas the USB 2.0 **S**erial **I**nterface **E**ngine(**SIE**) uses the **Port A**.

26 Image source: [XPS_USB]

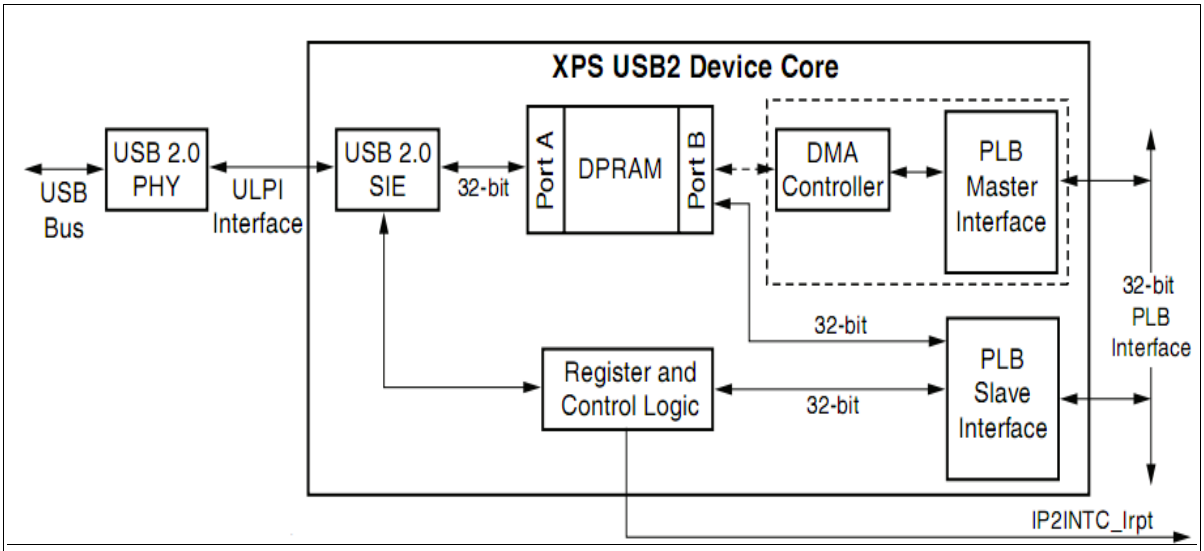


Figure 4.12: USB core structure

The SIE component is responsible for the serialization and de-serialization of USB traffic at the byte level and multiplexing and demultiplexing of USB data to and from the endpoints of the core. The SIE also handles USB 2.0 state transitions, such as suspend, resume, USB reset and remote wake-up signaling. The SIE interfaces to the PHY using a ULPI interface that requires 12 pins. Data to the FPGA from the USB is received from the PHY, error checked and loaded into the appropriate area of the DPRAM. Data from the FPGA that is to be sent over the USB, is loaded from the DPRAM, protocol wrapped, then when the protocol allows, presented to the PHY, one byte at a time. The status of the current USB transactions is signaled by the SIE to the Interrupt Status Register. The primary job of the PHY is to manage the bit level serialization and de-serialization of USB 2.0 traffic. In order to meet the USB requirements of 480 Mb/s, the PHY interface operates on a byte-level and uses a 60 MHz clock..

4.2.2.3 DDC IP Core

This section describes the design of the User IP Core which performs the DDC algorithm and handles the NPI memory transactions. Besides this, a Slave PLB interface ensures that status information as well as control flags can be manipulated by the Microblaze microprocessor using Software Registers. Figure 4.9 gave already a detailed overview of the communication

links between the main components in the design. As it can be seen from the drawing, the write process is independent of the read process. The DDC Core performs the complete signal preprocessing and writes the samples over the NPI Interface in the DDR2 Memory. In comparison, the Microblaze communicates with the USB 2.0 IP Core over the PLB Bus and transfers the data blocks over it. To acquire the data from the memory, both the Data and Instruction memory paths of the microprocessor are cached. To sum up, there are three independent paths between the cores which simplify the data transfer:

- DDC IP Core and MPMC using the NPI interface
- Microblaze and MPMC using Xilinx CacheLink interface
- Microblaze and USB 2.0 IP Core using the PLB Bus

The separation of the different data paths permits high data rates and prohibit bus collisions.

Figure 4.13 should provide a deeper evaluation of the DDC core functionality.

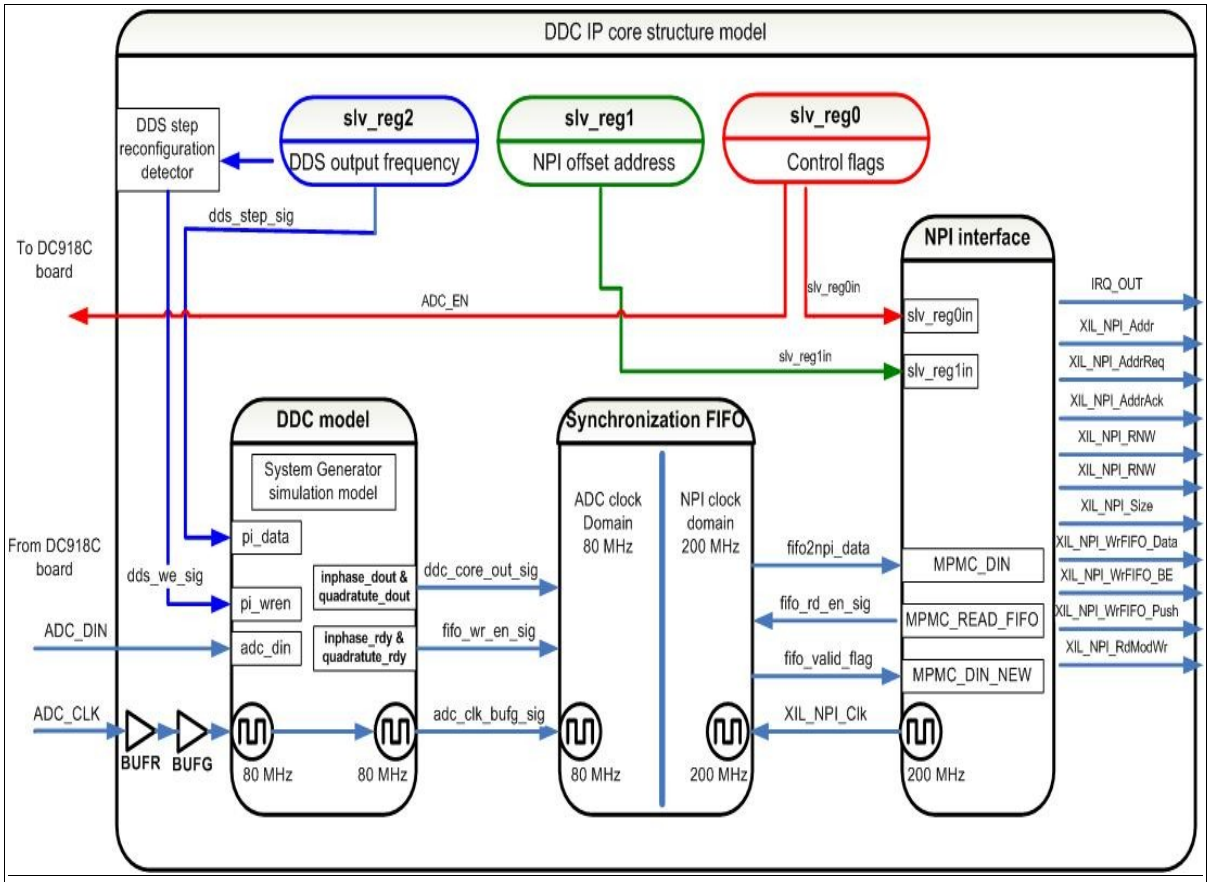


Figure 4.13: DDC IP core overview

After the processed sample leaves the System Generator DDC component, it must be stored in memory. Because of the fact that the ADC and the Microblaze run at different clock frequencies, both clock domains must be synchronized at this point. As proposed in [CLK_DOM], one possibility to solve this problem is to use an Asynchronous **F**irst-**I**n-**F**irst-**O**ut(**FIFO**) component. This way two independent clock regions can write data synchronous to the relevant clock and all synchronization issues are carried by the FIFO itself. Different flags provide information of the FIFO status in both domains and new read/write transactions must take these into account. Some remarks should be made with respect to the show diagram:

- As it can be seen from Figure 4.13, the core logic of the DDC block provides the "fifo_wr_en_sig" signal as well as the data to be written in the FIFO, whereas the NPI Core logic generates the "fifo_rd_en_sig" pulses and performs the write memory requests. Consequently, the NPI block wraps the incoming samples as shown on Figure 4.10 and routes the data to the MPMC. The memory controller then serves the request and the data is written on the provided address. Immediately after this the Microblaze can access the memory region and fetch the data for further processing.

- With these assumptions in mind, the last step to perform would be to notify the Microblaze microprocessor of the available data block. To reduce the sample rate of the system only a block of samples would be reported to the CPU instead of a single value. Moreover, it is better to transfer larger blocks of information so that the USB Bus negotiation is done only once. If the Microblaze receives an IRQ request coming form the DDC IP core then the corresponding ISR must update all relevant flags together with the value of the current sector being written. Due to the fact that one USB transfer block is larger than a single IRQ write size, it is important to buffer the data until a complete USB block is ready. Then, while buffering the next USB block, the already complete information packet is sent over the USB to the desktop PC. This ring buffer concept[27] allows for optimal memory utilization. Instead of reserving the complete DDR2 memory for the DDC Application, it is possible to allocate just a piece of it, 20MB in this case, and by dividing it into 4 sectors implement a ring

27 More information in [LDT], Chapter "FIFO Speicher"

buffer. This concept would work iff the desktop application is capable of reading a block before the write pointer of the ring buffer has reached its region again.

- For the sake of flexible software control a set of software registers, marked as "slv_reg0", "slv_reg1" and "slv_reg2" on Figure 4.13, which contain the most important DDC IP Core parameters, will be synthesized so that the Microblaze CPU can modify them. The DDC block starts operating as soon as the ADC is enabled over slv_reg0. The first step is to read a sample and send it through the DDC model. As next, the output value is written in the synchronization FIFO and after a certain amount of cycles the FIFO flags are updated. The NPI state machine is constantly generating a **read strobe** using the "fifo_rd_en_sig" and the "fifo_valid_sig" is used to determine the data validity. A counter is used to count the number of samples already buffered by the NPI state machine. If a block of 8 samples is available then a NPI transaction is issued to the memory controller and the NPI state machine recovers in the IDLE state. Furthermore the destination address of the NPI is increase by, 8 pairs of samples times 4 byte, 32 bytes so that the next transaction is correctly aligned. Subsequently, the interrupt generation logic observes the address counter and, based on the address value, produces an interrupt pulse if a 32KB boundary has been reached. By managing a software counter, the firmware is capable of evaluating how much memory has been already written.

The block diagram on Figure 4.14 depicts the introduced flow concept. As long as the DC918C board is enabled, the system is running and data is stored in the DDR2 memory. No flow-control is implemented between the hardware system and the desktop application and any data, not read on-time, will be overwritten after the ring buffer write pointer overflows.

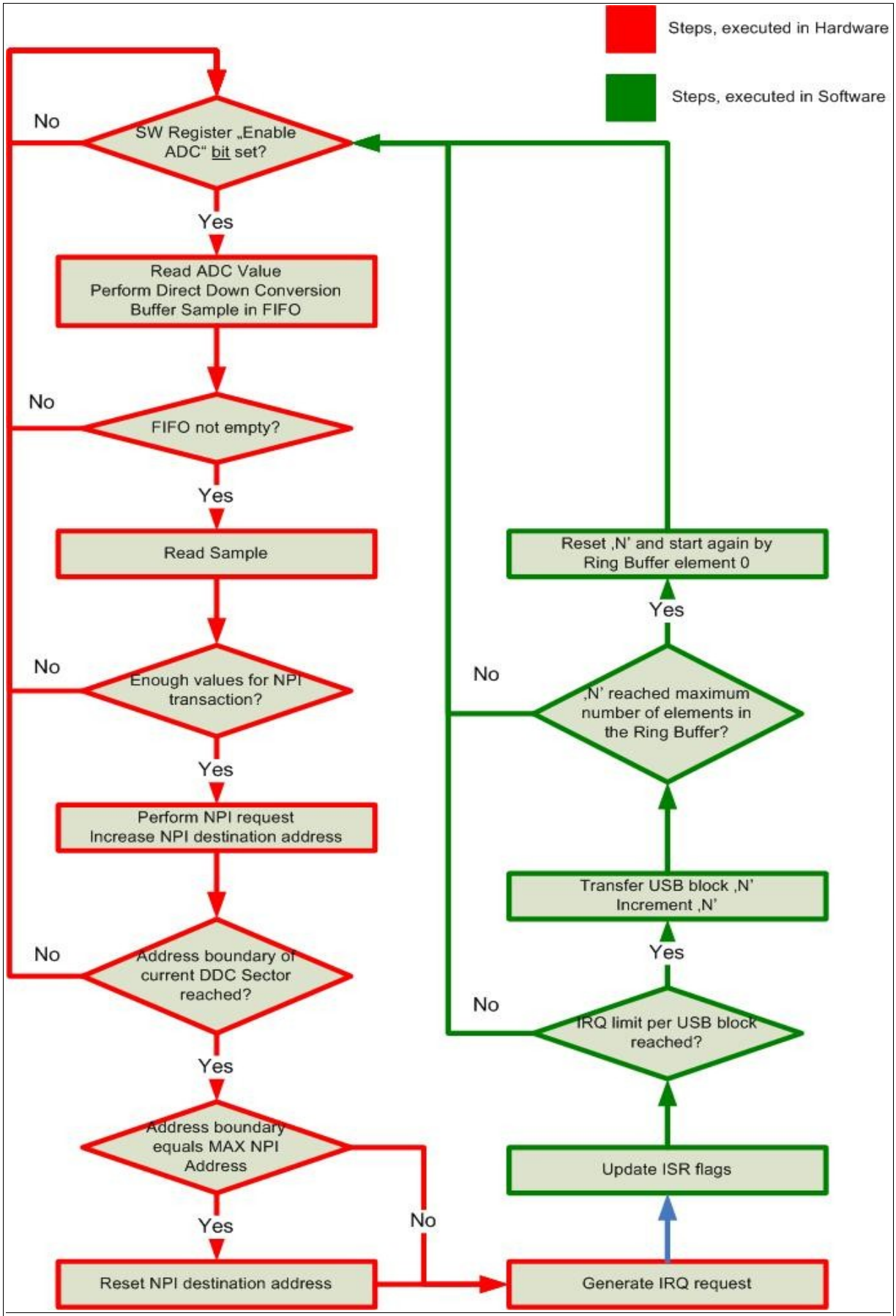


Figure 4.14: Flow concept of the DDC IP Core

- **Device Register Description**

The ddc_core has been synthesized with three 32-bit software registers which hold important DDC configuration parameters. Table 15, Table 16 and Table 17 contain a description of the register flags.

Bit	Name	Reset Value	Description
0-15	FAT Data Sectors	0	Provides the maximum address of the FAT Table, the NPI interface uses this value by the address generation
16-29	Reserved	0	
30	DDS synced	0	This bit is used to check if the DDS has recovered after being reconfigured
31	ADC Enable	0	Use this bit to enable/disable the DC918C board. When this bit is set, the DDC core starts operation and data is buffered in the DDR2 memory

Table 15: Software Register 1

Bit	Name	Reset Value	Description
0-31	Start address of the storage space in the DDR2 memory	0	This value provides the start address of the memory buffer. The samples are written and all previous data located on this address is **LOST**! The NPI interface uses this value together with the "FAT Data Sectors" to manage the NPI transactions and to generate an Interrupt whenever a certain amount of memory has been written.

Table 16: Software Register 2

Bit	Name	Reset Value	Description
0-31	DDS output frequency configuration	0	This value in **Hertz** provides the DDS output frequency which is to be configured. A reconfiguration of the DDS block is issued as soon as a new value is written to this field. The corresponding phase increment value is calculated and a WriteEnable pulse is sent to the DDS Block in the SystemGenerator model. The "DDS synced" field in Register 1 may be used to determine if the DDS output is stable.

Table 17: Software Register 3

- **ADC Clock integration and Clock domain crossing**

Due to the fact that the ADC Clock is seen as external source from the FPGA point of view, special considerations must be done to ensure the proper integration in the design. Generally speaking, the internal FPGA structure contains a global clock tree which is used to distribute the clock signal(s) over the complete chip. During the implementation step, **P**lace **A**nd **R**oute(**PAR**) collects all the information regarding the clock structure and then tries to route the design. As already mentioned, the ADC_CLK signal is fed into the FPGA from an external chip and must be properly attached to the global clock tree. As described in Chapter 1 "Clock Resources" of [UG190], there are two different clocking sources independent of the global clock network:

- **BUFR**

  All regional clock networks are independent of the global one and the regional clock source, BUFR, spans over only three regions. The usage of a BUFR is suitable for source-synchronization designs, where part of the implemented logic must be driven by the external source.

- **BUFIO**

  The I/O clock, which is the alternative to the BUFR, is limited to driving a single region only. A BUFIO component is restricted to only driving I/O Logic. This is due to the fact that the I/O clock network only reaches the I/O column in the same bank or clock region.

One constraint, which must be met in order to be able to integrate the external clock source, is that the signal must be routed to a "Clock Capable I/O" pin. Those specific pins are the only way to access a clock region of the FPGA with an external signal. Because of the fact that the clock source will be used to drive some logic components of the FPGA, a regional clock buffer, BUFR, must be instantiated. Figure 4.15[28] shows the main differences between the BUFR and BUFIO components. Once integrated in the FPGA , the BUFR clock can be spread to control the logic in the adjacent regions.

28 Image source: [UG190]

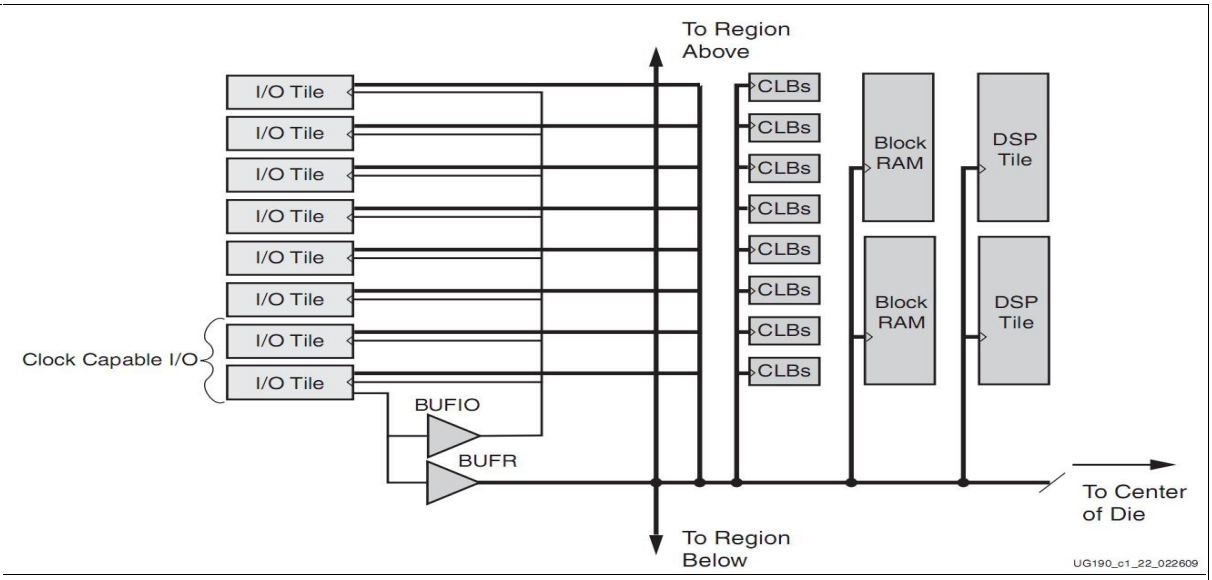


Figure 4.15: Regional Clock Buffers

- **MPD configuration**

After the VHDL implementation is accomplished, the interface information of the DDC IP core must be updated. Because of the fact that during the invocation of the "Create or Import Peripheral" Wizard no information is provided regarding the input/output ports of the system, a way must be found to make the configurations of the IP core global to the Microblaze system. This can be done by modifying the default **M**icroprocessor **P**eripheral **D**efinition(**MPD**) file. As described in [PSF] the MPD file lists the ports and default connectivity for the bus interfaces as well as provide different parameters and default values. Those values can be diverse VHDL generics which can be used to control the synthesis .

Although the PLB interface is already defined in the MPD file of the IP core, it is still required to list the NPI interface support. This way the EDK system will know that this IP is capable of connecting to a device with this kind of bus interface. The definition is done using the following notation:

BUS_INTERFACE BUS = **SPLB,** BUS_STD = **PLBV46,** BUS_TYPE = **SLAVE**
BUS_INTERFACE BUS = MPMC_PIM1, BUS_STD = XIL_NPI, BUS_TYPE = INITIATOR

Table 18 gives an overview of the MPD parameters and how exactly the hardware connection takes place.

Hardware IP core	BUS macro, as defined in the corresponding MPD file	BUS_STD macro, as defined in the corresponding MPD file	BUS_TYPE macro, as defined in the corresponding MPD file
MPMC[29]	MPMC_PIM1	XIL_NPI	TARGET
DDC IP Core[30]	MPMC_PIM1	XIL_NPI	INITIATOR

Table 18: Overview of the NPI interface connection configuration between the MPMC and the DDC cores

After the bus interfaces are configured, all port signals of the bus architecture must be listed and linked. As it can be seen in the following MPD code snippet[30] it is possible to define different types of ports and configure them with respect to the desired behavior.

```
...
PORT ADC_CLK = "", DIR = I, SIGIS = CLK
PORT ADC_DIN = "", DIR = I, VEC = [15:0]
PORT ADC_EN_OUT = "", DIR = O
PORT XIL_NPI_clk = "", DIR = I, SIGIS = CLK
PORT XIL_NPI_rst = "", DIR = I
PORT XIL_NPI_InitDone = InitDone, DIR = I, BUS = MPMC_PIM1
PORT XIL_NPI_Addr = Addr, DIR = O, VEC = [(C_PI_ADDR_WIDTH-1):0], BUS = MPMC_PIM1
...
```

One way of connecting a signal is to leave the MPD definition empty and then manipulate it using the EDK user interface. Another way is to use the BUS command and specify the bus label, which should absorb this signal. For the case of the XIL_NPI_InitDone signal, for example, the bus label is set to MPMC_PIM1 which means that it will be automatically associated with the corresponding signal from the bus specification. Hence, the connection is established in the background. All NPI relevant signals coming from the DDC IP Core must be configured to match the corresponding MPMC definition. The XIL_NPI_Clk signal is left

29 Information taken from mpmc_v2_1_0.mpd
30 Information taken from ddc_core_v2_1_0.mpd

unconnected due to the fact that it must be the same as the MPMC_Clk0 configuration. This association has been done using the EDK GUI. Finally, the ADC_CLK and ADC_DIN signals must be configured as external and all relevant pins must be listed in the UCF file of the project.

4.2.3 Design summary

As it can be seen from the values in Table 19, the design consumes a moderate part of the Virtex5 FPGA hardware resources available on the FPGA-chip. Furthermore, due to the extensive structure of the filters and the mixer, the DDC IP Core itself absorbs 11 of the 14 DSP48E macros that are used in the design.

Resource Type	Slice Registers	Slice LUTs	BlockRAMs	BUFG	DSP48E
DDC_IP_Core	3207	2313	3	1	11
MPMC	4525	2791	15	0	0
XPS_USB_IP	740	1770	4	0	0
Microblaze	1644	1629	4	0	3
lmb_bram	0	0	16	0	0
Entire Design	11754	9976	42	14	14
Available in the Virtex5 FPGA	44800	44800	148	32	128
Consumed in %	26	22	28	43	10

Table 19: Resource Consumption

Also, the design consumes 42 embedded BlockRAMs. This is due to the fact that a larger microprocessor internal memory has been synthesized. In fact, 16 BRAMs are consumed by the Microblaze so that the complete execution code of the application can be located there. This allows faster execution and removes the need of a boot-loader. In addition, the MPMC instantiates 15 embedded BRAMs, which are mostly used for the Microblaze CacheLink Interface. Moreover, the complete array, managed by the file system, is allocated in the DDR2 memory, and thus the expansion of this sector does not concern the FPGA fabrics.

5 Software Implementation

Main topics of this chapter will be the design of a file-system as well as its integration in the Mass Storage Support example project, provided with the Xilinx EDK. Also, the Windows application for data acquisition from the FPGA will be presented.

5.1 USB Firmware extension

Figure 5.1 shows an overview of the DDC software application. As it can be seen, the Xilinx **M**ass **S**torage **S**upport (**MSS**) example application has been used as a basis for this project.

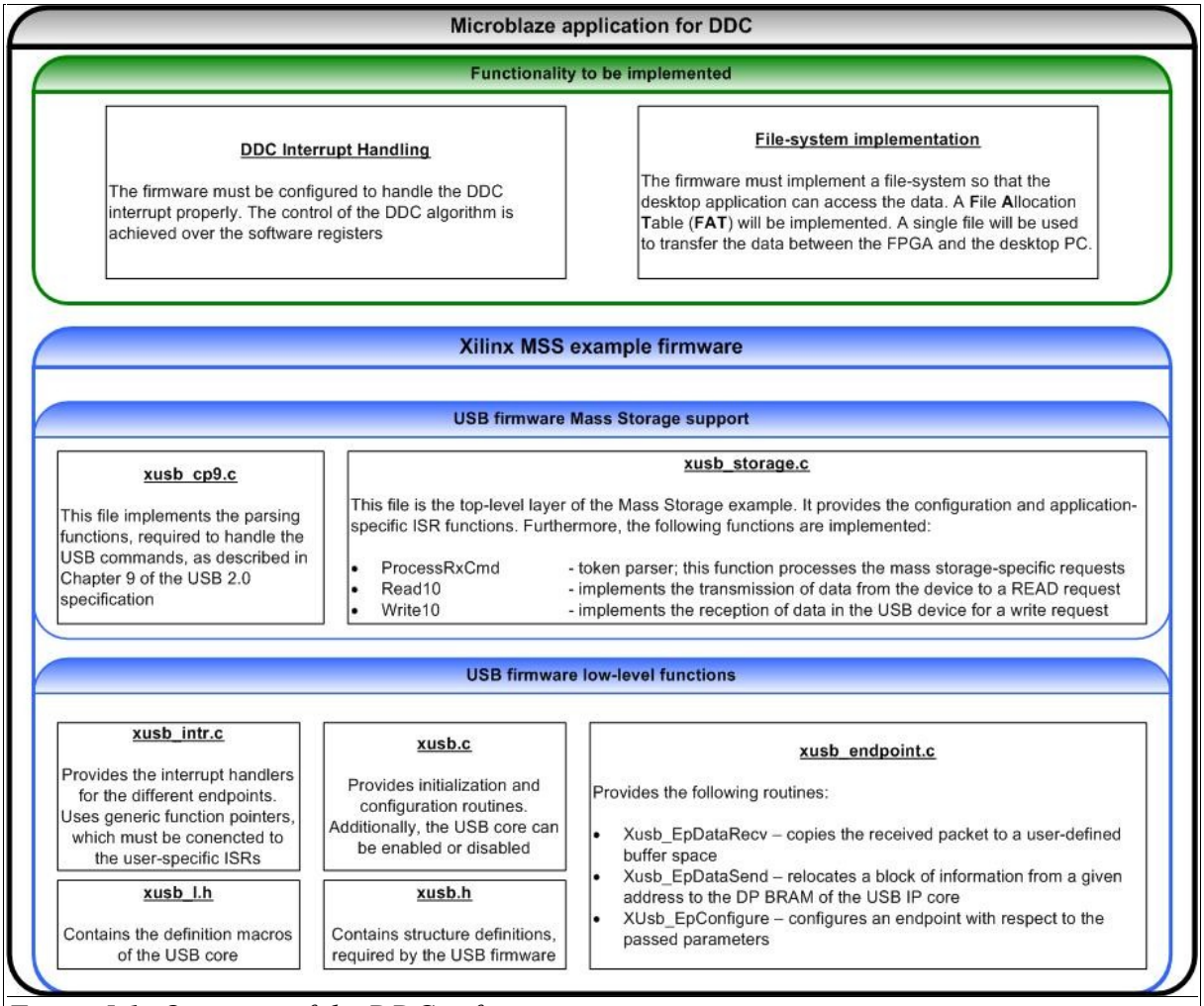


Figure 5.1: Overview of the DDC software project

The extension features the following two components:

- Interrupt handling for the DDC IP core

  The service routine should mark the number of interrupts received since the system is working. Each interrupt denotes that 64 sectors, each 512 bytes in size, have been written in memory. If a complete USB block, consisting of 10240 written sectors, is ready for transfer then the marker for the current block to be transferred should be updated. This should indicate to the desktop application that a new block is available.

- File-system implementation

  In order to make the DDR2 memory available to the USB interface, it should be managed by a file-system. This way, the desktop application can use the file, created by Microblaze, to transfer the data.

The software application configures two user endpoints and one control endpoint on the USB device. Endpoint 0 is configured as the control endpoint. Endpoint 1 is configured as BULK OUT and endpoint 2 is configured as BULK IN. Both endpoints are configured for a maximum packet size of 512 bytes. The remaining five endpoints are not used. According to [USBS], the benefits of Bulk only transfers are:

- Can be used to transfer large chunks of data
- CRC error detection
- Stream Pipe – unidirectional

- **File Allocation Table (FAT) and Data representation**

The initial state of the MSS example application is that the USB device is displayed as an unformatted one when connected to a host. This is due to the fact that the USB memory array is unmanaged. As a result, a file-system[31] should be introduced which will be responsible for the data management. Starting point here would be to implement a **FAT table** so that the complete memory is allocated and can be accessed by the desktop PC.

31 Mroe information on this topic can be found on Wikipedia, http://en.wikipedia.org/wiki/File_system [03.01.2011]

This type of file-system is characterized with a simple configuration interface. Every available sector is listed in a table and can be configured using the following markers:

- **0x0000** → marks a free cluster
- **0x0002 – 0xFFEF** → value of the next cluster in the cluster chain
- **0xFFF0 – 0xFFF6** → reserved cluster
- **0xFFF7** → BAD cluster
- **0xFFF8 – 0xFFFF** → used, last cluster in file

It has a limited sector count which is normally the label of the FAT version, 12, 16 or 32. This is the number of bits used to represent the maximum number of clusters supported. The total memory managed by the file system is simply the number of available clusters multiplied by the size of each sector. Some of the sectors are reserved and store status information used by the operating system when accessing the FAT table. Figure 5.2 provides a better overview of the FAT16 file system. As explained in [FAT16] there are several control blocks holding the complete information in a FAT:

- Boot Sector

  this sector holds the initialization code for the FAT table. Information such as sector size and count, as well as file-system type, is stored here.

- FAT Tables

  the size of the FAT table varies with the number of available sectors. The total number cannot go beyond 2^16, thus the name of the file-system – FAT16. Each available sector has a 16-bit pointer in the FAT table. The pointer contains one of the previously listed markers. For the case of this project, the table does not undergo any changes and as a results it is hard-coded during the initialization of the system.

- Root directory

  This section lists all created files and directories. Information such as filename, size, create date and create time are located there.

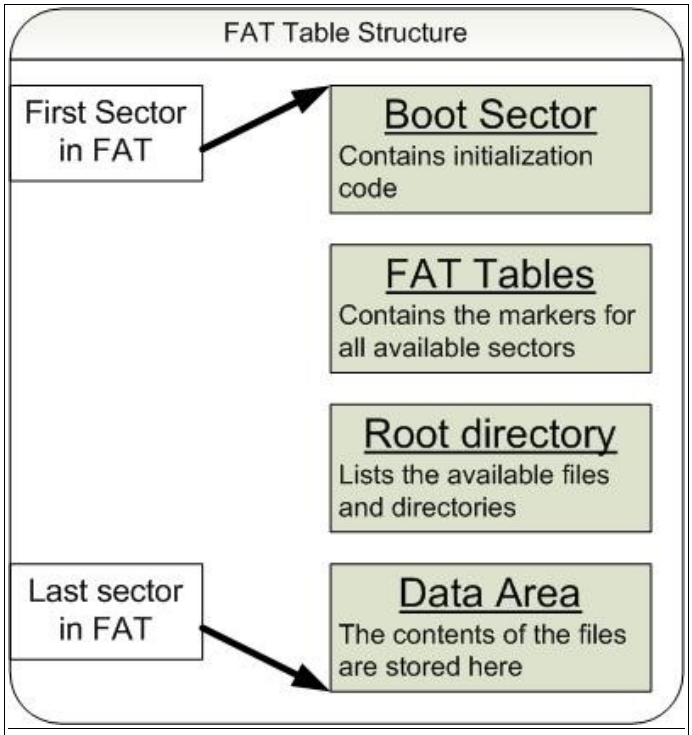


Figure 5.2: FAT Table overview

The easiest way to create the Boot Sector of the file-system is to let the operating system format the device. As a result, the Boot Sector is initialized with the required code and, if stored until the next plug in, the memory will appear in a formatted structure. Therefore, the first sector of the FAT table is always initialized with this code on start up. As next, a partition is configured with respect to the information located in the Boot Sector and a file is generated. This structure stores all relevant parameters of the file-system and provide easy software access.

As next, a file must be created. It will be used to transfer the data between the FPGA nad the desktop PC. The size of the file is 40961 Sectors * 512 Bytes = 20972032 Bytes. This value corresponds to the 20 MB, allocated for the ring buffer, discussed in Chapter 4.2.2.3, as well as a 512 byte long configuration sector, located at the very beginning of the memory, used to transfer control and status information. The following parameters, each 4 byte long, can be configured:

- Synchronization marker

  This field holds the system condition. Software can write this value to start the DDC system. Upon application termination, this marker should be updated to stop the hardware configuration.

- Current label of the sector to be transmitted

  This flag is updated directly in the DDC ISR. As soon as a new block is ready for transfer, its label is written in this field. The desktop application compares this value with its internal marker of the last read sector. If a change has been detected then the sector is read.

- DDS output frequency

  The desktop software application can use this field to set the DDS output frequency. This way, the mixer can be dynamically reconfigured and nearly any spectral region can be used as IF frequency.

The start address of the NPI interface and address limit of the DDR2 memory region are initialized and written to the registers of the core. The setup phase is then concluded by the USB IP core initialization settings and the interrupt configuration. The software then enters in an endless loop which processes any incoming commands.

The compiled application is used to initialize the internal BRAM memory of the Microblaze and it is loaded together with the hardware design. As soon as the system is started the hardware is ready for use and can be connected to a USB 2.0 compliant port.

5.2 Data Acquisition Application

After the overview of the Microblaze application it is time to pay attention to the software, responsible for the data acquisition. Not only is it important to make the DDR2 storage of the FPGA available to the USB bus, but additional timings regarding the operating system must be obeyed. Although the desktop PC is the master on the USB bus and issues all transactions, this is not the only task which consumes CPU power.

In general, when trying to establish a high performance communication between a device and a given operating system, the only choice is to design a device driver. Because this will . exceed the scope of this project and would result in long development period, another approach has been adopted. Given the fact that today's operating systems provide basic USB support, a generic driver can be used to manage the USB requests on the PC side. Because of

the fact that the DDC project requires only basic read/write USB support, using a generic driver would be sufficient. The communication files, created in the FPGA, will be then accessible by the operating system and data can be transferred.

The communication between the FPGA and the desktop PC is depicted on Figure 5.3.

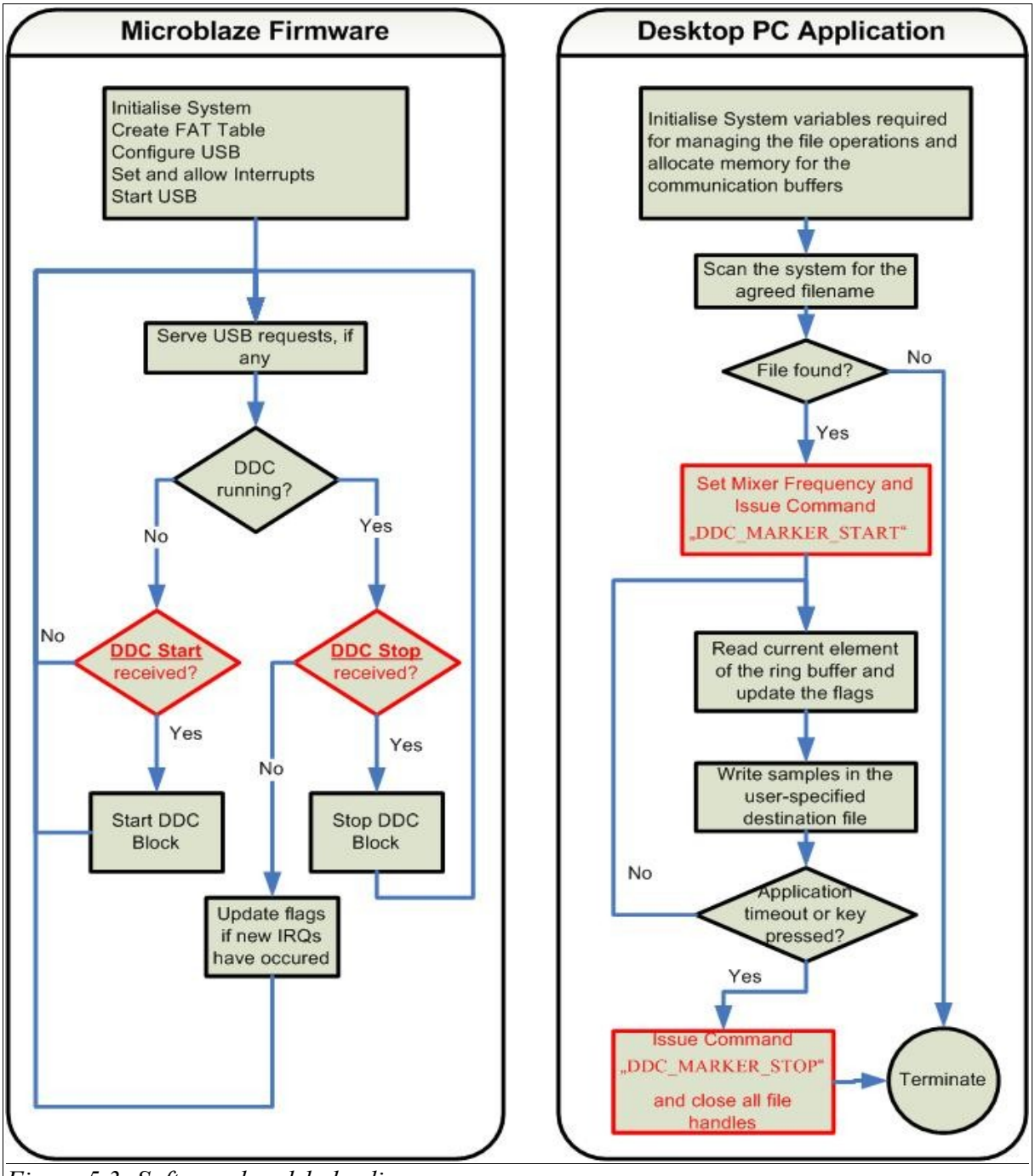


Figure 5.3: Software handshake diagram

As it can be seen, the desktop application configures the communication with one of the specific markers:

- DDC_MARKER_START

  This macro forces the Microblaze to enable the DC918C board so that new samples are acquired. The system is running as long as this marker is set.

- DDC_MARKER_STOP

  When the application software would like to terminate the data reception, it issues this command to the FPGA which stops the ADC device.

The next step concerning the data transmission is the design of the Acquisition software. For this purpose a Microsoft Visual Studio 2005 project has been created.

First of all, the application parameters, which are passed when the application is invoked, must be discussed. A total of three elements are required:

- Mixer frequency

  This allows the dynamic reconfiguration of the DDS at every run. The value must be in Hertz. The FPGA then internally calculates the new phase step and writes it in the DDC core.

- Destination filename

  In order to store different measurements, it has been considered useful to label them at every run. If the same filename is used twice then the former data is LOST!

- Runtime

  Due to the fact that the desktop application is the one, which trigger the data transmission process, a way must be found to terminate the communication link. For this reason, a runtime value, in milliseconds, is passed at the beginning of the application and the transfer process continues until a timeout has been detected.

Figure 5.4 depicts the flow of the desktop application.

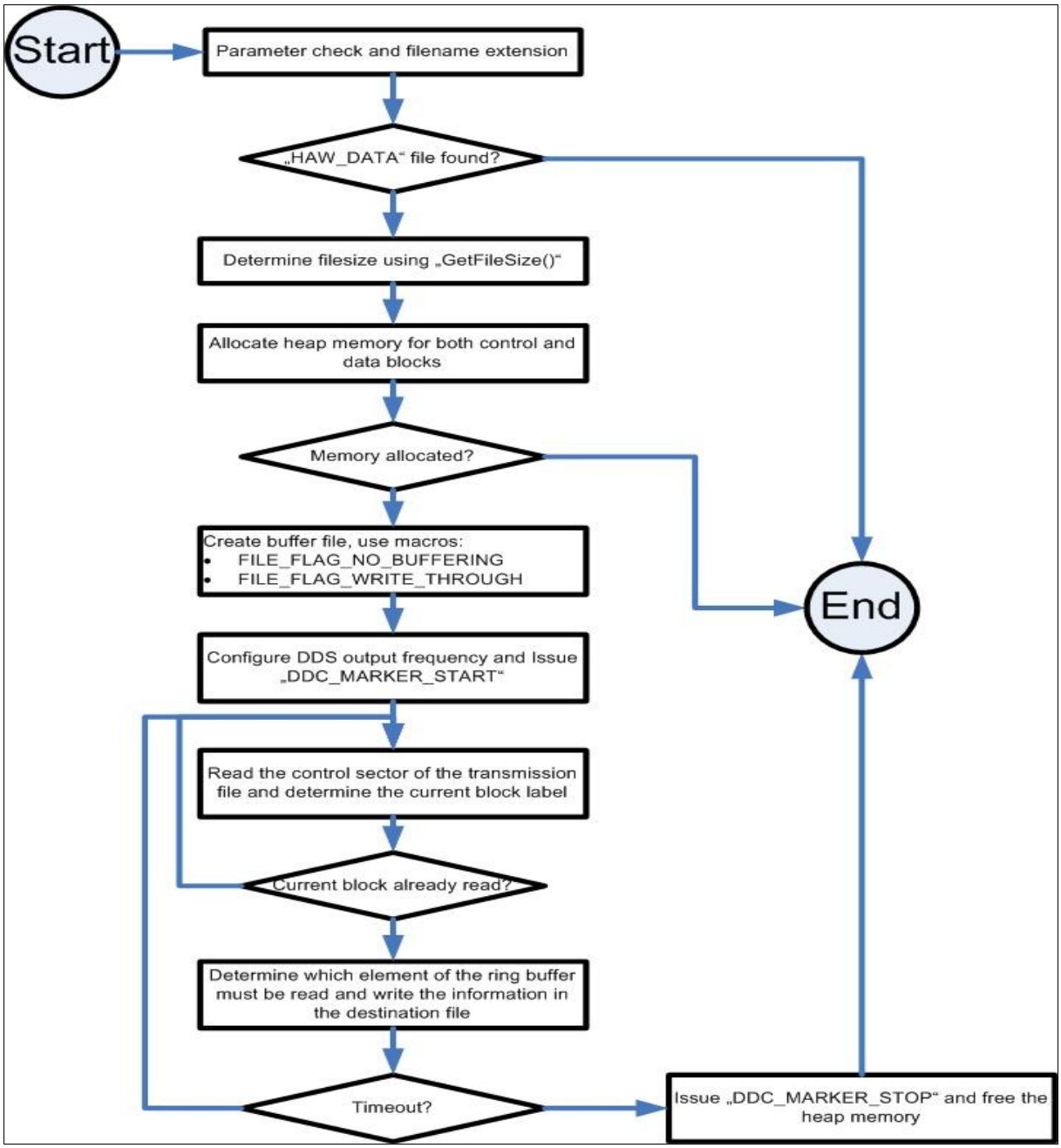


Figure 5.4: Overview of the desktop application structure

After the application is invoked, simple parameter checking is performed and the program terminates if any of the values is considered wrong. If all parameters are correct then the application searches for a filename called "HAW_DATA.txt", which is the one created in the FPGA. If the search is successful then heap memory is allocated for both the control and the data buffers. Furthermore, the storage file is created and its filename is extended to the maximum number of characters so that all filenames have the same length. This limitation is

necessary, because the Matlab script, which will be used to analyze the data, requires that all filenames have the same length. Moreover, an index at the end of the name is introduced, which is used to mark the files if their size exceed 100MB. A new file with the same name but incremented index is generated and the transmission process is not interrupted. The index is limited to 99 which means that if the received data exceeds 10GB or 33 minutes then a modification of the application is necessary.

One important remark, regarding the USB file access, must be mentioned. Due to the fact that the operating system assumes that it controls the FAT file-system, implemented in the FPGA, it is possible that changes, done by the Microblaze, may not be detected. For this reason, the following two attributes of the MSDN library function "CreateFileA" must be used:

- FLAG_NO_BUFFERING

  This flag removes any caching of information for the corresponding file, in this case "HAW_DATA.txt". This way, the operating system is forces to load the information every time it accesses the file. As a result, any change, done by the microprocessor, will be noticed and can be processed.

- FILE_FLAG_WRITE_THROUGH

  This attribute is used to directly route the data to the storage media. If omitted, the operating system uses intermediate caches before writing the information on the hard disc. For the reason that this is not required, this flag optimizes the data transfer.

As soon as the source file has been detected, heap memory is allocated and "DDC_MARKER_START" is written in the control block. This enables the FPGA algorithm and triggers the data transfer. As soon as the FPGA has gathered one block of data it sets the "current label" field with the acual value of the block and announces to the operating system that new data is available. The desktop application, on the other side, continuously polls the control block and compares the last processed label with the actual one. If a change has been detected then the corresponding block is read. As soon as the runtime-thread returns a timeout then the main loop is terminated and the "DDC_MARKER_STOP" is written in the control block. By doing this the FPGA is aware that no more data will be read and the ADC Enable bit can be cleared in Software Register 1. Eventually the desktop application closes all handles and frees the allocated heap memory. The application terminates and the data can be analyzed.

6 Results

First of all, the hardware configuration, consisting of the **ADC↔FPGA↔SMSC Dauther card** link, must be examined and the error-free transmission of information must be guaranteed. Furthermore, the DDC algorithm itself should be evaluated. Finally, the complete embedded solution must be examined and results should be compared with the theory.

6.1 Hardware Validation

Figure 6.1 depicts the interconnection layout of the system. As explained in Chapter 4.1, both the ADC and SMSC components are located on the expansion connectors J4 and J5, respectively.

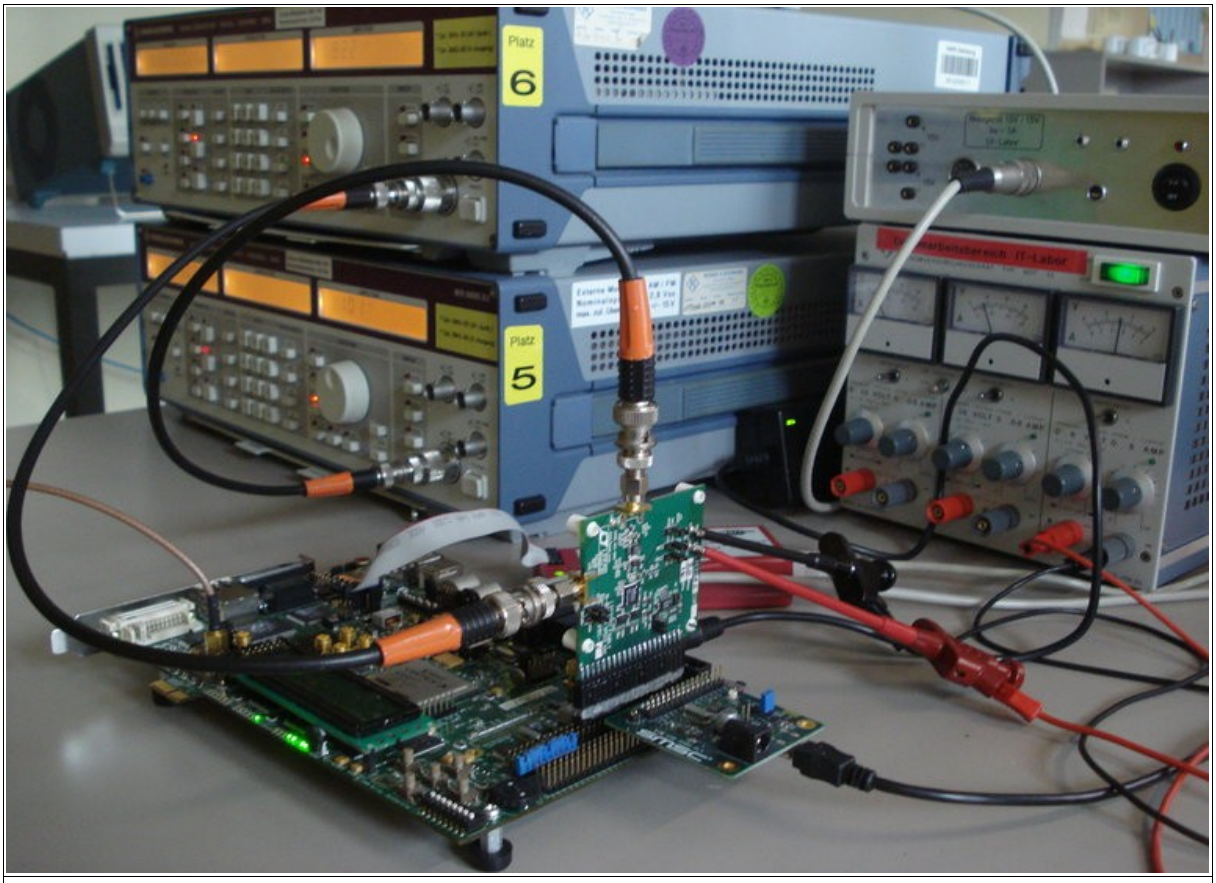


Figure 6.1: Design overview

Two signal generators of the type “Rohde & Schwarz SMG 0.1 MHz↔1000 MHz” have been employed to generate the ADC sampling clock and the input signal to be sampled. The sampling frequency has been set to 80 MHz and the complete DDC block is set up to that value. The DDS Compiler generates the mixing signals with respect to the sampling clock and the Undersampling principle, discussed in Chapter 3.4. Evaluating the listed equations for "n = 3" shows that the selected F_S = 80 MHz is in the allowed range.

A simple counter has been synthesized to demonstrate that all packets, sent over the USB bus, are received correctly. In contrast to the actual implementation, where the 16-bit InPhase and Quadrature components are concatenated, the counter generates a 32-bit value which is written to a single memory cell. With this in mind, there must be 1310720 values per block:

$$\frac{(5\mathrm{MB}_{(address\ space)})}{(4_{(bytes\ per\ value)})} = 1310720_{(values\ per\ USB\ sector)}.$$

Table 20 summarizes the size and limit values of the test blocks transmitted over the USB bus.

Sector Number	Start Address in HEX	Start Value in HEX	End Value in HEX	[(End – Start) * 4] in Bytes
0	0x00000000	0x00000000	0x0013FFFF	5242880
1	0x00500000	0x00140000	0x0027FFFF	5242880
2	0x00A00000	0x00280000	0x003FFFFF	5242880
3	0x00F00000	0x003C0000	0x004FFFFF	5242880
4	0x01400000	0x00500000	0x0063FFFF	5242880

Table 20: USB transmission validation

As it can be seen, the difference between the upper and lower address boundary of a 5MB sector equals 5*1024*1024 Bytes = 5242880_{10} = $0x00500000_{16}$, compare Table 20, column 2. Furthermore, due to the fact that a 32-bit, 4 Bytes, value is written at an address cell, a total of 5*1024*1024 Bytes / 4 Bytes = 1310720_{10} = $0x140000_{16}$ values are found in each sector, compare Table 20, columns 3 and 4.

6.2 Signal Quality Estimation

Since the transmission link has been verified, the next step would be to apply diverse RF test signals to the ADC board and validate their correct reception. First, the input signal will be visualized using the demonstration circuit DC718, which connects directly to the LTC DC918C board and together with an application-specific software provides good evaluation options. Additionally, measurement of an AM- as well as FM-modulated input waves should serve as a reference of how good the system performs.

6.2.1 Spectrum of the input signal

Before evaluating the test results, the spectrum of the input signal has been measured. As it can be seen on Figure 6.2, the SNR is 65dB and the two signals are clearly distinguished.

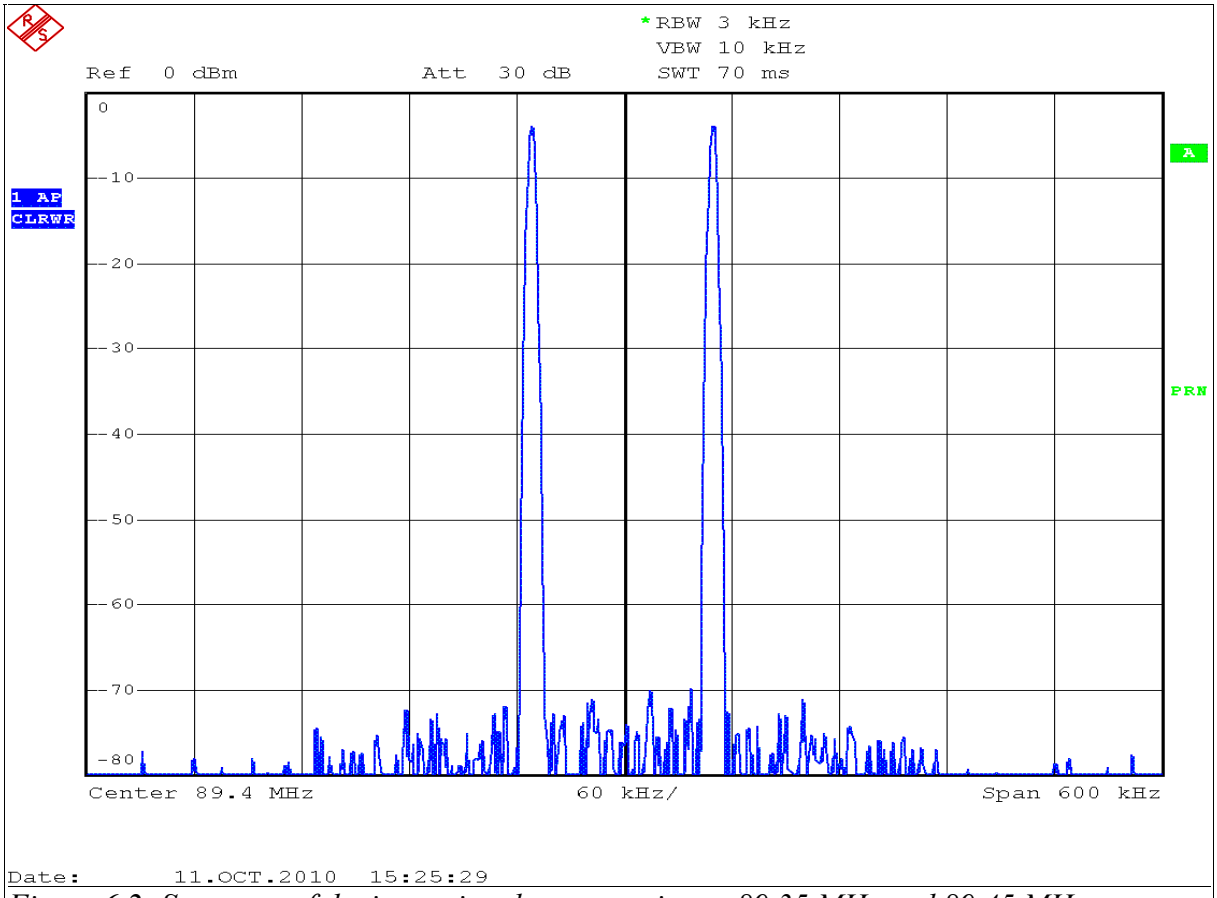


Figure 6.2: Spectrum of the input signal, two tone input, 89.35 MHz and 89.45 MHz

6.2.2 Test case 1 → Single-tone input

The first test to be performed is a measurement of a single-tone signal with the two different clock sources using the PScope software. Figure 6.3 and Figure 6.4 show the results.

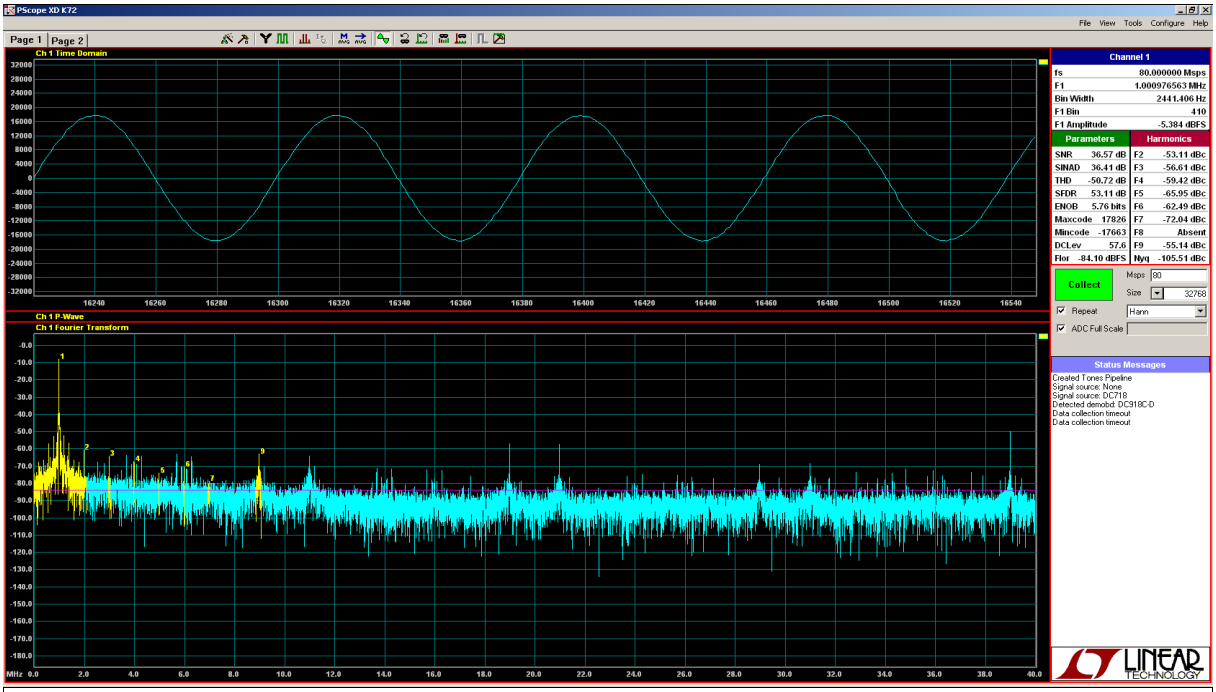


Figure 6.3: Single-tone measurement, FPGA DCM as clock source

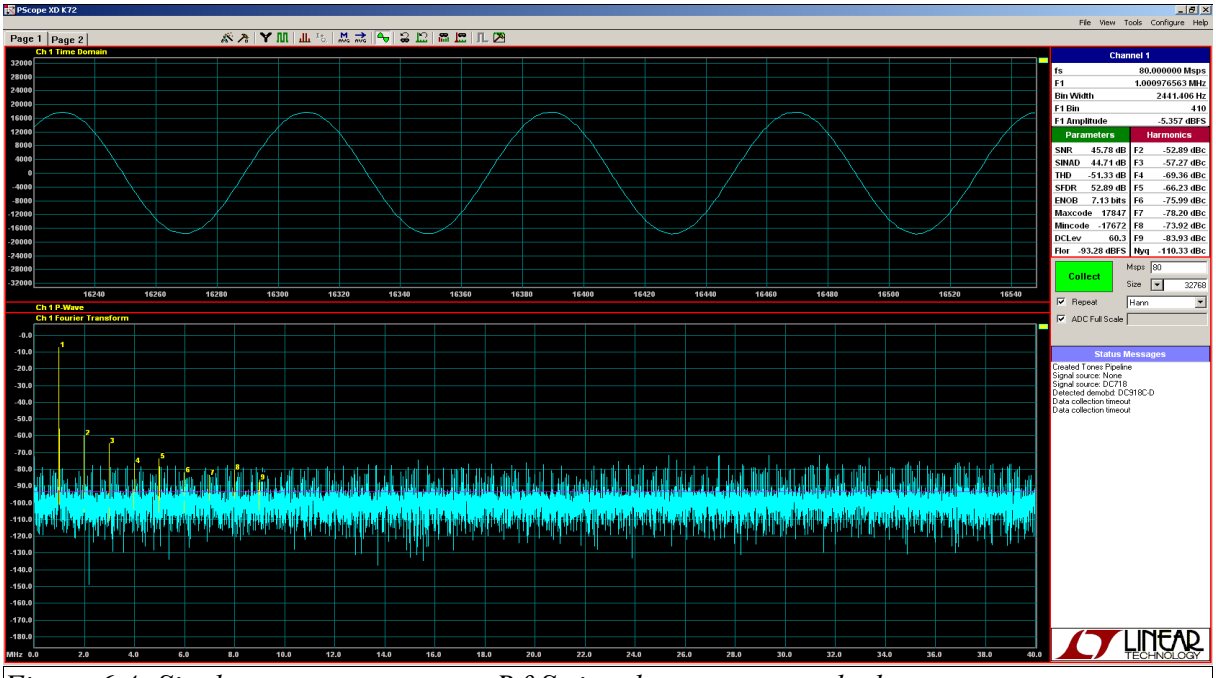


Figure 6.4: Single-tone measurement, R&S signal generator as clock source

Both measurements have been done using a Hanning window and an FFT size of 32768. As it can be seen, the spectrum of the input signal is distorted when sampled with the FPGA DCM clock. The leakage effect, seen around the main spectral component on Figure 6.3, has the effect that the original signal cannot be recovered and its energy "leaks" in the adjacent bins. In comparison, the second case provides a clear FFT plot and a SNR value of 45 dBc, which is with 10dB more then the former one. This reduction of the SNR can be justified with the duty cycle uncertainty as well as the period jitter, which were introduced in Chapter 3.5. For this reason, all further measurements have been taken using the R&S signal generator as source for the ENCODE input of the ADC circuit.

- **Amplitude measurements for a single-tone signal**

As next, several measurements have been taken to evaluate the amplitude deviation compared to the input signal. Table 21 lists the tested voltage levels. The same input sine wave with frequency f_{sin} = 89.45 MHz has been used for all test cases. The values for the input signal have been measured using the DC718 demonstration circuit and the PScope software, provided with the board, has been employed to visualize the measurements. The output of the DDC system has been evaluated using Matlab.

Input Voltage of the ADC Sine Wave(f_{sin} = 89.45 MHz) / mV_{RMS}	**PScope Measurement / converted value min/max**	**Output of the DDC system / converted value min/max**	**DDC system output deviation / in %**
101	-3539/3583	-3786/3787	~6.20
500	-17536/17625	-18785/18783	~6.43
682	-23905/23953	-25583/25581	~6.45
931	-32640/32687	-32768/32767	--
942	-32768/32767	-32768/32767	--

Table 21: Comparison of the peak-to-peak voltage samples, taken by the DDC sysetm

As it can be seen, the output of the DDC system, which is the sampled binary value of the peak-to-peak voltage of the input, deviates from the PScope measurements in about 6%. One reason for this is the rounding error which is introduced during the filter cascade computation.

6.2.3 Test case 2 → AM-modulated input

Amplitude modulation results in a variation of the carrier amplitude in proportion to the amplitude of the modulating signal. Figure 6.5 depicts an AM-modulted sine wave.

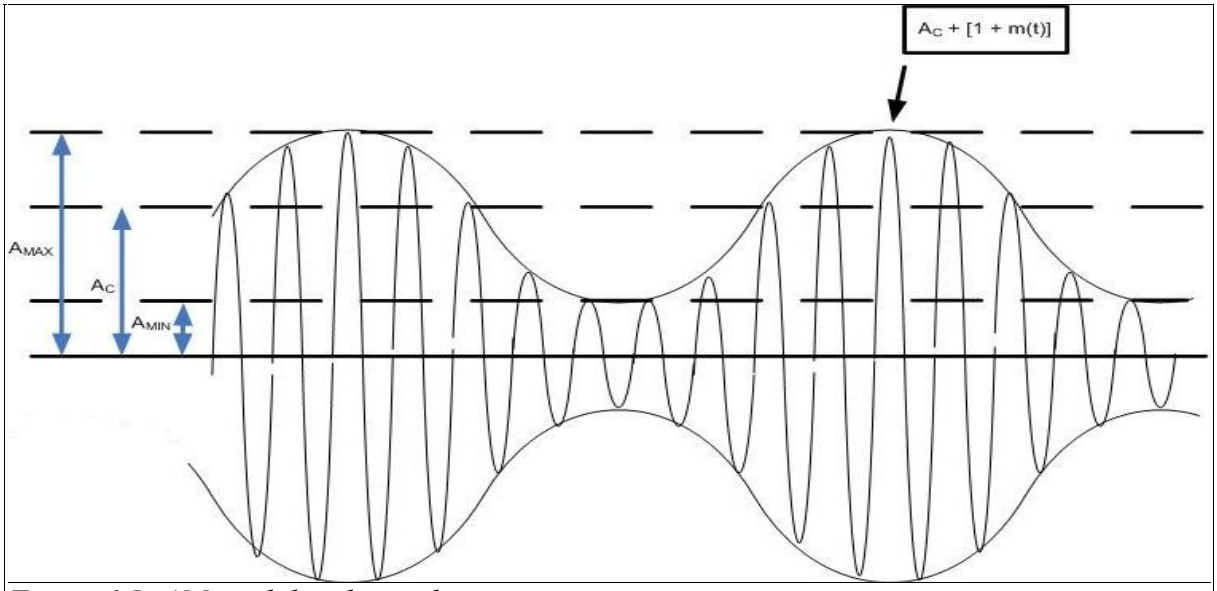


Figure 6.5: AM-modulated signal

The parameters A_{MAX} and A_{MIN} refer to the maximum and minimum of the signal's amplitude during the modulation. A_C denotes the amplitude in the unmodulated case. As it can be seen, the complex envelope of the signal is influenced by the modulating signal m(t). The degree of modulation, m, can be calculated using the following equation [DAC]:

$$m_{\%} = \frac{(A_{MAX} - A_{MIN})}{(2 * A_C)} * 100$$

Due to the characteristics of the modulated signal, it contains three different spectral components, the carrier and both lower and upper sidebands, which are the difference and the sum of the carrier and modulating frequency.

Figure 6.6 represents the AM-modulated signal which has been retrieved using the DDC system. The same input sine wave with frequency f_{sin} = 89.45 MHz and amplitude $A_C = 500mV_{RMS}$ has been used for all test cases. The AM modulation has been achieved using the internal 1 KHz source of the signal generator. For this reason three are three relevant spectral components and their ratio represents the degree of modulation.

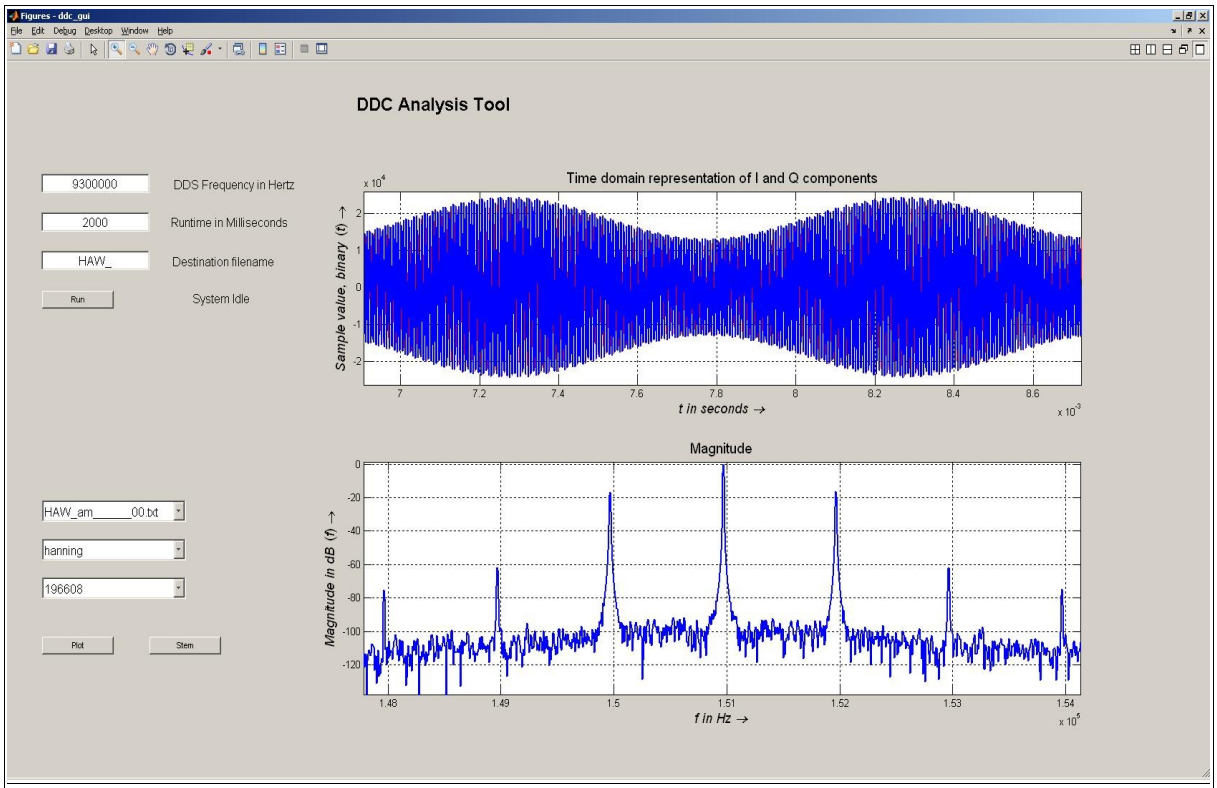


Figure 6.6: AM-modulated signal, degree of modulation 30%

The upper part of the image visualizes the time-domain representation of the I and Q components. The signal diagram is similar to the one, shown on Figure 6.5. the lower part represents the magnitude of the signal after a 192K FFT. In order to reduced the leakage effect a Hanning window has been applied before computing the FFT. The first pair of sideband peaks is located at ±1KHz with respect to the carrier frequency.

Different tests have been performed by varying the degree of modulation. A comparison between the expected and the measured values is listed in Table 22. The columns 2 and 3 contain the maximum and minimum measured values whereas columns 1 and 4 represent the configured and measured degree of modulation, respectively.

Degree of modulation, in %	A_{max} of the signal, 2's complement value	A_{min} of the signal, 2's complement value	Measured degree of modulation, in %
10	20508	16300	11,43
15	21551	15350	16,8
20	22483	14640	21,12
30	23119	12230	30,8

Table 22: Measurement results for the Amplitude Modulation test

The measurement values of the AM test show that there is a slight deviation of the theoretical value. One explanation for this error is the sampling process, which can only take a finite number of samples per period.

6.2.4 Test case 3 → FM-modulated input

Frequency modulation, together with Phase modulation, are commonly known as Angle modulation. This type of modulation is used in radio and television broadcasting as well as other signal transmission applications. Its advantage to provide increased noise immunity, even at the expense of higher bandwidth requirements, compared to AM modulation has been the main reason for FM being used in so many fields of analog and digital communication systems.

Frequency modulation can be defined as the instantaneous output frequency of a transmitter with respect to the modulating signal. The instantaneous voltage of a FM wave can be represented by the following equation [FM]:

$$e_{(FM(t))} = A_C * \sin(\omega_C * t + m_f * \sin(\omega_M * t))$$

The equation depends on both the carrier and modulating frequencies ω_C and ω_M, respectively, as well as on the rest-frequency peak amplitude, A_C, defined as the output frequency with no modulating signal applied. Furthermore, the value is influenced by the index of modulation, m_f. In contrast to AM, the amplitude of a FM-modulated signal is kept constant but an infinite

number of sidebands, equally spaced from each other by the modulation frequency, complement the spectrum. The amplitudes of the sidebands change with respect to the selected modulation index. The total power can be computed by evaluating the sum of the power values of the spectral components and should equal the power of the unmodulated signal. Figure 6.7 shows the spectrum of a FM-modulated signal for some modulation indices.

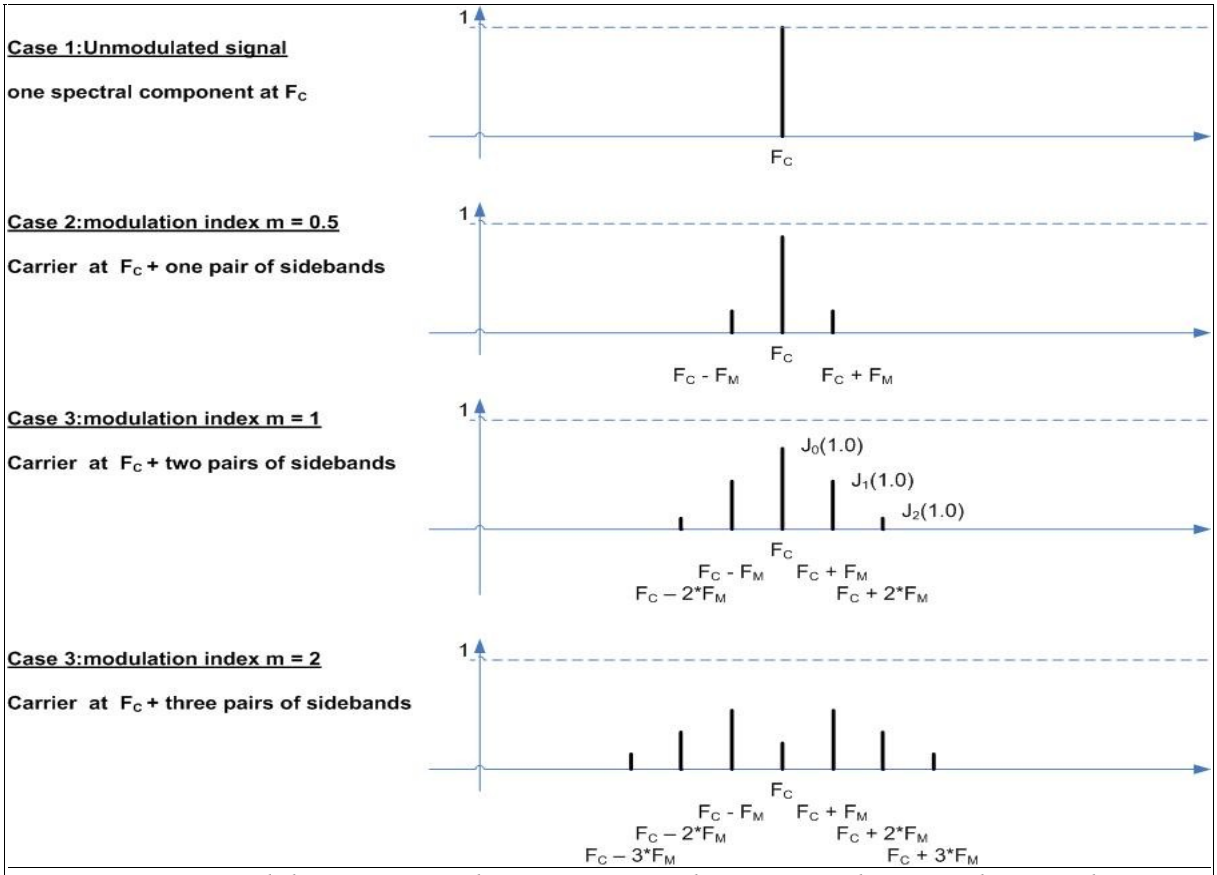


Figure 6.7: FM modulation, spectral components and connection between the Bessel function and index of modulation

The complex envelope of the output can be represented as follows:

$$g(t) = A_C * (e^{(j * m_f * \sin(w_m * t))})$$

Due to its periodicity with respect to the modulating frequency, the envelope can be expressed as a Fourier series [DAC, Chapter 5]:

$$g(t)=\sum_{n=-\infty}^{n=\infty}(c_n*(e^{(j*n*\omega_m*t)}))$$

with Fourier coefficients:

$$c_n=(\frac{A_C}{T_m})*[(\int_{-T_m/2}^{T_m/2}(e^{(j(m_f*\sin(\omega_m*t)))})*e^{(-j*n*\omega_m*t)}*dt)]=A_C*J_n(m_f)$$

For this reason, a way to express the relationship, between the carrier and the sidebands amplitudes, is to use the Bessel function representation, $J_n(m_f)$ in the above equation, depicted on Figure 6.8. As a matter of fact, each Bessel function contains a certain amount of the signal's energy. Plotting the magnitude of each Bessel function as a function of the index of modulation yields the power magnitude, contained in each spectral component, beginning from the carrier peak.

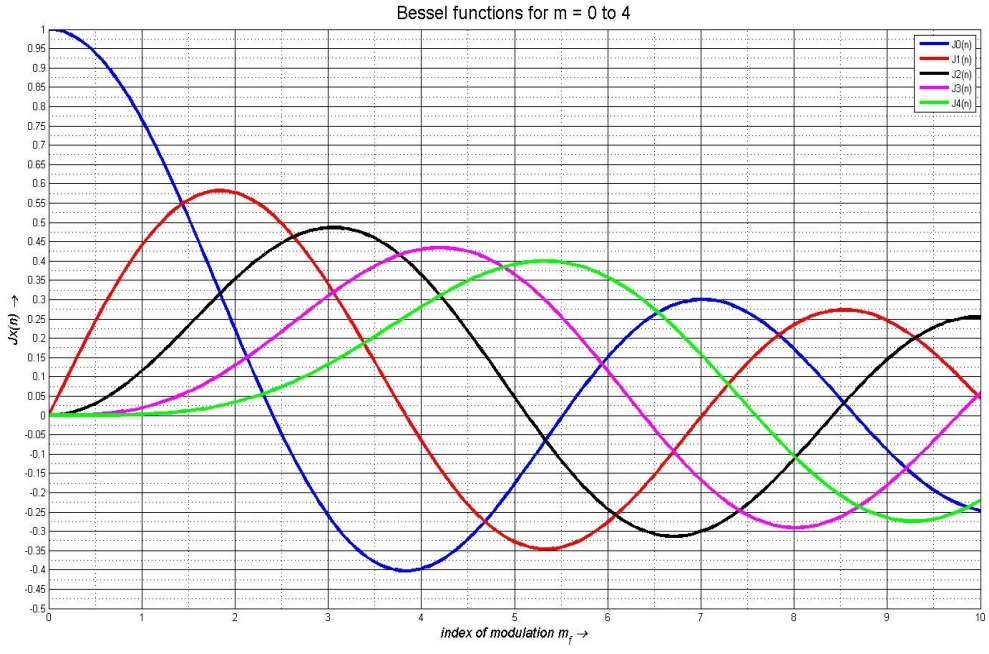


Figure 6.8: Bessel functions and relation to the index of modulation

The values from Table 23, marked as "expected", have been taken directly from Figure 6.8.

Figure 6.9 shows a sample view of the FM-modulated input signal after being processed by the DDC system.

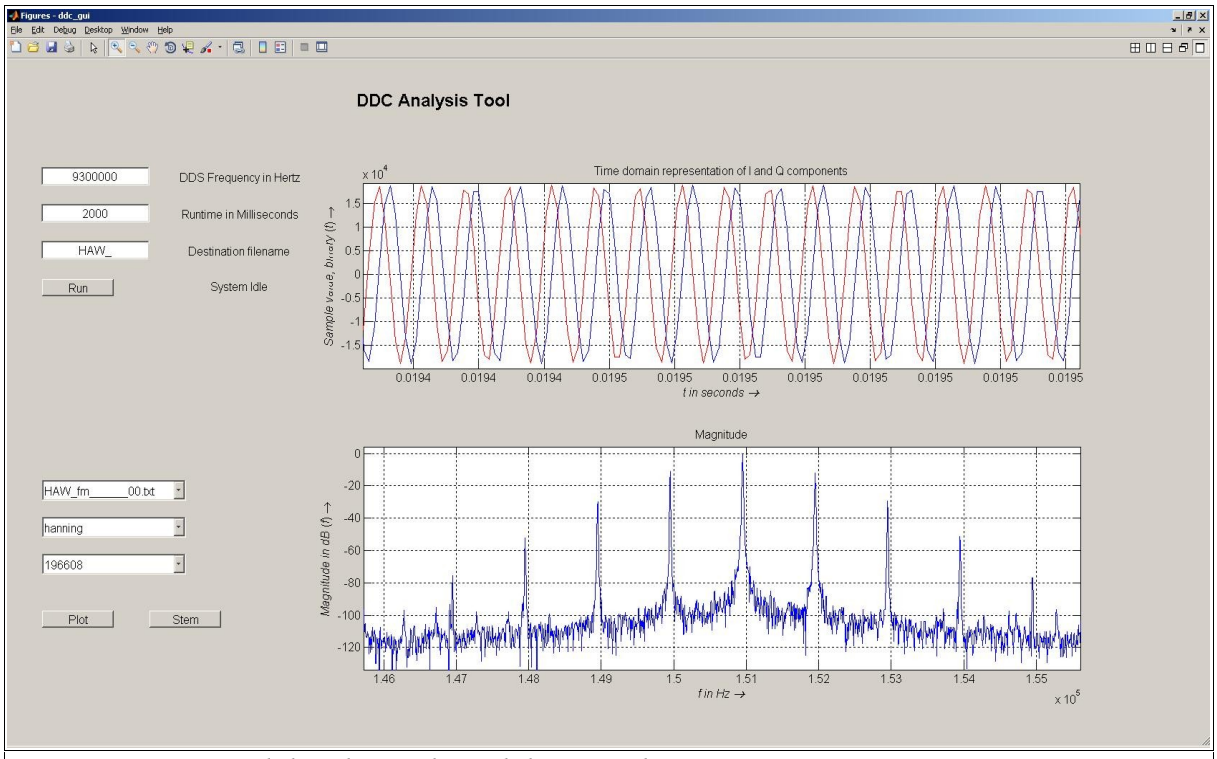


Figure 6.9: FM-modulated signal, modulation index m = 0.5

As it can be seen on the lower Magnitude plot, the carrier and sidebands are clearly distinguished and the measured amplitudes, summarized in Table 23, approach the theoretical expectations. The "Expected" values represent the theoretical estimation, whereas the "Measured" sections hold the data extracted using Matlab.

Bessel Func. ⟶	**$J_0(m_f)$**		**$J_1(m_f)$**		**$J_2(m_f)$**		**$J_3(m_f)$**		**$J_4(m_f)$**	
Mod. index, m_f ↓	**Expected**	**Measured**	**Expected**	**Measured**	**Expected**	**Measured**	**Expected**	**Measured**	**Expected**	**Measured**
0.5	0.9385	0.9431	0.2423	0.2402						
1	0.8652	0.7745	0.4401	0.4368	0.1149	0.1058				
2	0.2239	0.2417	0.5767	0.5786	0.3528	0.3250	0.1289	0.1105		
3	0.2601	0.2465	0.3391	0.3551	0.4861	0.4539	0.3091	0.2684	0.1320	0.1238

Table 23: Measurements for the FM-modulated signal

The measured results show that the system can detect the FM-modulated signal and that all relevant spectral information is received without significant losses.

7 Conclusion and Recommendations

As a conclusion, the main topics of this project will be summarized. At the beginning, the term “Software Defined Radio” has been used to introduce the reader to the field of real-time signal processing and Radio Frequency data transmission. The idea of a system, that manages the complete demodulation operation without using IF has been the main reason for establishing this master thesis. This work had the purpose to lay down the fundamentals of a SDR embedded system which should be able to recover a RF-signal and store it for further desktop processing. As the evolution of FPGA-based digital signal processing allowed for real-time computation of complex algorithms, the idea of shifting the appropriate processing blocks from the analog to the digital domain has been considered beneficial. The design of the complete communication link together with the implementation of the associated hardware components had the highest priority. A crucial role in the design plays the Direct-Down-Conversion algorithm, which has been employed because of its simplicity and efficiency.

After Chapter 2 presented the concept of this master thesis, Chapter 3 of this document introduced the theory concerning the DDC topic. Chapter 4 , Hardware Implementation and Chapter 5, Software Implementation, had the purpose to enlighten the used methodologies and programming tools to achieve the desired behavior of the hardware system, among which the VHDL implementation of the DDC IP Core and the microprocessor firmware extension. Finally, comprehensive tests have been performed to estimate the functionality of the architecture. Based on the measured results, listed in Chapter 6.2, it can be concluded that the design meets the project specifications, listed in Chapter 2.4. However, the estimated signal quality still differs from the theoretical values and an evaluation of the distortions may be necessary.

As already stated, this solution can be modified so that the resource utilization as well as design concept can be optimized. First of all, the filter cascade can be enhanced in a way that it consumes less resources but provides higher attenuation, which would result in better channel selection. An option for this would be a poly-phase FIR design, which may extract better filter characteristic and also lower the computation effort. Different combinations of a CIC and poly-phase FIR structures may yield better results in a particular case.

Furthermore, if additional flexibility is desired, then instead of using a function generator as

clock source, a Digital Clock Manager fabric may be used to clock the ADC board. Simple tests have been accomplished, compare Chapter 6.2.2, but due to jitter effects and duty cycle deviation the measured SNR degraded with 10dB. If, in contrast, the jitter is reduced then the ML507 Board can be used as a clock source for the DC918C demonstration circuit.

Eventually, if enhancement in the USB throughput is necessary, then the optional DMA interface of the core may be synthesized. A Master PLB interface allows the USB IP Core to take control of the PLB bus while the Microblaze is not active, compare Figure 7.1. This way, large chunks of memory can be moved independently of the microprocessor and by this leave the complete memory flow independent of the microprocessor.

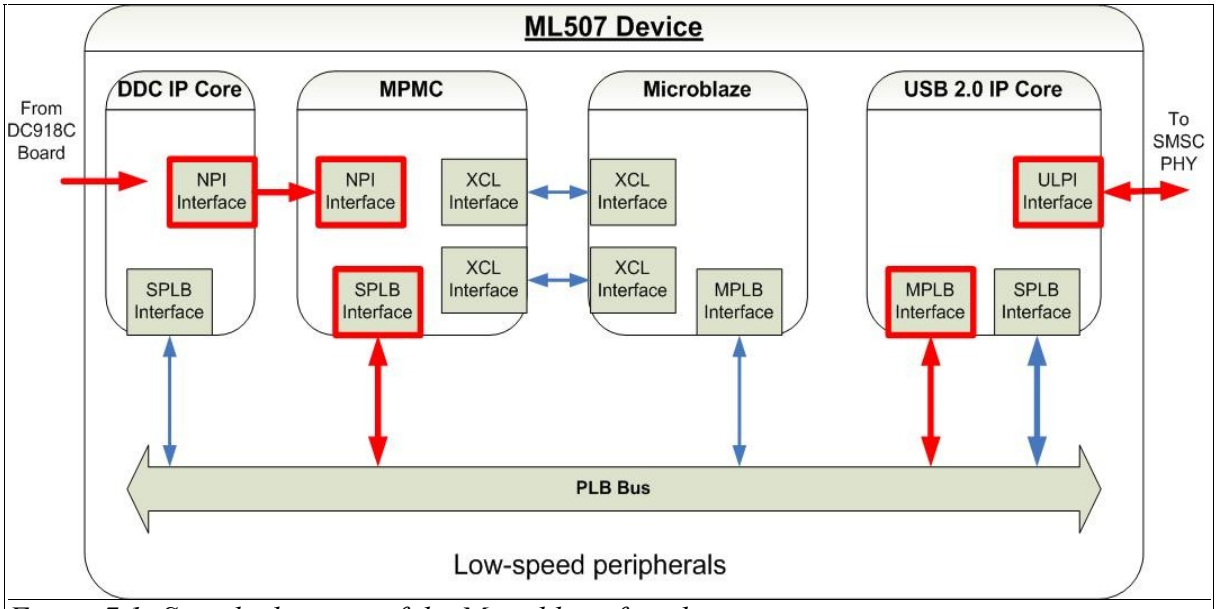


Figure 7.1: Sample diagram of the Microblaze-free data management

Finally, if the Mass Storage variant does not provide the required transmission speed then a USB streaming interface can be considered. This will come at the expense of a application-specific device-driver design, which may then optimize the communication between the FPGA board and the desktop computer.

Bibliography

AD_ADC: Analog Devices, Understand SINAD, ENOB, SNR, THD, THD + N, and SFDR so You Don't Get Lost in the Noise Floor ,
http://www.analog.com/static/imported-files/tutorials/MT-003.pdf , [12.01.2011]

AN1298: Agilent, Digital Modulation inCommunications Systems—An Introduction ,
http://cp.literature.agilent.com/litweb/pdf/5965-7160E.pdf , [12.01.2011]

AR24912: , 11.1 EDK, MPMC v5.00.a - How do I create an NPI Core and connect it to MPMC in EDK? ,
http://www.xilinx.com/support/answers/24912.htm , [12.01.2011]

CIC1: Matthew P. Donadio, CIC Filter Introduction ,
http://www.mikrocontroller.net/attachment/51932/cic2.pdf , [12.01.2011]

CLK_DOM: , Interfacing Two Clock Domains ,
http://www.asic-world.com/tidbits/clock_domain.html , [12.01.2011]

DAC: Leon W. Couch, Digital and Analog Communication Systems,
2006, ISBN 0-13-142492-0

DAC, Chapter 5: Leon W. Couch, Digital and Analog Communication Systems, 2006,

DC918: Linear Technology, QUICK START GUIDE FOR DEMONSTRATION CIRCUIT 918 ,
http://cds.linear.com/docs/Reference%20Design/dc918C_A-L.pdf , [12.01.2011]

DS558: Xilinx, LogiCORE IP DDS Compiler v4.0 ,
http://www.xilinx.com/support/documentation/ip_documentation/dds_ds558.pdf , [12.01.2011]

DS083: Xilinx, Virtex-II Pro and Virtex-II Pro X Platform FPGAs:Complete Data Sheet ,
http://www.xilinx.com/support/documentation/data_sheets/ds083.pdf , [26.01.2011]

DS100: , Virtex-5 Family Overview ,
http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf , [12.01.2011]

DS193: Xilinx, Virtex-5 FPGA XtremeDSP Design Considerations ,
http://www.xilinx.com/support/documentation/user_guides/ug193.pdf , [26.01.2011]

DSK: Texas Instruments, TMS320C6713 Floating-point Digital Signal Processor ,
http://focus.ti.com/lit/ds/symlink/tms320c6713.pdf , [26.01.2011]

FAT16: Jack Dobiash, FAT16 Structure Information ,
http://home.teleport.com/~brainy/fat16.htm , [12.01.2011]

FM: , Frequency Modulation ,
http://webtools.delmarlearning.com/sample_chapters/MU-04.PDF , [19.01.2011]

JIT: Bill Odom, National Semiconductors, Understand ADC clock jitter ,
http://www.eetindia.co.in/STATIC/PDF/200712/EEIOL_2007DEC17_SIG_TA_01.pdf?

SOURCES=DOWNLOAD , [27.01.2011]

LDT: Juergen Reichardt, Lehrbuch Digitaltechnik,
2009, ISBN 978-3-486-58908-5

MPMC: Xilinx, Multi-Port Memory Controller(MPMC) (v5.04.a) ,
http://gogo.jksw.cz/downloads/mpmc.pdf , [12.01.2011]

PSF: Xilinx, Platform Specification Format Reference Manual ,
http://www.xilinx.com/support/documentation/sw_manuals/edk10_psf_rm.pdf , [12.01.2011]

SP: John Proakis, Dimitris Manolakis, Digital Signal Processing, principles, algorithms and applications,
2007, ISBN 0-13-187374-1

SVG: Prof. Dr. Ulrich Sauvagerd, Interpolators_and_Decimators ,
http://users.etech.haw-hamburg.de/users/Sauvagerd/DSV_MES/protected/MES_DV_Decs_Ints.pdf , [01.02.2011]

UG190: Xilinx, Virtex-5 FPGA User Guide ,
http://www.xilinx.com/support/documentation/user_guides/ug190.pdf , [13.01.2011]

USB20: , Universal Serial Bus Specification ,
http://www.usb.org/developers/docs/usb_20_081810.zip , [12.01.2011]

USBS: , USB in a Nutshell ,
http://www.beyondlogic.org/usbnutshell/usb3.shtml#USBProtocols , [12.01.2011]

XAPP1113: Xilinx, Designing Efficient Digital Up and Down Converters for Narrowband Systems ,
http://www.xilinx.com/support/documentation/application_notes/xapp1113.pdf , [12.01.2011]

XPS_USB: Xilinx, LogiCORE IP XPS UniversalSerial Bus 2.0 Device (v5.00a) ,
http://www.xilinx.com/support/documentation/ip_documentation/xps_usb2_device.pdf , [12.01.2011]

Attachment A – List of the CD-contents

1. **VS_Masterthesis.pdf** →
   a) Master thesis document

2. **Project CoreGen** →
   a) Contains the cores, designed by the Xilinx CoreGenerator

3. **Project XPS** →
   a) Hardware project of the DDC system
   b) Software project of the Microblaze software application

4. **Projext SysGen** →
   a) System Generator project for the DDC model
   b) Initialization script of the DDC model
   c) Matlab script, used to analyze the data
   d) a simple file-search application, designed by Maximilien Chaumon

5. **Project VS2005** →
   a) Contains the Data Acquisition software application

6. **References** →
   a) Contains the refenreces, used in this master thesis in PDF format

Declaration

I declare that this Master Thesis has been completed by myself independently without outside help and only the defined sources and study aids were used. Sections that reflect the thoughts or works of others are made known through the definition of sources.

Hamburg, February 2011 _____ _____

Location, Date Signature